```
// Rhett Cheek, erc5ej@virginia.edu, phys3995
// pr_diff.cpp
// This is to model the preshower scintillator
// It has 4 fibers in a ring of radius 4.5-4.6cm (each fiber has diameter 0.1cm)

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

const double PI = 3.14159;
const double ROUNDER = 0.001;
// This is used to offset some issues that can occur
// in my reflection calculations due to the inexact nature of doubles in c++
// If you're better at coding than me you may know a more elegant solution

// At the end of the code is a short function that generates random doubles
// in some given range. Each time this program is run, the same random
// values are cycled through if it is initialized by the same parameters,
// giving the same output
double randd(double min, double max);


int main() {

  // This code block can shuffle the rng (random number generator) if
  // you wish to have different random values to start
  /*
  int shuffle_num = 0;// This can shuffle the rng generator a bit
  double rng_shuffler;
  printf("Enter an integer to shuffle the rng (0=default)\n");
  scanf("%d",&shuffle_num);
  for (int jj=0;jj<shuffle_num;jj++) {
    rng_shuffler = randd(0.0,1.0);
  }
  */

  // This code block set the size and a few other aspects of the scintillator
  double outr = 6.25; // length of hexagon's side (cm)
  double depth = 2.0; // depth of scintillator (cm)
  double inr = outr*(sqrt(3.0)/2); // inner radius of hexagon
  double fibr = 4.5;// fiber radius (as spun up in preshower device)
  double fibw = 0.1;// width/diameter of fiber itself
  //  double fib_radius = fibw/2.; // radius of the fiber itself
  double fibd = 0.4;// fiber depth (how far it winds into the presh. scin.
  double refr_ind = 1.5; // Refraction index of material
  double crit_phi = asin(1.0/refr_ind);// Critical angle

  // This code block allows user input for the reflectivity into the command lin
e
  // after the program has begun
  /*
  double crit_ref,norm_ref;
  printf("Enter the chance of transmission after a reflection at an angle\n");
  printf("greater than the critical angle [default=0.01]\n");
  scanf("%lf",&crit_ref);
  printf("Enter chance of transmission: smaller angle on the hexagon\n");
  printf("sides [default=0.2]\n");
  scanf("%lf",&norm_ref);
  */
  // Defaults for reflectivity are below
  double crit_ref = 0.05;// Chance of escape if a critcal reflection
  double norm_ref = 0.1;// Chance to escape if non-critical reflection
  // Technically this is the "chance of transmission"


  // Further initializations:
  int phonum;// = 2000; // Number of events
  printf("How many photon events (integer)?\n");
```



```
  scanf("%d",&phonum);
  double errorbar1; // Will be used in stat-analysis
  double errorbar2;
  int absorbnum = 0; // Number of photons absorbed
  double efficiency; // The efficiency of the scintillator in the simulation
  int fiber_col = 0;
  int inout; // Indicates where the photon spawns and its state in the system
  int inner_hit, outer_hit; // These indicate how the photon runs across the rin
g
  double phi1, phi2; // angles
  double P0 [3],v_dir [3], LdotN, PLdotN, DD, DI;//vector coordinates and place-
holders
  // LdotN is direction of line dot normal of plane
  // PLdotN is the point on plane-point on line dot normal of plane
  // DI is distance to intersection
  // These are used to find the points of collision with the boundaries

  // This creates and opens the output files
  FILE * fp;
  fp = fopen("points.dat","w");
  FILE * sp;
  sp = fopen("bounds.dat","w");
  FILE * op;
  op = fopen("output.dat","w");

  // This section defines points on the planes and normal vectors to the planes
  // to calculate the points of reflection for the photon's path
  /* The hexagon's orientation is as follows based on the drawing below ->
          __
         /  \
         \__/
  */
  double PP [8][3], NN [8][3];//[Boundary number][x,y,or z]
  // Future editors: consider having PP [8][6] where [3] [4] and [5] are NN's va
lues

  PP[0][0]=0.0; PP[0][1]=inr; PP[0][2]=0.0;//top of xy-plane
  PP[1][0]=0.0; PP[1][1]=-inr; PP[1][2]=0.0;//bottom of xy-plane
  PP[2][0]=outr/2.0; PP[2][1]=inr; PP[2][2]=0.0;//top right side of hex
  PP[3][0]=-outr/2.0; PP[3][1]=inr; PP[3][2]=0.0;//top left side of hex
  PP[4][0]=outr/2.0; PP[4][1]=-inr; PP[4][2]=0.0;//bottom right side of hex
  PP[5][0]=-outr/2.0; PP[5][1]=-inr; PP[5][2]=0.0;//bottom left side of hex
  PP[6][0]=0.0; PP[6][1]=0.0; PP[6][2]=0.0;//bottom of the z values
  PP[7][0]=0.0; PP[7][1]=0.0; PP[7][2]=depth;//top of the z values

  NN[0][0]=0.0; NN[0][1]=-1.0; NN[0][2]=0.0;//topxy
  NN[1][0]=0.0; NN[1][1]=1.0; NN[1][2]=0.0;//botxy (-n0)
  NN[2][0]=-sqrt(3.0)/2.0; NN[2][1]=-0.5; NN[2][2]=0.0;//topright
  NN[3][0]=sqrt(3.0)/2.0; NN[3][1]=-0.5; NN[3][2]=0.0;//topleft
  NN[4][0]=-sqrt(3.0)/2.0; NN[4][1]=0.5; NN[4][2]=0.0;//botright (-n3)
  NN[5][0]=sqrt(3.0)/2.0; NN[5][1]=0.5; NN[5][2]=0.0;//botleft (-n2)
  NN[6][0]=0.0; NN[6][1]=0.0; NN[6][2]=1.0;//botz
  NN[7][0]=0.0; NN[7][1]=0.0; NN[7][2]=-1.0; //topz (-n6)


  double Pchan [3],P2chan [3];// These are used to hold points of transition
  // for the purposes of my calculations
  double escape; // Gives escape chance for photon


  // The values below will be used to help check the ring vs. line intersections
  double xi1,xi2,xo1,xo2,yi1,yi2,yo1,yo2,zi1,zi2,zo1,zo2;// Line-ring intersect
coord's
  double absorb; // chance to be absorbed based on travel distance
  double a_resist; // random [0,1], and if this is greater than absorb, it trans
mits
  double travel_distance; // 3-space through the fiber the photon traverses

  double AA, BB, CC;// Used to simplify intersection calc expressions
```

```
  // I will make a 1st order approximation and say that instead of circular
  // fibers it is a rectangle spun around the 4.5-4.6cm radius
  /*  double fib_center [4]; // These are the Z values of the 4 fibers
  fib_center[0] = depth-0.04;fib_center[1] = depth-0.15;
  fib_center[2] = depth-0.25;fib_center[3] = depth-0.35;*/
  // These will describe the intersection points with the fiber ring

  //Attenuation variables
  double travel_total;// Will be used to calculate attenuation
  double lambda_b;
  double atten; // Chance the photon's been removed via attenuation
  int atten_num=0; // Number of photons that survived attenuation

  double coinflip; // A way to fix my earlier incorrect Azimuthal without alteri
ng
  // my calculations (I originally, incorrectly, had it in the range (0,2*PI))

  double XXX, YYY, ZZZ;
  printf("X=\n");
  scanf("%lf",&XXX);
  printf("Y=\n");
  scanf("%lf",&YYY);
  printf("Z=\n");
  scanf("%lf",&ZZZ);

  for (int ii = 1; ii<=phonum; ii++) {

    travel_total = 0.; // Reset the distance travelled by a new photon
    lambda_b =0.;

    P0[0]=XXX;
    P0[1]=YYY;
    P0[2]=ZZZ;
    //    printf("%d %d\n",ii, phonum);

    /*
    P0[0] = randd(-outr,outr); // RNG for initial points of event
    P0[1] = randd(-inr,inr);
    P0[2] = randd(0.0,depth);
    */

    phi1 = randd(0.0,2*PI); // Pi is twice as likely as any other
    coinflip=randd(0.0,2.0);
    if (coinflip>1.0) { // Azimuthal in range [0,pi/2], [3pi/2,2pi]
      phi2 = randd(3.*PI/2,2.*PI);
    }
    else {
      phi2 = randd(0.,PI/2.);
    }
    v_dir[0] = cos(phi1)*cos(phi2); v_dir[1] = sin(phi1)*cos(phi2);
    v_dir[2] = sin(phi2); // Internally consistent vector angle


   // This section checks if photon spawns inside hexagon and not corners of rec
tangle
    if ((P0[1]+2.0*P0[0])>(2.0*outr) || (P0[1]-2.0*P0[0])<(-2.0*outr) || \
        (P0[1]-2.0*P0[0])>(2.0*outr) || (P0[1]+2.0*P0[0])<(-2.0*outr)) {// In he
xagon?
      inout = 0; // if "if" is satisfied, it is outside the hex
    }
    else if ((((P0[1]*P0[1])+(P0[0]*P0[0]))>(fibr*fibr))\
             && (((P0[1]*P0[1])+(P0[0]*P0[0]))<(pow(fibr+fibw,2))) &&   \
             (P0[2]>depth-fibd && P0[2]<depth)) {
      // Check if it spawns on the fiber ring
      inout = 0;
      printf("Your inputs were bad and you should feel bad");
      break;
    }
    else {
```



```
      inout = 1;
    }
    if (inout == 0) {
      ii--;// It is either spawned outside or on the fiber
    }

    //// This section is what happens once a good starting position in generated
    else if (inout == 1){
      if(ii<1000){
        fprintf(fp,"%d %lf %lf %lf\n",ii,P0[0],P0[1],P0[2]);// Initial point for
 event
        fprintf(op,"%d %lf %lf %lf 8 0.0\n",ii,P0[0],P0[1],P0[2]);
      }

      while (inout == 1) {// While it is not absorbed or escaping

        inner_hit = 0;
        outer_hit = 0;

        //// Check if it hits the fibers

        // This is the complicated geometry that is unique to the preshower
        // Using the approximation that the fibers are essentially a rectangle o
f height
        // 4-millimeters
        // Calculate the intersection points with the rings (finite width)
        // Ax+By=C line, x^2+y^2=r^2 circle here
        // y-y0=tan(phi)*(x-x0)=> -tan(phi)*x+y=y0-tan(phi)*x0
        // A = -tan(phi1); B = 1; C = y0-tan(phi)*x0; r = 4.5 & 4.6
        AA = -tan(phi1);
        BB = 1.0;
        CC = P0[1]-tan(phi1)*P0[0];

        // It at least skims by the ring at some Z
        if ((pow(fibr+fibw,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2))>0) {

          xo1=((AA*CC)+BB*sqrt(pow(fibr+fibw,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2))
)/ \
            (pow(AA,2)+pow(BB,2));
          xo2=((AA*CC)-BB*sqrt(pow(fibr+fibw,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2))
)/ \
            (pow(AA,2)+pow(BB,2));
          yo1=((BB*CC)-AA*sqrt(pow(fibr+fibw,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2))
)/ \
            (pow(AA,2)+pow(BB,2));
          yo2=((BB*CC)+AA*sqrt(pow(fibr+fibw,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2))
)/ \
            (pow(AA,2)+pow(BB,2));

          outer_hit = 1;
        // Calculate z value at each intersection; see if it touches the ring
          zo1=tan(phi2)*sqrt(pow(xo1-P0[0],2)+pow(yo1-P0[1],2))+P0[2];
          zo2=tan(phi2)*sqrt(pow(xo2-P0[0],2)+pow(yo2-P0[1],2))+P0[2];
        }

        // It goes through the entire ring at some Z
        if ((pow(fibr,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2))>0) {

          xi1=((AA*CC)+BB*sqrt(pow(fibr,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2)))/ \
            (pow(AA,2)+pow(BB,2));
          xi2=((AA*CC)-BB*sqrt(pow(fibr,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2)))/ \
            (pow(AA,2)+pow(BB,2));
          yi1=((BB*CC)-AA*sqrt(pow(fibr,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2)))/ \
            (pow(AA,2)+pow(BB,2));
          yi2=((BB*CC)+AA*sqrt(pow(fibr,2)*(pow(AA,2)+pow(BB,2))-pow(CC,2)))/ \
            (pow(AA,2)+pow(BB,2));

          inner_hit = 1;
        // Calculate z value at each intersection; see if it touches the ring
```

```
          zi1=tan(phi2)*sqrt(pow(xi1-P0[0],2.)+pow(yi1-P0[1],2.))+P0[2];
          zi2=tan(phi2)*sqrt(pow(xi2-P0[0],2.)+pow(yi2-P0[1],2.))+P0[2];
        }

        // If the path goes through only the outer circle
        if ((outer_hit == 1) && (inner_hit == 0)) {
          if ((zo1>=0.) && (zo1<=depth) && (zo2>=0.) && (zo2<=depth)) {
            // Both outer collision points are within the scintillator
            travel_distance = sqrt(pow(xo1-xo2,2.)+pow(yo1-yo2,2.)+pow(zo1-zo2,2
.));
            absorb = 1.-pow(10.,(-12.76)*(travel_distance));
            a_resist = randd(0.,1.);
            fiber_col++;
            if (absorb>a_resist) {
              absorbnum++;

              if(sqrt(pow(P0[0]-xo1,2)+pow(P0[1]-yo1,2)+pow(P0[2]-zo1,2))>
\
                 sqrt(pow(P0[0]-xo2,2)+pow(P0[1]-yo2,2)+pow(P0[2]-zo2,2)))
                travel_total+=sqrt(pow(P0[0]-xo2,2)+pow(P0[1]-yo2,2)+pow(P0[2]-z
o2,2));
              else travel_total+=sqrt(pow(P0[0]-xo1,2)+pow(P0[1]-yo1,2)+pow(P0[2
]-zo1,2));

              lambda_b = 20.4*(1.-exp(-travel_total/13.1))+0.42*travel_total;
              // Constants provided by reference materials for the scintillator
              atten = 1.-exp(-travel_total/lambda_b);// Chance to lose it
              a_resist = randd(0.,1.); // Just needed a random int here
              if (a_resist>atten) { // This is (not) the inverted version
                atten_num++;
                //Record points and finish
                if(ii<1000) fprintf(op,"%d %lf %lf %lf -2 %lf\n",ii,P0[0],P0[1],
P0[2],travel_total);
                inout = 0;
                break;
              }

              else {
                if(ii<1000) fprintf(op,"%d %lf %lf %lf -1 %lf\n",ii,P0[0],P0[1],
P0[2],travel_total);
                inout=0;
                break;
              }
            }
          }
        }
        // If the path goes through both (far more likely)
        else if ((outer_hit == 1) && (inner_hit == 1)) {
          if ((zo1>=0.) && (zo1<=depth) && (zi1>=0.) && (zi1<=depth)) {
            // The first intersecting path is within the scintillator
            travel_distance = sqrt(pow(xo1-xi1,2.)+pow(yo1-yi1,2.)+pow(zo1-zi1,2
.));
            absorb = 1.-pow(10.,(-12.76)*(travel_distance));// I use cm not mm
            a_resist = randd(0.,1.);
            fiber_col++;
            if (absorb>a_resist) {
              absorbnum++;

              if(sqrt(pow(P0[0]-xo1,2)+pow(P0[1]-yo1,2)+pow(P0[2]-zo1,2))>
\
                 sqrt(pow(P0[0]-xo2,2)+pow(P0[1]-yo2,2)+pow(P0[2]-zo2,2)))
                travel_total+=sqrt(pow(P0[0]-xo2,2)+pow(P0[1]-yo2,2)+pow(P0[2]-z
o2,2));
              else travel_total+=sqrt(pow(P0[0]-xo1,2)+pow(P0[1]-yo1,2)+pow(P0[2
]-zo1,2));

              lambda_b = 20.4*(1.-exp(-travel_total/13.1))+0.42*travel_total;
              // Constants provided by reference materials for the scintillator
              atten = 1.-exp(-travel_total/lambda_b);// Chance to lose it
```



```
              a_resist = randd(0.,1.); // Just needed a random int here
              if (a_resist>atten) { // This is (not) the inverted version
                atten_num++;
                //Record points and finish
                if(ii<1000) fprintf(op,"%d %lf %lf %lf -2 %lf\n",ii,P0[0],P0[1],
P0[2],travel_total);
                inout = 0;
                break;
              }

              else {
                if(ii<1000) fprintf(op,"%d %lf %lf %lf -1 %lf\n",ii,P0[0],P0[1],
P0[2],travel_total);
                inout=0;
                break;
              }
            }
          }
          if ((zo2>=0.) && (zo2<=depth) && (zi2>=0.) && (zi2<=depth)) {
            // The second intersecting path is within the scintillator
            travel_distance = sqrt(pow(xo2-xi2,2)+pow(yo2-yi2,2)+pow(zo2-zi2,2))
;
            absorb = 1.-pow(10.,(-12.76)*(travel_distance));
            a_resist = randd(0.,1.);
            fiber_col++;
            if (absorb>a_resist) {
              absorbnum++;

              if(sqrt(pow(P0[0]-xo1,2)+pow(P0[1]-yo1,2)+pow(P0[2]-zo1,2))>
\
                 sqrt(pow(P0[0]-xo2,2)+pow(P0[1]-yo2,2)+pow(P0[2]-zo2,2)))
                travel_total+=sqrt(pow(P0[0]-xo2,2)+pow(P0[1]-yo2,2)+pow(P0[2]-z
o2,2));
              else travel_total+=sqrt(pow(P0[0]-xo1,2)+pow(P0[1]-yo1,2)+pow(P0[2
]-zo1,2));

              lambda_b = 20.4*(1.-exp(-travel_total/13.1))+0.42*travel_total;
              // Constants provided by reference materials for the scintillator
              atten = 1.-exp(-travel_total/lambda_b);// Chance to lose it
              a_resist = randd(0.,1.); // Just needed a random int here
              if (a_resist>atten) { // This is (not) the inverted version
                atten_num++;
                //Record points and finish
                if(ii<1000) fprintf(op,"%d %lf %lf %lf -2 %lf\n",ii,P0[0],P0[1],
P0[2],travel_total);
                inout = 0;
                break;
              }

              else {
                if(ii<1000) fprintf(op,"%d %lf %lf %lf -1 %lf\n",ii,P0[0],P0[1],
P0[2],travel_total);
                inout=0;
                break;
              }
            }
          }
        }

        if (inout==0) break;

        int boundary = 0;
        double score_to_beat = 1000.0;
        //// Calc distance to each planar intersection point
        // The shortest distance will be the collision point
        // This loop runs through each possible boundary to check which the phot
on will
        // be incident to
        for (int i=0;i<8;i++) {
```

```
          // These expressions are taken from the intersection of a plane and li
ne
          // d*l+l_0: point of intersection, d=(p_0-l_O).*n/l.*n
          // LdotN=>l.*n, PLdotN=>(p_0-l_0).*n where l is direction vector of li
ne,
          // l_0 is a point on the line, n is the normal vector of the plane, an
d
          // p_0 is a point on the plane

          LdotN=(NN[i][0]*v_dir[0])+(NN[i][1]*v_dir[1])+(NN[i][2]*v_dir[2]);
          PLdotN=((PP[i][0]-P0[0])*NN[i][0])+((PP[i][1]-P0[1])*NN[i][1])\
            +((PP[i][2]-P0[2])*NN[i][2]);

          DD = PLdotN/LdotN;
          DI = sqrt((DD*DD*v_dir[0]*v_dir[0])+(DD*DD*v_dir[1]*v_dir[1])\
                  +(DD*DD*v_dir[2]*v_dir[2]));//distance to intersection

          // First I use several tests to make sure the photon doesn't go
          // the opposite direction along the line that I want it to go
          if ((DI < score_to_beat) && (DI>0.0001)) {// Smaller distance is the c
ollision pt
            if ((i==0) && (phi1>PI)) {// These cut out any reversed rays
            }
            else if ((i==1) && (phi1<PI)) {
            }
            else if ((i==2) && ((phi1>(2.*PI/3.)) && (phi1<(5.*PI/3.)))) {
            }
            else if ((i==3) && ((phi1<PI/3.) || (phi1>(4.*PI/3.)))) {
            }
            else if ((i==4) && ((phi1>PI/3.) && (phi1<(4.*PI/3.)))) {
            }
            else if ((i==5) && ((phi1<(2.*PI/3.)) || (phi1>(5.*PI/3.)))) {
            }
            else if ((i==6) && (phi2<PI)) {
            }
            else if ((i==7) && (phi2>PI)) {
            }
            else {
              Pchan[0]=DD*v_dir[0]+P0[0];Pchan[1]=DD*v_dir[1]+P0[1];
              Pchan[2]=DD*v_dir[2]+P0[2];
              if ((Pchan[0]<=(outr+ROUNDER)) && (Pchan[0]>=(outr*(-1.0)-ROUNDER)
) &&
(Pchan[1]<=inr+ROUNDER) && (Pchan[1]>=(inr*(-1.0)-ROUNDER)) && (Pchan[2]<=depth+
ROUNDER) &&
(Pchan[2]>=0.0-ROUNDER) && (Pchan[1]+2.0*Pchan[0])<=(2.0*outr+ROUNDER) &&
(Pchan[1]-2.0*Pchan[0])>=(-2.0*outr-ROUNDER) && (Pchan[1]-2.0*Pchan[0])<=(2.0*ou
tr+ROUNDER) &&
 (Pchan[1]+2.0*Pchan[0])>=(-2.0*outr)-ROUNDER) {
                // Above condition makes sure the new coordinates are within the
 hexagon
                // BUT they're on the boundary!

                P2chan[0]=Pchan[0];
                P2chan[1]=Pchan[1];
                P2chan[2]=Pchan[2];
                score_to_beat = DI;
                boundary=i;
                //              fprintf(fp,"%d    %lf    \n",boundary,DI);
              }
            }
          }

        }// Closes loop that checks the distance to the intersection

        //// This section redirects the photon (or lets it escape) based on
        // the boundary that it hit
        // This is laboriously drawn out due to the geometric complexities in de
aling
        // with hexagons and normal-vectors (may possibly be simplified by futur
```



```
e edits)
        // Doing so would require a bit of care, however, and currently the syst
em in
        // place performs correct calculations
        escape = randd(0.0,1.0);

        if (boundary == 0){ // top of xy
          // only hits this bound if 0<phi<pi
          if (phi1<(PI/2.0)){// if phi is going to the right
            if (((PI/2.0)-phi1)<crit_phi) {// no total internal reflection
              if (escape<norm_ref) {// These are when the photon escapes
                inout = 0;
              }
              else {
                phi1 = randd(PI,2.*PI);
                coinflip=randd(0.0,2.0);
                if (coinflip>1.0) {
                  phi2 = randd(3.*PI/2,2.*PI);
                }
                else {
                  phi2 = randd(0.,PI/2.);
                }
              }
            }
            else {// total internal reflection
              if (escape<crit_ref) {
                inout = 0;
              }
              else {
                phi1 = (2.0*PI)-phi1; // New angle
              }
            }
          }

          else {// if phi is going to the left
            if ((phi1-(PI/2.0))<(crit_phi)) {// no tir
              if (escape<norm_ref) {
                inout = 0;
              }
              else {
                phi1 = randd(PI,2.*PI);
                coinflip=randd(0.0,2.0);
                if (coinflip>1.0) {
                  phi2 = randd(3.*PI/2,2.*PI);
                }
                else {
                  phi2 = randd(0.,PI/2.);
                }
              }
            }
            else {// tir
              if (escape<crit_ref) {
                inout = 0;
              }
              else {
                phi1 = (2.0*PI)-phi1; // New angle
              }
            }
          }


          while (phi1>2.0*PI){
            phi1 = phi1-2.0*PI;
          }
          while (phi1<0.0) {
            phi1 = phi1+2.0*PI;
          }

        }
```

```
      else if (boundary == 1) {//redirect at bot of xy
        // only hits if pi<phi<2pi
        if (phi1>(3.0*PI/2.0)) {// phi is going to the right
          if (phi1-(3.0*PI/2.0)<crit_phi) {
            if (escape<norm_ref) {
              inout = 0;
            }
            else {
              phi1 = randd(0.,PI);
              coinflip=randd(0.0,2.0);
              if (coinflip>1.0) {
                phi2 = randd(3.*PI/2,2.*PI);
              }
              else {
                phi2 = randd(0.,PI/2.);
              }
            }
          }
          else {
            if (escape<crit_ref) {
              inout = 0;
            }
            else {
              phi1 = (2.0*PI)-phi1; // New angle
            }
          }
        }

        else {// phi is going to the left
          if ((3.0*PI/2.0)-phi1<crit_phi) {
            if (escape<norm_ref) {
              inout = 0;
            }
            else {
              phi1 = randd(0.,PI);
              coinflip=randd(0.0,2.0);
              if (coinflip>1.0) {
                phi2 = randd(3.*PI/2,2.*PI);
              }
              else {
                phi2 = randd(0.,PI/2.);
              }
            }
          }
          else {
            if (escape<crit_ref) {
              inout = 0;
            }
            else {
              phi1 = (2.0*PI)-phi1; // New angle
            }
          }
        }

        while (phi1>2.0*PI){
          phi1 = phi1-2.0*PI;
        }
        while (phi1<0.0) {
          phi1 = phi1+2.0*PI;
        }
      }


      else if (boundary == 2) {// top right
        if (phi1>=(PI/6.0) && phi1<(2.0*PI/3.0)) {// upper half
          if ((phi1-(PI/6.0))<crit_phi){
            if (escape<norm_ref) {
```



```
              inout = 0;
            }
            else {
              phi1 = randd(2.*PI/3.,5.*PI/3.);
              coinflip=randd(0.0,2.0);
              if (coinflip>1.0) {
                phi2 = randd(3.*PI/2.,2.*PI);
              }
              else {
                phi2 = randd(0.,PI/2.);
              }
            }
          }
          else {
            if (escape<crit_ref) {
              inout = 0;
            }
            else {
              phi1 = (4.0*PI/3.0)-phi1;
            }
          }
        }

        else {//refl other way
          if (phi1<(PI/6.0)) {// not a negative angle
            if ((PI/6.0)-phi1<crit_phi){
              if (escape<norm_ref) {
                inout = 0;
              }
              else {
                phi1 = randd(2.*PI/3.,5.*PI/3.);
                coinflip=randd(0.0,2.0);
                if (coinflip>1.0) {
                  phi2 = randd(3.*PI/2.,2.*PI);
                }
                else {
                  phi2 = randd(0.,PI/2.);
                }
              }
            }
            else {
              if (escape<crit_ref) {
                inout = 0;
              }
              else {
                phi1 = (4.0*PI/3.0)-phi1;
              }
            }
          }
          else {// yes a "negative" angle
            if ((13.0*PI/6.0)-phi1<crit_phi){
              if (escape<norm_ref) {
                inout = 0;
              }
              else {
                phi1 = randd(2.*PI/3.,5.*PI/3.);
                coinflip=randd(0.0,2.0);
                if (coinflip>1.0) {
                  phi2 = randd(3.*PI/2.,2.*PI);
                }
                else {
                  phi2 = randd(0.,PI/2.);
                }
              }
            }
            else {
              if (escape<crit_ref) {
                inout = 0;
              }
```

```
              else {
                phi1 = (10.0*PI/3.0)-phi1;
              }
            }
          }
        }

        while (phi1>2.0*PI){
          phi1 = phi1-2.0*PI;
        }
        while (phi1<0.0) {
          phi1 = phi1+2.0*PI;
        }
      }


      else if (boundary == 3) {// top left
        if (phi1<5.0*PI/6.0){
          if ((5.0*PI/6.0)-phi1<crit_phi){
            if (escape<norm_ref) {
              inout = 0;
            }
            else {
              phi1 = randd(4.*PI/3.,7.*PI/3.);
              coinflip=randd(0.0,2.0);
              if (coinflip>1.0) {
                phi2 = randd(3.*PI/2.,2.*PI);
              }
              else {
                phi2 = randd(0.,PI/2.);
              }
            }
          }
          else {
            if (escape<crit_ref) {
              inout = 0;
            }
            else {
              phi1 = (2.0*PI/3.0)-phi1;
            }
          }
        }

        else {// phi greater than 5pi/6
          if (phi1-(5.0*PI/6.0)<crit_phi){
            if (escape<norm_ref) {
              inout = 0;
            }
            else {
              phi1 = randd(4.*PI/3.,7.*PI/3.);
              coinflip=randd(0.0,2.0);
              if (coinflip>1.0) {
                phi2 = randd(3.*PI/2.,2.*PI);
              }
              else {
                phi2 = randd(0.,PI/2.);
              }
            }
          }
          else {
            if (escape<crit_ref) {
              inout = 0;
            }
            else {
              phi1 = (8.0*PI/3.0)-phi1;
            }
          }
        }
```



```
        while (phi1>2.0*PI){
          phi1 = phi1-2.0*PI;
        }
        while (phi1<0.0) {
          phi1 = phi1+2.0*PI;
        }
      }


      else if (boundary == 4) {// bot right
        if (phi1<PI/3.0) {
          if (phi1+(PI/6.0)<crit_phi) {
            if (escape<norm_ref) {
              inout = 0;
            }
            else {
              phi1 = randd(PI/3.,4.*PI/3.);
              coinflip=randd(0.0,2.0);
              if (coinflip>1.0) {
                phi2 = randd(3.*PI/2.,2.*PI);
              }
              else {
                phi2 = randd(0.,PI/2.);
              }
            }
          }
          else {
            if (escape<crit_ref) {
              inout = 0;
            }
            else {
              phi1 = (2.0*PI/3.0)-phi1;
            }
          }
        }

        else if (phi1>=(11.*PI/6.0)) {
          if (phi1-(11.*PI/6.0)<crit_phi) {
            if (escape<norm_ref) {
              inout = 0;
            }
            else {
              phi1 = randd(PI/3.,4.*PI/3.);
              coinflip=randd(0.0,2.0);
              if (coinflip>1.0) {
                phi2 = randd(3.*PI/2.,2.*PI);
              }
              else {
                phi2 = randd(0.,PI/2.);
              }
            }
          }
          else {
            if (escape<crit_ref) {
              inout = 0;
            }
            else {
              phi1 = (8.0*PI/3.0)-phi1;
            }
          }
        }

        else {
          if ((11.0*PI/6.0)-phi1<crit_phi) {
            if (escape<norm_ref) {
              inout = 0;
            }
            else {
              phi1 = randd(PI/3.,4.*PI/3.);
```

```
            coinflip=randd(0.0,2.0);
            if (coinflip>1.0) {
              phi2 = randd(3.*PI/2.,2.*PI);
            }
            else {
              phi2 = randd(0.,PI/2.);
            }
          }
        }
        else {
          if (escape<crit_ref) {
            inout = 0;
          }
          else {
            phi1 = (8.0*PI/3.0)-phi1;
          }
        }
      }

      while (phi1>2.0*PI){
        phi1 = phi1-2.0*PI;
      }
      while (phi1<0.0) {
        phi1 = phi1+2.0*PI;
      }
    }


    else if (boundary == 5) {// bot left
      if (phi1>2.0*PI/3.0 && phi1<7.0*PI/6.0) {
        if (7.0*PI/6.0-phi1<crit_phi) {
          if (escape<norm_ref) {
            inout = 0;
          }
          else {
            coinflip=randd(0.0,2.0);
            if (coinflip>1.0) {
              phi2 = randd(3.*PI/2.,2.*PI);
              phi1 = randd(0.,2.*PI/3.);
            }
            else {
              phi2 = randd(0.,PI/2.);
              phi1 = randd(5.*PI/3.,2.*PI);
            }
          }
        }
        else {
          if (escape<crit_ref) {
            inout = 0;
          }
          else {
            phi1 = (4.0*PI/3.0)-phi1;
          }
        }
      }

      else { // phi greater than 7pi/6
        if (phi1-7.0*PI/6.0<crit_phi) {
          if (escape<norm_ref) {
            inout = 0;
          }
          else {
            coinflip=randd(0.0,2.0);
            if (coinflip>1.0) {
              phi2 = randd(3.*PI/2.,2.*PI);
              phi1 = randd(0.,2.*PI/3.);
            }
            else {
              phi2 = randd(0.,PI/2.);
```



```
              phi1 = randd(5.*PI/3.,2.*PI);
            }
          }
        }
        else {
          if (escape<crit_ref) {
            inout = 0;
          }
          else {
            phi1 = 2.0*PI+(4.0*PI/3.0)-phi1;
          }
        }
      }

      while (phi1>2.0*PI){
        phi1 = phi1-2.0*PI;
      }
      while (phi1<0.0) {
        phi1 = phi1+2.0*PI;
      }
    }


    else if (boundary == 6) {//bot of z
      if (phi2>(3.0*PI/2.0)) {// phi is going to the right
        if (phi2-(3.0*PI/2.0)<crit_phi) {
          if (escape<norm_ref) {// no critical angle, escape
            inout = 0;
          }
          else {//reflect
            phi2 = randd(0.,PI/2.);
            phi1= randd(0.,2.*PI);
          }
        }
        else {
          if (escape<crit_ref) {// yes critical angle, escape
            inout = 0;
          }
          else {// reflect
            phi2 = (2.0*PI)-phi2; // New angle
          }
        }
      }

      while (phi2>2.0*PI){
        phi2 = phi2-2.0*PI;
      }
      while (phi2<0.0) {
        phi2 = phi2+2.0*PI;
      }
    }


    else {// topz// Test else vs else if boundary 7
      if (((PI/2.0)-phi2)<crit_phi) {// no total internal reflection
        if (escape<norm_ref) {// no critical angle
          inout = 0;
        }
        else {// the reflect condition (diffused)
          phi2 = randd(3.*PI/2.,2.*PI);
          phi1= randd(0.,2.*PI);
        }
      }
      else {// total internal reflection
        if (escape<crit_ref) {
          inout = 0;
        }
        else {// reflect condition
          phi2 = (2.0*PI)-phi2; // New angle
```

```
          }
        }

        while (phi2>2.0*PI){
          phi2 = phi2-2.0*PI;
        }
        while (phi2<0.0) {
          phi2 = phi2+2.0*PI;
        }
      }

      // Record the reflection event in points.dat
      travel_total+=sqrt(pow(P0[0]-P2chan[0],2)+pow(P0[1]-P2chan[1],2)+pow(P0[
2]-P2chan[2],2));
      P0[0]=P2chan[0];
      P0[1]=P2chan[1];
      P0[2]=P2chan[2];
      v_dir[0] = cos(phi1)*cos(phi2);
      v_dir[1] = sin(phi1)*cos(phi2);
      v_dir[2] = sin(phi2);

      if (ii<1000){
      fprintf(fp,"%d %lf %lf %lf\n",ii,P0[0],P0[1],P0[2]);
      fprintf(sp,"%d %d\n",ii,boundary);
      fprintf(op,"%d %lf %lf %lf %lf %lf %d %lf\n",ii,P0[0],P0[1],P0[2]\
              ,phi1,phi2,boundary,travel_total);
      //      printf("%d\n",ii);
      }
    }// Closes while loop "while photon hasn’t escaped"
  }// Closes while loop when given photon is inside hex
}// Closes the iterations for the number of the photon events


efficiency = absorbnum*100./phonum;
errorbar1=sqrt((efficiency/100.)*(1-(efficiency/100.)))/sqrt(phonum);
errorbar1=errorbar1*100.;

double survive = atten_num*100./phonum;
errorbar2=sqrt((survive/100.)*(1-(survive/100.)))/sqrt(phonum);
errorbar2=errorbar2*100.;

printf("With starting position %lf %lf %lf (PS)\n",XXX,YYY,ZZZ);
printf("eff %lf %lf %lf %lf %lf %lf %lf\n",XXX,YYY,ZZZ,efficiency,survive,\
       errorbar1,errorbar2);
return (0);

}

/* Generate a random floating point number from min to max
This function uses RAND_MAX, so the randomness is limited to
some degree.

It uses the same values in each activation of the program.*/

double randd(double min, double max)
{
    double range = (max - min);
    double div = RAND_MAX / range;
    return min + (rand() / div);
}
```