LA-UR- 11-06085

Approved for public release; distribution is unlimited.

Title:

Developing a Monte Carlo mini-App for Exascale Co-Design

Author(s):

Lawrence J. Cox, Ph.D
Ryan C. Marcus
XCP-DO

Intended for:

SC11
Seattle, WA
11/13-18/2011

Distribution in conjunction with electronic poster in LANL Exhibition Booth

Developing a Monte Carlo mini-App for Exascale Co-Design

Larry J. Cox, Ryan C. Marcus
X-Computational Physics Division
Los Alamos National Laboratory

Abstract

Exascale computing research and development will need an understanding of the intended computational applications by the full spectrum of researchers. Co-design research teams will require representative applications to guide their research.

One of the important applications is Monte Carlo Radiation Transport, such as that supported by the ASC software package MCNP. This paper presents a Monte Carlo "mini-App" for Exascale R&D that encompasses many of the necessary algorithmic components of Monte Carlo transport. The physics in kept intentionally simple to avoid export control concerns. However, an effort was also made to make it representative of the "real" physics used in codes such as MCNP.

The primary goal for this MC mini-App is to clearly define the computational requirements and constraints while leaving open as much of the computer science as possible.

A secondary goal of this effort is to develop one or more sample implementations of the App that may help to advance Exascale R&D.

As the Exascale R&D progresses, this MC mini-App will provide a framework to include more realistic physics and to explore emerging hardware/software paradigms.

1. Introduction

Monte Carlo radiation transport methods are widely-used for studying the interaction of radiation with matter in almost every scientific and engineering discipline. Specific areas of application include, but are not limited to, radiation protection and dosimetry, radiation shielding, radiography, medical physics, nuclear criticality safety, detector design and analysis, nuclear oil well logging, isotope production, accelerator target design, fission and fusion reactor design, decontamination and decommissioning of nuclear facilities These methods are essential to many NNSA missions.

One example of a Monte Carlo software package is the family of codes referred to as MCNP. The latest version of this family, MCNP6, is a general-purpose, Monte Carlo N-Particle code that can be used for neutron, photon, electron, ion and exotic particle transport. MCNP6 is a merging of the MCNP5 and MCNPX efforts into a unified, multi-purpose code base. There are thousands of users in many nations, in many industries and at many universities around the world.

The goal of this effort is to develop a neutral-particle (n,γ) Monte Carlo radiation transport package tailored to solve one or more specific problems of interest and capable of running on most or all emerging hardware/software paradigms. Possible domains of interest include, but are not limited to, criticality, radiography, shielding. For now, criticality will be adopted as the canonical scientific application.

2. Project Objectives

Given a geometric configuration, its composition and a source of radiation, the computational goal of the MC mini-App is to compute the *scalar*, *energy-dependent* radiation flux in each geometrical zone.

Flux is defined as number per unit area (e.g., mol/cm^2). The radiation source may be a fixed (external) source, or it may be material reactivity (*e.g.*, fission in a nuclear reactor). In the later case, iteration will be needed to arrive at a self-consistent flux (addressed later).

The ***primary goal*** will be to clearly define the computational requirements for the mini-App in terms that can be used to create multiple implementations.

A ***secondary goal*** will be to create one or more sample implementations.

3. Assumptions and Definitions

Random Number Generator: A parallel-capable random number kernel will needed that provides a predictable, independent random number sequence to an arbitrary number of processes. The method(s) used in MCNP6 (src/mcnp_random.F90) will form the basis for this functionality and will be recast as needed into other languages.

(Initial) Source Specification: A description of a radiation source is a required component of a problem definition. This may be a user-defined external source (e.g., for radiography), or an initial guess at an internal source (e.g., for criticality). This data may need some processing, but will remain fixed after startup. It is needed only in computational step 1 (see Section 4).

Geometry: The geometry will be a "product" mesh (e.g., the geometry construct allowed in the LNK3DNT format). Each axis is specified as a list of intercepts. The geometry is the 1/2/3D outer product of the axis vectors. Cartesian (XYZ) will be the initial geometry addressed. However, extension to Cylindrical (RZ, RZθ) and spherical (R) will be kept in mind during development. Distances will be measured in centimeters (cm). Angles, when implemented, will be expressed in revolutions (1 rev = 2π radians = 360°).

Composition: The materials present in the geometry can be separately defined and stored. Since composition information is not needed at the same time geometric data is needed, keeping them separate offers more flexibility in memory placement and access. A set of composite mixtures (materials) will be defined based on their isotopics (specified in atom fractions). The isotopics will remain fixed during the calculation.

Cross Section Data: To calculate the interaction of the radiation flux with the geometry and its materials, isotopic cross section data is needed. Because the material mixtures will not change, cross section data can be loaded and partially preprocessed at startup. The nature of the radiation sources (radiation type(s), energy range, reaction channels of interest, etc.) will affect which cross section data will be needed. A computational kernel for calculating mean free path as a function of radiation type, energy and material will be needed. For creation of secondary emissions (child packets), double-differential cross section data may be required for some isotopes and reactions channels.

Type Defs: The following format is used to define data "types."

$Type(obj)$:
$$\left|\begin{array}{l} comp \\ comp2 \end{array}\right.$$

Type References: The type-component notation $obj.comp$ will be used to represent a reference to a component of a derived-type instance. For example, $p.pos(1)$ would be the x coordinate of a packet (see Section 6).

4. Startup

A significant amount of one-time preprocessing calculation can be done at startup. The following sections address some of the data that will remain fixed during the Monte Carlo calculation after initialization.

4.1 Source Description

Radiation will be described in *packets*, each representing an amount of a specific radiation type at a single energy. The intensity of a radiation packet will be represented by a *weight* that can change during the calculation.

Volume, plane and point sources may be required, depending on the type of calculation. Probability distributions in energy, position, direction and intensity may be needed. These PDFs can be loaded and preprocessed at startup. See Appendix A for more information on PDF definition and sampling.

4.2 Geometry Description

To define a Cartesian geometry, the following data is required:

$Type(geom)$:

$n(1{:}3)$	: The number of *segments* along each axis
$c_1(0{:}n(1))$	: The arrays of *bounds* for axis 1
$c_2(0{:}n(2))$	: The arrays of *bounds* for axis 2
$c_3(0{:}n(3))$	: The arrays of *bounds* for axis 3
$origin(1{:}3)$	: [optional]
$axis(1{:}2,1{:}3)$	: [optional]

The *origin* and *axis* components are needed if the geometry is not aligned with the default coordinate system (up to 9 real values needed). They are used to specify an offset and/or a rotation.

Even for very large mesh geometries, one can see that the data needed to define the geometry is relatively small. For example, a billion zones (1000^3) would only require 3012 real values (the bounds and orientation) and 3 integer values (the segment counts). The number of zones in a problem is the product of the segment counts along each dimension, $n_{zone} = \prod n(:)$.

The geometric zones will be indexed as a one dimension array $g \in (1{:}n_{zone})$. With $(i,j,k) \in (1{:}n(1), 1{:}n(2), 1{:}n(3))$ being the coordinate indices into the 2D bounds array, g is given by:

$$g(i,j,k) = k + n(3) \times [(j-1) + \mathrm{n}(2) \times (i-1)] \tag{1}$$

If any of the indices are out of bounds, then we define $g \equiv 0$, indicating that the location is not in the geometry.

The LNK3DNT file format stores the necessary counts and dimensions. It does *not* store orientation information (*origin*, *axis*). These quantities will need to be separate, user-defined inputs, if supported.

4.3 Material Composition Data

The LNK3DNT format also stores partial mass densities (g/cc) for each material in a zone. However, it does *not* store isotopics for the materials. A possible derived type for a material is this:

$Type(material)$:

n_{iso}	: number of isotopes in the material
$tbl(1{:}\,n_{iso})$	: cross section table identifiers
$za(1{:}\,n_{iso})$	: ZA numbers for each isotope
$af(1{:}\,n_{iso})$	: atom fractions for each isotope
$stp(1{:}\,n_E)$	: stopping power (See Section 4.4)

Isotopes are specified by $za(i) = 1000 \times Z(i) + A(i)$ where $Z(i)$ is the number of protons and $A(i)$ is the total number of nucleons (protons and neutrons) for isotope i. For example, the ZA for ^{238}U is 92238. The $tbl(:)$ component is an alphanumeric string that identifies the cross section table (or model) to use for each isotope. The atom fraction component, $af(:)$, is normalized so that $\sum af = 1$. Non-normalized atom fractions in the input will be permitted, setting a normalization requirement at startup. This will allow the specification of water as 2 hydrogen ($za = 1001$) and 1 oxygen ($za = 8016$).

Each geometric zone may have *zero* or more materials each with a specified partial density. A derived type could be defined that contains the number of materials, the material indices and the material partial densities for a single zone.

$Type(zone)$:

n_{mat}	: number of materials homogenized in the zone
$m(1{:}\,n_{mat})$	: indices into the materials array
$\rho(1{:}\,n_{mat})$	: partial mass densities (g/cc)

In this form, the composition data for a single zone can be kept nominally contiguous and perhaps more easily copied to where it is needed.

Voids (vacuum zones) are indicated by having no materials, $z.n_{mat} = 0$. The material and density information for each zone is stored in the LNK3DNT file format. The total density of a zone is the direct sum of the partial densities. The mass is that sum times the volume of the zone. Total density, mass and volume are values that can be computed as needed.

Other data-layout possibilities should also be considered. One example would be to assign the same number of material slots per zone (the problem maximum), which would make the array components uniform in size across all zones and eliminate the need to store a variable number of materials per zone.

4.4 Material Cross Section Data

The cross sections for radiation interaction with matter are dependent on the radiation type and energy, the isotopes present. One use of cross sections will be in calculating the *mean free path* of a radiation packet. This value can also be interpreted as an *attenuation length.* For each radiation type and isotope, there is a microscopic *total* cross section. These cross sections are isotope and radiation type specific and are a function of (radiation) energy. Other factors such as material temperature can affect cross section values.

The mean free path is a distance over which we expect to have $e^{-1} \cong 36.7\%$ attenuation (reduction in intensity) of a radiation source. The weight (representing the radiation intensity) of a radiation packet p traversing a distance s in a material with attenuation length λ would be $wgt(s) = p.w_0 \times e^{-s/\lambda}$. The average distance before any sort of reaction event, the *mean-free-path,* is this attenuation length by evaluating the following:

$$\langle s \rangle = \frac{\int s \times e^{-s/\lambda} ds}{\int e^{-s/\lambda} ds} = \lambda$$

Units analysis can illuminate the approach to calculating mean free paths. Interaction cross sections σ are in area units called *barns* (1 *barn* = 10^{-24} cm^2/nucleus). Mass densities ρ are in units of g/cm^3. Isotopic densities can be expressed in units of Avogadro's number ($A_v = 6.022 \times 10^{23}$ nuclei/mol). Isotope weights are in terms of atomic mass units, *AMU,* which are equivalent to mass per mol (g/mol). The product of these units is $(\sigma \times A_v \times \rho \div AMU) =$ (~~cm~~^{2}/~~nucleus~~)(~~nuclei~~/~~mol~~)(~~g~~/cm^3)(~~mol~~/~~g~~) = [cm^{-1}] or inverse distance. Summed over isotopes, this gives the reciprocal of the mean free path, λ. In this form, all terms except A_v are isotope specific.

Since densities can vary by zone, it is helpful to pull out the overall material density and replace it in the summation with isotopic atom fractions (a dimensionless quantity). This gives what can be called a *stopping power* [cm^2/g] for each material:

$$m.stp(E) = A_v \times \sum_i \left(\frac{\sigma_T(i,E) \times m.af(i)}{AMU(i)} \right)$$

Where the sum is over the isotopes, i, of material m. Note that microscopic total cross sections, $\sigma_T(i,E)$, and hence the stopping powers, are dependent on the radiation type and energy, as well as being isotope specific.

The mean free path, λ [in cm], for the mixture of materials in a zone, z, is obtained by summing the stopping powers over the materials, $m(j),\ j \in \{1, n_{mat}\}$, in the zone.

$$\lambda(z,E) = \left[\sum_j (z.\rho(j) \times z.m(j).stp(E))\right]^{-1}$$

For initial implementations, we may want to use discrete energy ranges ("energy groups") instead of *continuous energies*. Cross sections specified as constant over an energy range are called *multi-group* cross sections and represent a weighted average over the energy range for the group. The weighting is done with respect to a specified energy distribution, $\varphi(E)$.

$$\overline{\sigma_T}(g) = \frac{\int_{\Delta E(g)} \varphi(E)\sigma_T(E)dE}{\int_{\Delta E(g)} \varphi(E)dE}$$

This formula gives the energy-weighted, average total microscopic cross section, $\overline{\sigma_T}(g)$, over the energy group g where $\Delta E(g) = (E_g - E_{g-1})$. Again, note that these values are isotope and radiation type specific. With the multi-group approach, stopping powers can be calculated in advance for each material and energy group. If a continuous energy approach is adopted, some level of pre-calculation may be possible coupled with an interpolation scheme for obtaining values at energies not specifically calculated.

For internal source problems or fixed-source problems where scatter is of interest, the flux estimate arising from the initial source guess will be used as a new or additional source term. Sampling from a flux requires access to radiation producing reaction channel data, parts of the cross section data tables. Only for reaction channels that can generate radiation types of interest need be loaded. For example, if we are only interested in neutron transport, reaction details that can produce neutrons will be needed (fission, n2n, nXn...). As with other cross section data, reaction channel data will be fixed after startup processing and can be distributed as needed to processes.

5. Computation Steps for the MC mini-App

Given the geometric configuration, its composition and source of radiation, the computational goal of the MC mini-App is to compute the *scalar*, *energy-dependent* radiation flux in each geometrical zone. Flux is defined as number per unit area (e.g., moles/cm^2). The source may be a fixed (external) source, or it may be material reactivity (e.g., fission in a nuclear reactor). In the latter case, iteration will be needed to arrive at a self-consistent flux (addressed later).

The approach to calculating the flux consists of the following, separable steps. Each of theses steps has different data needs and opportunities for parallelism. Depending on the target hardware/software architecture, some of the steps can be combined. It may also be possible to breakdown the steps into smaller units.

Source: Generate a random sampling of radiation histories (packets): position (x, y, z), direction (u, v, w), energy (e), weight (wgt) and time (t). Packets will start with unit weight. The number of histories (per iteration) is a user-input parameter;

RayTrace: Calculate the track through the geometry by ray-tracing each packet from its origin to its exit from the geometry (or its intersection with an external tally plane, if any). The result is a list of positive distances, $trace(axis, bound)$, to boundary crossings along each axis. The list is ordered for each axis.;

Segment: Determine the *ordered* list of zones, $\{g\}$, that the track intercepts and the path length in each zone, $\{ds(g)\}$ [cm];

Attenuate: The weight is reduced along the track, $\{ds(g)\}$, zone-by-zone, using the *mean free path* [MFP] derived from the reaction cross section(s). Material and cross section data is needed for this step;

Tally: Accumulate the weighted path lengths, zone-by-zone ($wgt \times ds$) [cm] from a packet batch and its associated child batches (if any);

Spawn: The attenuation factors are used to create secondary (reaction emitted) packets. These "child" packets (descended from their "parents") will be used to calculate the contribution to tallies from a child batch.

Normalize: Determine flux in zones of interest (nominally all) by dividing the net weighted path lengths by the zone volumes [cm^3].

Each of these algorithm components is addressed in more detail in the sections that follow.

For internal sources (*e.g.*, criticality calculations), the initial packets are selected at random from an initial "guess" as to the flux distribution. When a new flux estimate is obtained from the initial source, it is used to generate a new source, which is propagated to refine the flux with steps 2-6. A self consistent flux is one where a source generated from it will result in the same flux. The number of iterations is a user-input parameter. In the future, provisions may be developed for estimating the (self) consistency of the flux, iteration to iteration.

For external sources (e.g., radiography or shielding calculations), the source is user defined and the transmitted packets may need to be accumulated into an image or dose. For these problems, the flux can either be ignored (eliminating step 5 completely and eliminating the need for knowing the element *order* in step 3) or used to estimate a scattered component (e.g., noise in an image).

6. Source – Generate a Batch of Packets

The first step is to obtain a batch of initial *packet* descriptions. Packets will be randomly generated in *batches* of independent packets to be processed together. Statistics (convergence, variance, etc.) will be performed on the variation between batches. This means that each batch can be considered independent with respect to the other computational steps. See Appendix A for information on source PDFs and sampling techniques.

Inputs:	Source descriptions (spatial, energy, time);
	May also need access to geometric and composition information;
Parallelism:	Each packet can be independently sourced;
Memory:	Each packet will occupy ~80bytes.
	Other data needed will vary in size, but be unchanging.

Packet data includes the packet number ($pnum$), radiation type ($rtype$: i.e., neutron, photon…), position [$pos(1{:}3)$], direction [$dir(1{:}3)$], energy (e), initial weight (w_0) and initial time (t_0) [if needed]. Direction is specified in *direction cosines* with respect to the x, y, z axes. Therefore a packet consists of nine (9) real values and two (2) integer values ($pnum$, $rtype$). Note that packets will always be described in a Cartesian (XYZ) coordinate system, even if a non-Cartesian geometry model is used. A packet object will be expressed as follows.

$Type(packet)$:

$pnum$	: packet number/index
$rtype$	: radiation type (neutron, photon)
$pos(3)$	: Cartesian location (xyz)
$dir(3)$	: Cartesian direction cosines $[-1,1]$
e, t, wgt	: energy (MeV), time (s), weight

With double precision real values, the data-size of a packet is o(80) bytes (10 words). As the MC capabilities are developed, other packet attributes may become necessary. However, trade offs between storage and memory footprint can be made by (re)computing other values when needed. One possible reduction would be to use only two direction cosines. Because the sum of the squares of the cosines must be unity, we could do with only two, for instance with only $dir(1{:}2)$, $dir(3) = \sqrt{1 - dir(1)^2 - dir(2)^2}$. For computational simplicity (to be shown below), it may be better to have all three exist as part of a packet object. This can be a computer-science design decision. For algorithmic discussions, we assume all three direction cosines are available.

7. RayTrace – Determine the Boundary Crossings for Each Dimension

With a batch of packets, the next step is to calculate the intersection points with the geometry. Only the geometry data and a packet's position and direction are needed for each trace. A *trace* is a list of the distances to intersections with the geometry boundaries. The trace from each packet is independent (actually each axis of the trace is independent of the others), offering parallelism opportunities needing relatively small amounts of data.

Inputs:	*packet(s), geometry*
Parallelism:	Each axis of each raytrace is independent.
Memory:	A trace will be a data object consisting of approximately $\sum(n(i)+1)$ **words.**

A canonical data structure is:

$Type(trace)$:

p -> *packet* pointer (or object? or packet number?)
$d_1(0\colon p.n(1))$
$d_2(0\colon p.n(2))$
$d_3(0\colon p.n(3))$

A reference to the parent packet is included because its energy and initial weight will be needed in later steps.

For Cartesian geometries, the distance algorithm is quite simple. From position *p* and direction-cosine *dc* both measured perpendicular to a plane, the general formula for the distance *d* to the plane at *b* is given by $[d = (b-p)/dc]$. More complicated formulae are needed for cylindrical or spherical surfaces.

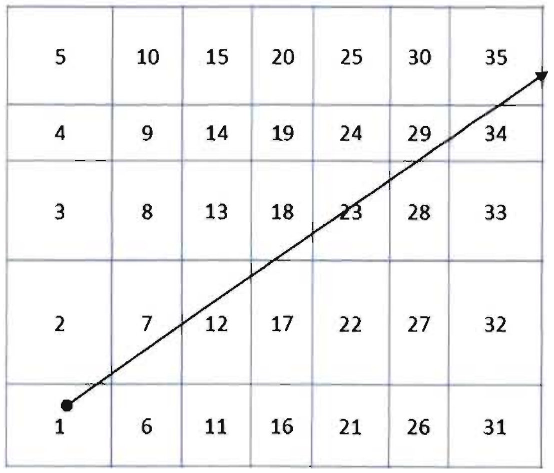

Figure 1: A sample RayTrace through a 2D, 5x7 mesh. The packet trace originates in zone 1 and exits the mesh through the maximum *x* boundary in zone 35.

The resulting distance d is measured along the direction of motion, $p.dir$, from the packet origin, $p.pos$. For instance, if the plane is perpendicular to the x-axis, then $p = x$, $dc = u$ and b is the x-position of the plane. Negative values indicate the intercept is behind the current position. If $dc = 0$, this means the path is parallel to the plane and there is no intercept.

In array-syntax pseudo code, the calculation of all intersections of a may look like this:

```
For axes $i = 1, 3$ :
    Where $trace.p.dir(i) \neq 0$,  $trace.d_i(:) = \left(\frac{geom.c_i(:) - trace.p.pos(i)}{trace.p.dir(i)}\right)$;
    Otherwise  $trace.d_i(:) = -\infty$;
```

Any non-positive value can be assigned to axes with $p.dir(i) = 0$. This is done to separate infinite and backwards distances from those of interest (finite and positive). This formula gives the three intercept array components of a trace, one for each axis. The component $trace.d_1(:)$ contains the distances to all of the x bounds given by $geom.c_1(:)$. The y and z intercept arrays are in $trace.d_i(:)$ with $i = 2$ and 3, respectively.

8. Segment – Determine Zone-by-zone Path Lengths

The intercepts specified in a *trace* are in order for each *axis*, but the three lists need to be merged into a single, ordered list coupled with a list of the associated axis indices (zones intercepted). An additional complication is that where $p.dir(a) < 0$, the ordering of the distances for axis a is backwards. Note that we can ignore axes a where $p.dir(a) = 0$ as all those distances will be infinite.

Inputs: A $trace$ object is an input to this step.
Parallelism: Each conversion of a *trace* to a *segment* is independent
Memory: A *segment* is an object containing ordered lists of path lengths and intercepted zones.

```
$Type(segment)$
    $e = t.p.e$
    $w_0 = t.p.w_0$
    $ns$
    $zone(1{:}ns)$
    $ds(1{:}ns)$ [can also act as $s(:)$ temporarily]
```

This operation is parallelizable on traces, an extension of the parallelism on packets in the Trace step.

Here is a pseudo-code approach to a merge-sort. First find the *first* and *last* indices, and the *step* for each axis:

```
For axes $i = 1,3$ :
    $step(i) = sign(p.dir(i))$
    $first(i) =$ index of ( $min( trace.d(:) > 0 )$ )
    $last(i) =$ index of ( $max( trace.d_i(:) )$ )
```

These three triplets give the starting, ending and increment indices for each axis. If $p.dir(i) = 0$, the *first* and *last* indices will be the same (the starting index on that axis). This condition needs to be identified and dealt with appropriately when setting *first*, *last* and *step*.

The first distance to intercept, $s(1)$, will be $min(trace.d_l(first(l)), l = 1{,}3)$ from which we also need to obtain the axis l on which this first intercept lies.

$$l = \text{index of } (\min(trace.d_l(first(l)), l = 1{,}3))$$

The first zone intercepted is given by $segment.zone(1) = g(i,j,k)$ where $(i,j,k) = (first(1), first(2), first(3))$ using Equation (1). This *should* be the element in which the trace starts. Care must be taken to identify the first zone intersected. The algorithm above should be correct as the zone index is calculated using the upper bound indices in Equation 1. However, the logic needs to be checked.

To get the next distance, the index for the axis l found for $s(1)$ is incremented by $step(l)$. For example, if l=1 (the x axis), $i = i + step(1)$; j and k do not change. Also increment the number of segments, $segment.ns$ by one. With the updated (i,j,k) triplet, the next distance, $s(2)$, and the next element, $segment.zone(2)$, are obtained using the method described above.

This process is repeated until an axis $last(:)$ index is exceeded. Because all of the distances are calculated to unbounded planes, there may still be positive distances along the other axes. But those intercepts are all out of bounds (outside the geometry). There cannot be more than ($[\sum n(a)]$- 1) zones intercepted, potentially many less.

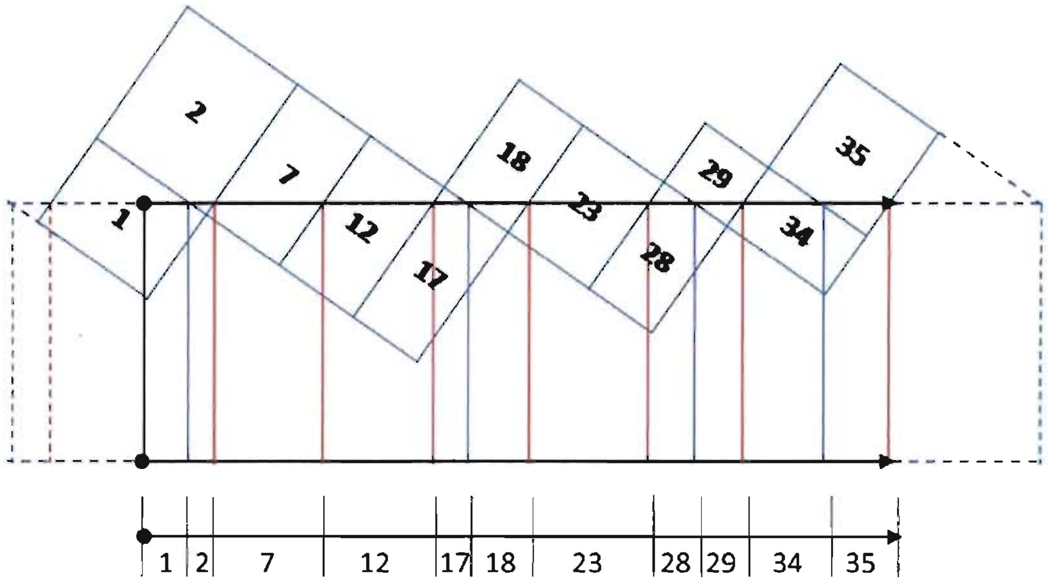

Figure 2: **Segment example in a 5x7 mesh**. The Segment step merges the components of a Trace. The trace from **Figure 1** has seven positive x intercepts and five positive y intercepts, the last of which is off the grid. There are eleven zones intercepted, in this order: $g \in \{1,2,7,12,17,18,23,28,29,34,35\}$. The intercepts from the x and y axes are shown in red and blue, respectively, demonstrating the merging of the trace components.

Adding a distance to an image plane is relatively trivial. Only the position and orientation of the plane are needed. Note that an image plane may be arbitrarily oriented with respect to the geometry.

One last step is to convert the distance list, $s(:)$, into zone-lengths, $segment.ds(:)$, thusly:

$$segment.ds(1) = s(1);$$
$$segment.ds(l) = s(l) - s(l-1), \text{for } l \in \{2, segment.ns\}.$$

9. Attenuate – Estimate Bulk Effect on Intensity

In addition to a segment object s, this step requires information about the materials and cross sections to calculate the mean free path $\lambda(z, E)$ for a packet at energy, $E = segment.E$, in zones $z \in \{s.zone(:)\}$. The calculation of mean free paths was defined in Section 4.4. For any segment object s, the energy is constant, simplifying the cross section requirements considerably.

Inputs: A segment object; material/energy dependent mean free path.

Parallelism: On segments (packets), potentially on zones

Memory:

The attenuate step converts the segment list from length units [cm] to mean-free-path units to allow for simple, cumulative attenuation along the trace.

Recall that attenuation of intensity (weight in our context) from passing through a distance ds in a material with mean free path λ is given by:

$$wgt(ds) = s.w_0 \times e^{-ds/\lambda}$$

By converting lengths ds into units of mean-free-path, $d\ell = ds/\lambda$, the equation simplifies to this:

$$wgt(\ell) = s.w_0 \times e^{-d\ell}$$

With our segment list converted to mean free paths, attenuation through multiple zones is given by:

$$wgt(i) = s.w_0 \times e^{-\sum d\ell(j)} \text{ for } j < i, \text{ or}$$

$$wgt(i) = s.w_0 \times e^{-\ell(i)} \text{ for } \ell(i) = \sum_{j=1}^{i-1} d\ell(j)$$

This is the weight of the path segment in the ith zone intercepted. Note that $wgt(1) = s.w_0$; that is, the weight in the zone of origin is the original (un-attenuated) weight.

10. Tally – Accumulating Results

For each batch of packets (parent or child), tallies represent the accumulation of some quantity as an estimate of a desired attribute of the entire system. The accumulation can be over zones (a flux tally) or on an image plane (image tally). This section discusses the similarity and differences between the two tally types.

10.1.1 Flux Tally

A (scalar) flux tally can be used to estimate the radiation flux as a function of energy in the mesh. The radiation flux (units of [cm^{-2}]) is a measure of the intensity of radiation as a function of energy, zone by zone.

The basic *dimensionless* flux tally is simply the accumulation of the mean free path lengths, $\Delta\ell_E(\mathrm{i}) = (\mathrm{i}) - \ell_E(\mathrm{i}-1)$, appropriately attenuated, traveled in each zone.

$$\varphi(z(i), E) = \sum\left(\Delta\ell_E(\mathrm{i}) \times \mathrm{e}^{-\ell_E(\mathrm{i}-1)}\right)$$

Conversion to a radiation flux is done by multiplying by the mean free path, $\lambda(z, E)$, and dividing by the zonal volume, $\mathrm{Vol}(z)$.

$$\phi(z, E) = \frac{\lambda(z, E) \times \varphi(z, E)}{\mathrm{Vol}(z)}$$

Other flux-based tally types can be estimated by adding appropriate factors in the flux summation. For instance, an energy flux (a single number per zone) could be obtained by multiplying each contribution by the packet energy. Similarly, an energy *deposition* tally could be made by applying a (modified) material stopping power to each contribution.

10.1.2 Image Tally

It is also possible to accumulate image tallies from the radiation packets (attenuated) that are transmitted through the geometry. This work has not yet been started

11. Spawn – Creating Child Packets

With the calculated list of attenuation lengths, $\ell(i)$, found above, we can generate randomly sample secondaries (child packets) emitted from reactions of the parent packet with the materials in the geometry. These packets will be used to calculate additional contributions to tally estimates. In a radiography calculation, for example, the (first generation) "parent" batch contributions represent a direct image. Child packet batches will represent a scatter contribution to calculated images.

A reaction distance in mean free paths can be sampled using the formula $d\ell' = -\ln(\text{rand})$ where a random value, rand $\in \{0,1\}$ is used. *Note that this is always a positive value.*

If we sample *multiple* steps accumulating the random strides until we exceed the maximum length (in mean free paths) along a trace, we obtain the number child packets to generate and enough information to determine their initial locations.

$$\ell'(k) = \ell'(k-1) + [-\ln(rand)] \text{ for } k \in \{1, n \in: \ell'(n) < \ell_{\max}\}$$

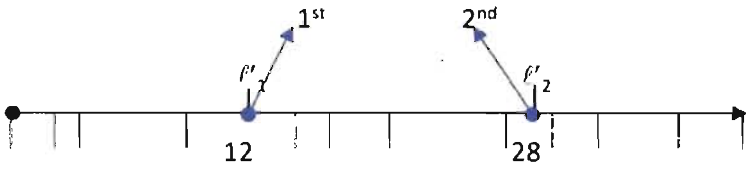

Figure 3: Secondaries Example. In this example continued from **Figure 2**, two child packets are created in zones 12 and 28, respectively. Their origins will lie on the original track. The energy, direction and weight will be sampled using reaction kinematics for the respective materials using the original packet energy and direction as input.

The child locations found are given in cumulative mean free paths. These values are used to identify the zones in which the reactions occur and are also converted back into physical distances (from the parent's origin) to establish the secondary origins. A child packet is created in the nth zone of a segment, $s.zone(n)$, where $\ell(n-1) \leq \ell' < \ell(n)$. The physical distances from the parent's origin (measured along the parent's direction) to the child packet origin is given by:

$$d'(i) = \lambda(z(n), E) \times \left(\ell'(i) - \ell(n)\right) + \left(\sum_{j=1}^{n-1} [\lambda(z(j), E) \times \ell(j)]\right)$$

The new origins (as points) are obtained from the distances $d'(:)$ and the parent data, p_O.

$$p'(:).pos = p_O.pos + p_O.dir \times d'(:)$$

The direction, energy and *final* weight of each child will be randomly sampled using reaction kinematics defined by the cross sections[1]. The *colliding* weight is the parent weight, $p_O.wgt$, attenuated by the distance to the collision.

$$p'(:).wgt = p_O.wgt \times e^{-\ell'(:)}$$

Reaction kinematics can increase or decrease a child packet's weight. For example, if the collision is a capture (absorption), the kinematic weight is zero and we can discard the child. But if the collision is an $(n, 2n)$ reaction (one neutron in, two out), the kinematic weight is two. In this case, we can either generate two child packets each of the colliding weight or one child of twice the colliding weight.

After creation, a batch of these child packets (all arising from a parent batch) will be processed as its own "batch" with results being added to appropriate parent batch tallies. Child packets are processed through the same computational as their parents: RayTrace, Segment, Attenuate, Spawn (if more generations are needed or requested) and Tally. The Normalize step can be deferred until all required/requested generation batches are completed.

It is possible to spawn multiple generations (parent -> child -> grandchild...) and accumulate the results. For now, the number of levels will be taken as zero (no children) or one (one generation) with an option for more generations as a runtime parameter.

[1] See Appendix B for consideration of Center of Momentum [CM] corrections to reaction kinematics

Appendix A: Source Distributions and Sampling

Generation of initial radiation packets requires the provision of probability distribution functions (PDFs) and sampling techniques to be used with them. This appendix describes some common source distributions, simple descriptions for them and methods for randomly sampling from them.

Source descriptions may include intensity (weight), location, direction, energy and perhaps time. The radiation type of a source must also be known and may include more than one radiation type. For a single radiation type, the source can have up to nine (9) independent variables – a 9D phase space. It is possible for the PDFs to be inseparable functions of all of these variables, however, in practice, the function is separable (or approximately separable) into simple functional forms of the different variables.

The general form for sampling a random value (x) from a PDF described by a function $f(x)$ is

$$\varepsilon = \frac{\int_{c_{min}}^{x} f(c)dc}{\int_{c_{min}}^{c_{max}} f(c)dc}$$

where $\varepsilon \in (0,1]$. For many analytic PDFs, a closed form for x can be derived. More complicated forms may require a rejection-based sampling scheme. Examples of both for sampling a direction randomly in 4π are given in Appendix B.

For the purposes of this application, energy, weight and time will be considered as independent (uncorrelated) variables in the phase space. PDFs for these variables will be supported as piecewise-constant (histogram), piecewise-linear and polynomial forms.

Location will be considered to be independent of direction in most cases. Note that this is an oversimplification for many applications, but will suffice for the purposes of this mini-App. Only for uniform spherical sources and planar sources coupled with non-isotropic directional PDFs will a correlation be supported. In these cases, the direction sampled will be with respect to [w.r.t.] the radial location vector (for spherical volumes) and w.r.t. the surface normal vector (for surface sources). Hence, in those cases, a rotation of the sampled direction is required to convert it to the default Cartesian system.

Location: Volume, surface and point distributions will be supported. Point sources are obvious. For a single point, each packed will originate at that point. For multiple points, each point will be given a probability. A packet's origin will be chosen randomly from the list using the probabilities.

Surface sources will be supported from planes and spheres.

Volume sources that vary in Cartesian location (x,y,z) or radius (R) will be permitted with polynomial forms. The most basic form is a uniform distribution within a spherical or parallelepiped volume.

Uniform Parallelepiped: Sampling a location in a uniform parallelepiped is straightforward. A position along each axis is sample uniformly within the limits along that axis, i.e., $x = x_{min} + \varepsilon \times (x_{max} - x_{min})$ for $\varepsilon \in (0,1]$. If the parallelepiped is not aligned with the default Cartesian coordinate system, the simplest approach is to sample as if it is aligned and apply the necessary, fixed rotation.

Uniform Sphere: For a uniform spherical distribution of radius R_{max}, a radius is sampled using the cube-root of one random number $\varepsilon \in (0,1]$:

$$r = R_{max} \times \varepsilon^{1/3}$$

This form is obtained by substituting $f(c)dc = r^2dr$ into the general form given above with radial limits of $[0, R_{max}]$. The orientation within the sphere is calculated using the methods described in Appendix B for choosing a random isotropic direction in $4\pi, \overrightarrow{\langle u, v, w\rangle}$. For this purpose, the result is equivalent to a random rotation with respect to the default *z*-axis. Multiplying a random rotation by a random radius r gives a random Cartesian position: $p.pos(1{:}3) = r \times \overrightarrow{\langle u, v, w\rangle}$.

Direction: Direction distributions can be isotropic or angularly biased. The forms this mini-App will support are isotropic and cylindrically-symmetric anisotropic.

Isotropic: See Appendix B for methods to sample from a fully isotropic distribution.

Anisotropic: A cylindrically symmetric anisotropic distribution is one that varies as a function of the angle from a specified symmetry axis. The symmetry axis may be arbitrarily defined by a constant unit direction or may be correlated to a chosen location. See the discussion above for how this axis is correlated to some location PDFs. Azimuthally around the symmetry axis, the distribution is assumed to be uniform, *i.e.*, the azimuthal angle is a random value $\phi \in (0,2\pi]$. Axial distributions will be supported as functions of the *cosine* of the axial angle, *e.g.*, $f(c)dc = \cos(\theta)d\theta$, in the general form. Limits on the range of θ will also be permitted, specifically being helpful for radiography point sources. One commonly used set of functions are the Legendre polynomials [P_n]. Note that the isotropic distribution is a special case of this with $f(c) = 1$ [P$_0$]. There is a recursive form for Legendre polynomials:

$$P_0 = 1$$
$$P_1 = \mathrm{x}$$
$$P_{n+1}(x) = \frac{(2n + 1)xP_n(x) - nP_{n-1}(x)}{n + 1}$$

Appendix B: Sampling a Random Direction in 4π

Sampling a direction randomly in 3-space can be equated to uniformly sampling locations on a unit sphere. Such locations can be specified by two angles, an axial angle in the range $[0,\pi]$; and an azimuthal angle in the range $[0,2\pi]$. The azimuthal angle is uniformly distributed in *angle*. However, the axial angle is uniformly distributed in the *cosine of the angle*.

There are direct and rejection approaches to sampling random values that meet these conditions.

Direct Technique:

The z direction cosine, w, is randomly distributed between $(-1,1]$, so it can be chosen simply with one random number, $\varepsilon \in (0,1]$:

$$w = 2\varepsilon - 1$$

The *transverse* direction cosine, call it β, is given by:

$$\beta = \sqrt{1 - w^2}$$

Note that $\beta = \sqrt{u^v + v^2}$ as well.

The azimuthal angle is randomly distributed between $(0,2\pi]$. For such a random angle, $\varepsilon \in (0,2\pi]$, the direction *cosines* for x and y are given by

$$u = \beta \times \cos(\varepsilon)$$
$$v = \beta \times \sin(\varepsilon)$$

2D Rejection Technique[2]:

It may be faster to reject some values than calculate trig functions. The azimuthal direction cosines, u and v, can be sampled using a rejection scheme.

With w and β calculated as above, choose two random values, a and b, in the range $(-1,1]$.

- If $(a^2 + b^2) > 1$, reject the pair and choose again. (~21.5% likelihood of rejection)
- When accepted values are found (~78.5% likelihood for any pair), u and v are given by:

$$u = \beta \times \frac{(a^2 - b^2)}{(a^2 + b^2)}$$
$$v = \beta \times \frac{(2ab)}{(a^2 + b^2)}$$

[2] *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*, Lux & Koblinger, CRC Press, 1990

Appendix C: Center of Momentum Corrections for Kinematics

Cross section tables describe the kinematics in the collision's center-of-momentum [CM] frame of reference. If the material is taken to be stationary (the lab frame of reference), the collision CM is determined by the energy and direction of the parent and the mass of the isotope with which it collides. To sample the cross section tables in the CM frame, we need to find the parent packet energy in the CM. For this MC mini-App, Galilean transformations will be used for simplicity.[3]

In the CM, the packet momentum (a vector quantity) is equal to and opposite the nucleus' momentum. Starting with momentum formula $\vec{p} = m\vec{v}$, energy formula $E = \frac{\vec{p}\cdot\vec{p}}{2m}$, and an initial nucleus velocity of zero, we solve for a change in speed $\Delta v \in : m_p(|v_p| - \Delta v) = m_n \Delta v$, yielding $\Delta v = \frac{m_p|v_p|}{(m_p+m_n)}$. Substitution into the energy formula gives the collision energy in the CM, $E_p{}'$.

$$E_p{}' = \frac{1}{2}m_p\left(v_p - \frac{m_p|v_p|}{(m_p+m_n)}\right)^2 = E_p\left(\frac{m_n}{m_p+m_n}\right)^2 \leq E_p$$

This reduced *projectile* energy is used to sample the cross section kinematics to create the CM-based child packets, p_c'. The resulting CM-energy, $E_c{}'$, and CM-direction, $p_c'.dir$, are transformed back to the laboratory frame. These transformations are accomplished by adding the *vector* momentum component, $m_c\Delta v \times p_O.dir$ to the *vector* CM-momentum of the child $\sqrt{2m_c E'_c} \times p_c'.dir$. This vector sum gives the child's lab-frame momentum, from which its direction and energy can be derived.

[3] Special relativity corrections to energy and momentum should to be considered in CM transformations to ensure the physics is accurately represented. Further adjustment to the CM for random nuclear motion in "hot" bulk material may also be important.