Chapter 10

Geometry

In particle transport or ray tracing for applications such as the Monte Carlo simulation of particles being transported through media or vacuum, the inclusion of boundaries permits the development of applications of particle transport in geometrical objects. These objects can be very simple, such as a planar interface between two semi-infinite media or a collection of geometrical objects that define a nuclear power vessel or a complex radiation detector.

The technique to address both of these problems is essentially the same. When a particle is about to be transported it knows that it is in a region of space identified by a region number (or some other unique characteristic), its position, its direction and its proposed flight distance (*e.g.* the distance to an interaction point). It also knows that its current location in space is bounded by a finite set of surfaces. So, it interrogates the geometry code associated with each of these to find out which surface it hits, if any, and the distance to that surface.

For most Monte Carlo applications the essential questions to be answered by the geometry modelling code are:

- If my particle starts off at position $\vec{x}_0$ with direction vector μ, and it "flies" in a straight line up to a maximum distance s, will it hit a given surface?
- If the answer to the above is "yes", what is the distance to the intersection?
- And is the answer to the first question above is "yes", what is the region number that lies on the other side?

In this chapter the general problem of solving for the distance to the intersection point of a directed straight line with an arbitrary plane or an arbitrary quadric surface is developed. Quadric surfaces include, for example, spheres, cylinders and cones as well as a few less familiar surfaces. A general strategy for boundary-crossing logic is presented which circumvents ambiguities associated with numerical precision and end-of-step directional uncertainties.

The specific examples of surfaces given are planes, circular cylinders, spheres and circular cones with arbitrary orientation and position. Care is taken to develop the mathematical equations so that they can be computed with numerical accuracy and a discussion on the influence of machine precision on the accuracy of results is given.

This simple design allows the Monte Carlo transport routine to concern itself only with the problem of transport of particles in infinite media and it only needs to know the composition of the medium that particles are being transported in. Thus, when the new region number is communicated to the Monte Carlo transport routine, it merely has to check in its look-up tables to see whether or not the medium has changed and then take appropriate action. This decoupling of transport physics and geometry is a very powerful generalizing feature and permits arbitrary flexibility in specifying the geometry in such a way that the underlying transport mechanisms are identical for all applications.

10.1 Boundary crossing

Let us imagine that a particle is being transported in vacuum, that its initial position is $\vec{x}_0$, its region number is N_0, it has direction $\vec{\mu}$ and that there is only one surface bounding its current region. The geometry code is then interrogated to see of the flight path will strike the surface.

If the geometry modelling code replies:

- Yes, I will hit a certain surface along my flight path,
- The distance to the intersection is s,
- The region number that lies on the other side is N,

particle transport is effected using the equation:

$$\vec{x} = \vec{x}_0 + \vec{\mu}s, \tag{10.1}$$

and one assumes naturally that mathematically (or logically) that the new position is exactly $\vec{x}$ and that any test of where the particle is will yield the answer N for its region number.

However, numerics and mathematics often differ.

In an "ideal" computer where floating point numbers could be specified to absolute accuracy, there would be no ambiguity. However, real computers represent real numbers to within a finite, non-zero precision.[1] Is the particle on a surface or not? Is it exactly on the surface or

[1]It is possible to recode geometry transport in integer arithmetic, thereby avoiding ambiguities. This "quantisation" of space approach has its drawbacks and further discussion would take us out of the scope of this discussion.

has truncation caused an "undershoot" or round-up caused an overshoot? The problem of the finite precision of floating point numbers in computers introduces three position-dependent possibilities that one must consider:

undershoot The transport distance does not quite reach the surface,

exact Numerically, the particle is exactly on the surface,

overshoot The transport distance is slightly overestimated so that the surface is actually crossed.

All of these possibilities occur with varying frequency during the course of a Monte Carlo calculation. In fact, it is correct to say that if you run a geometry code through enough examples with stochastic selection of input parameters, then everything that can happen *will* happen. Therefore, it is necessary to write geometry coding that is robust enough to handle all of these possibilities and also to be aware of the error handling that must be introduced. It has been suggested that all geometry be coded in double or extended precision to avoid these kinds of ambiguities. However, one must realise that double or extended precision does not mean absolute precision. Higher precision reduces the *size* of the undershoot or overshoot but does nothing to cure undershoot or overshoot ambiguities. A geometry code that survives using single precision arithmetic will work at higher precision providing that the coding does not make some intrinsic assumptions on precision or scale. The converse is not true. The routines developed for this course will work for both single and higher precision and the regions of validity for use with both single and double precision is investigated.

There is yet another complicating factor. Transport to a surface can cause the direction of a particle to change! If the boundary represents a reflecting surface, the direction will change according to the law of reflection (discussed in a later chapter). Many charged particle transport schemes apply multiple scattering angular deflection to electron/positron trajectories in media when the flight path is interrupted by a boundary. (This point is discussed further in a later chapter.) Or, with less probability is the possibility that an exact boundary intersection corresponds with the exact numerical point of interaction causing a deflection. The various possibilities are depicted in Figure 10.1.

Fortunately there is a general error handling strategy that resolves these ambiguities:

- Do not calculate flight distances to boundaries that the particle is assumed to be headed away from based on its known direction. The handles the undershoot and exact hit problem in the case the particle assumes that it is still directed away from the surface that it just crossed. However, the particle has to know in advance whether it is "inside" or "outside" of a surface to start with.

- If the flight distance returned is less than or equal to zero, set the transport step to zero and assume that the transport step causes a region change. This strategy resolves

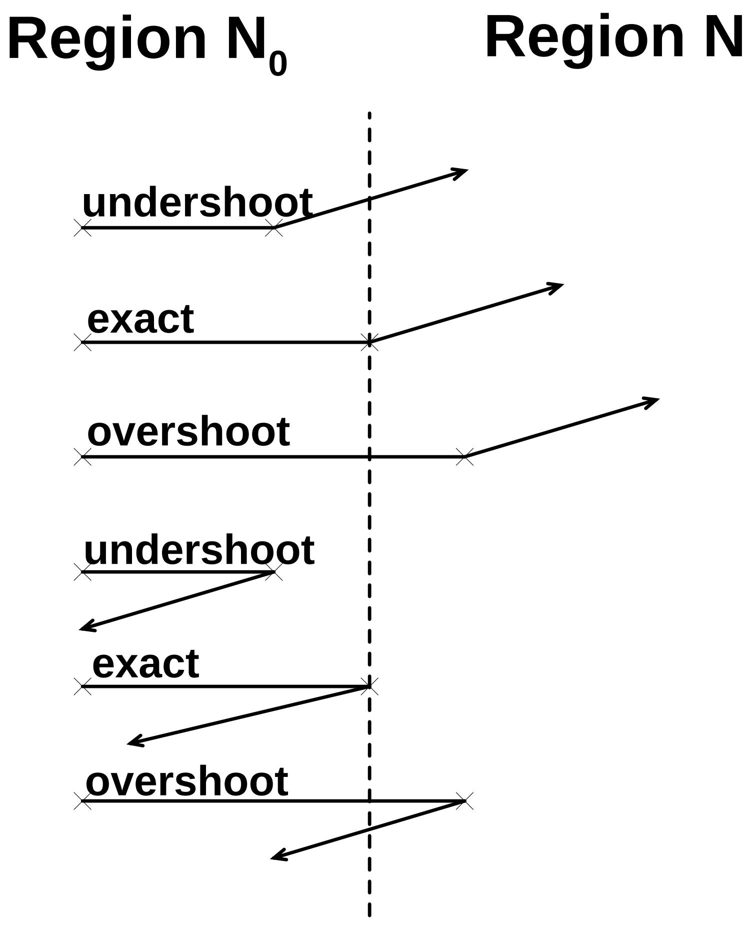

Figure 10.1: Boundary crossing of a particle across a boundary. There are six possibilities corresponding to undershoot, exact surface position, and overshoot with either forward scatter or backscatter.

ambiguities associated with undershoots and exact hits when the particle backscatters form the surface.

Remarkably, this simple logic resolves all boundary transport ambiguities. Its calculational efficiency is excellent, it is completely independent of the scale of the geometry with respect to the transport pathlengths and it heals precision errors "on the spot" not allowing them to accumulate.

Let us enumerate all the possibilities of boundary crossing and demonstrate how this boundary-crossing logic resolves all problems.

undershoot, forward scatter The transport distance does not quite reach the surface, but assumes logically that it is in the next region. The subsequent transport step ignores the surface because it is assumed to be heading away from it. Note that the positioning error is healed "on the spot" in the next transport step.

exact, forward scatter Numerically, the particle is exactly on the surface, but assumes that it is in the next region. The subsequent transport step ignores the surface because it is assumed to be heading away from it.

overshoot, forward scatter The transport distance is slightly overestimated so that the surface is actually crossed. There is no ambiguity in this case since the numerical position and the logical placement of the particle are in agreement.

undershoot, backscatter The transport distance does not quite reach the surface, but assumes logically that it is in the next region. The subsequent transport step checks this surface because it is assumed to be heading toward it. The calculation of the transport distance to the surface yields a small, negative distance. Set the transport distance to zero and let the next step transport the particle to this surface, doing nothing more than resetting the region number to the original. Note that the error is absorbed by the next transport step.

exact, backscatter The transport distance brings the particle exactly to the surface, and the logic assumes that it is in the next region. The subsequent transport step checks this surface because it is assumed to be heading toward it. The calculation of transport distance to the surface yields an exact zero. Employ this zero transport distance and let the next step transport the particle to this surface, doing nothing more than resetting the region number to the original.

overshoot, backscatter The transport distance is slightly overestimated so that the surface is actually crossed. There is no ambiguity in this case since the numerical position and the logical placement of the particle are in agreement. The next transport will be a small, positive step.

It is clear that if there is an overshoot, there is no ambiguity. It is tempting to think that by arranging for an overshoot (a small amount related to the relative or absolute floating point precision) in every case will resolve all ambiguities and be efficient as well. The problem with this approach is that is works well for normal incidence but not very well for grazing incidence. Certain implementations of this strategy introduce a scale dependence into the transport scheme, assume that errors in steps of a certain size can be ignored. Such strategies will either be ineffective if the geometries are very large or introduce positioning errors if the geometrical elements are very small.

10.2 Solutions for simple surfaces

10.2.1 Planes

The general equation for a plane of arbitrary orientation is:

$$\vec{n} \cdot (\vec{x} - \vec{P}) = 0 \tag{10.2}$$

where $\vec{n}$ is the unit normal of the plane and $\vec{P}$ is any point on the surface of the plane. Note the use of the inner product, $\vec{n} \cdot \vec{x} \equiv n_x x + n_y y + n_z z$.

Intersection distance

Inserting Equation 10.1 into Equation 10.2 and solving for s gives:

$$s = -\frac{\vec{n} \cdot (\vec{x}_0 - \vec{P})}{\vec{n} \cdot \vec{\mu}} \tag{10.3}$$

We remark that there is no solution ($s = \infty$) when the particle direction is perpendicular to the normal of the plan ($\vec{n} \cdot \vec{\mu} = 0$). This is the solution of a particle travelling parallel to a plane and never hitting it. Only positive solutions for s are acceptable and this depends upon whether or not the particle is travelling towards the plane.

Adopting the convention that a particle is considered to be outside the plane of it is on the side that the unit normal, $\vec{n}$, is pointing, we enumerate the possibilities:

Case I $\underline{\vec{n} \cdot \vec{\mu} = 0}$

Trajectory is parallel to the plane, no solution

Case II $\underline{\vec{n} \cdot (\vec{x}_0 - \vec{P}) \geq 0 \text{ and the particle is assumed to start from the outside}}$

1. If $\vec{n} \cdot \vec{\mu} < 0$, $s = -\vec{n} \cdot (\vec{x}_0 - \vec{P})/\vec{n} \cdot \vec{\mu}$

2. Elseif $\vec{n} \cdot \vec{\mu} > 0$, no solution

Case III $\underline{\vec{n} \cdot (\vec{x}_0 - \vec{P}) \leq 0 \text{ and the particle is assumed to start from the inside}}$

1. If $\vec{n} \cdot \vec{\mu} > 0$, $s = -\vec{n} \cdot (\vec{x}_0 - \vec{P})/\vec{n} \cdot \vec{\mu}$
2. Elseif $\vec{n} \cdot \vec{\mu} < 0$, no solution

Case IV $\underline{\vec{n} \cdot (\vec{x}_0 - \vec{P}) < 0 \text{ but the particle is assumed to start from the outside}}$

1. If $\vec{n} \cdot \vec{\mu} < 0$, $s = 0$ effecting a region change
2. Elseif $\vec{n} \cdot \vec{\mu} > 0$, no solution

Case V $\underline{\vec{n} \cdot (\vec{x}_0 - \vec{P}) > 0 \text{ but the particle is assumed to start from the inside}}$

1. If $\vec{n} \cdot \vec{\mu} > 0$, $s = 0$ effecting a region change
2. Elseif $\vec{n} \cdot \vec{\mu} < 0$, no solution

The case of a parallel trajectory is handled by **Case I**. The two "normal" conditions in **Case II** and **Case III** handle the eventuality where the particle is exactly on the plane, $\vec{n} \cdot (\vec{x}_0 - \vec{P}) = 0$. **Case IV** and **Case V** handle the anomalies. In the case of an undershoot and backscatter out of the region where the particle thinks it is, then a zero distance is returned so that the next transport step will switch to the correct region number. If the case of an undershoot and forward scatter, no solution is given allowing other surfaces in the geometry to determine the intersection. Note that no correction is made for the undershoot distance and this will be included in the next transport step. Therefore, numerical inaccuracies are not allowed to accumulate.

The FORTRAN codes which accomplishes this for a plane normal to the z axis is:

```
C23456789|123456789|123456789|123456789|123456789|123456789|123456789|12

      subroutine zplane(p0,z0,w,s,inside,hit)
      implicit none

C  Calculates the distance of a particle to a planar surface normal to the
C  z-axis

      logical
     *    inside ! Input:  inside = .true.  => particle thinks it is inside
     *    ,      ! Input:  inside = .false. => particle thinks it is outside
     *    hit    ! Output: hit    = .true.  => particle would hit  the surface
                 ! Output: hit    = .false. => particle would miss the surface
```

```
 real
*    p0,    ! Input:  Point at which the plane cuts the z-axis
*    z0,    ! Input:  z-coordinate of the particle
*    w ,    ! Input:  z-axis direction cosine of the particle
*    s      ! Output: Distance to the surface (if hit)

 if (
*    (inside.and.w.gt.0e0)      !headed towards the surface
*    .or.
*    (.not.inside.and.w.lt.0e0) !headed towards the surface
*   )
*then
     hit = .true.
     s   = max(0e0,(p0-z0)/w)
     return
 else
     hit = .false.
     return
 endif

 end
```

10.3 General solution for an arbitrary quadric

Borrowing from the notation of Olmsted [Olm47], an arbitrary quadric surface in 3(x,y,z)-space[2] can be represented by:

$$f(\vec{x}) = \sum_{i,j=0}^{3} a_{ij}x_i x_j = 0. \tag{10.4}$$

The a_{ij}'s are arbitrary constants and the 4-vector x_i has components $(1, x, y, z)$. The zeroth component is unity by definition allowing a very compact representation and a_{ij} is symmetric with respect to the interchange of i and j, that is $a_{ij} = a_{ji}$. Equation 10.4 is very general and encompasses a wide variety of possibilities including solitary planes (*e.g.* only a_{0i} non-zero), intersecting planes (*e.g.* only a_{11} and a_{22} non-zero), cylinders (circular, elliptical, parabolic and hyperbolic), spheres, spheroids and ellipsoids, cones (circular and elliptical), hyperboloids of one and two sheets and elliptic and hyperbolic paraboloids. These surfaces

[2]The only variance with Olmsted's notation is that the 4th component is labelled as the 0th component in this work.

can be combined to make geometrical objects of arbitrary complexity and are extremely useful in Monte Carlo modeling of physical objects.

Despite having apparently 10 independent constants, Equation 10.4 represents only 10 independent real surfaces (including the simple plane), unique after a translation and rotation to standard position. The three cross terms (a_{ij} for $i \neq j$ and $i, j \geq 1$) can be eliminated by rotation. The resultant equation then only involves terms like x_i^2 and x_i. In addition, providing that a given variable's quadratic constant is non-zero, the linear terms can be eliminated by a translation. The result is that there are only two generic forms:

$$f(\vec{x}) = \sum_{i=1}^{3} a_i x_i^2 + c = 0, \tag{10.5}$$

and

$$f(\vec{x}) = \sum_{i=1}^{2} a_i x_i^2 + b x_3 = 0. \tag{10.6}$$

Equations 10.5 and 10.6 describe only 10 distinct possibilities with real solutions.

1. **ellipsoids:** $a_1^2 x_1^2 + a_2^2 x_2^2 + a_3^2 x_3^2 - c^2 = 0$.
2. **cones:** $a_1^2 x_1^2 + a_2^2 x_2^2 - a_3^2 x_3^2 = 0$.
3. **cylinders:** $a_1^2 x_1^2 + a_2^2 x_2^2 - c^2 = 0$.
4. **hyperboloids of one sheet:** $a_1^2 x_1^2 + a_2^2 x_2^2 - a_3^2 x_3^2 - c^2 = 0$.
5. **hyperboloids of two sheets:** $a_1^2 x_1^2 + a_2^2 x_2^2 - a_3^2 x_3^2 + c^2 = 0$.
6. **elliptic paraboloids:** $a_1^2 x_1^2 + a_2^2 x_2^2 + a_3 x_3 = 0$.
7. **hyperbolic paraboloids:** $a_1^2 x_1^2 - a_2^2 x_2^2 + a_3 x_3 = 0$.
8. **hyperbolic cylinders:** $a_1^2 x_1^2 - a_2^2 x_2^2 + c^2 = 0$.
9. **parabolic cylinders:** $a_1^2 x_1^2 + a_3 x_3 = 0$.
10. **simple planes:** $a_3 x_3 + c = 0$.

The first nine of these are shown[3] in Figure 10.2. (The magnitude of the above constants were all chosen to be unity for the purposes of display. Consequently, the first six of these surfaces shown exhibit at least one axis of rotational symmetry.)

There are other imaginary surfaces (*e.g.* imaginary ellipsoids $a_1^2 x_1^2 + a_2^2 x_2^2 + a_3^2 x_3^2 + c^2 = 0$) that we will not consider nor will we consider quadrics that can be made up of two

[3]Thanks to Keath Borg (wherever you are!) for generating the data for these figures.

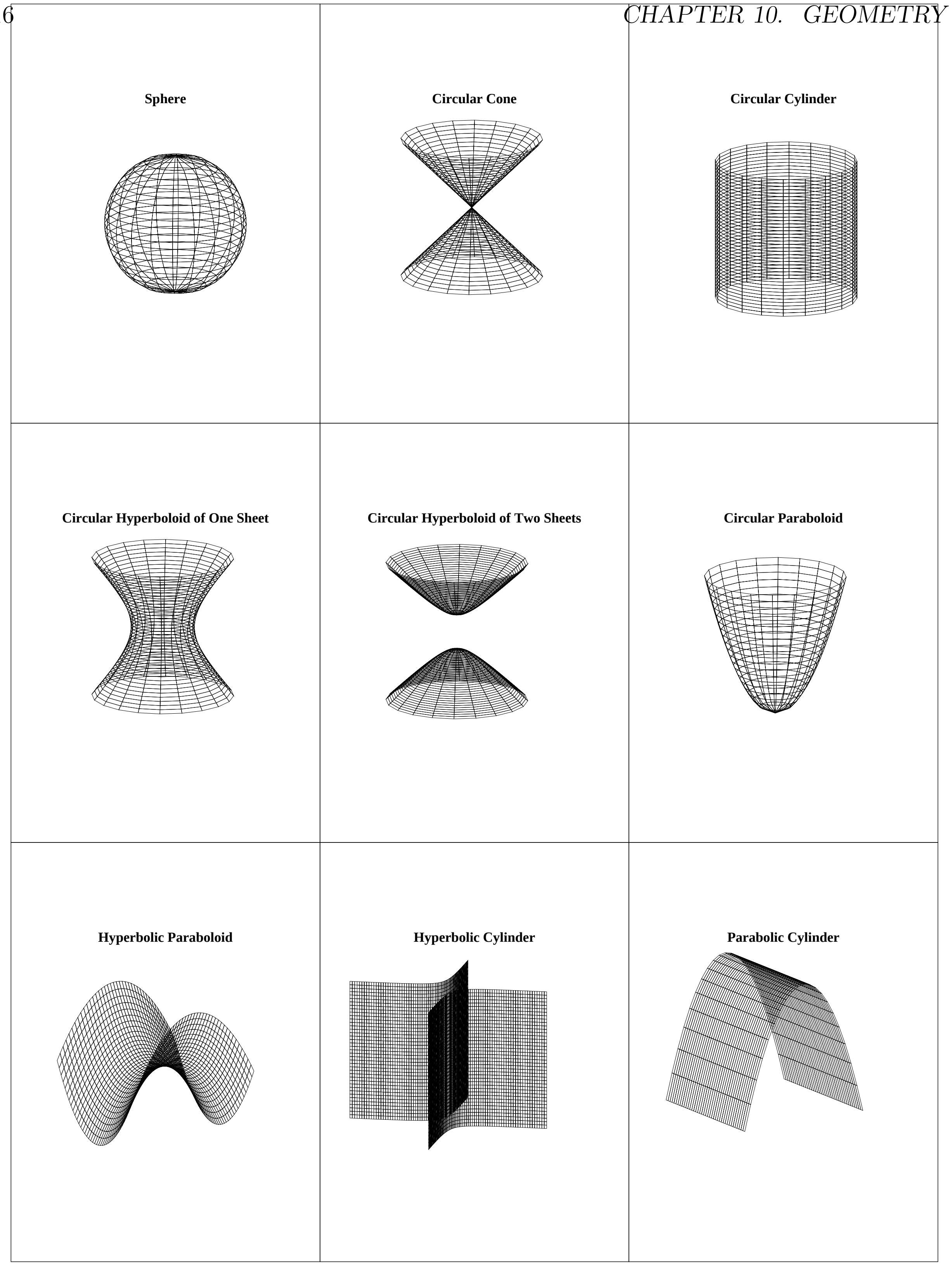

Figure 10.2: The nine real non-planar quadric surfaces.

independent planes in various orientations (*e.g.* intersection planes $a_1^2x_1^2 - a_2^2x_2^2 = 0$, parallel planes $a_1^2x_1^2 - c^2 = 0$, and coincident planes $a_1^2x_1^2 = 0$).

For more information on the reduction to canonical form, the reader is encouraged to read Olmsted's book [Olm47]. Olmsted also gives the classification of the surfaces and lists the entire set of 17 canonical quadric forms.

10.3.1 Intercept to an arbitrary quadric surface?

We now change notation slightly and write the particle trajectory by the parametric equation:

$$\vec{x} = \vec{p} + \vec{\mu}s \tag{10.7}$$

where the starting position of the particle is $\vec{p} = (p_x, p_y, p_z)$. A positive value of s expresses a distance along the direction that the particle is going (forward trajectory) and a negative value is associated with a distance that the particle came from (backward trajectory). Thus, negative solutions that are found for s below will be rejected.

In Monte Carlo particle transport calculations as well as ray-tracing algorithms a common problem is to find the distance a particle has to travel in order to intersect a surface. This is done by substituting for $\vec{x}$ from Equation 10.7 in Equation 10.4 to give:

$$s^2\left(\sum_{i,j=0}^{3} a_{ij}\mu_i\mu_j\right) + 2s\left(\sum_{i,j=0}^{3} a_{ij}p_i\mu_j\right) + \left(\sum_{i,j=0}^{3} a_{ij}p_ip_j\right) = 0, \tag{10.8}$$

where we have adopted the convention that $\mu_0 = 0$ and $p_0 = 1$. This is a quadratic equation in s of the form $A(\vec{\mu})s^2 + 2B(\vec{\mu},\vec{p})s + C(\vec{p}) = 0$ where $A(\vec{\mu}) = \sum_{i,j=0}^{3} a_{ij}\mu_i\mu_j$, $B(\vec{\mu},\vec{p}) = \sum_{i,j=0}^{3} a_{ij}p_i\mu_j$ and $C(\vec{p}) = \sum_{i,j=0}^{3} a_{ij}p_ip_j$.

Interpretation of the quadratic constants

The constant $C(\vec{p})$ is identically zero when $\vec{p}$ is on the surface. When $\vec{p}$ is not on the surface, the sign of $C(\vec{p})$ can be interrogated to see if the particle is inside or outside. There is some arbitrariness in the definition of what is "inside" or "outside". A sphere with radius R, for example, has the form $\vec{p}^2 - R^2 = 0$ and in this case $C(\vec{p}) > 0$ when $|\vec{p}| > R$. So, for this example, $C(\vec{p})$ is positive when $\vec{p}$ is outside and negative when inside. However, the same sphere is defined by $R^2 - \vec{p}^2 = 0$ giving opposite interpretation for the signs of $C(\vec{p})$ for points inside and outside. It is best to adopt a constant interpretation and be aware that two points on opposite sides of the surface in the sense that a line joining them intersects the surface only once, have different signs.

For planes and most of the other surfaces, "inside" and "outside" are arbitrary since multiplying Equation 10.5 or Equation 10.6 by a minus sign leaves the surface intact. However, there is one natural interpretation provided by the calculation of the normal to the surface, $\nabla f(\vec{p})$, where $\vec{p}$ is a point on the surface, *i.e.* $C(\vec{p}) = 0$. In the way they were defined, $\nabla f(\vec{p})$ points to the "outside" region which can be defined as follows: If more than one line can be drawn through a point such that the surface is not intersected in either the forward or backward direction, then this point is on the outside. If at most only one such line exists, then the point is on the "inside". This defines the inside and outside in a unique and natural way (inside a sphere, for example). There are three exceptions to this rule, the simple plane, the hyperboloid of one sheet and the hyperbolic paraboloid. In their standard forms given below Equation 10.6, the outside or inside of a plane is completely arbitrary, the outside of a hyperbolic paraboloid contains the positive x_3-axis and the inside of the hyperboloid of one sheet contains the x_3-axis which also seems to be a "natural" choice.

The constant $B(\vec{\mu}, \vec{p})$ is related to the inner product of the particle's direction $\vec{\mu}$ with the normal to the surface at a point $\vec{p}$ when $\vec{p}$ is on the surface. Specifically, $B(\vec{\mu}, \vec{p}) = \sum_{i,j=0}^{3} a_{ij} p_i \mu_j = \frac{1}{2}\vec{\mu} \cdot \nabla f(\vec{p})$. When $\vec{p}$ is on the surface $\nabla f(\vec{p})$ is its normal there. This can be exploited to decide to which side of a surface a particle is going if it happens to be on the surface and is pointed in some direction. Imagine a particle on the surface at point $\vec{p}$ with some direction $\vec{\mu}$ and consider an infinitesimal step ϵ. The sign of $C(\vec{p} + \vec{\mu}\epsilon) = 2\epsilon B(\vec{\mu}, \vec{p}) + O(\epsilon^2)$ will have the sign of $B(\vec{\mu}, \vec{p})$. If $B(\vec{\mu}, \vec{p}) = 0$ for $\vec{p}$ on the surface, it means that the particle is moving in the tangent plane to the surface at that point.

When a particle is on the surface, the constant $A(\vec{\mu})$ can be related to the curvature of the surface. It can be shown[4] that the radius of curvature at the point $\vec{p}$ on the surface in the plane containing the normal to the surface there, $\nabla f(\vec{p})$ and the direction of the particle on the surface, $\vec{\mu}$, is given by $|\nabla f(\vec{p})|/|A(\vec{\mu})|$. There is one case among the surfaces we consider where both $|\nabla f(\vec{p})|$ and $|A(\vec{\mu})|$ vanish simultaneously and that is of a point on the vertex of a cone. In this anomalous case we can take the radius of curvature to be zero.

$A(\vec{\mu})$ vanishes when the particle is travelling parallel to a "ruled line" of the surface, whether on the surface or not. A ruled line is a line that lies entirely on the surface. Quadrics with one or more vanishing quadratic constants (one of the a_i's in Equation 10.5 or 10.6) always possess ruled lines, as do planes, cones, hyperboloids of one sheet and hyperbolic paraboloids. The constant $A(\vec{\mu})$ can also vanish for a particle having a trajectory that is parallel to an asymptote of a hyperboloid, or pointed at the "nose" of a paraboloid or in the plane perpendicular to it.

$A(\vec{\mu})$ can be used to decide where a particle is in relation to a surface in the case that $B(\vec{\mu}, \vec{p})$ and $C(\vec{p})$ vanish, that is when the particle is on the surface and in the plane tangent to it at that point. In this case an infinitesimal transport $C(\vec{p} + \vec{\mu}\epsilon) = A(\vec{\mu})\epsilon^2$ will have the sign of

[4]The way to do this is consider a particle at point $\vec{p}$ on the surface with an initial direction $\vec{\mu}$ tangent to the surface and moving in the plane defined by the normal orthogonal vectors $\vec{\mu}$ and $\nabla f(\vec{p})/|\nabla f(\vec{p})|$. The trajectory of the particle is then described by $f(\vec{p} + \vec{\mu}s_u + (\nabla f(\vec{p})/|\nabla f(\vec{p})|)s_n) = 0$ where s_u and s_n are projections of the particle's position vector on the $\vec{\mu}$ and $\nabla f(\vec{p})/|\nabla f(\vec{p})|$ axes, respectively. This yields the equation of a conic. The radius of curvature is then obtained by the standard equation for motion in a plane, $R_{\rm c} = \{[1 + ({\rm d}s_n/{\rm d}s_u)^2]^{3/2}\}/|{\rm d}^2 s_n/{\rm d}s_u^2|$.

$A(\vec{\mu})$. So, if ($A(\vec{\mu}) > 0$, $B(\vec{\mu}, \vec{p}) = 0$, $C(\vec{p}) = 0$) the particle is headed outside, if ($A(\vec{\mu}) < 0$, $B(\vec{\mu}, \vec{p}) = 0$, $C(\vec{p}) = 0$) the particle is headed inside, and if ($A(\vec{\mu}) = 0$, $B(\vec{\mu}, \vec{p}) = 0$, $C(\vec{p}) = 0$) the particle is on the surface and directed along a ruling and there is no intercept in this case.

For planar surfaces ($A(\vec{\mu}) = 0$ always in this case) there is always a solution for s unless the particle's trajectory is exactly parallel to the plane. If the solution for s is negative, it is rejected since it not a forward solution. If the solution for s is positive, then it represents a solution along the forward trajectory of the particle.

For the non-planar surfaces, the equation for s is quadratic. In general, when $B(\vec{\mu}, \vec{p})^2 - A(\vec{\mu})C(\vec{p}) < 0$, there are no solutions to the quadratic equation, which means that the particle's trajectory misses the surface. If the surface in question is one of the seven with the intuitive inside-outside interpretation, then one might guess that one does not have to test for the positiveness of $B(\vec{\mu}, \vec{p})^2 - A(\vec{\mu})C(\vec{p})$ when the particle is inside, thereby saving computer time. However, there are conditions where the limited numerical precision causes $B(\vec{\mu}, \vec{p})^2 - A(\vec{\mu})C(\vec{p})$ to be negative even when the particle is inside one of the natural surfaces. In this case it can be shown that the particle's trajectory is very close to that of being along a ruling and very close to the surface ($A(\vec{\mu}) \approx 0$, $B(\vec{\mu}, \vec{p}) \approx 0$, $C(\vec{p}) \approx 0$). It is consistent, therefore, to assume that there is no solution in this case even if the trajectory is not exactly on the surface or exactly tangent to it. In this case the particle is assumed to travel along the ruling until it hits another surface in the problem or an interaction effects a change of direction whereupon a decision can be made whether the particle is headed inside or outside the surface.

Employing the error recovery strategy of the previous section, a general algorithm for an arbitrary quadric surface may be sketched:

```
IF $\underline{B^2 - AC < 0}$ Particle does not intersect the surface.
ELSEIF The particle thinks it is outside
    IF $\underline{B \geq 0}$
        IF $\underline{A \geq 0}$ No solution.
        ELSE $s = -(B + \sqrt{B^2 - AC})/A$.
    ELSE $s = \max(0, C/[\sqrt{B^2 - AC} - B])$.
ELSE The particle thinks it is inside
    IF $\underline{B \leq 0}$
        IF $\underline{A > 0}$ $s = (\sqrt{B^2 - AC} - B)/A$.
        ELSE No solution.
    ELSE $s = \max(0, -C/[\sqrt{B^2 - AC} + B])$.
```

All quadric surfaces are special cases that can be solved by this algorithm. Only the constants A, B, C need to be specified for any case. Indeed, this algorithm will work for planes as well but the simplicity of planes motivates the construction of a more efficient algorithm specific to planes only.

The FORTRAN codes which solves the general solution of distance to intercept of a quadric surface is:

```
C23456789|123456789|123456789|123456789|123456789|123456789|123456789|12

      subroutine quadric(A,B,C,s,inside,hit)
      implicit none

C  Calculates the first positive distance to an arbitrary quadric surface

      logical
     *    inside ! Input:  inside = .true.  => particle thinks it is inside
     *    ,      ! Input:  inside = .false. => particle thinks it is outside
     *    hit    ! Output: hit    = .true.  => particle would hit  the surface
                 ! Output: hit    = .false. => particle would miss the surface

      real
     *     A,    ! Input: Quadratic coefficient A
     *     B,    ! Input: Quadratic coefficient B
     *     C,    ! Input: Quadratic coefficient C
     *     s     ! Output: Distance to the surface (if hit)

      real
     *     Q     ! Internal: quadratic coefficient


      Q = B**2 - A*C

      if (Q .lt. 0e0) then
          hit = .false.
          return
      end if

      if (inside) then              !inside the surface
          if (B .le. 0e0) then      !headed away from surface
              if (A .gt. 0e0) then  !but, surface curving up
                  hit = .true.      !always a hit in this case
                  s = (sqrt(Q) - B)/A
```

```
                return
            else                    !surface curving away and headed in
                hit = .false.       !never a hit in this case
                return
            end if
        else                        !headed toward the surface
            hit = .true.            !always a hit in this case
            s = max(0e0, -C/(sqrt(Q) + B))
            return
        end if
    end if

    !Must be outside the surface

    if (B .ge. 0e0) then      !headed away
        if (A .ge. 0e0) then !surface curves away
            hit = .false.    !never a hit in this case
            return
        else                 !surface curves up
            hit = .true.     !always a hit in this case
            s = -(sqrt(Q) + B)/A
            return
        end if
    else                     !headed toward the surface
        hit = .true.         !always a hit in this case
        s = max(0e0, C/(sqrt(Q) - B))
        return
    end if

    end
```

Given this algorithm, the only thing one needs to do for any quadric surface is to specify the constants A, B, C and present them to the quadratic solver. The following sections describe some examples of this.

10.3.2 Spheres

The general equation for a sphere is:

$$(\vec{x} - \vec{X})^2 - R^2 = (x - X)^2 + (y - Y)^2 + (z - Z)^2 - R^2 = 0 \qquad (10.9)$$

where $\vec{X} \equiv (X, Y, Z)$ is the location of the center of the sphere and R is its radius.

Intercept distance

Substituting the equation for particle trajectory, Equation 10.1, into the above yields a quadratic equation of the form $As^2 + 2Bs + C = 0$, where the quadratic constants, A, B and C, are:

$$\begin{aligned} A &= 1 \\ B &= \vec{\mu} \cdot (\vec{x}_0 - \vec{X}) \\ &= u(x_0 - X) + v(y_0 - Y) + w(z_0 - Z) \\ C &= (\vec{x}_0 - \vec{X})^2 - R^2 \\ &= (x_0 - X)^2 + (y_0 - Y)^2 + (z_0 - Z)^2 - R^2 \end{aligned} \tag{10.10}$$

These constants may be employed in the general quadric surface forward-distance solver algorithm described previously.

The FORTRAN code that calculates the quadratic constants for the case when the sphere is centered at the origin is:

```
C23456789|123456789|123456789|123456789|123456789|123456789|123456789|12

      subroutine csphere(R,x0,y0,z0,u,v,w,s,inside,hit)
      implicit none

C  Calculates the distance to a sphere centered at (0,0,0)

      logical
     *    inside ! Input:  inside = .true.  => particle thinks it is inside
     *    ,      ! Input:  inside = .false. => particle thinks it is outside
     *    hit    ! Output: hit    = .true.  => particle would hit  the surface
                 ! Output: hit    = .false. => particle would miss the surface

      real
     *    R,     ! Input:  Radius of the cylinder
     *    x0,    ! Input:  x-coordinate of the particle
     *    y0,    ! Input:  y-coordinate of the particle
     *    z0,    ! Input:  z-coordinate of the particle
     *    u ,    ! Input:  x-axis direction cosine of the particle
     *    v ,    ! Input:  y-axis direction cosine of the particle
     *    w ,    ! Input:  w-axis direction cosine of the particle
     *    s      ! Output: Distance to the surface (if hit)

      real
```

```
*     A,    ! Internal: Quadratic coefficient A
*     B,    ! Internal: Quadratic coefficient B
*     C     ! Internal: Quadratic coefficient C

      A = 1e0    ! i.e. u**2 + v**2 + w**2 = 1
      B = u*x0  + v*y0  + w*z0
      C = x0**2 + y0**2 + z0**2 - R**2

      call quadric(A,B,C,s,inside,hit) ! Get the generic quadric solution

      return
      end
```

10.3.3 Circular Cylinders

The general equation for a circular cylinder is:

$$(\vec{x}-\vec{P})^2 - [(\vec{x}-\vec{P})\cdot\vec{U}]^2 - R^2 = 0 \tag{10.11}$$

where $\vec{P} \equiv (P_x, P_y, P_z)$ is any fixed point on the axis of the cylinder, $\vec{U}$ is the direction vector of the axis of the cylinder, and R is its radius.

Intercept distance

Substituting the equation for particle trajectory, Equation 10.1, into the above yields a quadratic equation of the form $As^2+2Bs+C=0$, where the quadratic constants, A, B and C, are:

$$\begin{aligned} A &= 1-(\vec{\mu}\cdot\vec{U})^2 \\ B &= \vec{\mu}\cdot\{(\vec{p}-\vec{P})-\vec{U}[(\vec{p}-\vec{P})\cdot\vec{U}]\} \\ C &= (\vec{p}-\vec{P})^2-[(\vec{p}-\vec{P})\cdot\vec{U}]^2-R^2 \end{aligned} \tag{10.12}$$

These constants may be employed in the general quadric surface forward-distance solver algorithm described previously.

The FORTRAN code that calculates the quadratic constants for the case when the cylinder is cylindrical, aligned and centered on the z axis is:

```
C23456789|123456789|123456789|123456789|123456789|123456789|123456789|12
```

```
      subroutine ccylz(R,x0,y0,u,v,s,inside,hit)
      implicit none

C  Calculates the distance to a circular cylinder centered and aligned along
C  the z-axis

      logical
     *    inside ! Input:  inside = .true.  => particle thinks it is inside
     *    ,      ! Input:  inside = .false. => particle thinks it is outside
     *    hit    ! Output: hit    = .true.  => particle would hit  the surface
                 ! Output: hit    = .false. => particle would miss the surface

      real
     *    R,     ! Input:  Radius of the cylinder
     *    x0,    ! Input:  x-coordinate of the particle
     *    y0,    ! Input:  y-coordinate of the particle
     *    u ,    ! Input:  x-axis direction cosine of the particle
     *    v ,    ! Input:  y-axis direction cosine of the particle
     *    s      ! Output: Distance to the surface (if hit)

      real
     *     A,     ! Internal: Quadratic coefficient A
     *     B,     ! Internal: Quadratic coefficient B
     *     C      ! Internal: Quadratic coefficient C

      A = u**2  + v**2
      B = u*x0  + v*y0
      C = x0**2 + y0**2 - R**2

      call quadric(A,B,C,s,inside,hit) ! Get the generic quadric solution

      return
      end
```

10.3.4 Circular Cones

In standard quadric form, the general equation for a cone is:

$$\cos^2\Theta\{(\vec{x}-\vec{P})-\vec{U}[(\vec{x}-\vec{P})\cdot\vec{U}]\}^2-\sin^2\Theta[(\vec{x}-\vec{P})\cdot\vec{U}]^2=0 \tag{10.13}$$

where $\vec{P} \equiv (P_x, P_y, P_z)$ is the vertex point of the cone, $\vec{U}$ is the direction vector of the symmetry axis of the cone and Θ is its opening angle. This form, depicted in Figure 10.2, is actually two cones on the same axis situated point-to-point. To avoid ambiguities, we adopt the convention that $0 < \Theta < \pi/2$ and use $\vec{U}$ to orient the cone. (The special case, $\Theta = \pi/2$, corresponds to the quadric surface for coincident planes, while the special case, $\Theta = 0$ corresponds to a zero-radius cylinder.) Both cones are to be regarded as valid surfaces for which the intercept distance is to be calculated. If an application requires only one cone, then it will be assumed that the other "reflection" cone has been eliminated through the use of another surface that isolates only one of the cones.

Intercept distance

Substituting the equation for particle trajectory, eq. 10.1, into the above yields a quadratic equation of the form $As^2 + 2Bs + C = 0$, where the quadratic constants, A, B and C, are:

$$\begin{aligned} A &= \cos^2\Theta[\vec{\mu} - \vec{U}(\vec{\mu}\cdot\vec{U})]^2 - \sin^2\Theta(\vec{\mu}\cdot\vec{U})^2 \\ B &= \cos^2\Theta\vec{\mu}\cdot\{(\vec{p}-\vec{P}) - \vec{U}[(\vec{p}-\vec{P})\cdot\vec{U}] - \sin^2\Theta\vec{U}[(\vec{p}-\vec{P})\cdot\vec{U}]\} \\ C &= \cos^2\Theta\{(\vec{p}-\vec{P}) - \vec{U}[(\vec{p}-\vec{P})\cdot\vec{U}]\}^2 - \sin^2\Theta[(\vec{p}-\vec{P})\cdot\vec{U}]^2 \end{aligned} \tag{10.14}$$

These constants may be employed in the general quadric surface forward-distance solver algorithm described previously.

10.4 Using surfaces to make objects

Now that we know everything about surfaces, it is time to put them together to make objects. That is, we want to develop techniques to delineate regions of space and use them to define the geometrical elements that constitute some object in a Monte Carlo application.

10.4.1 Elemental volumes

We define first the concept of an *elemental* volume, a region of space that can be specified *uniquely* by a set of logical conditions related to being inside or outside the constituent surfaces.

Here are some examples:

A single plane $z = 0$

The gradient of $f(z) = z = 0$ is $\nabla f(z) = (\partial z/\partial z)\hat{z} = \hat{z}$. So, the normal to this surface points along the positive z axis. Therefore, using our definition, all points in space with

$z < 0$ are "inside" and all points in space with $z > 0$ are "outside". (One immediately sees how arbitrary this definition is!).

What if you were asked to locate the point $(x_0, y_0, 0)$? Based on its position, this point is neither inside nor outside. To answer this question you would require more information, the particle direction, $\vec{\mu} = (u, v, w)$. You would base your decision on where the particle is going. If the particle is on the surface, form the product $\vec{\mu} \cdot \hat{n}$ where n is the normal to the plane. In this case $\vec{\mu} \cdot \hat{n} = w$. If $\vec{\mu} \cdot \hat{n} > 0$, it means that the particle is directed outside. If $\vec{\mu} \cdot \hat{n} < 0$, it means that the particle is directed inside. This would place the particle based on where it is going. What if $z = 0$ and $\vec{\mu} \cdot \hat{n} = 0$, or $w = 0$ in our case? This means that the particle is on the plane and has a trajectory that is parallel to the plane. In this case, the choice is arbitrary. Unless there is a special source of particles that specifically chooses this set of conditions (and would specify the logical location of the particle), the probability that a particle will transport and scatter into this condition is quite small. In this case, choose either "inside" or "outside". Eventually the particle will scatter either inside or outside the plane and its position will be resolved at that point.

A single sphere of radius R centered at $(0, 0, 0)$

The equation of this surface is:

$$\vec{x}^2 - R^2 = x^2 + y^2 + z^2 - R^2 = 0 \ . \qquad (10.15)$$

Given a particle at position $\vec{x}_0 = (x_0, y_0, z_0)$ and direction $\vec{\mu} = (u, v, w)$, the quadratic constants, A, B and C, are:

$$\begin{aligned} A &= 1 \\ B &= \vec{\mu} \cdot \vec{x}_0 \\ &= ux_0 + vy_0 + wz_0 \\ C &= \vec{x}_0^2 - R^2 \\ &= x_0^2 + y_0^2 + z_0^2 - R^2 \end{aligned} \qquad (10.16)$$

The sphere delineates two elemental volumes, the interior of the sphere and the exterior of the sphere as shown in Figure 10.4.

If you were asked to locate the particle, first you would look at $C = x_0^2 + y_0^2 + z_0^2 - R^2$. If $C < 0$ the particle is inside the sphere and if the $C > 0$ the particle is outside the sphere. (In this example, the choice of "inside" and "outside" seems a little more natural.) If $C = 0$ the location of the particle is on the sphere and we would then look at the constant B. If $C = 0$ and $B < 0$, the particle is headed inside. If $C = 0$ and $B > 0$, the particle is headed outside. If $C = 0$ and $B = 0$ the particle is on the surface and has a trajectory that is tangent to the sphere. In this case we would appeal to the constant A. For a sphere is always has the same value, 1. It is positive, which means the surface is curving away from the particle trajectory and the particle is headed outside.

A single circular cylinder of radius R centered and directed along the z axis

The equation of this surface is:

$$x^2 + y^2 - R^2 = 0 \ . \tag{10.17}$$

Given a particle at position $\vec{x}_0 = (x_0, y_0, z_0)$ and direction $\vec{\mu} = (u, v, w)$, the quadratic constants, A, B and C, are:

$$\begin{aligned} A &= u^2 + v^2 \\ B &= ux_0 + vy_0 \\ C &= x_0^2 + y_0^2 - R^2 \end{aligned} \tag{10.18}$$

The cylinder delineates two elemental volumes, the interior of the cylinder and the exterior of the cylinder as shown in Figure 10.5.

If you were asked to locate the particle, first you would look at $C = x_0^2 + y_0^2 - R^2$. If $C < 0$ the particle is inside the cylinder and if the $C > 0$ the particle is outside the cylinder. (In this example, the choice of “inside” and “outside” also seems a little more natural.) If $C = 0$ the location of the particle is on the cylinder and we would then look at the constant B. If $C = 0$ and $B < 0$, the particle is headed inside. If $C = 0$ and $B > 0$, the particle is headed outside. If $C = 0$ and $B = 0$ the particle is on the surface and has a trajectory that is tangent to the cylinder. in this case we would appeal to the constant A. For a cylinder this can be zero if the particle's direction is identical to the axis of the cylinder. In this case, the choice is again arbitrary. Unless there is a special source of particles that specifically chooses this set of conditions (and would specify the logical location of the particle), the probability that a particle will transport and scatter into this condition is quite small. In this case, choose either “inside” or “outside”. Eventually the particle will scatter either inside or outside the cylinder and its position will be resolved at that point.

An elemental volume made up of several surfaces

For this example, let us consider the planes $P_1 : z = 0$, $P_2 : z = 1$ and the cylinder $C_1 : x^2 + y^2 - R^2 = 0$. The geometry is depicted in Figure 10.6.

The first thing to notice (if you have not already) is that planes and cylinders are infinite and all of space is “carved up” by a set of surfaces. In fact this arrangement delineates 6 elemental volumes! With the notation S_n means outside of surface S_n, we can delineate the 6 elemental volumes as:

Region	Location	Description
1	$\overline{P}_1 \cap C_1 \cap P_2$	outside P_1, inside C_1, inside P_2
2	$P_1 \cap C_1$	inside P_1, inside C_1
3	$C_1 \cap \overline{P}_2$	inside C_1, outside P_2
4	$P_1 \cap \overline{C}_1$	inside P_1, outside C_1
5	$\overline{P}_1 \cap \overline{C}_1 \cap P_2$	outside P_1, outside C_1, inside P_2
6	$\overline{C}_1 \cap \overline{P}_2$	outside C_1, outside P_2

The location of a particle within one of these elemental volumes can be determined uniquely by finding which of these conditions is satisfied.

General considerations for elemental volumes

The first thing to notice that each elemental volume defines a *unique region of space* that can be specified by a *unique set of logical conditions.* Tracking particles through elemental volumes is extremely fast. First of all, the positioning errors mentioned earlier in this chapter cause no problem and are absorbed on subsequent transport steps. Correction schemes never have to be invoked. Particles that become "lost", that is, lose or never acquire a sense of which elemental volume they are in, can be found using the location techniques just described.[5] Finally, if a particle leaves an elemental volume by one of its surfaces and this surface is employed by another elemental volume, its next elemental volume is known by flipping the logical switch that orients the particle with respect to that surface.

10.5 Tracking in an elemental volume

We now consider the case where we are tracking a particle within an elemental volume. There are some nice consequences of tracking within elemental volumes. The first is that any given surface defining the elemental volume can only be intercepted once. Otherwise the "insideness" or the "outsidesness" would not be unique. The other consequence is that the intercept is defined as the shortest intercept to any surface bounding the elemental volume without consideration of location of the intercept. Recall that a surface extends to the exterior of any elemental volume because surfaces are infinite (except for ellipsoids) and volumes are (usually) finite. The consequence of this is that the shortest intercept with any surface is guaranteed to be in the elemental surface volume.

This is best demonstrated by example. Consider the interior of the right circular cylinder

[5]If each surface of each elemental volume joins uniquely onto other elemental volume, it can never become "lost" through tracking. Source particles may be "lost" before tracking starts and will have to acquire its location in an elemental volume by searching. Sometimes the surface of an elemental volumes join onto two or more other elemental volumes. A search algorithm may have to be initiated at this point to locate the particle uniquely in an elemental volume. Another difficulty occurs if "voids" are created by poorly constructed geometry code. Good coding defines all of space uniquely.

defined by the planes $P_1 : z = 0$, $P_2 : z = 1$ and the cylinder $C_1 : x^2 + y^2 - R^2 = 0$. The geometry is depicted in Figure 10.7 and some representative trajectories are given there.

Consider trajectory 1. It has intercepts with both C_1 and P_2. However, the distance to the intercept with C_1 is shorter and is the answer in this case. Contrast this with trajectory 2. It has intercepts with both C_1 and P_2. However, the distance to the intercept with P_2 is shorter and is the answer in this case.

Recall that the interior of the right circular cylinder is represented by the condition $\overline{P}_1 \cap C_1 \cap P_2$. Therefore, the distance to exit the elemental volume is given by the following coding:

```
TransportDistance = infinity ! (Some very large number)

! Check the distance to each bounding surface

! Check the leftmost plane
call zplane(0e0,z0,w,s,.false.,hit) ! 0e0 = zplane position, .false. => outside
if (hit .and. (s .le. TransportDistance)) TransportDistance = s

! Check the rightmost plane
call zplane(1e0,z0,w,s,.true.,hit) ! 1e0 = zplane position, .true. => inside
if (hit .and. (s .le. TransportDistance)) TransportDistance = s

! Check the cylinder
call ccylz(1e0,x0,y0,u,v,s,.true.,hit) ! 1e0 = radius, .true. => inside
if (hit .and. (s .le. TransportDistance)) TransportDistance = s
```

After this code segment is executed, the variable `TransportDistance` is the distance to exit the elemental volume by any surface irrespective of its direction, as long as it is logically placed within this elemental volume.

The equivalent coding for a general elemental volume would be:

```
TransportDistance = infinity ! (Some very large number)

! Check the distance to each bounding surface

! {...} are the parameters that define the particle's position and direction
! [...] are the parameters that define the surface
! [.true.|.false.] true or false depending upon orientation of surface
```

```
! hit is always returned .true. or .false.
! If hit is returned as .true., s represents the distance to that surface

call Surface_1({...},[...],s,[.true.|.false.],hit)
if (hit .and. (s .le. TransportDistance)) TransportDistance = s

call Surface_2({...},[...],s,[.true.|.false.],hit)
if (hit .and. (s .le. TransportDistance)) TransportDistance = s

call Surface_3({...},[...],s,[.true.|.false.],hit)
if (hit .and. (s .le. TransportDistance)) TransportDistance = s

.
.
.

! Until all the bounding surfaces are exhausted
```

After this code segment is executed, the variable `TransportDistance` is the distance to exit the elemental volume by any surface irrespective of its direction, as long as it is logically placed within this elemental volume.

10.6 Using elemental volumes to make objects

10.6.1 Simply-connected elements

A simply-connected object is one in which one element connects to only one other element through a common surface. It is best to explain this by means of an example.

Consider the simple problem of creating the object depicted in Figure 10.8. This object is made up of the planes $P_1 : z = 0$, $P_2 : z = 1$ and the cylinder $C_1 : x^2 + y^2 - R^2 = 0$ and the sphere $S_1 : x^2 + y^2 + z^2 - R^2 = 0$. We will define three regions:

Region	Location	Description
1	$\overline{P}_1 \cap C_1 \cap P_2$	outside P_1, inside C_1, inside P_2
2	$P_1 \cap S_1$	inside P_1, inside S_1
0	All the rest	This is the “outside” of the object

We will adopt the convention that region “0” is the exterior of the object. By exterior we mean that if a particle is tracked from the interior to the exterior it is considered to “vanish” from the simulation.

Note that the way to define this object is not unique! Equivalently we may define it as follows:

Region	Location	Description
1	$\overline{S}_1 \cap C_1 \cap P_2$	outside S_1, inside C_1, inside P_2
2	S_1	inside S_1
0	All the rest	This is the "outside" of the object

However, we will employ the first definition in our example. We note that we do require <u>two</u> elemental volumes to define this object. The reason for this is that the interior space of the object is made up of regions that are both "inside" and "outside" the sphere. The division into two elemental surfaces can be done in a number of ways.

The logic to handle tracking in the object is to realize that if the particle is inside region 1 and tracked to either C_1 or P_2, it escapes on the next step. If the particle is inside region 2 and tracked to either S_1 or P_1, it escapes on the next step. If the particle is inside region 1 and tracked to P_1 it enters region 2 on the next step. Conversely, if the particle is inside region 2 and tracked to P_1 it enters region 1 on the next step.

We will now write the "pseudo-code" for this geometry. However, we must now extend the particle phase-space concept introduced in Chapter 7. In that chapter, in Equation 7.1, a particle's phase-space was defined as:

$$\{\vec{x}, \vec{u}\} , \tag{10.19}$$

where $\vec{x}$ is the particle's absolute location in space referred back to the origin of some laboratory coordinate system, and $\vec{u}$ is its direction referred back to a fixed set of axes in the same laboratory coordinate system. The phase-space is now expanded to:

$$\{\vec{x}, \vec{u}, N_{\text{elem}}\} , \tag{10.20}$$

where N_{elem} is the elemental volume number associated with its current position.

The pseudo-code for this geometry is:

```
IF $N_{\text{elem}} = 0$, stop transport, particle vanishes.

ELSEIF $N_{\text{elem}} = 1$

    IF particle hits $P_1$, $N_{\text{elem}} = 2$, next region is region 2
    ELSEIF particle hits $P_2$, $N_{\text{elem}} = 0$, next region is the outside
    ELSEIF particle hits $C_1$, $N_{\text{elem}} = 0$, next region is the outside
    ELSE No surface is hit, continue transport

ELSEIF $N_{\text{elem}} = 2$
```

IF particle hits P_1, $N_{\text{elem}} = 1$, next region is region 1

ELSEIF particle hits S_1, $N_{\text{elem}} = 0$, next region is the outside

ELSE No surface is hit, continue transport

Now, let us write the Fortran code for this application:

```
.
.
.

! Determine the distance to a scattering point
TransportDistance = DistanceToScatteringPoint()

NextElement = Nelem ! Assume particle stays in the same region

if (Nelem .eq. 0) then
    ! Particle escapes
    ! Special coding or a logic transfer point must be specified to
    ! give us some particle phase-space to work with

elseif (Nelem eq.1) then

    ! In the first region

    ! Check the distance to each bounding surface of region 1

    ! Check plane 1 at z = 0
    call zplane(0e0,z0,w,s,.false.,hit) ! .false. => outside
    if (hit .and. (s .le. TransportDistance)) then
        TransportDistance = s
        NextElement = 2
    end if

    ! Check plane 2 at z = 1
    call zplane(1e0,z0,w,s,.true.,hit) ! .true. => inside
    if (hit .and. (s .le. TransportDistance)) then
        TransportDistance = s
        NextElement = 0
    end if
    ! Check the cylinder
    call ccylz(1e0,x0,y0,u,v,s,.true.,hit) ! 1e0 = radius, .true. => inside
    if (hit .and. (s .le. TransportDistance))
        TransportDistance = s
        NextElement = 0
    end if

elseif (Nelem .eq. 2) then

    ! In the second region

    ! Check the distance to each bounding surface of region 2

    ! Check plane 1 at z = 0
    call zplane(0e0,z0,w,s,.false.,hit) ! .true. => inside
    if (hit .and. (s .le. TransportDistance)) then
        TransportDistance = s
        NextElement = 1
    end if

    ! Check the sphere
    call csphere(1e0,x0,y0,z0,u,v,w,s,.true.,hit) ! 1e0 = radius, .true. => inside
    if (hit .and. (s .le. TransportDistance))
        TransportDistance = s
        NextElement = 0
    end if

endif

! Transport the particle

x0 = x0 + u*s
y0 = y0 + v*s
z0 = z0 + w*s

! Change region numbers

Nelem = NextRegion

.
.
.
```

Note that if the surface is hit, it must also shorten the transport distance in order to serve as a candidate for the surface through which a particle leaves the elemental volume.

The generalization to more complex objects is clear. With the knowledge of which elemental volume the particle is in cycle over all the bounding surfaces of the element. If the particle strikes this surface along its flight path and if this distance shortens the proposed transport distance, then the next element can be specified. It can also be superseded by a shorter intersection to another surface. *The shortest one is the one that matters!*

10.6.2 Multiply-connected elements

A multiply-connected object is one in which one element can connect to more than one element through one or more common surface. It is best to explain this by means of an example as well.

Consider the problem of creating the object depicted in Figure 10.9. This object is made up of two planes $P_1 : x = 0$, $P_2 : y = 0$. We will define three regions:

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

Regions 2 and 3 are simply connected to each other an region 1, but region 1 is multiply-connected to regions 2 and 3 through the common surface P_1.

In objects that are not too complex one can do a fake transport of the particle and see in which region its terminal position resides in. In this case we would test to see whether `y0 + v*s` os greater or less than zero to make a decision. Each case requires special coding and techniques to resolve. If one can turn a multiply-connected object into an simply-connected one, it often generates faster code at the expense of introducing more regions into the problem.

A general strategy for resolving this problem would take us beyond the scope of this course. A basic solution that is applied in some applications is first to do a search for position in all the candidate elements. If the search comes up empty, it means that the particle did not escape its current region by virtue of an undershoot that that surface. One can reset the region back to the original and try again. If the subsequent transport still does not resolve the problem the solution is to provide small boosts to the transport step starting with the smallest resolvable floating-point number and allow these boosts to grow geometrically until the surface is crosses, at least numerically.

It is ugly but it works and is fast. Unlike the simply-connected surfaces where some elegant logic solves the problem, the logic for multiply-connected surfaces gets quite involved and is usually not worth the effort in terms of code efficiency.

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

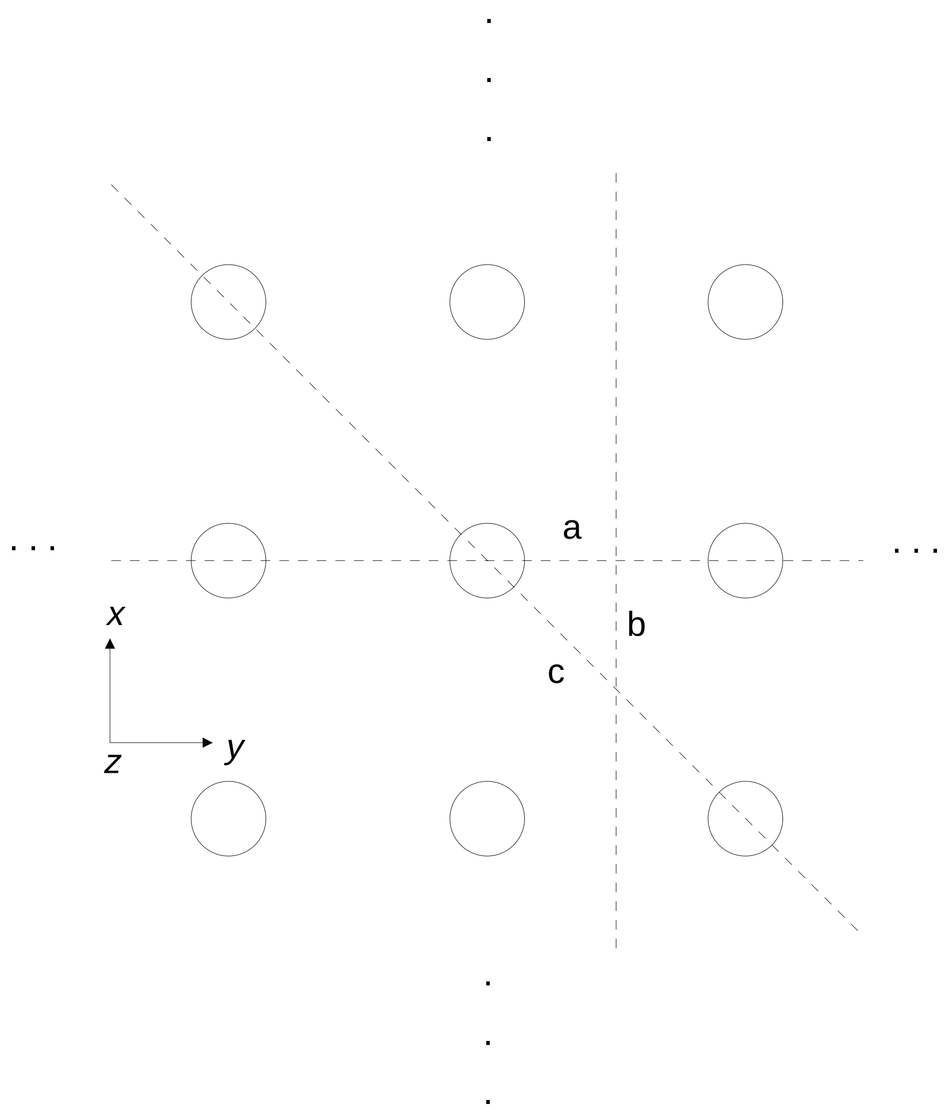

Figure 10.10: Top end view of an infinite square lattice of cylinders. Three planes of symmetry are drawn, **a**, **b**, and **c**. A complete simulation of the entire lattice may be performed by restricting the transport to the interior of the three planes. When a particle strikes a plane it is reflected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

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 to $\hat{n}$, and $\vec{u}_{\perp}$, perpendicular to $\vec{n}$. The parallel part gets reflected, $\vec{u}'_{\|} = -\vec{u}_{\|}$, and the perpendicular part remains unchanged, $\vec{u}'_{\perp} = \vec{u}_{\perp}$. That is, the new direction vector is $\vec{u}' = -\vec{u}_{\|} + \vec{u}_{\perp}$. 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.

Bibliography

[Olm47] J. M. H. Olmsted. Solid Analytic Geometry. *(Appleton-Century-Crofts Inc, New York)*, 1947.

Problems

1. Consider a right circular cylinder of radius R and length L where R and L are in units of, say, cm's. The interior of this object is defined by the planes $z = 0$, $z = L$ and the cylinder $x^2 + y^2 = R^2$.

   A beam of <u>at least</u> 10^4 particles is incident on the middle of the flat end of the geometry at $\vec{x} = 0$ with unit direction vector $\vec{u} = \hat{z}$. These particles exhibit discrete interactions with an attenuation factor of $\Sigma = 1$ cm^{-1}. Thus, the interaction probability distribution function at location z is $p(z)\mathrm{d}z = \Sigma \exp(-\Sigma z)\mathrm{d}z$. These particles scatter isotropically once they reach the scattering point.

   Write a Monte Carlo code to simulate this problem and answer the following questions:

   (a) The total pathlength/history is the sum of the distance from start to interaction (if it occurs) <u>plus</u> the drift distance to exit the geometry. For $R = 0$ and $L = \infty$, what is the average total pathlength/history and its associated estimated error, $\bar{t} \pm s_{\bar{t}}$?

   (b) For $L = 10$ cm, what is the average pathlength and its associated estimated error, $\bar{t} \pm s_{\bar{t}}$ when R is 0.1,0.2,0.5,1,2,5,10,20,50,100,200,500,1000 cm's? Plot the results and discuss the small-R and large-R limits.

   (c) For $R = 10$ cm, what is the average pathlength and its associated estimated error, $\bar{t} \pm s_{\bar{t}}$ when L is 0.1,0.2,0.5,1,2,5,10,20,50,100,200,500,1000 cm's? Plot the results and discuss the small-L and large-L limits.

2. Consider a right circular cylinder of radius R and length L. The interior of this object is defined by the planes $z = 0$, $z = L$ and the cylinder $x^2 + y^2 = R^2$. For this exercise, $R = L = 1$ cm. A beam of <u>at least</u> 10^4 particles is incident on the middle of the flat end of the geometry at $\vec{x} = 0$ with unit direction vector $\vec{u} = \hat{z}$. These particles

exhibit discrete interactions with an attenuation factor Σ which will be given later in units of cm^{-1}'s. Thus, the interaction probability distribution function at location z is $p(z)\ \mathrm{d}z = \Sigma \exp(-\Sigma z)\ \mathrm{d}z$. At the interaction point, the particle scatters into a polar, Θ, and an azimuthal, Φ, angle with probability distribution:

$$p(\Theta, \Phi)\,\mathrm{d}\Theta\ \mathrm{d}\Phi = \frac{\eta(2+\eta)}{4\pi} \frac{\sin\Theta\ \mathrm{d}\Theta\ \mathrm{d}\Phi}{(1-\cos\Theta+\eta)^2}\ ,$$

where η is a dimensionless constant that describes forward scattering for $\eta \longrightarrow 0$ and isotropic scattering for $\eta \longrightarrow \infty$. After the first interaction, second and subsequent interactions can occur with the same interaction probability until the particle escapes from the geometry, possibly having scattered repeatedly.

Write a Monte Carlo code to simulate this problem and answer the following associated questions:

(a) i. Show that in the limit, $\eta \longrightarrow 0$, $p(\Theta)\sin\Theta\ \mathrm{d}\Theta\ \mathrm{d}\Phi \longrightarrow \delta(1-\cos\Theta)\sin\Theta\ \mathrm{d}\Theta\ \mathrm{d}\Phi/2\pi$.
   ii. Show that in the limit, $\eta \longrightarrow \infty$, $p(\Theta)\sin\Theta\ \mathrm{d}\Theta\ \mathrm{d}\Phi \longrightarrow \sin\Theta\ \mathrm{d}\Theta\ \mathrm{d}\Phi/4\pi$.

(b) For the set of Σ's, $\Sigma = 0.1,\ 1,\ 10$ cm^{-1}'s and both forward ($\eta \longrightarrow 0$) and isotropic ($\eta \longrightarrow \infty$) scattering, calculate the backscatter and transmission coefficients (number backscattered or transmitted divided by the number of histories), and the associated estimated statistical uncertainty.

(c) For $\Sigma = 10$ cm^{-1}, do simulations for a set of η's with enough points such that both asymptotic limits are nearly reached and the η behavior of the backscatter and transmission coefficients for intermediate η are fully described.

(d) Repeat 2(b) but with the back plane of the geometry being a reflective surface.

(e) Repeat 2(b) but with both the back plane and the cylindrical surface of the geometry reflective but with absorption included in the following way—at each interaction point, there is a 0.8 probability that the particle will scatter in the way previously described and a 0.2 probability that it will be absorbed.

(f) Repeat 2(b) but with grazing incidence of the starting point, *i.e.* the particles are incident on the middle of the flat end of the geometry at $\vec{x} = 0$ with unit direction vector $\vec{u} = \hat{y}$.

3. A cylindrical beam of particles with the following starting characteristics:

$$\psi(\vec{x}, \vec{u}) = \frac{2}{R_{\mathrm{beam}}^2}\Theta(R_{\mathrm{beam}}^2 - x^2 - y^2)\delta(z)\delta(\hat{z} - \vec{u})$$

is incident on a geometry defined by the plane, $z = 0$ and the cylinder $x^2 + y^2 = R_{\mathrm{cyl}}^2$. Once inside the target (*i.e.* $z \geq 0$), the particle scatters isotropically with elastic scattering cross section, $\Sigma_s = 1$ and absorption cross section $\Sigma_a = 0.25$. Outside the target (*i.e.* $z < 0$), $\Sigma_s = \Sigma_a = 0$. *Note to take care that you inform the particles whether they are inside the cylinder or outside.*

(a) Let $R_{\rm cyl} = 1$. Tally and plot the average path-length/$R^2_{\rm cyl}$ (per particle) inside the cylinder and its estimated error as a function of $0.1 \leq R_{\rm beam} \leq 10$ using 10000 incident particle histories.

(b) Now let $R_{\rm beam} = 1$. Tally and plot the average path-length/$R^2_{\rm cyl}$ (per particle) inside the cylinder and its estimated error as a function of $0.1 \leq R_{\rm cyl} \leq 10$ using 10000 incident particle histories.

(c) Compare both the average values and the estimated errors of the above two examples. What conclusions do you make? Which is the more efficient way to solve the problem?

4. A geometry is defined by the infinite square lattice of cylinders

$$\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (x - di)^2 + (y - dj)^2 = R^2_{\rm cyl},$$

where i and j are integers and d is the lattice spacing constant. 10000 particles start out from the origin with an isotropically symmetric direction. Inside the cylinders there is only absorption, with an absorption constant Σ_a. Outside the cylinders there is only isotropic elastic scattering, with an interaction constant Σ_s. Starting with default values of $R_{\rm cyl} = 0.25$, $\Sigma_s = 1$, $\Sigma_a = 1$, tally and plot the average square cylindrical radius distance $\overline{x^2 + y^2}$ (and its estimated error) where the particle is absorbed as a function of the following geometrical variables:

(a) d, taking note that $d < 0.5$

(b) Σ_s

(c) Σ_a

5. The particle interaction scheme we will consider is that of either isotropic or forward scattering with a scattering constant, $\Sigma_{\rm scat}$, and particle absorption with the constant, $\Sigma_{\rm abs}$. In this example, $\Sigma_{\rm scat} = 1$ cm^{-1} and $\Sigma_{\rm abs} = 0.05$ cm^{-1}. The particles are incident normally on a planar geometry consisting of 21 planes normal to the z-axis separated by 1 cm. That is, $z = 0, 1, 2, ...20$ cm. Tally the average pathlength in each planar zone. Once within the geometry, if it hits the plane at $z = 0$ or $z = 20$, it escapes.

A working version of transport portion of the code is attached. The difficulty in doing this exercise is writing an efficient `subroutine geometry`. Use the `subroutine zplane()` from the code library to solve this problem. (It is possible to do it in about 15 lines of executable code.)

Plot the results for isotropic and forward scattering. Compare and explain the results. Hand in only your plot(s), your `subroutine geometry` and associated discussion.

```
      ! Starting position
      x = 0
      y = 0
      z = 0

      ! Starting direction
      u = 0
      v = 0
      w = 1

      iregion = 1 ! Starts incident on plane 1

1     continue ! Beginning of the transport loop

      t = -log(1 - rng())/(Sigma_scat + Sigma_abs)

      call geometry
     *(
     *    t,          ! On input, this is the proposed transport distance
     *                ! On output, if hit = .true., t is set to the
                      ! distance to the plane which it hits
     *    hit,        ! Set to .true. if the input distance t is greater
                      ! than the transport distance to a plane
     *    iregion,    ! Input: the current region number
     *    new_region  ! The new region number.
                      ! If new_region is set to 0 then is escape from either
                      ! the front or the back of the geometry
     *)

      ! Transport the particle
      x = x + u*t
      y = y + v*t
      z = z + w*t

      path(iregion)  = path(iregion)  + t    ! Tally the pathlength in region
                                             ! iregion
      path2(iregion) = path2(iregion) + t**2 ! Tally its square for statistics

      if (hit) then
          ! Particle moves out of the current region
          if (new_region .eq. 0) then
              ! It has escaped the geometry, do the next particle
```

```
                continue
            else
                iregion = new_region ! It has changed its region number
                goto 1 ! Continue transport in the new region
            end if
        else
            if (rng() .lt. Sigma_scat/(Sigma_scat + Sigma_abs)) then
                ! Particle interacts isotropically
                ! For forward scattering, comment out the next two lines
                call isotropic(costhe)
                call rotate(u,v,w,costhe)
                goto 1 ! Continue transport in the same region
            else
                ! Particle is absorbed, do the next particle
                continue
            end if
        end if
```

6. A geometry consists of a unit cube at the center of which is a sphere with diameter D, where $D \leq 1$. The planar surfaces are reflecting with an efficiency of $\epsilon = 0.95$. That means, if a particle strikes a planar surface, it has $p = 0.95$ of being reflected and $p = 0.05$ of being absorbed. The sphere, on the other hand, is totally absorbing. That is, if a particle strikes the sphere, it has $p = 1$ of being absorbed. The source for this is distributed uniformly throughout the box and radiates isotropically. The interior of the box is vacuum.

   Write a Monte Carlo code to simulate this problem and tally the number of reflections a particle experiences before it is absorbed as a function of the diameter of the sphere. Tally the number of particles absorbed by the sphere *vs.* the number of particles absorbed by the planes as a function of the diameter of the sphere.

   Use your discretion as to how many histories to execute to get a reasonable result and how fine a grid spacing in a to produce reasonable results.