```
// sh_diff.cpp, Rhett Cheek, erc5ej@virginia.edu
// Simulation of the shower scintillator
// Command-line input(s) will be requested in text

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

const double PI = 3.14159;
const double ROUNDER = 0.00001; // This is used to avoid issues that can occur
// in my reflection calculations due to the inexact nature of doubles in c++
/*
note:
grep eff screen* > root_eff_input.dat
then edit the file to remove texts
change screen output to include initial x,y,z
read x,y,z,eff1,eff2 from root -> make plots
*/

// 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;
  double rng_shuffler;
  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 and outer radius (cm)
  double depth = 0.15; // depth of scintillator (cm)
  double inr = outr*(sqrt(3.0)/2.); // A consequence of hexagon geometry
  double fibw = 0.1;// diameter of fibers
  double fibr = fibw/2.;
  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,floor_ref;
  printf("Chance of transmission after a reflection (>critical angle, default 0.
01)\n");
  scanf("%lf",&crit_ref);
  printf("Chance of transmission: smaller angle on the hexagon sides [default=0.
2]\n");
  scanf("%lf",&norm_ref);
  printf("Chance of transmission: smaller angle on floor [default=0.2]\n");
  scanf("%lf",&floor_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
  double floor_ref = 0.1;// Chance to escape if floor or ceiling
  // The latter was added to this one for interestingness
```



```
  // Further initializations, as well as setting the number of photon events to
run
  int phonum;// = 2000; // Number of events ran
  printf("How many photon events do we want (integer)?\n");
  scanf("%d",&phonum);
  double errorbar1;
  double errorbar2;
  int absorbnum = 0; // Initialize the number of photons absorbed
  double efficiency; // The efficiency of the scintillator in the simulation
  int fiber_col = 0; // The number of times any fiber is impacted
  int inout;
  double phi1, phi2; // angles
  double P0 [3],v_dir [3], LdotN, PLdotN, DD, DI;//some vector-coordinates
  // 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)


  // 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 defines points on the planes and normal vectors to the planes which ar
e used
  // 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 the escape chance for a photon


  // This block writes the coordinates of the center of each fiber strand
  // This will take a while since there are 96 of them distributed throughout
  // the scintillator. I define them in the system shown in a model and then
  // do a transformation on all of them that shifts them 30 degrees over, agreei
```

```
ng
  // with my original orientation of the hexagon
  // All of my length measurements are in centimeters, my origin is the hexagon
  // center
  // This neglects the 0.0025cm uncertainty in the measurements

  double Xcen [96], Ycen [96]; // will fill up to Xcen/Ycen[95]

  // First the Y's
  for (int j=0;j<=9;j++) {// 3 lines have 10 fibers
    Ycen[j] = 0;// j:[0,9]
    Ycen[j+19] = 1.876;//j:[19,28]
    Ycen[j+62] = -1.876;//j:[62,71]
  }
  for (int j=10;j<=18;j++) {// 4 lines have 9 fibers
    Ycen[j] = 0.938; // fibers are spaced (y-axis) by 0.938cm
    Ycen[j+19] = 2.814;//j:[10,18],[29,37]
    Ycen[j+43] = -0.938;//j:[53,61]
    Ycen[j+62] = -2.814;//j:[72,80]
  }
  for (int j=38;j<=45;j++) {// 2 lines have 8 fibers
    Ycen[j] = 3.752;//j:[38,45]
    Ycen[j+43] = -3.752;//j:[81,88]
  }
  for (int j=46;j<=50;j++) {// 2 lines have 5 fibers
    Ycen[j] = 4.690;//j:[46,50]
    Ycen[j+43] = -4.690;//j:[89,93]
  }
  Ycen[51] = 5.625;// These are just the last 4 I did manually
  Ycen[52] = 5.625;
  Ycen [94] = -5.625;
  Ycen [95] = -5.625;

  // Now the X's
  // Sorry but I'm going to brute-force this

  double x_sep = 1.083;// The spacing of the scinitillator's fibers
  // along the x-axis
  Xcen[0]=-4.874; Xcen[19]=Xcen[0]; Xcen[62]=Xcen[0];
  // That's just where it is on the scintillator
  // Displaced by 4.874 cm at its furthest
  for (int j=1;j<=9;j++) { // These finish the lines of 10
    Xcen[j] = Xcen[j-1]+x_sep;//j:[1,9]
    Xcen[j+19] = Xcen[j-1]+x_sep;//j:[20,28]
    Xcen[j+62] = Xcen[j-1]+x_sep;//j:[63,71]
  }
  for (int j=5;j<=9;j++) {
    Xcen[j]+=0.001;
    Xcen[j+19]+=0.001;
    Xcen[j+62]+=0.001;
  }

  Xcen[10]=-4.332; Xcen[29]=Xcen[10];
  Xcen[53]=Xcen[10]; Xcen[72]=Xcen[10];
  for (int j=11;j<=18;j++) {// These finish the lines of 9
    Xcen[j] = Xcen[j-1]+x_sep;//j:[11,18]
    Xcen[j+19] = Xcen[j-1]+x_sep;//j:[30,37]
    Xcen[j+43] = Xcen[j-1]+x_sep;//j:[54,61]
    Xcen[j+62] = Xcen[j-1]+x_sep;//j:[73,80]
  }

  Xcen[38]=-3.791; Xcen[81]=Xcen[38];
  for(int j=39;j<=45;j++) {// These finish the lines of 8
    Xcen[j] = Xcen[j-1]+x_sep;//j:[39,45]
    Xcen[j+43] = Xcen[j-1]+x_sep;//j:[82,88]
  }
  for (int j=42;j<=45;j++) {
    Xcen[j]+=0.001;
    Xcen[j+43]+=0.001;
```



```
  }

  Xcen[46]=-2.166;Xcen[89]=Xcen[46];
  for(int j=47;j<=50;j++) {
    Xcen[j] = Xcen[j-1]+x_sep;//j:[47,50]
    Xcen[j+43] = Xcen[j-1]+x_sep;//j:[90,93]
  }
  Xcen[51] = 0.542; Xcen[52] = -0.542;
  Xcen[94] = 0.542; Xcen[95] = -0.542;

  // Whew, all done with that. Now to rotate the axis by pi/6
  // x'=xcos(phi)+ysin(phi) and y'=-xsin(phi)+ycos(phi) (I hope)

  for (int j=0;j<96;j++) {
    Xcen[j] = Xcen[j]*cos(PI/6.)+Ycen[j]*sin(PI/6.);
    Ycen[j] = -Xcen[j]*sin(PI/6.)+Ycen[j]*cos(PI/6.);
  }
  // And that should be all of the fibers :)


  // This block creates variables that will make the circle collision calcs easi
er
  double MM,slope,del;
  double x1,x2,y1,y2,z1,z2,dz1,dz2;
  // absorb is calculated from the photon's path through the fiber, a_resist is
  // the random double to test if the photon is absorbed. absorb is higher for
  // a more likely absorption.
  double absorb; // double between 0 and 1, with 1 as 100%  absorb chance
  double a_resist; // random double[0,1], if less than absorb it's absorbed
  double travel_distance;// 3-space through the fiber the photon traverses

  //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

  int ref_in_fiber;// A binary condition that checks if a photon's point of boun
dary
  // collision is inside one of the fibers, which case I don't let it escape
  // This is due to the nature of the apparatus: the fibers run through many sho
wer
  // hexes, so the photon would pass into the next one.
  double coinflip;// A way to fix my earlier incorrect Azimuthal without alterin
g
  // 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);
    //    if (ii<2)    printf("%lf %lf - initial\n",P0[0],P0[1]);
    // Switch X and Y to the proper axes
    P0[0] = P0[0]*cos(PI/6.)+P0[1]*sin(PI/6.);
    P0[1] = -P0[0]*sin(PI/6.)+P0[1]*cos(PI/6.);
```

```
    //    if (ii<2)    printf("%lf %lf - final\n",P0[0],P0[1]);
    /*
    P0[0] = 0.;// Using constant start-points for the _ study
    P0[1] = 0.;
    P0[2] = 0.1;

    P0[0] = randd(-outr,outr);
    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 [0,2pi]
    coinflip=randd(0.0,2.0);
    if (coinflip>1.0) {
      phi2 = randd(3.*PI/2,2.*PI);
    }
    else {
      phi2 = randd(0.,PI/2.);
    }
    ref_in_fiber= 0;
    v_dir[0] = cos(phi1)*cos(phi2); v_dir[1] = sin(phi1)*cos(phi2);
    v_dir[2] = sin(phi2);


    // This section checks if photon spawns inside hexagon and not corners of re
ctangle
    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)) {
      inout = 0;
    }
    else {
      inout = 1;
    }

    for (int j=0;j<96;j++) { // This checks if it spawns inside a fiber
      if (pow(P0[0]-Xcen[j],2)+pow(P0[1]-Ycen[j],2)<=pow(fibr,2)) {
        inout = 0;
        printf("Your inputs were bad and you should feel bad");
        break;
      }
    }

    if (inout == 0) { // Reset bad initial points
      ii--;
      printf("Bad %lf %lf %lf\n",P0[0],P0[1],P0[2]);
    }


    //// This section is what happens once a good starting position in generated
    else if (inout == 1){
      // Record initial point in points.dat file
      if (ii<1000){
        fprintf(fp,"%d %lf %lf %lf\n",ii,P0[0],P0[1],P0[2]);
        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

        //// This section test for fiber intersections
        // Line has eqn. [y=mx+d], Circle has eqn. [(x-a)^2+(y-b)^2=r^2]
        // Point-slope form gives y-P0[1]=tan(phi1)*(x-P0[0])
        // m=tan(phi1), d=P0[1]-tan(phi1)*P0[0], a=>Xcen[j], b=>Ycen[j], r=>fibr
        // m=>MM, d=>slope
        MM = tan(phi1);
        slope = P0[1]-(tan(phi1)*P0[0]);

        for (int j=0;j<96;j++) {// This loop goes through each fiber to collisio
n check
          del = pow(fibr,2.)*(1+pow(MM,2.))-pow((Ycen[j]-MM*Xcen[j]-slope),2.);
          if (del>0) {
```



```
            // There is a real solution to the intersection
            x1 = (Xcen[j]+(Ycen[j]*MM)-(slope*MM)+(sqrt(del)))/(1+pow(MM,2.));
            x2 = (Xcen[j]+(Ycen[j]*MM)-(slope*MM)-(sqrt(del)))/(1+pow(MM,2.));
            y1 = (MM*x1)+slope;// A fairly simple assumption as it must still be
            y2 = (MM*x2)+slope;// on the line
            // Find z1 and z2 for the intersections: dz=magnitude distance
            // in xy plane * tan(phi2), the vertical angle
            dz1 = tan(phi2)*sqrt(pow(P0[0]-x1,2.)+pow(P0[1]-y1,2.));
            dz2 = tan(phi2)*sqrt(pow(P0[0]-x2,2.)+pow(P0[1]-y2,2.));
            z1=P0[2]+dz1;// Now need to check if these are inside
            z2=P0[2]+dz2;// the scintillator or not
            if (((z1>=0.0) && (z1<=depth)) && ((z2>=0.0) && (z2<=depth))) {
              // This is the easiest version; there are no boundary hits midway
              travel_distance = sqrt(pow(x2-x1,2.)+pow(y2-y1,2.)+pow(z2-z1,2.));
              absorb = 1.0-pow(10.0,(-1.276)*(travel_distance*10.));
              // Travel distance * 10 is because I used cm instead of mm
              a_resist = randd(0.0,1.0);
              fiber_col++;
              if (absorb>a_resist) {
                absorbnum++;

                if(sqrt(pow(P0[0]-x1,2)+pow(P0[1]-y1,2)+pow(P0[2]-z1,2))>\
                   sqrt(pow(P0[0]-x2,2)+pow(P0[1]-y2,2)+pow(P0[2]-z2,2)))
                  travel_total+=sqrt(pow(P0[0]-x2,2)+pow(P0[1]-y2,2)+pow(P0[2]-z
2,2));
                else travel_total+=sqrt(pow(P0[0]-x1,2)+pow(P0[1]-y1,2)+pow(P0[2
]-z1,2));

                lambda_b = 20.4*(1.-exp(-travel_total/13.1))+0.42*travel_total;
                // Constants provided by reference materials for the scintillato
r
                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<100) fprintf(op,"%d %lf %lf %lf -2 %lf\n",ii,P0[0],P0[1]
,P0[2],travel_total);
                  inout = 0;
                  break;
                }

                else {
                  if(ii<100) fprintf(op,"%d %lf %lf %lf -1 %lf\n",ii,P0[0],P0[1]
,P0[2],travel_total);
                  inout=0;
                  break;
                }
              }
            }

            // Now need to check in case the photon is hitting a bound before es
caping
            // In this case, the photon cannot escape the system
            else if (((z1<=0.0) || (z1>=depth)) && (z2>=0.0) && (z2<=depth)) {
              ref_in_fiber = 1;
              travel_distance = sqrt(pow(x2-x1,2.)+pow(y2-y1,2.)+pow(z2-z1,2.));
              absorb = 1.0-pow(10.0,(-1.276)*(travel_distance*10.));
              // Travel distance * 10 is because I used cm instead of mm
              a_resist = randd(0.0,1.0);
              fiber_col++;
              if (absorb>a_resist) {
                absorbnum++;

                if(sqrt(pow(P0[0]-x1,2)+pow(P0[1]-y1,2)+pow(P0[2]-z1,2))>\
                   sqrt(pow(P0[0]-x2,2)+pow(P0[1]-y2,2)+pow(P0[2]-z2,2)))
                  travel_total+=sqrt(pow(P0[0]-x2,2)+pow(P0[1]-y2,2)+pow(P0[2]-z
2,2));
                else travel_total+=sqrt(pow(P0[0]-x1,2)+pow(P0[1]-y1,2)+pow(P0[2
```

```
]−z1,2));

                lambda_b = 20.4*(1.−exp(−travel_total/13.1))+0.42*travel_total;
                // Constants provided by reference materials for the scintillato
r
                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) {
                  atten_num++;
                //Record points and finish
                  if(ii<100) fprintf(op,"%d %lf %lf %lf −2 %lf\n",ii,P0[0],P0[1]
,P0[2],travel_total);
                  inout = 0;
                  break;
                }

                else {
                  if(ii<100) fprintf(op,"%d %lf %lf %lf −1 %lf\n",ii,P0[0],P0[1]
,P0[2],travel_total);
                  inout=0;
                  break;
                }
              }
            }

            else if ((z1>=0.0) && (z1>=depth) && ((z2<=0.0) || (z2>=depth))) {
              ref_in_fiber = 1;
              travel_distance = sqrt(pow(x2−x1,2.)+pow(y2−y1,2.)+pow(z2−z1,2.));
              absorb = 1.0−pow(10.0,(−1.276)*(travel_distance*10.));
              // Travel distance * 10 is because I used cm instead of mm
              a_resist = randd(0.0,1.0);
              fiber_col++;
              if (absorb>a_resist) {
                absorbnum++;

                if(sqrt(pow(P0[0]−x1,2)+pow(P0[1]−y1,2)+pow(P0[2]−z1,2))>\
                   sqrt(pow(P0[0]−x2,2)+pow(P0[1]−y2,2)+pow(P0[2]−z2,2)))
                  travel_total+=sqrt(pow(P0[0]−x2,2)+pow(P0[1]−y2,2)+pow(P0[2]−z
2,2));
                else travel_total+=sqrt(pow(P0[0]−x1,2)+pow(P0[1]−y1,2)+pow(P0[2
]−z1,2));

                lambda_b = 20.4*(1.−exp(−travel_total/13.1))+0.42*travel_total;
                // Constants provided by reference materials for the scintillato
r
                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) {
                  atten_num++;
                //Record points and finish
                  if(ii<100) fprintf(op,"%d %lf %lf %lf −2 %lf\n",ii,P0[0],P0[1]
,P0[2],travel_total);
                  inout = 0;
                  break;
                }

                else {
                  if (ii<100) 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;
        //// Boundary collisions: Calc the point where the photon hits the scint
```



```
illator's
        // outer boundary
        int boundary = 0;
        double score_to_beat = 1000.0;


        // Calculate distance to each planar intersection point (the shortest on
e will be
        // 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]));// the distance to intersection

          if ((DI <= score_to_beat) && (DI>0.0001)) {

            // 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 ((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*outr+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;
              }

            }

          }
        } // Closes loop that checks the distances 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, and may possibly be simplified
        // 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 (ref_in_fiber ==1) escape = 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<floor_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<floor_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 and new point 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<100){
          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 %d %lf\n",ii,P0[0],P0[1],P0[2],boundary,tra
vel_total);
          // printf("%d\n",ii);
        }
      }// Closes the "while photon hasn't escaped" loop
    }// Closes the photon number that is in the hex loop
  }// Closes the photon number loop


  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 (SS)\n",XXX,YYY,ZZZ);
  printf("eff %lf %lf %lf %lf %lf %lf %lf\n",XXX,YYY,ZZZ,efficiency,survive,\
         errorbar1,errorbar2);
  return (0);
}

// RNG function
/* 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);
}
```