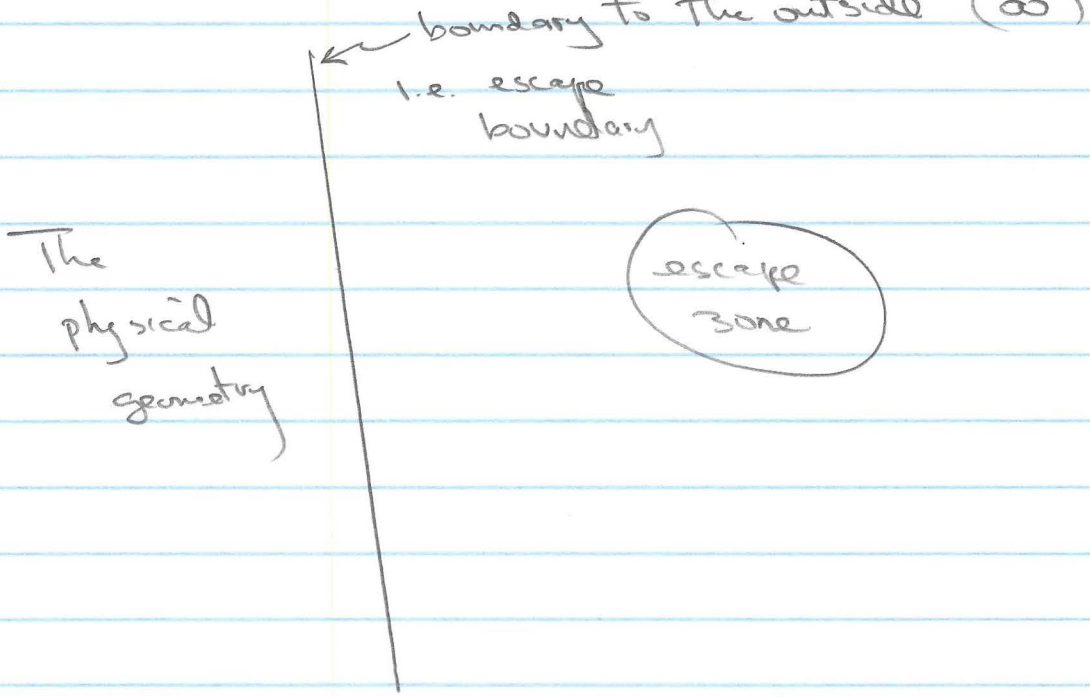
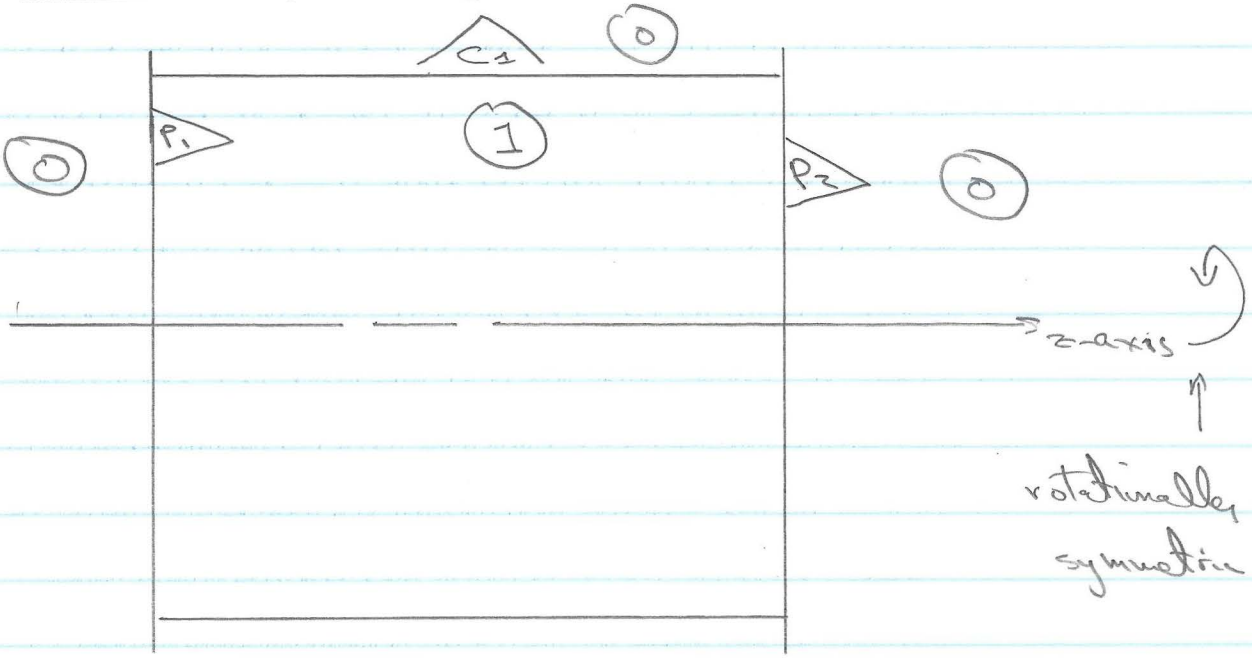
Lecture 8 The end of Geometry

Escape zones



There are several approaches with physics interpretations and efficiency considerations.

10.6 Using elemental volumes to make volumes



one zone $\bar{P}_1 \cap C_1 \cap P_2$

outside P_1, inside $P_2 \& C_1$

escape zone strategy 1)

- if a particle arrives or is created in zone (0) it is not transported, it is effectively discarded, and a new "history" is started
its angle of trajectory is ignored

```
% p1p2c1es1.m short for Plane 1, Plane2, Cylinder 1, escape zone strategy 1
% Class Example: See the hand-written notes L8 pages 1 and beyond

clear all, close all, format compact

diagnostics = true;   % Reorder this and the following statement to enable
diagnostics = false;
if (diagnostics)  disp('Diagnostic mode is on: <return> to continue.'), pause, end

% Geometry definition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Region 1 physical volume: Outside P1, inside P2, inside C1
% Region 0 escape region  : everything not in Region 1
Nr = 1; % Number of elemental subvolumes
z1 = 0; % P1, z = z1
z2 = 1; % P2; z = z2
R  = 1; % C1 circular cylinder x^2 + y^2 -R^2 = 0
L = z2 - z1;
rplane = R/(R + L); % probability that a particle emanating from the surface
                    % originates on one of the planes.

% Tally set up %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xspot = []; % Tally vector used for plotting later

% Diagnostic tallies
vsrhofails = 0; % Volume source rho failure counter
vszfails   = 0; % Volume source z   failure counter
ssrhofails = 0; % Surface source rho failure counter

% Source type %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
choice = input('0 for surface source, 1 for volume source: ' );
if     (choice == 0) surfaceSource = true; volumeSource  = false;
elseif (choice == 1) surfaceSource = false; volumeSource  = true;
else   disp('No choice for source type, exiting'), return, end

if     (surfaceSource) disp('Surface source is modeled')
elseif (volumeSource)  disp('Volume source is modeled'), end

% Scattering type %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
choice = input('0 for forward, 1 for backward, 2 for isotropic scattering: ' );
if (choice == 0)
   scatteringForward   = true;
   scatteringBackward  = false;
   scatteringIsotropic = false;
elseif (choice == 1)
   scatteringForward   = false;
   scatteringBackward  = true;
   scatteringIsotropic = false;
elseif (choice == 2)
   scatteringForward   = false;
   scatteringBackward  = false;
   scatteringIsotropic = true;
else disp('No choice for scattering type, exiting'), return, end

if (scatteringBackward)  disp('Backward scattering is modeled' ), end
if (scatteringForward)   disp('Forward scattering is modeled'  ), end
if (scatteringIsotropic) disp('Isotropic scattering is modeled'), end

% Simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nh = 1000; % Number of histories to simulate
for n = 1:Nh
   if (diagnostics) n, end
   % Sample the source ==========================================================
   r = 1; % Source particle starts in region 1
   if (volumeSource)
```

```
   % Define the starting position vector ----------------------------------
   % Samping loop for x and y
   repeat = true; % This is a rejection loop for circular collimation
   while (repeat)
      x(1) = 1 - 2*rand;
      x(2) = 1 - 2*rand;
      if (x(1)^2 + x(2)^2 < 1) repeat = false; % Note the "<" logical operator
      else vsrhofails = vsrhofails + 1; end % Tally the fails
   end
   x = R*x;
   % Samping loop for z
   repeat = true;
   while (repeat)
      x(3) = z1 + (z2 - z1)*rand;
      if (x(3) > z1 && x(3) < z2) repeat = false; % Note the "<" logical operator
      else vszfails = vszfails + 1; end % Tally the fails
   end
elseif (surfaceSource)
   % Define the starting position vector ----------------------------------
   if (rand < rplane) % Particle emerges from a plane
      % Samping loop for x and y
      repeat = true; % This is a rejection loop for circular collimation
      while (repeat)
         x(1) = 1 - 2*rand;
         x(2) = 1 - 2*rand;
         if (x(1)^2 + x(2)^2 < 1) repeat = false; % Note the "<" logical operator
         else ssrhofails = ssrhofails + 1; end
      end
      x = R*x;
      % Pick a plane, left or right
      if (rand < 0.5) % Left plane is chosen
         x(3) = z1;
         if (diagnostics) disp(sprintf('Starting on plane at z = %g', z1)), end
      else
         x(3) = z2;
         if (diagnostics) disp(sprintf('Starting on plane at z = %g', z2)), end
      end
   else % particle emerges from the surface
      phi = 2*pi*rand; x(1) = R*cos(phi); x(2) = R*sin(phi); x(3) = z1 + (z2 - z1)*rand;
      if (diagnostics)
         disp(sprintf('Starting on cylinder at x = [%g,%g,%g]',x(1),x(2),x(3)))
      end
   end
end % Starting position now given

xspot(end + 1,:) = x; % Tally the starting point

% Define a random starting direction vector
u = rotate([0,0,1],1 - 2*rand); % Compact use of the rotate function

% Transport through the geometry ===========================================
while (r > 0 & r <= Nr) % True if particle is inside the geometry
   t0 = inf;            % Distance to interaction (in vacuum)

   if (diagnostics)
      str = 'Before transport r:x:u,t0: = %g:[%g,%g,%g]:[%g,%g,%g],%g';
      disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3),t0))
   end

   if (r == 1) % If particle is in region 1
      [hit,t] = zplane(z1,x(3),u(3),false); if (hit & (t <= t0)) r = 0; t0 = t; end
      [hit,t] = zplane(z2,x(3),u(3),true ); if (hit & (t <= t0)) r = 0; t0 = t; end
      [hit,t] = ccylz(R,x,u,true         ); if (hit & (t <= t0)) r = 0; t0 = t; end
   end
   x = x + u*t0; % Transport the particle
```

sample within a unit circle

scale to arbitrary R, radius

this would be really rare

```
      if (diagnostics)
         str = 'After  transport r:x:u,t0: = %g:[%g,%g,%g]:[%g,%g,%g],%g';
         disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3),t0))
      end

      % Deflect the particle at the end of the transport step (should have no effect)
      if     (scatteringForward  ) w =  1;
      elseif (scatteringBackward ) w = -1;
      elseif (scatteringIsotropic) w =  1 - 2*rand; end
      u = rotate(u,w); % Scetter the particle
      if (diagnostics)
         str = 'After  rotation  r:x:u =     %g:[%g,%g,%g]:[%g,%g,%g]';
         disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3)))
      end
      xspot(end + 1,:) = x; % Record the intercept point

      if (diagnostics) pause, end

   end % End of the transport loop ==========================================

   if (diagnostics) % Graphical output each history in diagnostic mode only
      figure(1)
      plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
      xlabel('x')
      ylabel('y')
      zlabel('z')
      axis([-2,2,-2,2,0,1])
      pause
   end

end % End of simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Outputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1)
plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
xlabel('x'), ylabel('y'), zlabel('z'), title('Full geometry')
axis([-2,2,-2,2,0,1])

figure(2)
plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
xlabel('x'), ylabel('y'), zlabel('z'), title('Chopping the ends off')
title('Chopping the ends off')
axis([-2,2,-2,2,0.001,0.999])

% Diagnostic tallies (always) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (volumeSource)
   f = vsrhofails/Nh;
   s = sqrt(f*(1-f)/Nh);
   t = 1-pi/4;
   t = t/(1-t); % Account for multiple misses
   disp(sprintf('Fraction of rho rejections = %g +/- %g compared to theory %g',f, s, t))
   disp(sprintf('Fraction of z rejections = %g compared to theory %g',vszfails/Nh, 0))
elseif (surfaceSource)
   f = ssrhofails/Nh;
   s = sqrt(f*(1-f)/Nh);
   t = 1-pi/4;
   t = t/(1-t)*rplane; % Account for multiple misses, note the rplane fraction
   disp(sprintf('Fraction of rho rejections = %g +/- %g compared to theory %g',f, s, t))
end

pause, close all
```

In this escape zone strategy, which is most suitable when the intersection with geometry does not cause a deflection, is efficiency.

Counter examples : - some charged particle algs
- mirrored surfaces

Pros efficient
Cons see above + no info transmitted regarding where particles leave the geometry

Some new features in this code

Volume source — discuss
Surface source — discuss

Diagnostic mode

Note the diagnostics switch where much info, useful for debugging is printed (very useful)
In diagnostic mode
A full phase space should be output everytime any element of phase space is changed

$\Delta \vec{x} \neq 0$

$\Delta \vec{u} \neq 0$

$\Delta r \neq 0$ region change

$\Delta E \neq 0$ energy

creation (source)

escape

.

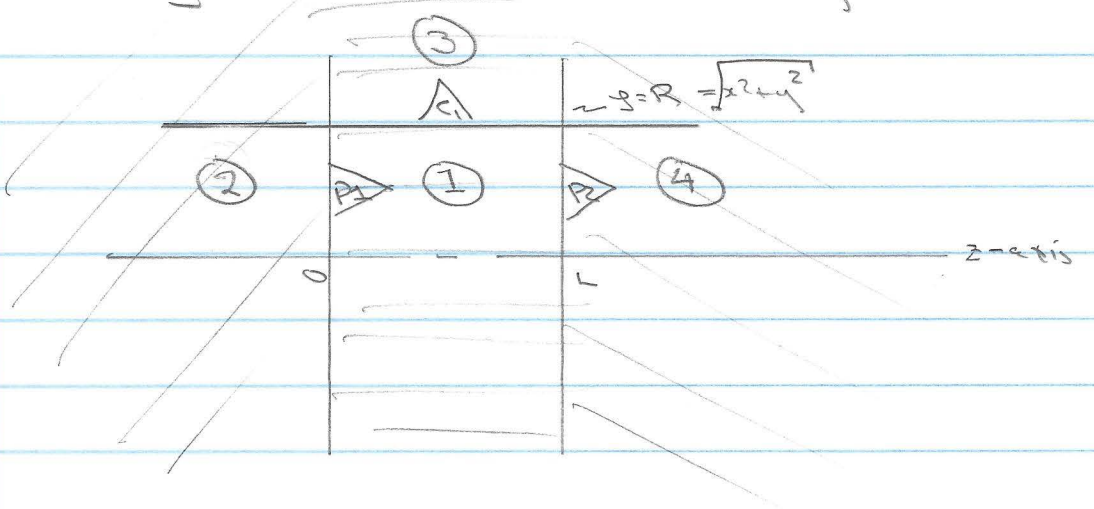
.

.

next topic

escape strategy 2) delineate all of space

Consider the following object, the volume enclosed by $P1: z=0$, $P2: z=L$, $C1: x^2+y^2-R^2=0$



Region #	Location	Description
(1)	$\overline{P_1} \cap C_1 \cap P_2$	outside P1, inside C_1 & P_2
(2)	P_1	inside P1
(3)	$\overline{C_1} \cap \overline{P_1} \cap P_2$	outside C_1, P_1, inside P_2
(4)	$\overline{P_2}$	outside P_2

note: <u>all</u> of space is "numbered"

geometry code: $z_1=0$, $z_2=L$, R all defined

r → region number of particle being transported

Nr total # of regions including escape zone

tφ: proposed transport distance

```
% z1=0, z2=L, R, r, Nr, tφ, xφ are all known
while (r>0 & r<=Nr)
    ri=r; % starting region#
    if (r==1)  % particle is in ①, the physical geometry
        [hit,t]=zplane(z1,x(3),u(3),false); % outside z1
        if (hit & t<=tφ) r=2; tφ=t; end
        [hit,t]=zplane(z2,x(3),u(3),true); % inside z2
        if (hit & t<=tφ) r=4; tφ=t; end
        [hit,t]=ccylz(R,x,u,true); % inside C1
        if (hit & t<=tφ) r=3; tφ=t; end
    elseif (r==2) % Left of z1
        [hit,t]=zplane(z1,x(3),u(3),true); % inside z1
        if (hit & t<=tφ) r=1; tφ=t; end
        else r=0; end
    elseif (r==4) % Right of z2
        if (hit & t<=tφ) r=1; tφ=t; end
        else r=0; end
    elseif (r==3)
        [hit,t]=zplane(z1,x(3),u(3),false); % outside P1
        if (hit & t<=tφ) r=2; tφ=t; end
        [hit,t]=zplane(z2,x(3),u(3),true); % inside P2
        if (hit & t<=tφ) r=4; tφ=t; end
        [hit,t]=ccylz(R,x,u,false); % outside C1
        if (hit & t<=tφ) r=2; tφ=t; end
    end  % all space is covered
    x=xφ+u*t;
    if (r==ri)  % Particle stays in the same region
        % Scattering would go here
    end
```

region ① check

check region ②

check ④

check region ③

not really needed see code

Pro's:

diagnostics re exit region can be captured

no new surfaces are created

next topic

Escape strategy 3)

minimum encapsulation

```
% p1p2c1es1.m short for Plane 1, Plane2, Cylinder 1, escape zone strategy 2
% Class Example: See the hand-written notes L8 pages 8 and beyond

clear all, close all, format compact

diagnostics = true;
diagnostics = false; % Comment out this statement to enable diagnostics
if (diagnostics)  disp('Diagnostic mode is on: <return> to continue.'), pause, end

% Geometry definition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Region 1 physical volume: Outside P1, inside P2, inside C1
% Region 2: escape zone   : Inside  P1
% Region 3: escape zone   : Outside P1, inside P2, Outside C1
% Region 4: escape zone   : Outside P2
Nr = 4; % Number of elemental subvolumes
z1 = 0; % P1, z = z1
z2 = 1; % P2; z = z2
R  = 1; % C1 circular cylinder x^2 + y^2 -R^2 = 0
L = z2 - z1;
rplane = R/(R + L); % probability that a particle emanating from the surface
                    % originates on one of the planes.

% Tally set up %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xspot = []; % Tally vector used for plotting later
esc = zeros(Nr,1);; % Tallies escapes from any region

% Diagnostic tallies
vsrhofails = 0; % Volume source rho failure counter
vszfails   = 0; % Volume source z   failure counter
ssrhofails = 0; % Surface source rho failure counter

% Source type %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
choice = input('0 for surface source, 1 for volume source: ' );
if     (choice == 0) surfaceSource = true;  volumeSource  = false;
elseif (choice == 1) surfaceSource = false; volumeSource  = true;
else   disp('No choice for source type, exiting'), return, end

if     (surfaceSource) disp('Surface source is modeled')
elseif (volumeSource)  disp('Volume source is modeled'), end

% Scattering type %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
choice = input('0 for forward, 1 for backward, 2 for isotropic scattering: ' );
if (choice == 0)
   scatteringForward   = true;
   scatteringBackward  = false;
   scatteringIsotropic = false;
elseif (choice == 1)
   scatteringForward   = false;
   scatteringBackward  = true;
   scatteringIsotropic = false;
elseif (choice == 2)
   scatteringForward   = false;
   scatteringBackward  = false;
   scatteringIsotropic = true;
else disp('No choice for scattering type, exiting'), return, end

if (scatteringBackward)  disp('Backward scattering is modeled' ), end
if (scatteringForward)   disp('Forward scattering is modeled'  ), end
if (scatteringIsotropic) disp('Isotropic scattering is modeled'), end

% Simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nh = 1000; % Number of histories to simulate
for n = 1:Nh
   if (diagnostics) n, end
```

```
% Sample the source ========================================================
r = 1; % Source particle starts in region 1
if (volumeSource)
   % Define the starting position vector ----------------------------------
   % Samping loop for x and y
   repeat = true; % This is a rejection loop for circular collimation
   while (repeat)
      x(1) = 1 - 2*rand;
      x(2) = 1 - 2*rand;
      if (x(1)^2 + x(2)^2 < 1) repeat = false; % Note the "<" logical operator
      else vsrhofails = vsrhofails + 1; end % Tally the fails
   end
   x = R*x;
   % Samping loop for z
   repeat = true;
   while (repeat)
      x(3) = z1 + (z2 - z1)*rand;
      if (x(3) > z1 && x(3) < z2) repeat = false; % Note the "<" logical operator
      else vszfails = vszfails + 1; end % Tally the fails
   end
elseif (surfaceSource)
   % Define the starting position vector ----------------------------------
   if (rand < rplane) % Particle emerges from a plane
      % Samping loop for x and y
      repeat = true; % This is a rejection loop for circular collimation
      while (repeat)
         x(1) = 1 - 2*rand;
         x(2) = 1 - 2*rand;
         if (x(1)^2 + x(2)^2 < 1) repeat = false; % Note the "<" logical operator
         else ssrhofails = ssrhofails + 1; end
      end
      x = R*x;
      % Pick a plane, left or right
      if (rand < 0.5) % Left plane is chosen
         x(3) = z1;
         if (diagnostics) disp(sprintf('Starting on plane at z = %g', z1)), end
      else
         x(3) = z2;
         if (diagnostics) disp(sprintf('Starting on plane at z = %g', z2)), end
      end
   else % particle emerges from the surface
      phi = 2*pi*rand; x(1) = R*cos(phi); x(2) = R*sin(phi); x(3) = z1 + (z2 - z1)*rand;
      if (diagnostics)
         disp(sprintf('Starting on cylinder at x = [%g,%g,%g]',x(1),x(2),x(3)))
      end
   end
end % Starting position now given

xspot(end + 1,:) = x; % Tally the starting point

% Define a random starting direction vector
u = rotate([0,0,1],1 - 2*rand); % Compact use of the rotate function

% Transport through the geometry ============================================
while (r > 0 & r <= Nr) % True if particle is inside the geometry
   t0 = inf;            % Distance to interaction (in vacuum)

   if (diagnostics)
      str = 'Before transport r:x:u,t0: = %g:[%g,%g,%g]:[%g,%g,%g],%g';
      disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3),t0))
   end

   ri = r; % Record the starting region number for later
   if (r == 1) % Particle is in region 1
      [hit,t] = zplane(z1,x(3),u(3),false); if (hit & (t <= t0)) r = 2; t0 = t; end
```

```
      [hit,t] = zplane(z2,x(3),u(3),true ); if (hit & (t <= t0)) r = 4; t0 = t; end
      [hit,t] = ccylz(R,x,u,true         ); if (hit & (t <= t0)) r = 3; t0 = t; end
   elseif (r == 2) % Particle is in region 2
      [hit,t] = zplane(z1,x(3),u(3),true );
      if (hit & (t <= t0)) r = 1; t0 = t;
      else esc(r,1) = esc(r,1) + 1;
           if (diagnostics) disp(sprintf('Particle escape from region %g',ri)); end
           r = 0;
      end
   elseif (r == 4) % Particle is in region 4
      [hit,t] = zplane(z2,x(3),u(3),false); if (hit & (t <= t0)) r = 1; t0 = t;
      else esc(r,1) = esc(r,1) + 1;
           if (diagnostics) disp(sprintf('Particle escape from region %g',ri)); end
           r = 0;
      end
   elseif (r == 3) % Particle is in region 3
      %[hit,t] = zplane(z1,x(3),u(3),false); if (hit & (t <= t0)) r = 2; t0 = t; end
      %[hit,t] = zplane(z2,x(3),u(3),true ); if (hit & (t <= t0)) r = 4; t0 = t; end
      [hit,t] = ccylz(R,x,u,false        ); if (hit & (t <= t0)) r = 1; t0 = t;
      else esc(r,1) = esc(r,1) + 1;
           if (diagnostics) disp(sprintf('Particle escape from region %g',ri)); end
           r = 0;
      end
   end
   x = x + u*t0; % Transport the particle
   if (diagnostics)
      str = 'After  transport r:x:u,t0: = %g:[%g,%g,%g]:[%g,%g,%g],%g';
      disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3),t0))
   end

   % Deflect the particle at the end of the transport step (should have no effect)
   if (ri == 1) % Deflections only in physical regions
      if     (scatteringForward  ) w =  1;
      elseif (scatteringBackward ) w = -1;
      elseif (scatteringIsotropic) w =  1 - 2*rand; end
      u = rotate(u,w); % Scatter the particle
      if (diagnostics)
         str = 'After  rotation  r:x:u =     %g:[%g,%g,%g]:[%g,%g,%g]';
         disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3)))
      end
   end
   xspot(end + 1,:) = x; % Record the intercept point

end % End of the transport loop ==========================================

if (diagnostics) % Graphical output each history in diagnostic mode only
   figure(1), plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
   xlabel('x'), ylabel('y'), zlabel('z')
   axis([-2,2,-2,2,0,1])
   pause
end

end % End of simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Outputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1), plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
xlabel('x'), ylabel('y'), zlabel('z'), title('Full geometry')
axis([-2,2,-2,2,0,1])

figure(2), plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
xlabel('x'), ylabel('y'), zlabel('z'), title('Chopping the ends off')
title('Chopping the ends off')
axis([-2,2,-2,2,0.001,0.999])

% Diagnostic tallies (always) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if (volumeSource)
   f = vsrhofails/Nh;
   s = sqrt(f*(1-f)/Nh);
   t = 1-pi/4;
   t = t/(1-t); % Account for multiple misses
   disp(sprintf('Fraction of rho rejections = %g +/- %g compared to theory %g',f, s, t))
   disp(sprintf('Fraction of z rejections = %g compared to theory %g',vszfails/Nh, 0))
elseif (surfaceSource)
   f = ssrhofails/Nh;
   s = sqrt(f*(1-f)/Nh);
   t = 1-pi/4;
   t = t/(1-t)*rplane; % Account for multiple misses, note the rplane fraction
   disp(sprintf('Fraction of rho rejections = %g +/- %g compared to theory %g',f, s, t))
end

for r = 1:Nr disp(sprintf('Region %g had %g escapees',r,esc(r,1))), end

pause, close all
```

encapsulation

Any quadric can encapsulate a finite physical geometry.

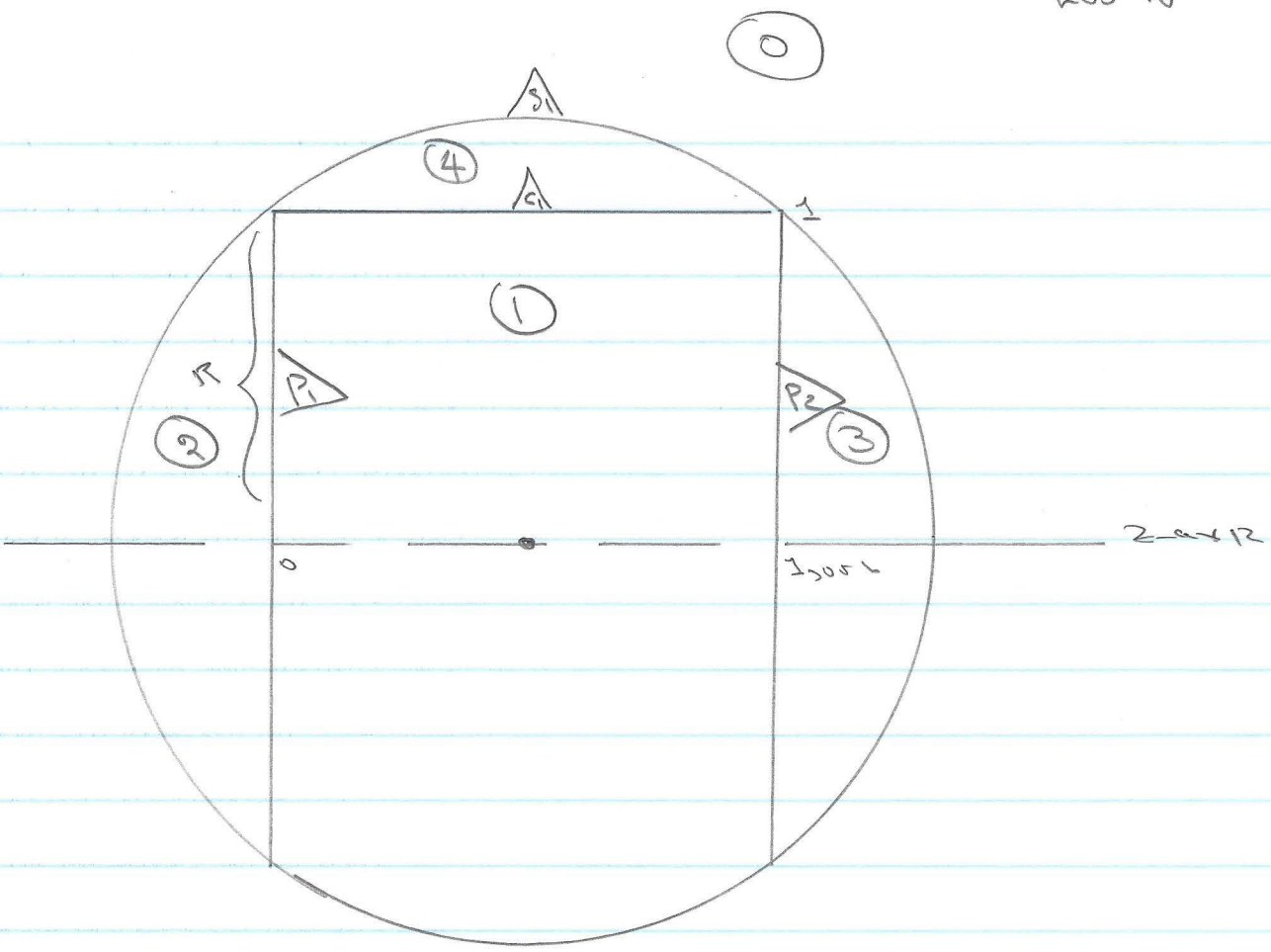
A sphere or spheroid can minimally encapsulate a physical volume, and the process can be automated.

For example, the right circular cylindrical solid we have been working with, has the minimally encapsulating sphere

$$x^2 + y^2 + \left(z - \overbrace{\frac{(z_2 - z_1)}{2}}^{L/2}\right)^2 - \frac{(L^2 + R^2)}{4} = 0$$

The zone delimiting them are

See code



P1 : $z=0$

P2 : $z=1$ (or L)

C1 : $x^2+y^2-R^2=0$

S1 : $x^2+y^2+\left(z-\frac{L}{2}\right)^2-\left(\frac{L^2}{4}+R^2\right)=0$

① $\bar{P}_1 \cap C_1 \cap P_2$

② $\bar{P}_1 \cap S_1$

③ $\bar{P}_2 \cap S_1$

④ $\bar{C}_1 \cap S_1$

⓪ $\bar{S}_1$

```
% p1p2c1es3.m short for Plane 1, Plane2, Cylinder 1, escape zone strategy 3
%             using a minimally encapsulating sphere
% Class Example: See the hand-written notes L8 pages 15 and beyond

clear all, close all, format compact

diagnostics = true;
diagnostics = false; % Comment out this statement to enable diagnostics
if (diagnostics)  disp('Diagnostic mode is on: <return> to continue.'), pause, end

% Geometry definition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nr = 4; % Number of elemental subvolumes
z1 = 0; % P1, z = z1
z2 = 1; % P2; z = z2
R  = 1; % C1 circular cylinder x^2 + y^2 -R^2 = 0
L  = z2 - z1;
xs = [0,0,L/2]; % center of sphere in 3 space
Rs = sqrt((L/2)^2 + R^2); % center of sphere S1: x^2 + y^2 + (z - L/2)^2 - Rs^2
rplane = R/(R + L); % probability that a particle emanating from the surface
                    % originates on one of the planes
% Region 0: escape zone    : Outside S1
% Region 1: physical volume: Outside P1, inside P2, inside C1
% Region 2: boundary zone  : Inside  P1, inside S1
% Region 3: boundary zone  : Outside P1, inside P2, Outside C1
% Region 4: boundary zone  : Outside P2
% All zones except 1 should be considered to be filled with vacuum and not allow
% any particle deflections

% Tally set up %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xspot = []; % Tally vector used for plotting later
esc = zeros(Nr,1);; % Tallies escapes from any region

% Diagnostic tallies
vsrhofails = 0; % Volume source rho failure counter
vszfails   = 0; % Volume source z   failure counter
ssrhofails = 0; % Surface source rho failure counter

% Source type %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
choice = input('0 for surface source, 1 for volume source: ' );
if     (choice == 0) surfaceSource = true;  volumeSource  = false;
elseif (choice == 1) surfaceSource = false; volumeSource  = true;
else   disp('No choice for source type, exiting'), return, end

if     (surfaceSource) disp('Surface source is modeled')
elseif (volumeSource)  disp('Volume source is modeled'), end

% Scattering type %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
choice = input('0 for forward, 1 for backward, 2 for isotropic scattering: ' );
if (choice == 0)
   scatteringForward   = true;
   scatteringBackward  = false;
   scatteringIsotropic = false;
elseif (choice == 1)
   scatteringForward   = false;
   scatteringBackward  = true;
   scatteringIsotropic = false;
elseif (choice == 2)
   scatteringForward   = false;
   scatteringBackward  = false;
   scatteringIsotropic = true;
else disp('No choice for scattering type, exiting'), return, end

if (scatteringBackward)  disp('Backward scattering is modeled' ), end
if (scatteringForward)   disp('Forward scattering is modeled'  ), end
```

```
if (scatteringIsotropic) disp('Isotropic scattering is modeled'), end

% Simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nh = 1000; % Number of histories to simulate
for n = 1:Nh
   if (diagnostics) n, end
   % Sample the source ==========================================================
   r = 1; % Source particle starts in region 1
   if (volumeSource)
      % Define the starting position vector ---------------------------------
      % Samping loop for x and y
      repeat = true; % This is a rejection loop for circular collimation
      while (repeat)
         x(1) = 1 - 2*rand;
         x(2) = 1 - 2*rand;
         if (x(1)^2 + x(2)^2 < 1) repeat = false; % Note the "<" logical operator
         else vsrhofails = vsrhofails + 1; end % Tally the fails
      end
      x = R*x;
      % Samping loop for z
      repeat = true;
      while (repeat)
         x(3) = z1 + (z2 - z1)*rand;
         if (x(3) > z1 && x(3) < z2) repeat = false; % Note the "<" logical operator
         else vszfails = vszfails + 1; end % Tally the fails
      end
   elseif (surfaceSource)
      % Define the starting position vector ---------------------------------
      if (rand < rplane) % Particle emerges from a plane
         % Samping loop for x and y
         repeat = true; % This is a rejection loop for circular collimation
         while (repeat)
            x(1) = 1 - 2*rand;
            x(2) = 1 - 2*rand;
            if (x(1)^2 + x(2)^2 < 1) repeat = false; % Note the "<" logical operator
            else ssrhofails = ssrhofails + 1; end
         end
         x = R*x;
         % Pick a plane, left or right
         if (rand < 0.5) % Left plane is chosen
            x(3) = z1;
            if (diagnostics) disp(sprintf('Starting on plane at z = %g', z1)), end
         else
            x(3) = z2;
            if (diagnostics) disp(sprintf('Starting on plane at z = %g', z2)), end
         end
      else % particle emerges from the surface
         phi = 2*pi*rand; x(1) = R*cos(phi); x(2) = R*sin(phi); x(3) = z1 + (z2 - z1)*rand;
         if (diagnostics)
            disp(sprintf('Starting on cylinder at x = [%g,%g,%g]',x(1),x(2),x(3)))
         end
      end
   end % Starting position now given

   xspot(end + 1,:) = x; % Tally the starting point

   % Define a random starting direction vector
   u = rotate([0,0,1],1 - 2*rand); % Compact use of the rotate function

   % Transport through the geometry =============================================
   while (r > 0 & r <= Nr) % True if particle is inside the geometry
      t0 = inf;            % Distance to interaction (in vacuum)

      if (diagnostics)
         str = 'Before transport r:x:u,t0: = %g:[%g,%g,%g]:[%g,%g,%g],%g';
```

```
      disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3),t0))
   end

   ri = r; % Record the starting region number for later
   if (r == 1) % Particle is in region 1
      [hit,t] = zplane(z1,x(3),u(3),false); if (hit & (t <= t0)) r = 2; t0 = t; end
      [hit,t] = zplane(z2,x(3),u(3),true ); if (hit & (t <= t0)) r = 4; t0 = t; end
      [hit,t] = ccylz(R,x,u,true         ); if (hit & (t <= t0)) r = 3; t0 = t; end
   elseif (r == 2) % Particle is in region 2
      [hit,t] = zplane(z1,x(3),u(3),true);
      if (hit & (t <= t0)) r = 1; t0 = t; end
      [hit,t] = sphere(Rs,xs,x,u,true);
      if (hit & (t <= t0)) r = 0; t0 = t; end
      if (r == 0)
         esc(ri,1) = esc(ri,1) + 1;
         if (diagnostics) disp(sprintf('Particle escape from region %g',ri)); end
      end
   elseif (r == 4) % Particle is in region 4
      [hit,t] = zplane(z2,x(3),u(3),false);
      if (hit & (t <= t0)) r = 1; t0 = t; end
      [hit,t] = sphere(Rs,xs,x,u,true);
      if (hit & (t <= t0)) r = 0; t0 = t; end
      if (r == 0)
         esc(ri,1) = esc(ri,1) + 1;
         if (diagnostics) disp(sprintf('Particle escape from region %g',ri)); end
      end
   elseif (r == 3) % Particle is in region 3
      [hit,t] = zplane(z1,x(3),u(3),false); if (hit & (t <= t0)) r = 2; t0 = t; end
      [hit,t] = zplane(z2,x(3),u(3),true ); if (hit & (t <= t0)) r = 4; t0 = t; end
      [hit,t] = ccylz(R,x,u,false        ); if (hit & (t <= t0)) r = 1; t0 = t; end
      [hit,t] = sphere(Rs,xs,x,u,true);
      if (hit & (t <= t0)) r = 0; t0 = t; end
      if (r == 0)
         esc(ri,1) = esc(ri,1) + 1;
         if (diagnostics) disp(sprintf('Particle escape from region %g',ri)); end
      end
   end
   x = x + u*t0; % Transport the particle
   if (diagnostics)
      str = 'After  transport r:x:u,t0: = %g:[%g,%g,%g]:[%g,%g,%g],%g';
      disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3),t0))
   end

   % Deflect the particle at the end of the transport step (should have no effect)
   if (ri == 1) % Deflections only in physical regions
      if     (scatteringForward  ) w =  1;
      elseif (scatteringBackward ) w = -1;
      elseif (scatteringIsotropic) w =  1 - 2*rand; end
      u = rotate(u,w); % Scatter the particle
      if (diagnostics)
         str = 'After  rotation  r:x:u =     %g:[%g,%g,%g]:[%g,%g,%g]';
         disp(sprintf(str,r,x(1),x(2),x(3),u(1),u(2),u(3)))
      end
   end
   xspot(end + 1,:) = x; % Record the intercept point

end % End of the transport loop ==========================================

if (diagnostics) % Graphical output each history in diagnostic mode only
   figure(1), plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
   xlabel('x'), ylabel('y'), zlabel('z')
   axis equal
   pause
end
```

```
end % End of simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Outputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(1), plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results
xlabel('x'), ylabel('y'), zlabel('z'), title('Full geometry')
axis equal

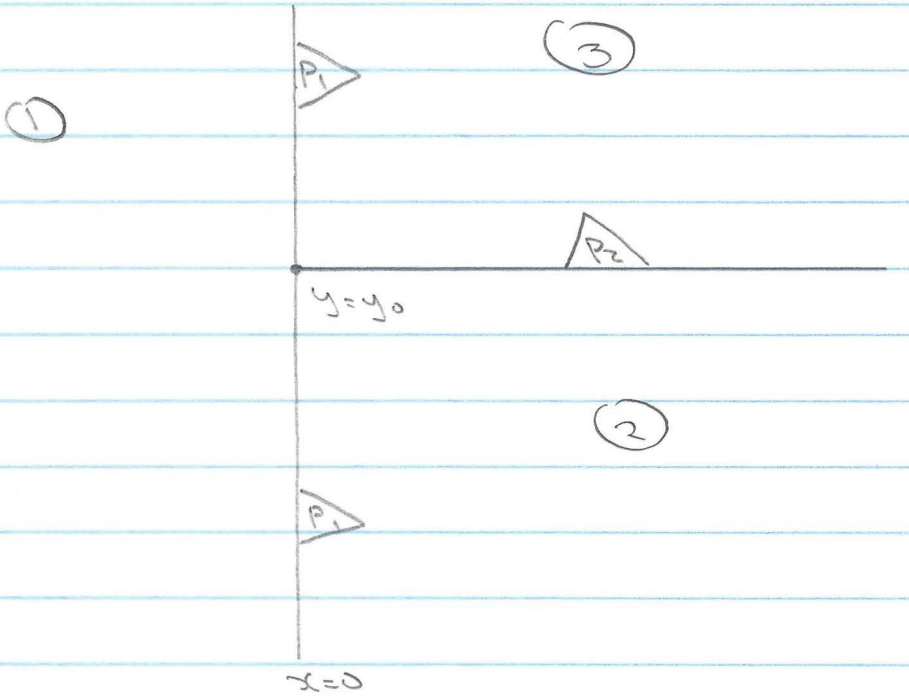
% Diagnostic tallies (always) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (volumeSource)
   f = vsrhofails/Nh;
   s = sqrt(f*(1-f)/Nh);
   t = 1-pi/4;
   t = t/(1-t); % Account for multiple misses
   disp(sprintf('Fraction of rho rejections = %g +/- %g compared to theory %g',f, s, t))
   disp(sprintf('Fraction of z rejections = %g compared to theory %g',vszfails/Nh, 0))
elseif (surfaceSource)
   f = ssrhofails/Nh;
   s = sqrt(f*(1-f)/Nh);
   t = 1-pi/4;
   t = t/(1-t)*rplane; % Account for multiple misses, note the rplane fraction
   disp(sprintf('Fraction of rho rejections = %g +/- %g compared to theory %g',f, s, t))
end

for r = 1:Nr disp(sprintf('Region %g had %g escapees',r,esc(r,1))), end

pause, close all
```

10.6.2 Multiply-connected elements

Consider



Zones

1	P_1	inside P1
2	$\overline{P}_1 \cap P_2$	outside P1, inside P_2
3	$\overline{P}_1 \cap \overline{P}_2$	outside P1, outside P_2

all of space defined

(3) and (2) are simply connected to each other and simply connected to P1

(1) is multiply connected to P_2 & P_3
— transport out of P_1 does not result in a unique zone

Resolution

```
% transport loop                          3 in this case
while (r>0 ; r<=Nr)
    t0 = inf;   % no scattering in this example
    if (r==1)                                  0 in this case
        [hit, t] = xplane(x1, x(1), u(1), true);
        if (hit ; t<=t0)
            t0 = t;
            if ( x(2) + u(2)*t0 <= y0 )
                r=2;
            else
                r=3;
            end
        end
    end
```

In this case, there is no alternative to checking where the particle is based on its future position

Particle location, search routines

Another time when we must check for region number based on particle position is when a particle is started, as in a source.

The algorithm would look like the following, using the last example

$\vec{x}$ is known

$\vec{x} = (x, y, z)$

```
if ( x <= 0 )
    r=1; % in region 1
elseif ( y <= y0 )
    r=2; % in region 2
else
    r=3; % in region 3
end
```

in the case of the p1p2c1es3 example

```
c1 =   x(1)^2 + x(2)^2 - R^2 ;
s1 =   x(1)^2 + x(2)^2 + (x(3) - L/2)^2 - Rs^2

if ( c1 <=0 )  inside c1 = true; end
if ( s1 <=0 )  inside s1 = true; end
if ( z-z1 <=0 ) inside p1 = true; end
if (z-z2 <=0 )  inside p2 = true; end

if ( inside c1 & ~inside p1 & inside p2)
    r = 1;
elseif ( inside p1)
    r = 2;
elseif ( ~inside p2)
    r = 4;
elseif ( ~inside c1 & ~inside p1 & inside p2
        & inside s1 )
    r = 3;
else
    r = 0;
end
```

Note that locating a source particle in region 0 involves another search since region 0 is multiply connected to region 1, 2, 3, 4

Note: It is inadvisable to locate a source particle in an escape zone!

L 08-25

10.6.3 Combinatorial geometry

So far we have introduced elemental volumes and indicated how they may be combined to make objects. We have also introduced a simple yet powerful tracking algorithm for simply-connected surfaces and have given some idea as to the subtleties involved in multiply-connected elements.

The general discussion of this is called *combinatorial geometry* and would take us beyond the scope of this book. Before dismissing the topic, however, the following capabilities of combinatorial codes are either necessary or desirable:

1. The ability for a particle to locate itself unambiguously in a unique elemental volume based upon its position and direction.
2. A systematic numbering/identification scheme for the elemental volumes. For small applications this can be coded by the user. For involved applications, the combinatorial code should be able to do this on its own.
3. The ability to model reflecting, absorbing or partially absorbing surfaces.
4. The ability to "build" objects in standard position and then translate and rotate them into actual position.
5. The ability to re-use objects, once, twice, or in repetitive patterns.
6. The ability to model various sources of particles, either originating interior or exterior to the geometry.
7. The ability to see the results of geometry construction graphically.
8. A graphical tool to build geometries.

10.7 Law of reflection

The use of some of the inherent symmetry of the geometry can realize some real simplifications. We will consider the use of reflecting planes to mimic some of the inherent symmetry of a geometry.

For example, consider the geometry depicted in fig. 10.10. In this case, an infinite square lattice of cylinders is irradiated uniformly from the top. The cylinders are all uniform and aligned. How should one approach this problem? Clearly, one can not model an infinite array of cylinders. If one tried, one would have to pick a finite set and decide somehow that it was big enough. Instead, it is much more efficient to exploit the symmetry of the problem. It turns out that in this instance, one needs to transport particles in only 1/8'th

L09-26

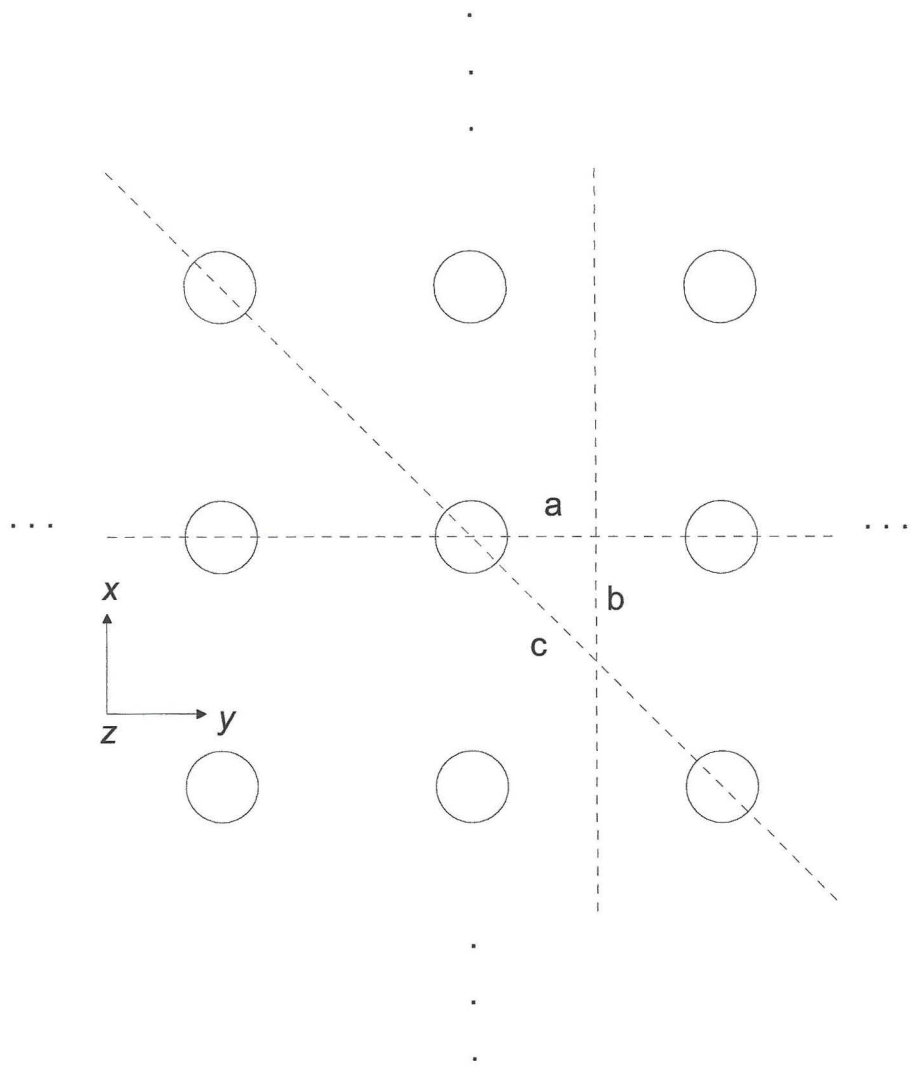


Figure 10.10: Top end view of an infinite square lattice of cylinders. Three planes of symmetry are drawn, **a**, **b**, and **c**. A complete simulation of the entire lattice may be performed by restricting the transport to the interior of the three planes. When a particle strikes a plane it is reflected back in, thereby mimicking the symmetry associated with this plane.

L09-27

of a cylinder! To see this we find the symmetries in this problem. In fig. 10.10 we have drawn three planes of symmetry in the problem, planes **a**, **b**, and **c**[6]. There is reflection symmetry for each of these planes. Therefore, to mimic the infinite lattice, any particles that strike these reflecting planes should be reflected. One only needs to transport particles in the region bounded by the reflecting planes. Because of the highly symmetric nature of the problem, we only need to perform the simulation in a portion of the cylinder and the "response" functions for the rest of the lattice is found by reflection.

The rule for particle reflection about a plane of arbitrary orientation is easy to derive. Let $\vec{u}$ be the unit direction vector of a particle and $\hat{n}$ be the unit direction normal of the reflecting plane. Now divide the particle's direction vector into two portions, $\vec{u}_\parallel$, parallel to $\hat{n}$, and $\vec{u}_\perp$, perpendicular to $\vec{n}$. The parallel part gets reflected, $\vec{u}'_\parallel = -\vec{u}_\parallel$, and the perpendicular part remains unchanged, $\vec{u}'_\perp = \vec{u}_\perp$. That is, the new direction vector is $\vec{u}' = -\vec{u}_\parallel + \vec{u}_\perp$. Another way of writing this is,

$$\vec{u}' = \vec{u} - 2(\vec{u} \cdot \vec{n})\vec{n} \ . \tag{10.21}$$

Applying eq. 10.21 to the problem in fig. 10.10, we have: For reflection at plane **a**, $(u'_x, u'_y, u'_z) = (-u_x, u_y, u_z)$. For reflection at plane **b**, $(u'_x, u'_y, u'_z) = (u_x, -u_y, u_z)$. For reflection at plane **c**, $(u'_x, u'_y, u'_z) = (-u_y, -u_x, u_z)$. The use of this reflection technique can result in great gains in efficiency. Most practical problems will not enjoy such a great amount of symmetry but one is encouraged to make use of any available symmetry. The saving in computing time is well worth the extra care and coding.

[6]Note that this symmetry applies only to a square lattice, where the spacing is the same for the x and y-axes. For a rectangular symmetry, the planes of reflection would be somewhat different. There would be no plane **c** as for the square lattice in fig. 10.10.

```
% zplanes3a.m Class Example - 3 parallel planes, 1 cylinder, interacting beam
% a very simple implementation of multiple planes
% This is an adaptation of zplane3.m adding a cylinder and interactions

clear all, close all

% Source definition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This example is a beam incident from the left, normal incidence
Rbeam = 1; % Radius of collimation of the beam

% Geometry definition %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nr = 2; % Number of elemental subvolumes

% Region 0 (escape zone) definition, bounded by:
z1 = 0; % z = z1; z-plane

% Region 1 definition, bounded by z1 defined above plus
z2 = 1; % z = z2; z-plane at z2
r1 = 2* Rbeam; % Cylinder with radius 2x that of the beam

% Region 2 definition, bounded by z2 and r1 defined above plus
z3 = 2; % z = z3; z-plane at z3

% Region 3 definition (escape zone), bounded by z3 defined above

% Region 4 definition, bounded by (escape zone)
% r1 defined above

% Physics set up %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = linspace(0.1,10,Nr); % Macroscopic Sigma in the exponential interaction law
% Note that it is region dependent and hence we are modelling an inhomogeneous
% object. Region 1 will have little scattering, Region 2 will have lots.
% For Nr = 2, S = [0.1,10];

% Tally set up %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xspot = []; % Tally vector used for plotting later

% Simulation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nh = 500; % Number of histories to simulate
for n = 1:Nh

   % Sample the source
   r = 1; % Source particle starts in region 1
   % Define the starting position vector
   repeat = true; % This is a rejection loop for circular collimation
   while (repeat)
      x(1) = Rbeam*(2*rand - 1);
      x(2) = Rbeam*(2*rand - 1);
      if (x(1)^2 + x(2)^2 <= Rbeam^2) repeat = false; end
   end
   x(3) = z1;            % Starts on leftmost plane
   u = [0, 0, 1];        % Normal incidence
   xspot(end + 1,:) = x; % Record the starting point

   % Transport through the geometry

   while (r > 0 & r <= Nr) % True of particle is inside the geometry
   t0 = -log(rand)/S(r); % Distance to an interaction, note region dependence
   ri = r; % Current region

      % zplane() and ccylz() are NERS544 library routines

      if (r == 1) % Particle is in region 1
```

```
      [hit,t] = zplane(z1,x(3),u(3),false); % Outside z1
      if (hit & (t <= t0)) r = 0; t0 = t; end
      [hit,t] = zplane(z2,x(3),u(3),true); % Inside z2
      if (hit & (t <= t0)) r = 2; t0 = t; end
      [hit,t] = ccylz(r1,x,u,true); % Inside r1
      if (hit & (t <= t0)) r = 4; t0 = t; end

   elseif (r == 2) % Particle is in region 2
      [hit,t] = zplane(z2,x(3),u(3),false); % Outside z2
      if (hit & (t <= t0)) r = 1; t0 = t; end
      [hit,t] = zplane(z3,x(3),u(3),true); % Inside z3
      if (hit & (t <= t0)) r = 3; t0 = t; end
      [hit,t] = ccylz(r1,x,u,true); % Inside r1
      if (hit & (t <= t0)) r = 4; t0 = t; end
   end

   x = x + u*t0; % Transport the particle
   if (r == ri) % The particle stays in the same region thus must interact
      ws = 2*rand - 1; % Isotropic scattering
      u = rotate(u,ws); % NERS544 library routine
   end

   xspot(end + 1,:) = x; % Record the intercept or interaction point

end

pause(0.001) % Matlab has a multithread error causing me to do this
          % Plotting starts before the data is written!
plot3(xspot(:,1),xspot(:,2),xspot(:,3),'k.') % View results

end % End of simulation
```