SANDIA REPORT
SAND2019-3660
Printed Click to enter a date

Sandia National Laboratories

Photon Radiation Scatter from Heterogeneous Shields using Green's Functions

Gregory G. Thoreson, Steven M. Horne, and Dean J. Mitchell

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

The effect of shielding on ionizing photon radiation can be estimated using radiation transport simulations. This report covers the methodology and implementation of using Green's Functions to pre-compute this effect, which allows the radiation field exiting a variety of shielding configurations to be quickly computed. It also covers a weighting function that makes a relatively small pre-computed library applicable to a large variety of heterogeneous shields. The method enables rapid computation of the intensity versus energy for scattered radiation exiting a variety of shield materials and thicknesses without running a full transport simulation.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page left blank

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
GADRAS	Gamma Detector Response and Analysis Software

1. INTRODUCTION

Common sources of ionizing photon radiation are gamma decay, x-ray fluorescence, and bremsstrahlung. Many applications require shielding to limit the radiation dose from the source. In other applications, shielding may be undesirable but unavoidable, and the radiation emitted from the shield is indicative of the source and shield. In all cases, estimating the effect of the shield is typically done with radiation transport simulations in the form of Monte Carlo or deterministic solvers. A simpler approach is to use build-up factors that pre-compute the effect of the dose changing with shielding. This is similar to the methodology outlined in this report, except that the build-up approach typically integrates the exiting radiation field with a dose response function, removing much of the information observable with spectral radiation detectors.

A common complication is heterogeneous shields. The sequence of shielding materials is important because radiation exiting the shield from a source shielded first by a low atomic number (e.g. graphite) followed by a high atomic number (e.g. lead) will have a different spectral shape than if the two materials were reversed. This contrasts with attenuation of uncollided gamma-rays, where the transmission probability is the product of the probabilities of not interacting with all shielding layers, which is independent of material sequence. Accordingly, when the order of two shielding materials is reversed, the gamma-ray spectra exhibit the same photopeak intensities, but the configuration with the higher atomic-number material on the outside of the radiation source produces less scatter continuum.

This report briefly covers the theory of Green's Functions and how they are applicable to this problem. The weighting function that makes them applicable to heterogeneous shields is also discussed. An example implementation with Python 2.7 [1] source code is covered, and a light-weight shield-scatter library file is also available upon request.

2. METHODOLOGY

2.1. Green's Functions

Green's Functions describe the response of a system to an excitation or an impulse. They are a convenient means to encapsulate complex physics into a re-usable function that is evaluated from a simple integral, rather than solving a differential equation. The differential equation at hand is the Boltzmann transport equation,

$$\left\{\frac{1}{v}\frac{\partial\psi}{\partial t}\right\} = \{q_{ex} + q_s + q_f\} - \{\sigma_t\psi\} - \{\hat{\Omega}\cdot\nabla\psi\}, \tag{1}$$

where $\psi = \psi(\vec{r},t,E,\hat{\Omega})$ is the particle flux ($cm^{-2}\cdot s^{-1}\cdot eV^{-1}\cdot str^{-1}$) in the volume d^3r about $\vec{r}$ (cm) traveling in direction $d\hat{\Omega}$ about $\hat{\Omega}$ (steradians, str) with energy between E and $E+dE$ (electron-volts, eV) at time t (seconds, s), $v = v(E)$ is the velocity of the particle in ($cm\cdot s^{-1}$), σ_t is the total attenuation cross section in (cm^{-1}), and q_{ex}, q_s, and q_f are source terms from external sources, scattering, and fission, respectively, in ($cm^{-3}\cdot s^{-1}\cdot eV^{-1}\cdot str^{-1}$). Let L be the differential operator such that it satisfies the equation

$$L\psi = q_{ex}. \tag{2}$$

Furthermore, let λ represent the phase-space of the source term such that $d\lambda = d^3\vec{r}\,dt\,dE\,d\hat{\Omega}$. Generalizing the source term in this equation is accomplished by replacing q_{ex} with the Dirac delta function about phase-space and replacing the particle flux ψ with the Green's Function G,

$$LG(\lambda;\lambda') = \delta(\lambda-\lambda'). \tag{3}$$

Therefore, the source is now a general point function about phase-space λ'. If the solution to G is known, then the particle flux (response of the system), to any source (excitation of the system) is calculated by integrating over all phase-space of the source Λ,

$$\psi = \int_{\Lambda} d\lambda' G(\lambda;\lambda')q_{ex}(\lambda'),\ \lambda'\in\Lambda. \tag{4}$$

The benefit of the Green's Function method is that the computation time is pushed up-front in the calculation of G, and the computational evaluation of the integral in Equation (4) is trivial compared to solving Equation (2).

Equation (4) contains an integral over all phase-space. The challenge in applying Green's Functions is deciding which parameters to include in the phase-space integration, which to pre-integrate over, and which to parameterize over. For shielding calculations, we can significantly simplify this by assuming time independence, pre-integrate over the spatial and angular distribution of the source by assuming an isotropic point source, and pre-integrate the flux over the exterior of a spherical shield. This reduces the phase-space integration to a simple energy transformation,

$$\psi(E) = \int_0^\infty dE' G(E;E') q_{ex}(E'). \quad (5)$$

Although we assumed a point source in the center of a spherical shield, the same assumptions could be made for a variety of setups, including a beam source on a slab shield. The approximations are applicable because isotropic sources are common for radiation shielding and spherical geometries provide the necessary symmetry to integrate over the exterior of the shield to disregard angle and space.

For a simple energy transformation in multigroup form, the Green's Function integration can be reduced to a matrix transformation on a source vector. The integration over energy is the summation of the product of the matrix row elements with the source column vector elements. Another alternative, and the one chosen in this example implementation, is to use a Dirac delta function in Equation (5) for a discrete source energy with multigroup energy output. This is discussed further in the implementation section.

The final topic for Green's Functions is parameterization to make them as applicable to as wide a problem space as possible. The parameterization can be thought of as a library of Green's Functions on which to interpolate. For shielding calculations, the primary parameters of interest are atomic number and thickness (or areal density) of the shield. By pre-computing a library of Green's Functions over a variety of atomic numbers and thicknesses for homogeneous shields, this allows us to later interpolate and weight them for heterogeneous shields. Additional parameterization is discussed later.

2.1.1. A Note on Secondary Photon Production

Photons can produce fluorescent x-rays, annihilation photons, delta-rays, and gammas from photonuclear reactions in shields, depending on the source energy ranges considered for the application. Care must be taken to recognize which effects are modeled by the Green's Function, and which are considered separate source terms against which the Green's Function is applied to avoid either missing or double-counting a physical effect. In addition, some multi-group photon cross-section sets for deterministic transport include some of these secondary production terms in the differential scattering cross-section, which may or may not be desirable.

2.2. Weighting for Heterogenous Shields

There are two types of heterogeneity to consider. First, the Green's Function library may be single atomic numbers, in which case a single shield layer with more than one element needs to be weighted. The second is the type already discussed, where multiple distinct layers with different materials are modeled. Weighting functions allow an average atomic number and areal density to be used as the interpolants over the homogeneous Green's Function set to replicate either or both heterogeneous shield types.

There are two alternatives to creating a weighting function. First, the homogeneous Green's Functions may be applied sequentially in order of the shielding layers. Another integral over energy is added to Equation (5) for each layer, which requires additional computation time. This is still much faster than solving Equation (4) and may be appropriate for some applications. For precise shielding applications, care must be taken to account for the angular distribution of the photon flux between layers as well. The weighting function methodologies covered in this report also suffer the

same error from disregarding the angular distribution between shields. A second alternative is to parameterize the Green's Functions over the different shielding orders expected and for specific material types. Some applications may have a limited set of material types and ordering, in which case this may be feasible. The application driving this report required a very computationally efficient approach that the two alternatives could not satisfy.

It is helpful to rephrase the meaning of homogeneous Green's Functions as the probability, p, of observing scattered radiation of energy E exiting the shield in each energy group, given that a photon of source energy E_0 is incident on the interior of the shield as a function of atomic number, Z, and areal density, t,

$$p(E,E_0,Z,t). \tag{6}$$

The simplest example of a heterogeneous shield is two layers, shown in Figure 1. This is a useful starting model to discuss the effect of the various weighting schemes.

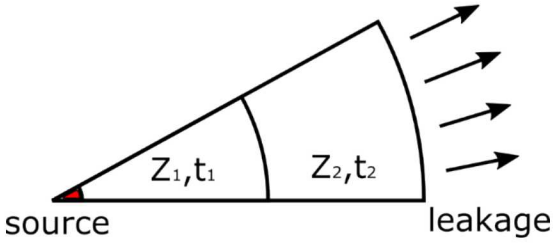

Figure 1. Two Layer Heterogeneous Shield

The Green's Function set is parameterized over atomic numbers, Z, and areal densities, t. The weighting scheme must transform N layers of atomic numbers, with possibly more than one atomic number per layer, and areal densities into a single atomic number, $\bar{Z}$, and areal density, $\bar{t}$.

2.2.1. Areal Density Weighting

The simplest approach to the computation of the weighting functions is to weight all atomic numbers by their respective areal densities and mass fractions and to sum the areal densities together, as described by the following equations:

$$\bar{Z} = \left(\sum_{i=1}^{N} t_i\right)^{-1} \sum_{i=1}^{N} t_i \sum_{j=1}^{M_i} Z_{i,j} w_{i,j} \tag{7}$$

$$\bar{t} = \sum_{i=1}^{N} t_i \tag{8}$$

where M_i is the number of elements in the i^{th} shield layer, $Z_{i,j}$ is the atomic number of the j^{th} element in the i^{th} layer, and $w_{i,j}$ is the mass fraction of the j^{th} element in the i^{th} layer's material.

The merit of this approach is that it is very fast and source-energy-independent. Thicker or denser shielding layers are weighted most heavily, and the total areal density is preserved. However, this

approach applies linear weighting to the areal density whereas the importance of each layer is closer to an exponential function. Furthermore, if the source is volumetrically distributed within one of the shielding layers, it does not properly weight the importance of different source locations with varying shielding characteristics.

2.2.2. Areal Density and Leakage Weighting

If a source is distributed within a shield or a radioactive material has substantial self-shielding (e.g., uranium metal), then the atomic number and areal density most consistent with the scatter radiation will be from the less-attenuated points in the source, such as those that are closer to the exterior surface of the shielding material. Accounting for this effect requires weighting by the transmission probability; thus, there will be different average atomic numbers and areal densities for each source energy.

If there are Q number of source voxels or points in the shield, then Equations (7) and (8) can be modified to account for their respective leakage importance. Let l_k be the leakage rate (s^{-1}) from the k^{th} source point,

$$l_k = q_k \prod_{i=1}^{N} e^{-\frac{\mu_{t,i}}{\rho_i} t_i} \tag{9}$$

where q_k is the source rate at the k^{th} source point (s^{-1}), $\mu_{t,i}/\rho_i$ is the total attenuation coefficient for the i^{th} shield layer, and t_i is the areal density of the i^{th} layer (unless it is the same layer as the source layer, in which case it is the areal density from the source point to the exterior of the source layer). The weighted atomic number and areal density then become,

$$\bar{Z} = \left(\sum_{k=1}^{Q} l_k \right)^{-1} \sum_{k=1}^{Q} \left[l_k \left(\sum_{i=1}^{N} t_i \right)^{-1} \sum_{i=1}^{N} t_i \sum_{j=1}^{M_i} Z_{i,j} w_{i,j} \right] \tag{10}$$

$$\bar{t} = \left(\sum_{k=1}^{Q} l_k \right)^{-1} \sum_{k=1}^{Q} \left[l_k \sum_{i=1}^{N} t_i \right] \tag{11}$$

Equations (10) and (11) have similar forms for an analytical source expression rather than a source mesh or voxels, with the summation over Q voxels replaced with an integration over the source volume. In some cases, integration may be performed over a variety of path lengths. For example, if the total leakage into 4π from a spherical source is computed, the path lengths can be discretized over all azimuthal and polar angles from a point in the model. For brevity, this additional integral is ignored in this formulation.

In addition to uniformly distributed sources, Equations (10) and (11) allow for unevenly distributed sources within the shield, such as neutron-induced secondary photons. It can also be used to account for the same source energy in different shields or layers.

This approach is quite accurate for most applications and is strongly recommended over the simple areal density weighting approach. However, notice that the transmission weighting is the product of all transmission probabilities from the source layer to the exterior of the shield. Thus, the weight is

the same for non-source layers even if their order is changed. It fails to account for the effect of reversing the order of a heterogeneous shield composed of a low-atomic number and a high-atomic number.

2.2.3. Escape Probability Weighting

Equations (10) and (11) can be improved by accounting for the increases in probability that the scattered photons escape from external shells. When scattering Green's Functions are normalized to the uncollided transmission probability, they describe the probability of observing a scattered photon per escaping uncollided photon. Thus, the absolute transmission or attenuation of the source by the shield does not make a difference. As the shielding becomes very thick, the scatter radiation normalized to the uncollided fraction becomes invariant. This is a similar concept to the infinite-thickness self-shielding concept used for radioactive materials such as uranium metal.

The largest effect of the atomic number of the shield, when normalized to uncollided leakage, is the shape of the scatter continuum. Lower atomic numbers have a longer low energy tail while higher atomic numbers will attenuate this via the photoelectric effect. The largest effect of the areal density is the magnitude of the continuum relative to the uncollided leakage, which asymptotically approaches some infinite scatter thickness. Average atomic numbers for heterogeneous shields are evaluated to give the best representation for a particular configuration and used to determine areal densities that are applicable to each shell.

The average atomic number that determines the continuum shape most consistent with the actual shielding arrangement will be closest to the outermost layers of the shield, provided the scatter radiation exiting that layer can escape through all outer layers. For simplicity, the assumption is made that this escape probability is roughly consistent with the original source photon energy. As the shielding becomes more heterogeneous and the uncollided component is reduced, this assumption becomes worse.

Let N be the number of layers in the shield, and $l_i(cm)$ be the upper-bound of the i^{th} layer such that the thickness of it is $(l_i - l_{i-1})$. The layer thicknesses can correspond to a physical shield for a point source at the center of a sphere. More generally, they are the layers of various materials traversed by a ray from the source point directed toward a detection point.

Let x be the dimension along the path of the ray. The average atomic number is a continuous weighting function of the areal density and escape probability along that path,

$$\bar{Z} = \left(\int_0^{l_N} dx\, \rho\, e^{-\mu(l_N - x)} \right)^{-1} \int_0^{l_N} dx\, \rho\, Z\, e^{-\mu(l_N - x)} \tag{12}$$

where ρ is the density as a function of position $(g \cdot cm^{-3})$, and μ is the total cross-section (cm^{-1}). This integral must be broken into a summation over segments or layers in which the density and cross-section are constant. The exponential term can then be separated into a product of the attenuation in the i^{th} shell and the attenuation in all outer shells. For brevity, let T_i be the transmission in the i^{th} shell,

$$T_i = e^{-\mu_i(l_i - l_{i-1})} \tag{13}$$

where μ_i is the total cross-section for the i^{th} layer (cm^{-1}). Let T_i^+ be the transmission through all shells outside of the i^{th} shell,

$$T_i^+ = \prod_{j=i+1}^{N} e^{-\mu_j(l_j - l_{j-1})}. \tag{14}$$

Including leakage-weighting from earlier, the average atomic number from Equation (14) evaluates to

$$\bar{Z} = \left(\sum_{k=1}^{Q} l_k\right)^{-1}\left(\sum_{i=1}^{N}\frac{\rho_i}{\mu_i}(1-T_i)T_i^+\right)^{-1}\sum_{k=1}^{Q}\left[l_k\sum_{i=1}^{N}\frac{\rho_i Z_i}{\mu_i}(1-T_i)T_i^+\right]. \tag{15}$$

The total areal density is computed by only weighting the portions of the shield along the path that are similar to the average atomic number. The areal density of a differential path dx is reduced by accounting for interactions that remove the unattenuated photon beam from the average atomic number's interactions. The probabilities considered at $x+dx$ along the path are (1) interacting before x, (2) interacting between x and $x+dx$ and being absorbed, and (3) interacting between x and $x+dx$ and forward-scattering, and (4) interacting between x and $x+dx$, back-scattering, and being attenuated by internal materials. These probabilities translate into the four terms in the square brackets in Equation (16):

$$\bar{t} = \int_0^{l_N} dx\,\rho\frac{\mu_s/\mu}{\bar{\mu}_s/\bar{\mu}}\left[\left(1-e^{-\mu x}\right) + e^{-\mu x}\left(1-\frac{\mu_s}{\mu}\right) + e^{-\mu x}\left(\frac{\mu_s}{\mu}\right)(1-B) + e^{-\mu x}\left(\frac{\mu_s}{\mu}\right)B\,T_d^-\right], \tag{16}$$

where μ_s is the scatter cross-section (cm^{-1}), B is the back-scatter probability, and T_d^- is the differential internal transmission given by

$$T_d^- = e^{-\mu_d x} \tag{17}$$

where μ_d is the difference between the actual total cross section and the total cross-section for the average atomic number, bounded to a minimum of zero. In addition, the areal density is adjusted by the ratio of the actual material's cross-sections (μ_s/μ) to the average atomic number's cross-sections ($\bar{\mu}_s/\bar{\mu}$). The differential attenuation attempts to account for strong absorbers that are interior to a given spatial location. The backscatter probability, B, is the probability that the photon will backscatter (i.e., > 90 degrees deflection from original path). The backscatter probability is an integral over the differential Klein-Nishina differential scattering cross-section from 90 to 180 degrees, normalized to the total scatter cross-section.

Equation (16) evaluates to

$$\bar{t} = \sum_{i=1}^{N}\rho_i\frac{\mu_{s,i}/\mu_i}{\bar{\mu}_{s,i}/\bar{\mu}_i}\left[(l_i - l_{i-1}) - \frac{\mu_{s,i}}{\mu_i}B\left(\frac{1}{\mu_i}\left[e^{-\mu_i l_{i-1}} - e^{-\mu_i l_i}\right]T_i^- - \frac{1}{\mu_i+\mu_{d,i}}\left[e^{-(\mu_i+\mu_{d,i})l_{i-1}} - \epsilon\right.\right.\right. \tag{18}$$

where $\bar{T}_i$ is the product of all transmission from shells internal to the i^{th} shell,

$$\bar{T}_i = \prod_{j=1}^{i-1} e^{-\mu_j (l_j - l_{j-1})}, \tag{19}$$

and $\bar{T}_{d,i}$ is the product of all differential transmission from shells internal to the i^{th} shell,

$$\bar{T}_{d,i} = \prod_{j=1}^{i-1} e^{-\mu_{d,j} (l_j - l_{j-1})}, \tag{20}$$

2.2.4. Hydrogen Weighting

The incoherent scatter cross-section varies smoothly between different atomic numbers, except for hydrogen, which has twice the scattering cross-section per unit mass of any other element, as shown in Figure 2. For applications that utilize hydrogen-rich shielding materials, such as plastics, the weighting functions for atomic number will not be sufficient as an interpolant on a parameterized set of Green's Functions to capture the degree of scattering these materials produce.

Accounting for this requires that the Green's Functions also be parameterized with respect to hydrogen fraction in the material, and the average hydrogen weight fraction is weighted in a similar way, but separately, to the atomic number. The atomic numbers in Equation (15) are simply replaced with the hydrogen mass-fractions in the materials.

The atomic number weighting in Equation (15) can include hydrogen, or not, depending on how the Green's Functions are parameterized, and whether the average atomic number in the Green's Function set includes hydrogen.

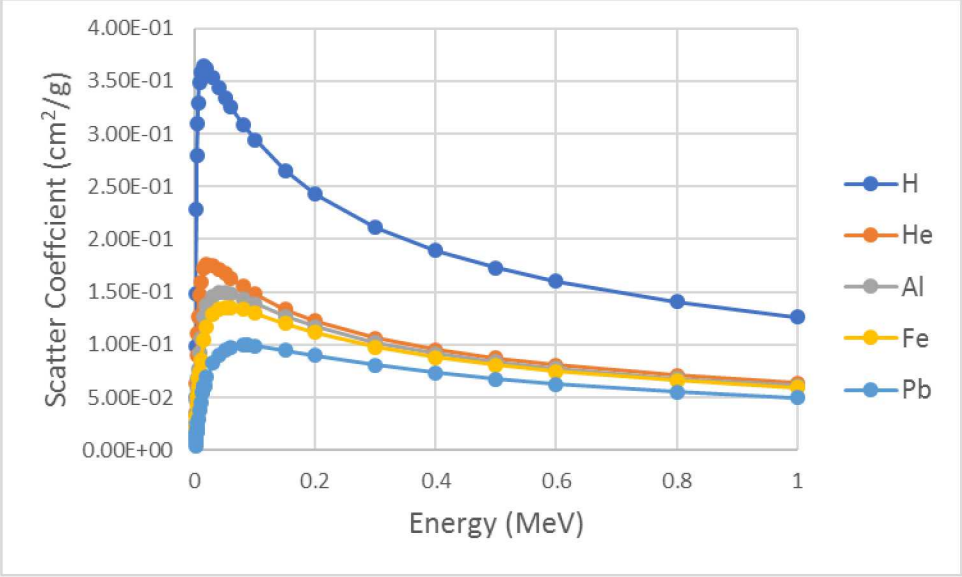

Figure 2. Scatter Attenuation Coefficients for Different Atomic Numbers

2.2.5. Shield Shape

For a spherical shielding library, in addition to the areal density, the inner and outer radius of the shield also has a noticeable effect on the overall intensity of the scatter spectrum. A point source in the center of a solid sphere has more scatter than a point source with a gap between it and a shield of the same radial thickness. This can be accounted for by generating multiple Green's Functions for different inner and outer radii with the same radial thickness and interpolating between them. A convenient metric for this shape is the ratio of the inner radius to the outer radius, which is zero for a solid sphere and unity as the inner radius approaches infinity.

For a homogeneous shield, the shape factor is trivial to calculate and understand, as there is only one shielding layer. However, because we are collapsing a heterogeneous shield into a single layer with an average atomic number and areal density, computation of the effective inner and outer radii must account for different layers. Conceptually, the objective is to capture the inner and outer radius of the bulk of the material as it contributes to the total areal density. For example, a low-density layer on the inner side of the shield should not influence the shape factor significantly because it does not influence the leakage of scattered photons much.

The effective outer radius can be computed by weighting sequential mean-free-paths for the actual shielding material versus a solid sphere with the average material properties, as given below:

$$\bar{r}_o = r_o - \left(\sum_{i=1}^{\infty} e^{-(i-1)} \right)^{-1} \sum_{i=1}^{\infty} e^{-(i-1)} (r^{-}(i\lambda) - i\bar{\lambda}) \tag{21}$$

where i is the mean free path index, r_o is the physical outer radius of the actual shield or the end of shielding material encountered by the path or ray, and $r^{-}(i\lambda)$ is the actual radius in the shield that achieves i mean free paths of the original source energy, moving inward from the exterior of the shield to the source point. For distributed sources, the radii are averaged over all source voxels, each of which may have different paths through the shielding materials.

The effective inner radius is computed by

$$\bar{r}_i = r_o - \frac{\bar{t}}{\bar{\rho}} \tag{22}$$

where $\bar{\rho}$ is the average density of the shielding material calculated by using the same areal density weighting from Equation (19).

Computationally, differences in the weights become exceedingly small beyond a few mean free paths, so the infinite summation can be capped at seven mean free paths or less (infinite thickness). One disadvantage of this weighting scheme for the inner and outer radii is that this formulation can result in non-physical radii (e.g. negative inner radius), so they should be bounded to the physical size of the actual shield. Furthermore, some shields do not contain enough material to represent one mean free path. It is unclear that the shape factor even matters for such thin shielding materials; nevertheless, Equation (21) can be modified by reducing the mean-free paths (both average and actual) such that the average shield is exactly the number of scaled mean-free paths used in the summation.

3. EXAMPLE IMPLEMENTATION

For this implementation the PARTISN [2] transport code was selected to generate the Green's Function because of its speed for one-dimensional calculations against a fairly large number of transport simulations. This should not be misinterpreted as a recommendation, however. With available super-computers, the entire premise of Green's Functions is to push complicated physics simulations up-front, and the appropriate simulation package or methodology should be chosen for the physics desired in the application.

The cross-section set used is a 150 group, 3rd order Legendre scatter set based on a version of GAMLEGJR [3] modified with the Biggs-Lighthill [4][5][6] analytic cross-section formulas for the photoelectric effect. The SN quadrature order used is 8, the spatial mesh throughout the shield is uniform and the width of one mesh interval is approximately one mean-free-path of the source photon energy in that material.

Three sets of one-dimensional spherical input files were generated for PARTISN with three layers:

(1) Void source layer with radial thickness 1 mm to simulate a point source volume

(2) Void intermediate layer with variable outer radius (for different shape factors)

(3) Variable shield type and thickness

The intermediate void layer is varied between 0 cm, 10 cm, and 200 cm to span a range of shielding shapes among which we can interpolate. The three void layer thicknesses are the three aforementioned sets. The 1-mm-radius source volume in the center of the model has a uniformly distributed source with one of the sixteen discrete energies listed in Table 1. The source energies were chosen to effectively capture the slope in the emitted scatter radiation as a function of source energy, and to bound the K-shell discontinuities from the six atomic numbers chosen in Table 2. These six elements were chosen as representative of the most common shielding materials encountered for the application. The jump from iron to lead is large; however, intermediate elements like tin are rarely encountered in significant quantities relative to other more common shielding materials like iron.

The varying areal densities of the shield layer are shown in Table 3, which are simply powers of two to capture the slope of the exponential behavior of scatter radiation emitted from a shield as a function of thickness. For lead at a density of 11.35 g/cm^3, the range covers thicknesses between 800 μm and 22.6 cm. For areal densities below 1.0 g/cm^2, the interpolation is linear between the scatter field emitted at 1.0 g/cm^2 and a zero scatter field.

Lastly, the amount of hydrogenous material is varied as well, shown in Table 4. At zero hydrogen amounts, the materials are the pure elements shown in Table 2. A hydrogen mass fraction of 0.0279 corresponds to a generic high-explosive CHON. A mass fraction of 0.144 applies to polyethylene (or any plastic with a hydrogen-to-carbon ratio of 2). This represents the maximum hydrogen content we can expect for common shielding applications.

PARTISN is run three times for each model. First, no secondary photons in the form of x-ray fluorescence or annihilation are considered and a parameterized Green's Function set is created from the leakage PARTISN calculates. Second, the photon flux, as a function of radius, from the first PARTISN solution is integrated against the pair-production cross-section from Biggs-Lighthill to create a distributed source term within the shield. PARTISN is used to solve this second problem to get the leakage scatter from annihilation photons. A one-dimensional ray-tracer is used to get the

discrete annihilation line intensity. The third run again uses the photon flux from the first, this time integrating against the MCNP EPDL97 [7] x-ray fluorescence cross-section set to get the leakage scatter from the x-rays. The ray tracer is used again to get the discrete x-ray line intensities. The two additional runs to compute the secondary photon scatter are only done for the model set with the 10 cm void.

The output leakage from the first PARTISN run is post-processed to remove the uncollided contribution in the group containing the discrete source energy and regrouped, using discontinuous linear weighting, into a 31 group structure that is simply 31 evenly spaced energies from 0 to the discrete source energy. This allows a group structure to be created for any source energy interpolant easily, and its values are the interpolation between the same indices in the different group structures of two different source energies.

Similarly, the annihilation scatter leakage is processed to remove the uncollided annihilation peak and regrouped into 31 evenly spaced groups from 0 to 511 keV. The x-ray scatter leakage is processed to remove the uncollided x-ray peaks and regrouped into ten evenly spaced groups from 0 to the maximum fluorescent x-ray energy for that material. However, the annihilation peak intensities and x-ray peak intensities are stored and preserved for use in shielding applications that desire secondary particle production.

For source energies, atomic numbers, areal densities, and hydrogen fractions that lie in the range of computed values shown in Table 1 through Table 4, a group-to-group logarithmic interpolation is utilized. For extrapolations, the group intensities are first scaled by the inverse uncollided transmission probability for each respective source energy through the shield first, and a group-to-group linear interpolation is utilized. Finally, the groups are scaled by the actual source energy transmission.

Extra care must be taken when interpolating between the atomic numbers when high-atomic-number shields are present, especially when photoelectric absorption is a dominant effect in the material. For example, if a heterogeneous shield has mixed polyethylene and uranium (atomic numbers of approximately 5.28 and 92, respectively), but the average atomic number is less than lead (atomic number 82), it is probably more appropriate to interpolate between iron (atomic number 26) and uranium to get some effect of the k-edge discontinuity in the uranium material. If the interpolation is between iron and lead, the two closets interpolants one would naturally choose, the lead k-edge discontinuity effects would appear instead. Thus, it is advised that in any practical implementation to keep track of what materials are actually present in the model to constrain the atomic number interpolants.

The mesh, scatter leakages, and peak intensities are stored in a binary file named "sandia.shieldscatter.db", which is available from the authors upon request. The format of this file is discussed in Appendix A. A Python 2.7 source code that parses this database is provided in Appendix B. The mesh points are listed in Table 1 through Table 4.

Table 1. Source Energies (keV).

60	87	89	115	116	121	122	200	300	400	600	1000	1600	2600	4000	9000

Table 2. Atomic Numbers (and Elements)

6	13	26	82	92	94
C	Al	Fe	Pb	U	Pu

Table 3. Areal Densities (g/cm^2)

1	2	4	8	16	32	64	128	256

Table 4. Hydrogen Mass Fractions

0	0.279	0.1444

4. RESULTS

Measurements of various sources using various gamma radiation detectors were collected. The source and detector configurations are described in the following sections. The detector response models (e.g. photopeak intensity, partial energy absorption continuum) were created in GADRAS [8]. The following sections demonstrate the effect of shielding on the computed spectrum and how it compares to the measurement.

4.1. Point Sources in Homogeneous Shielding

A 95% efficient high-purity germanium detector (HPGe) was placed at 100 cm from the center of various calibration sources (Cs-137, Y-88, U-232) in nylon, iron, and tungsten shields. This provides a wide range of energies and shielding atomic numbers. The results using the shield scatter library interpolation versus the scattering output directly from PARTISN (i.e. no library interpolation) is compared to evaluate the accuracy of this method.

The first shield is a 12-cm-thick nylon sphere with inner radius of 3 cm. The density of the nylon is 1.157 g/cc. This relatively thick low-atomic-number material provides plenty of scatter continuum. Figure 3 through Figure 5 illustrate the difference between using the scatter continuum from PARTISN versus the pre-computed scatter table lookup and how both compare to the measurement. The agreement between them is very good. There is some artificial tailing in the PARTISN-computed calculations around the photopeaks in some of the plots due to errors in combining the discrete and group components in the spectrum. The important component to consider is the scatter continuum below the photopeaks.

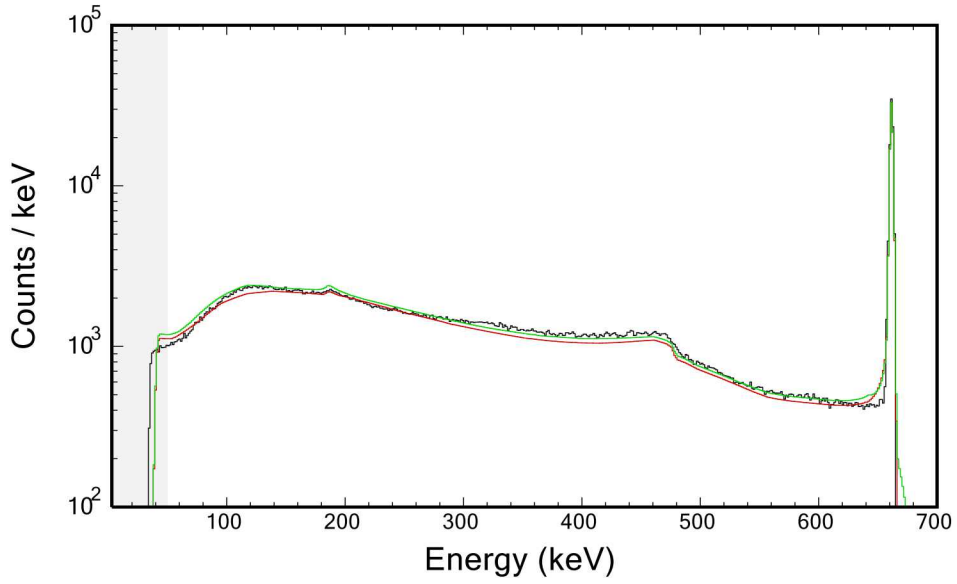

Figure 3. Cs-137 in 12-cm-thick Nylon Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

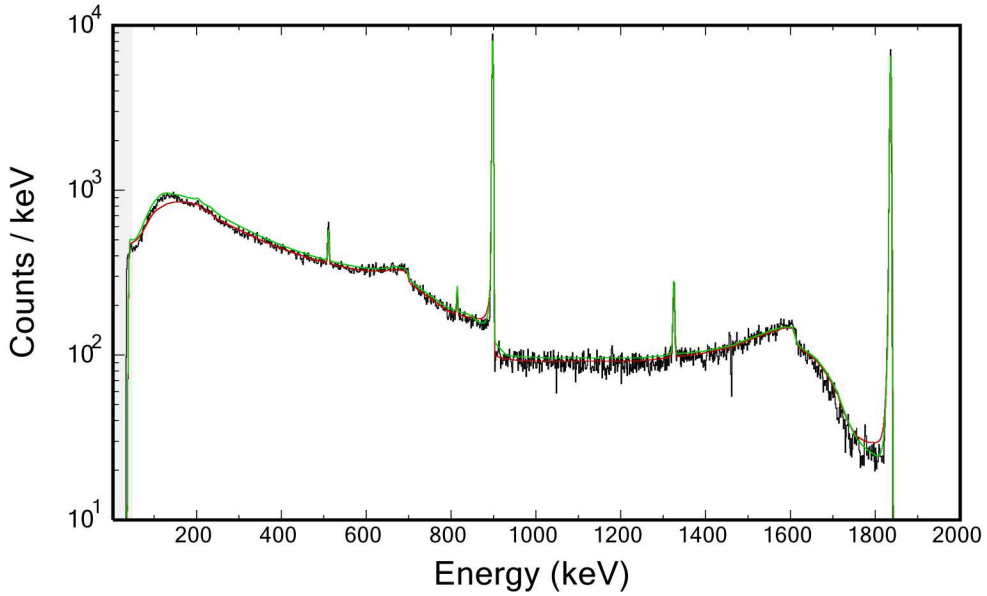

Figure 4. Y-88 in 12-cm-thick Nylon Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

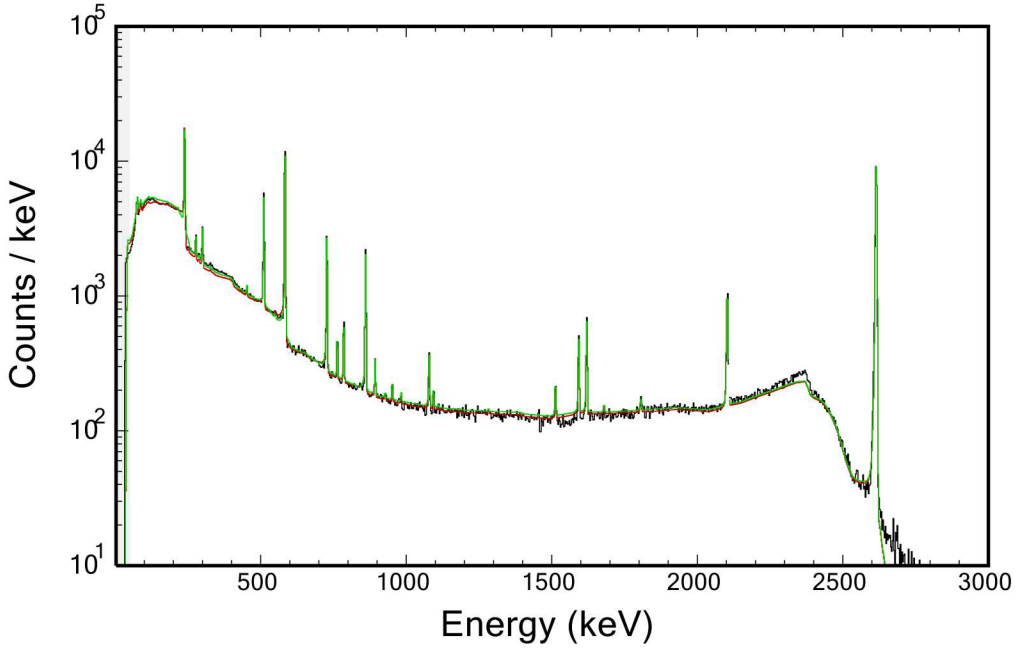

Figure 5. U-232 in 12-cm-thick Nylon Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

The second shield is a 4-cm-thick iron shell with inner radius of 3 cm. The density of the iron is 7.66 g/cc. This has a higher atomic number and areal density than the nylon shield. Figure 6 through Figure 8 compare the shield scatter library interpolation to the PARTISN output. The agreement is again very good.

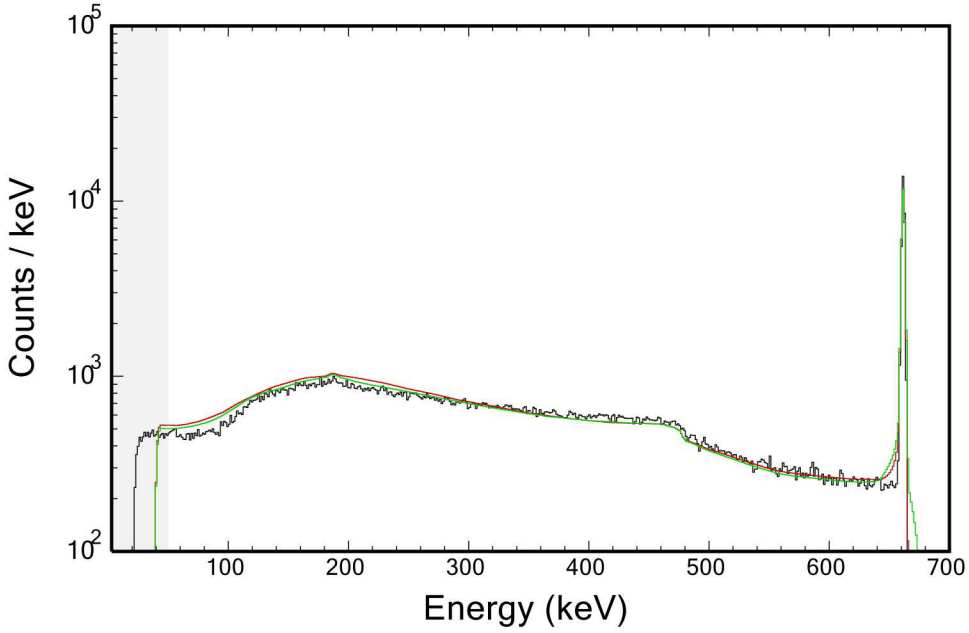

Figure 6. Cs-137 in 4-cm-thick Iron Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

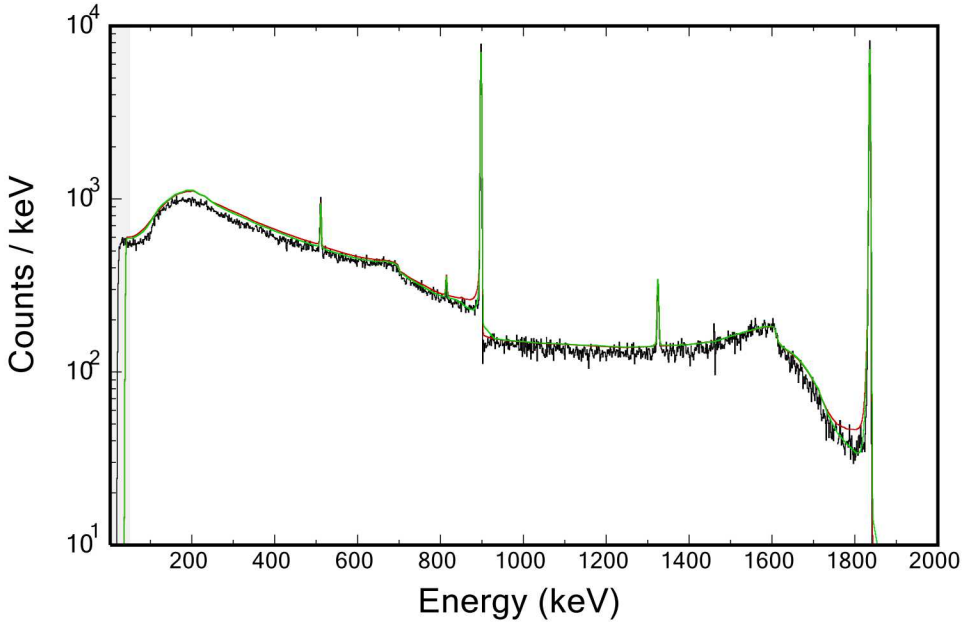

Figure 7. Y-88 in 4-cm-thick Iron Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

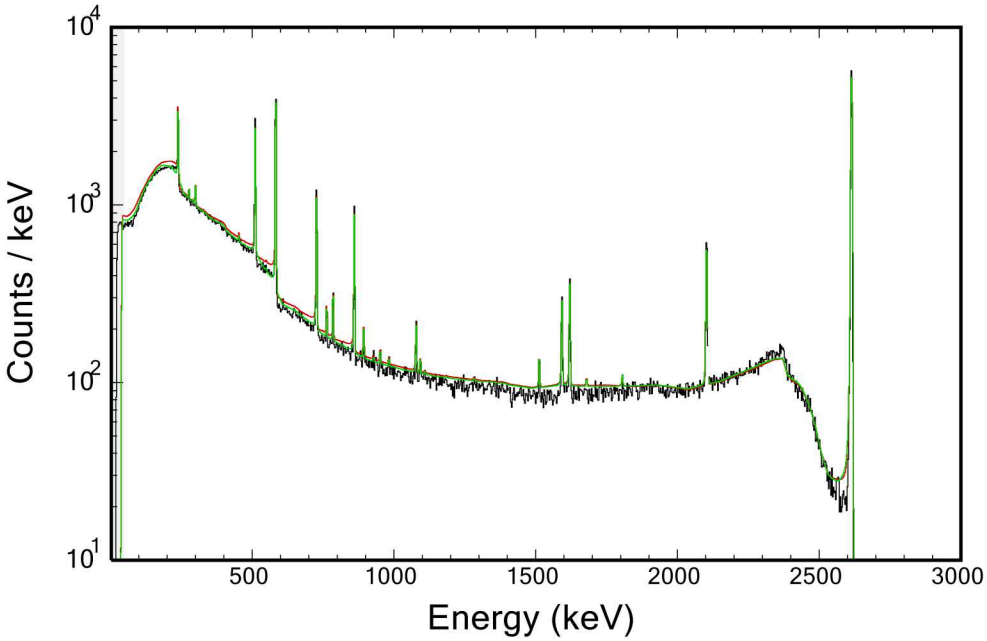

Figure 8. U-232 in 4-cm-thick Iron Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

The third shield is a 1-cm-thick tungsten shell with inner radius of 3 cm. The density of the tungsten is 16.65 g/cc. This has a similar areal density to the nylon shield, but with much higher atomic number and produces much less scattering. Figure 9 through Figure 11 again compare the shield scatter library interpolation to the PARTISN output. The agreement is again very good, except in Figure 9 for Cs-137 where the low-angle-scattering for PARTISN is erroneously too high compared to the measurement.

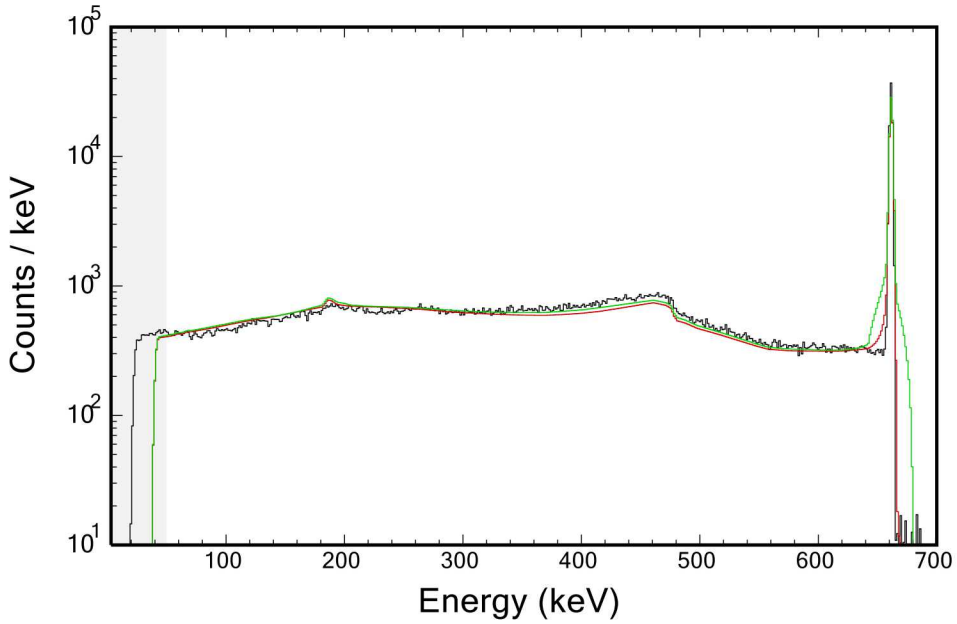

Figure 9. Cs-137 in 1-cm-thick Tungsten Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

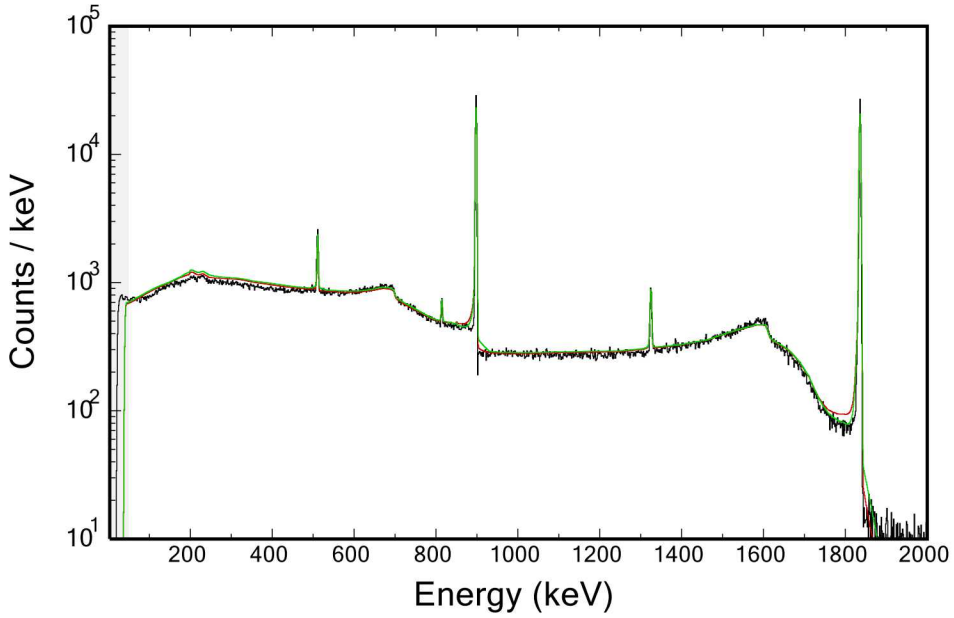

Figure 10. Y-88 in 1-cm-thick Tungsten Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

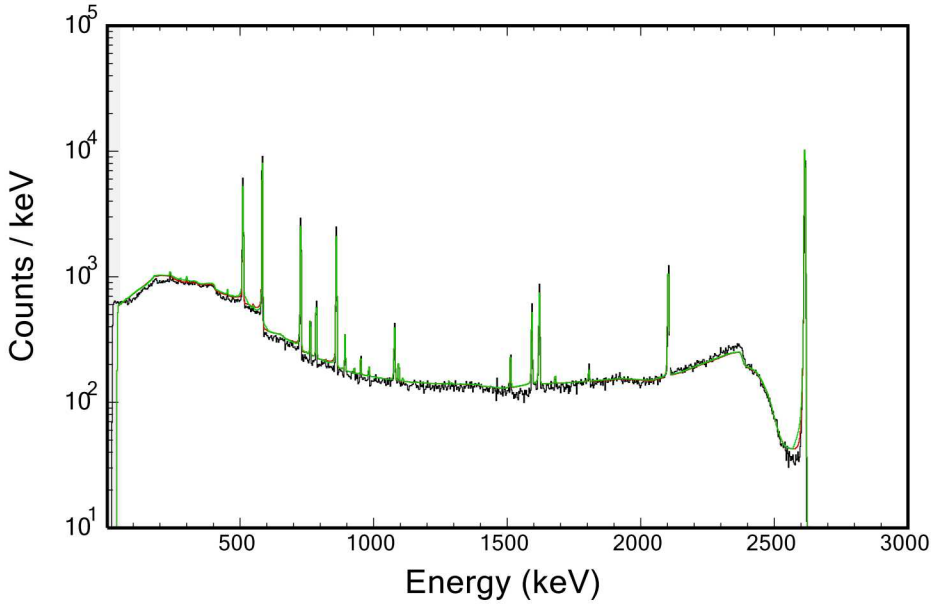

Figure 11. U-232 in 1-cm-thick Tungsten Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

4.2. Distributed Sources in Homogenous and Heterogeneous Shielding

A useful object in this regard is the beryllium reflected plutonium (BeRP) ball. The details of this object can be found in reference [9]; in summary it is a 4.5 kilogram sphere of alpha-phase, weapons-grade plutonium, surrounded by a 0.03 cm steel cladding. There is no beryllium around the BeRP ball any longer. Many measurements of the BeRP ball in various shielding using various radiation detectors have been recorded over the years. We will focus on the bare case with no additional shielding and a composite shield of iron and polyethylene. Both shielding configurations were performed with a large HPGe detector (roughly 140% relative efficiency).

The bare case is useful because even without external shielding, the self-shielding effect from the plutonium is an important aspect of modeling. Figure 12 compares the measurement to the shield-scatter interpolation and to the PARTISN scatter estimate. Here the spectrum's upper energy range is reduced to 500 keV to highlight the low-energy region with the most difference. The shield-scatter library is consistently lower than the PARTISN output below 250 keV by approximately 5%. However, this puts it closer to the measurement and it's not clear, due to compounding errors from the detector response function, which is more accurate. Nevertheless, the match between the two methods is very good.

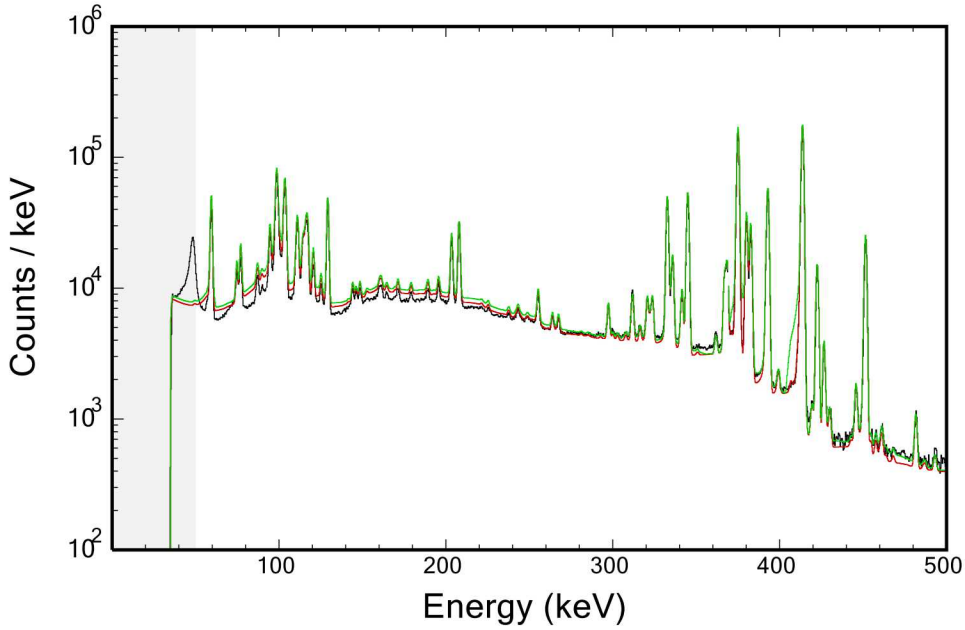

Figure 12. Bare BeRP Ball, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

The two-layer shield has 1 inch of iron followed by 2 inches of polyethylene around the BeRP ball. This heterogeneous configuration is a good test of the weighting scheme developed in this report to determine effective atomic numbers, areal densities, hydrogen fractions, and shape factors. The comparison between the shield-scatter library, the measurement, and PARTISN is shown in Figure 13. The agreement between all three is good across the entire energy range of the spectrum. The

green curve overlaps the red curve enough that they are indistinguishable. This gives confidence in the heterogenous weighting scheme.

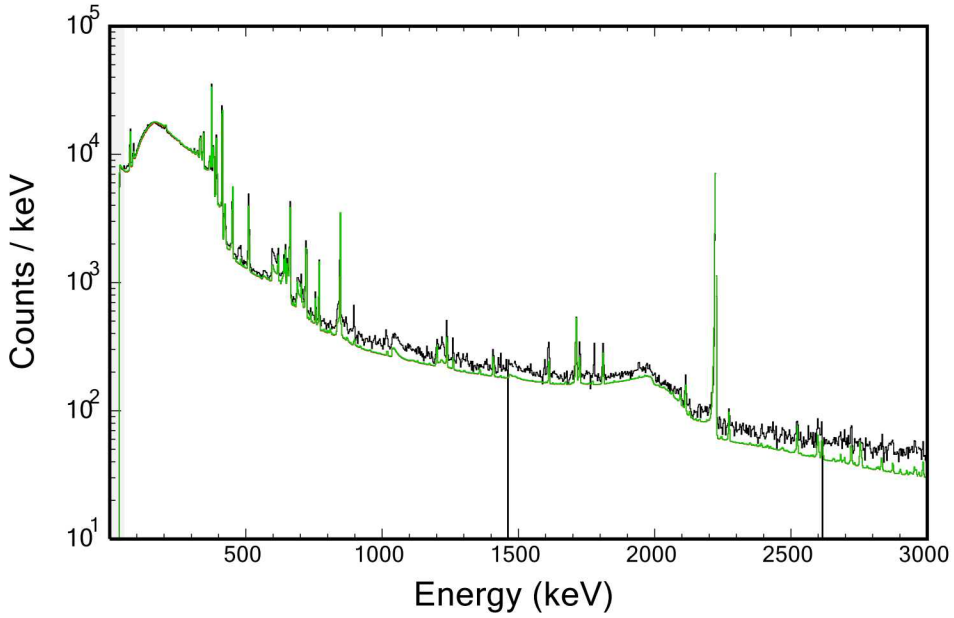

Figure 13. BeRP Ball in 1-inch Iron and 2-inches of Polyethylene Shield, Comparison of Measurement (black), PARTISN (green), and Shield-Scatter Library (red)

4.3. Computational Comparisons

Measurements are useful to establish which method is closer to the ground-truth. However, the detector response required to transform the photon leakage into a spectrum adds uncertainty. Computational comparisons do not have ground truth, but there is no detector response and any hypothetical shield design is possible. It is still helpful to understand where the two methods deviate, even if it's not clear which one is more accurate.

During the testing of this weighting methodology and shield-scatter library, it was found that the most difficult shielding cases for the weighting method were highly different atomic numbers in heterogeneous thick shields. For example, polyethylene and lead are common shielding materials, and their atomic numbers (roughly 5.3 and 82, respectively) are very different. Furthermore, source energies in the range of 200 keV to 600 keV have a high probability of interacting and producing scatter. Energies below 200 keV are dominated by photoelectric absorption in the lead and the scatter is not important. Energies above 600 keV begin to converge regardless of shield ordering, most likely because the photons become very penetrating and their first-interaction is more uniform throughout the shield. Thus, 200 keV and 600 keV are the two source energies discussed here as they are the most problematic for thick heterogeneous shields.

A 200 keV and 600 keV hypothetical discrete source was placed in the center of a 10-cm-thick void sphere, with various thicknesses of polyethylene and lead surrounding the void. Areal densities of 5

g/cm^2 and 20 g/cm^2 were used for both the polyethylene and lead, and their order reversed, making four different shielding combinations. The notation used in the plot legends below indicates the shield order and the areal density.

The PARTISN scatter output is compared to the scatter estimated with the shield-scatter library interpolation method and weighting scheme in Figure 14 through Figure 17. Overall, the agreement is reasonable given that it is not clear which method is more accurate. The two methods have the same trends when the shield order is reversed and the overall magnitudes are similar. However, for the 600 keV source in Figure 16 and Figure 17, there is a significant discontinuity in the scatter leakage around the lead k-edge. The average atomic number is not capturing that discontinuity effect as well as PARTISN does. That said, this effect is not observable in gamma radiation measurements as the detector signal is overwhelmed with partial-energy-absorption of the primary photon.

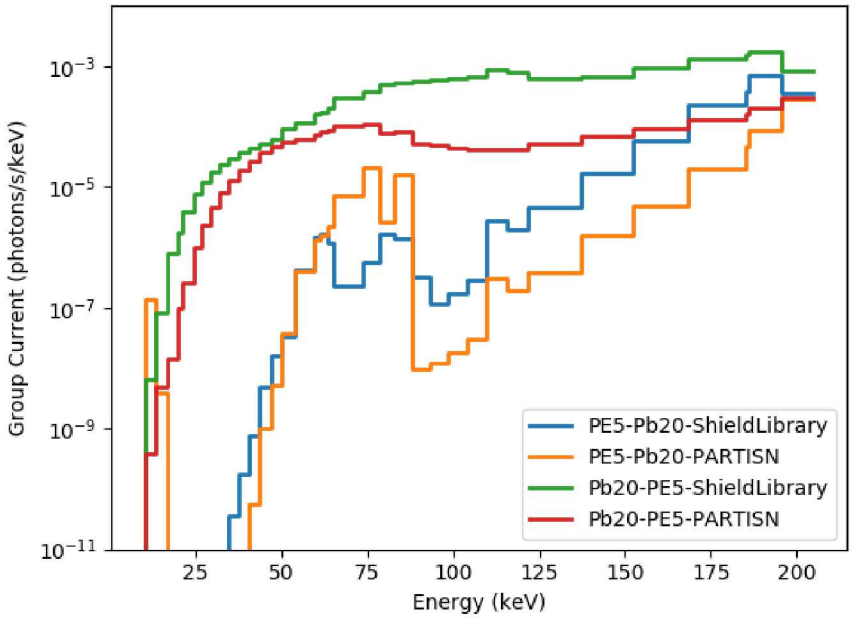

Figure 14. 200 keV Source in 5 g/cm^2 Polyethylene and 20 g/cm^2 Lead, Comparison of PARTISN and Shield-Scatter Library

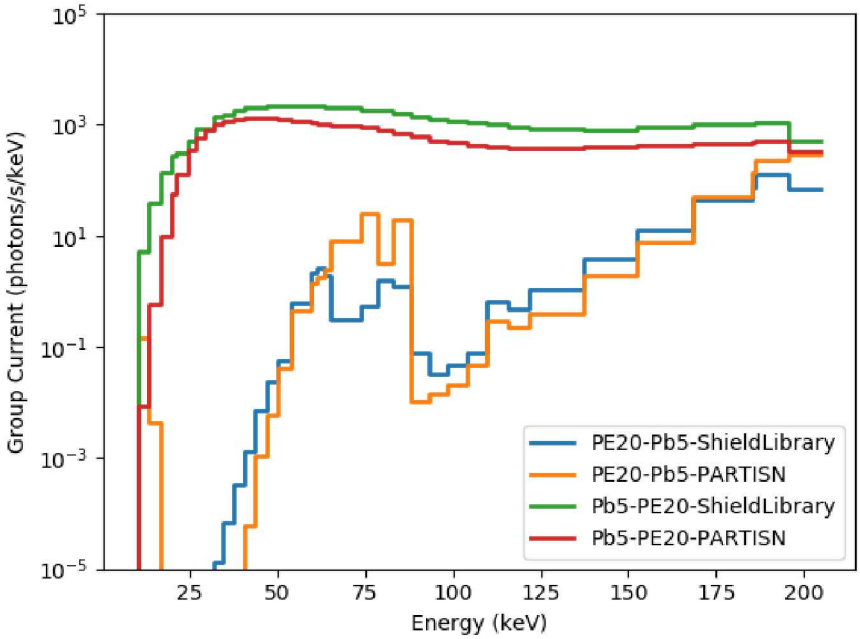

Figure 15. 200 keV Source in 5 g/cm² Lead and 20 g/cm² Polyethylene, Comparison of PARTISN and Shield-Scatter Library

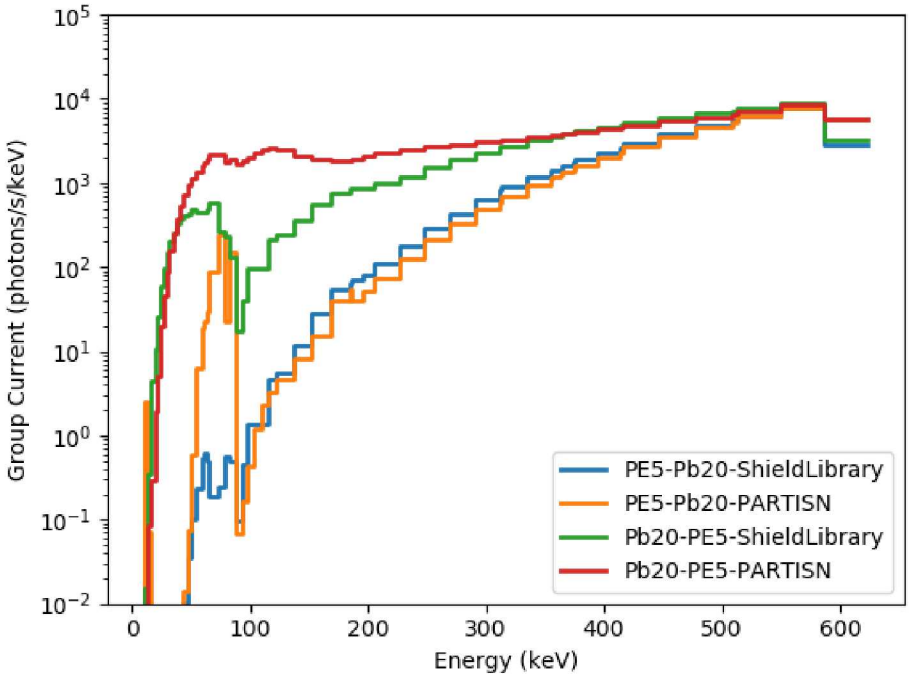

Figure 16. 600 keV Source in 5 g/cm² Polyethylene and 20 g/cm² Lead, Comparison of PARTISN and Shield-Scatter Library

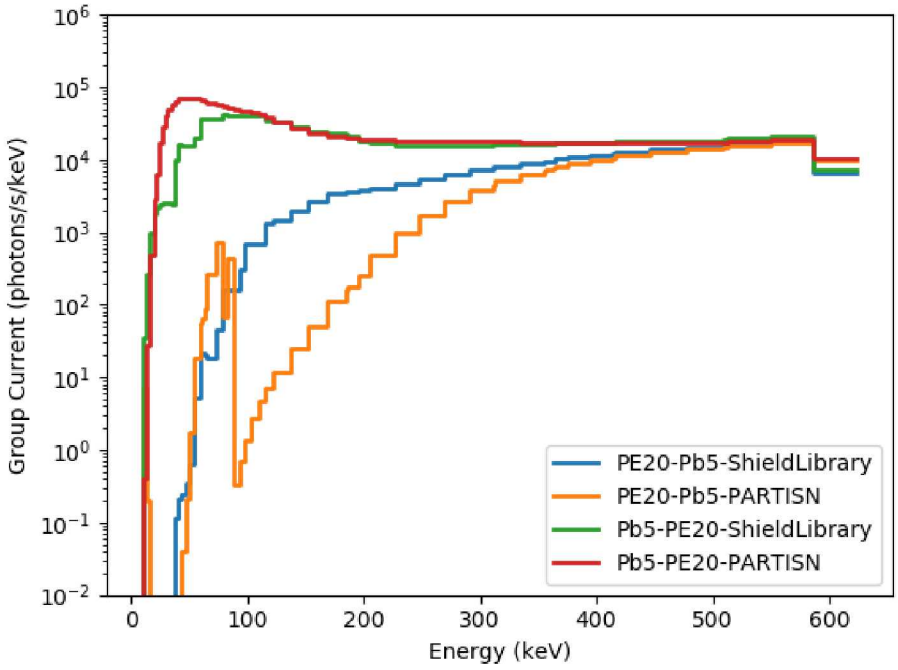

Figure 17. 600 keV Source in 5 g/cm² Lead and 20 g/cm² Polyethylene, Comparison of PARTISN and Shield-Scatter Library

4.4. Additional Comparisons

There is a separate report of additional comparisons that can be made available upon request to the authors of this report. The content of the report is not unlimited release, and the audience is restricted.

5. CONCLUSIONS

The weighting scheme developed in this report allows Green's Functions for homogeneous shields to be used for more complex heterogeneous shields. This approach requires a substantial initial investment in computation time to generate the Green's Functions as a function of various parameters such as atomic number and thickness. However, this only needs to be done once, and can be thoroughly checked for errors. The interpolation of these Green's Functions, based on the interpolants determined from the weighting scheme, to generate the estimated scatter from any applicable source and shield is computationally efficient and can be more reliable because of the thorough checking done on the library.

The shield-scatter library was compared to output from PARTISN for measured gamma-ray spectra. GADRAS was used for the detector response. Between various atomic numbers and thicknesses, the results between the two are mostly indistinguishable

The output from the shield-scatter library and PARTISN were also directly compared for hypothetical shields and sources for which no measurement exists. Thick shields of alternating order with very different atomic numbers were used to stress test the two methods. It's not clear which is producing more accurate results, but the two methods follow the same trends when the shield order is changed.

The primary benefit of this method is the computational efficiency. Thousands of gamma energies with their shielding interpolants can have individual scattering contributions estimated and summed much faster than a one-dimensional deterministic transport run in PARTISN. Although the shield-scatter library and comparisons were all done for one-dimensional spheres, with an appropriate weighting scheme for higher-order geometries this approach can also be applied to three-dimensional sources. The computational speed compared to a three-dimensional transport calculation is many orders of magnitude faster.

REFERENCES

[1] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org

[2] R. E. Alcouffe, R. S. Baker, J. A. Dahl, S.A. Turner, and Robert Ward, *PARTISN: A Time-Dependent, Parallel Neutral Particle Transport Code System*, Los Alamos National Laboratory, LA-UR-05-3925, May 2005.

[3] S. Miyasaka and K. Minami, *A Production Code of Multi-group Cross Sections and Energy Absorption Coefficients for Gamma Rays*, Japan Atomic Energy Research Institute, JAERI-M 6936, February 1977.

[4] F. Biggs and R. Lighthill, *Analytical Approximations for Total and Energy Absorption Cross Sections for Photon-Atom Scattering*, Sandia Laboratories, SC-RR-72 0685, December 1972.

[5] F. Biggs and R. Lighthill, *Analytical Approximations for X-Ray Cross Sections III*, Sandia National Laboratories, SAND87-0070, August 1988.

[6] F. Biggs and R. Lighthill, *Analytical Approximations for Total Pair-Production Cross Sections,* Sandia Laboratories, SC-RR-68-619,, September 1968.

[7] D.E. Cullen et. al., *Tables and Graphs of Photon Interaction Cross Sections from 10 eV to 100 GeV Derivted from the LLNL Evaluated Photon Data Library (EPDL)*, Lawrence Livermore National Laboratory, UCRL-50400, 1989.

[8] S.M. Horne et. al., *GADRAS Version 18 User's Manual*, Sandia National Laboratories, October 2018.

[9] J. Mattingly, *Polyethylene-Reflected Plutonium Metal Sphere: Subcritical Neutron and Gamma Measurements*, Sandia National Laboratories, SAND2009-5804, December 2009.

APPENDIX A. SHIELD LIBRARY FILE FORMAT

This describes the format of the binary file "sandia.shieldscatter.db".

Data Type	Meaning
4-byte signed integer	Number of scatter groups (M)
4-byte signed integer	Number of source energies (I)
4-byte floating point array	Source energies (I values)
4-byte signed integer	Number of atomic numbers (J)
4-byte floating point array	Atomic numbers (J values)
4-byte signed integer array	Number of x-rays for each atomic number (J values) (X(j) for all j in J)
4-byte floating point array	X-ray energies for each atomic number (loop over J, then X(j))
4-byte floating point array	K-shell energies for each atomic number (J values)
4-byte signed integer array	Indices (0-based) in source energies array corresponding to left edge of K-shell (J values)
4-byte signed integer	Number of areal densities (K)
4-byte floating point array	Areal densities (K values)
4-byte signed integer	Number of hydrogen mass fractions (L)
4-byte floating point array	Logarithm of scatter intensities for 0 cm intermediate void, no secondary scatter (L x K x J x I x M values, in that loop order)
4-byte floating point array	Logarithm of scatter intensities for 0 cm intermediate void, with secondary scatter (L x K x J x I x M values)
4-byte floating point array	Logarithm of scatter intensities for 10 cm intermediate void, no secondary scatter (L x K x J x I x M values)
4-byte floating point array	Logarithm of scatter intensities for 10 cm intermediate void, with secondary scatter (L x K x J x I x M values)
4-byte floating point array	Logarithm of scatter intensities for 200 cm intermediate void, no secondary scatter (L x K x J x I x M values)
4-byte floating point array	Logarithm of scatter intensities for 200 cm intermediate void, with secondary scatter (L x K x J x I x M values)
4-byte floating point array	Logarithm of 511 keV annihilation photon intensity (L x K x J x I values)
4-byte floating point array	Logarithm of X-ray intensities (L x K x J x I x X(j) values)
4-byte floating point array	Ratio of inner-to-outer radii for 0 cm intermediate void (K x J values)
4-byte floating point array	Ratio of inner-to-outer radii for 10 cm intermediate void (K x J values)
4-byte floating point array	Ratio of inner-to-outer radii for 200 cm intermediate void (K x J values)

APPENDIX B. SHIELD LIBRARY PARSING CODE

The following Python v2.7 code can be used to parse the shield scatter database, which is available from the authors upon request. The python text files are also available upon request. Example use in your python script may look something like:

Example python script

```
from scatlib import *
lib = ScatLib()
lib.load('sandia.shieldscatter.db')
an = 13.0 # Al
ad = 2.7 # g/cm^2, 1 cm thick for Al at 2.7 g/cc
ah = 0.0 # no hydrogen weight fraction
erg = 661.7 # Cs–137 primary emission in keV
sf = 0.0 # point source in solid sphere of shielding
# get the group bounds and intensities for this shield
bounds,intensities = lib.get_scatter(erg, an, ad, ah, sf)
```

scatlib.py

```
from parseutils import *
import numpy as np

class ScatLib(object):
    def __init__(self):
        self.group_count = 0
        self.source_energy_count = 0
        self.atomic_number_count = 0
        self.areal_density_count = 0
        self.hydrogen_fraction_count = 0
        self.kedge_energies = []
        self.kedge_indices = []
        self.xray_counts = []
        self.xray_energies = []
        self.primary_scatter_point = []
        self.total_scatter_point = []
        self.primary_scatter_basis = []
        self.total_scatter_basis = []
        self.primary_scatter_slab = []
        self.total_scatter_slab = []
        self.annih_peak = []
        self.xray_peaks = []
        self.shape_factor_point = []
        self.shape_factor_basis = []
        self.shape_factor_slab = []

    def load(self, filename):
        f = open(filename,'rb')
        # scatter group count
        self.group_count = read_int32(f)
        # source energies
        self.source_energy_count = read_int32(f)
        self.energies = read_float32_array(f, self.source_energy_count)
        # atomic numbers
        self.atomic_number_count = read_int32(f)
        self.atomic_numbers = read_float32_array(f, self.atomic_number_count)
        # number of x-rays for each for each atomic number
        self.xray_counts = read_int32_array(f, self.atomic_number_count)
        # x-ray energies for each atomic number
        self.xray_energies = []
        for m in range(self.atomic_number_count):
            self.xray_energies.append([])
            for m in range(self.xray_counts[m]):
```

```
            self.xray_energies[-1].append(read_float32(f))
        # actual k-edge energy for each atomic number
        self.kedge_energies = read_float32_array(f, self.atomic_number_count)
        # index of source energies for left side of kedge (0 based index)
        self.kedge_indices = read_int32_array(f, self.atomic_number_count)
        # areal densities
        self.areal_density_count = read_int32(f)
        self.areal_densities = read_float32_array(f, self.areal_density_count)
        # hydrogen fractions
        self.hydrogen_fraction_count = read_int32(f)
        self.hydrogen_fractions = read_float32_array(f, self.hydrogen_fraction_count)
        # primary scatter (no secondaries) data for 0 cm intermediate void
        dims = [self.hydrogen_fraction_count,
                self.areal_density_count,
                self.atomic_number_count,
                self.source_energy_count,
                self.group_count]
        self.primary_scatter_point = read_float32_ndarray(f, dims)
        self.primary_scatter_point = self.primary_scatter_point.transpose()
        # total scatter (with secondaries) data for 0 cm intermediate void
        self.total_scatter_point = read_float32_ndarray(f, dims)
        self.total_scatter_point = self.total_scatter_point.transpose()
        # primary scatter (with secondaries) data for 10 cm intermediate void
        self.primary_scatter_basis = read_float32_ndarray(f, dims)
        self.primary_scatter_basis = self.primary_scatter_basis.transpose()
        # total scatter (with secondaries) data for 10 cm intermediate void
        self.total_scatter_basis = read_float32_ndarray(f, dims)
        self.total_scatter_basis = self.total_scatter_basis.transpose()
        # primary scatter (with secondaries) data for 200 cm intermediate void
        self.primary_scatter_slab = read_float32_ndarray(f, dims)
        self.primary_scatter_slab = self.primary_scatter_slab.transpose()
        # total scatter (with secondaries) data for 200 cm intermediate void
        self.total_scatter_slab = read_float32_ndarray(f, dims)
        self.total_scatter_slab = self.total_scatter_slab.transpose()
        # annihilation peak intensities
        dims = [self.hydrogen_fraction_count,
                self.areal_density_count,
                self.atomic_number_count,
                self.source_energy_count]
        self.annih_peak = read_float32_ndarray(f, dims)
        self.annih_peak = self.annih_peak.transpose()
        # xray peak intensities
        for l in range(self.hydrogen_fraction_count):
            self.xray_peaks.append([])
            for m in range(self.areal_density_count):
                self.xray_peaks[-1].append([])
                for n in range(self.atomic_number_count):
                    self.xray_peaks[-1][-1].append([])
                    for o in range(self.source_energy_count):
                        self.xray_peaks[-1][-1][-1].append(read_float32_array(f, self.xray_counts[n]))
        # shape factors
        dims = [self.areal_density_count, self.atomic_number_count]
        self.shape_factor_point = read_float32_ndarray(f, dims)
        self.shape_factor_point = self.shape_factor_point.transpose()
        self.shape_factor_basis = read_float32_ndarray(f, dims)
        self.shape_factor_basis = self.shape_factor_basis.transpose()
        self.shape_factor_slab = read_float32_ndarray(f, dims)
        self.shape_factor_slab = self.shape_factor_slab.transpose()
        f.close()

    def generate_group_bounds(self, source_energy):
        width = source_energy / self.group_count
        bounds = []
        for n in range(self.group_count+1):
            bounds.append(width * n)
        return bounds

    def get_scatter(self, source_energy, atomic_number, areal_density, hydrogen_fraction, shape_factor):
        # get energy interpolation indices (i and ip)
```

```
        if source_energy < self.kedge_energies[self.atomic_number_count-1]:
            # source energy is in k-edge region
            for n in range(self.atomic_number_count):
                if source_energy < self.kedge_energies[n]:
                    # source energy is on left side of edge
                    # keep interpolation to left of edge
                    i = self.kedge_indices[n] - 1
        else:
            i = self.source_energy_count - 2
            while self.energies[i] > source_energy:
                i = i - 1
                # break if we hit right side of highest energy k-edge
                if i == self.kedge_indices[self.atomic_number_count-1] + 1:
                    break
        ip = i + 1
        fi = (source_energy - self.energies[i]) / (self.energies[ip] - self.energies[i])
        # get atomic number interpolation indices (j and jp)
        j = self.atomic_number_count - 2
        while self.atomic_numbers[j] > atomic_number:
            j = j - 1
            if j == 0:
                break
        jp = j + 1
        fj = (atomic_number - self.atomic_numbers[j]) / (self.atomic_numbers[jp] - self.atomic_numbers[j])
        # get areal density interpolation indices (k and kp)
        k = self.areal_density_count - 2
        while self.areal_densities[k] > areal_density:
            k = k - 1
            if k == -1:
                # k = -1 is flag to extrapolate to zero scatter
                break
        kp = k + 1
        if k > -1:
            fk = (areal_density - self.areal_densities[k]) / (self.areal_densities[k+1] -
self.areal_densities[k])
        else:
            fk = areal_density / self.areal_densities[kp]
        # get hydrogen fraction interpolation indices (m and mp)
        m = self.hydrogen_fraction_count - 2
        while self.hydrogen_fractions[m] > hydrogen_fraction:
            m = m - 1
            if m == 0:
                break
        mp = m + 1
        fm = (hydrogen_fraction - self.hydrogen_fractions[m]) / (self.hydrogen_fractions[m+1] -
self.hydrogen_fractions[m])
        # determine average shape factors for each set
        if k == -1:
            point_shape_factor = (1.0 - fj) * self.shape_factor_point[j][kp] + fj *
self.shape_factor_point[jp][kp]
            basis_shape_factor = (1.0 - fj) * self.basis_factor_point[j][kp] + fj *
self.basis_factor_point[jp][kp]
            slab_shape_factor = (1.0 - fj) * self.slab_factor_point[j][kp] + fj *
self.slab_factor_point[jp][kp]
        else:
            point_shape_factor = (1.0 - fj) * ((1.0 - fk) * self.shape_factor_point[j][k] + fk *
self.shape_factor_point[j][kp]) \
                + fj * ((1.0 - fk) * self.shape_factor_point[jp][k] + fk *
self.shape_factor_point[jp][kp])
            basis_shape_factor = (1.0 - fj) * ((1.0 - fk) * self.shape_factor_basis[j][k] + fk *
self.shape_factor_basis[j][kp]) \
                + fj * ((1.0 - fk) * self.shape_factor_basis[jp][k] + fk *
self.shape_factor_basis[jp][kp])
            slab_shape_factor = (1.0 - fj) * ((1.0 - fk) * self.shape_factor_slab[j][k] + fk *
self.shape_factor_slab[j][kp]) \
                + fj * ((1.0 - fk) * self.shape_factor_slab[jp][k] + fk * self.shape_factor_slab[jp][kp])
        # determine weights for linear interpolation between each set
        point_weight = 0.0
        basis_weight = 0.0
```

```
        slab_weight = 0.0
        if shape_factor < point_shape_factor:
            point_weight = 1.0
        elif shape_factor > slab_shape_factor:
            slab_weight = 1.0
        elif shape_factor > basis_shape_factor:
            slab_weight = (shape_factor - basis_shape_factor) / (slab_shape_factor - basis_shape_factor)
            basis_weight = 1.0 - slab_weight
        else:
            basis_weight = (shape_factor - point_shape_factor) / (basis_shape_factor - point_shape_factor)
            point_weight = 1.0 - basis_weight
        # interpolate matrices
        group_intensities = self.interpolate(i, ip, j, jp, k, kp, m, mp, fi, fj, fk, fm, point_weight,
basis_weight, slab_weight)
        # generate group bounds for requested source energy
        group_bounds = self.generate_group_bounds(source_energy)
        return group_bounds, group_intensities

    def get_matrix_element(self, l, i, j, k, m, pw, bw, sw):
        if k == -1:
            return 0.0
        return pw * self.total_scatter_point[l][i][j][k][m] + bw * self.total_scatter_basis[l][i][j][k][m]
+ sw * self.total_scatter_slab[l][i][j][k][m]

    def interpolate(self, i, ip, j, jp, k, kp, m, mp, fi, fj, fk, fm, pw, bw, sw):
        ri = 1.0 - fi
        rj = 1.0 - fj
        rk = 1.0 - fk
        rm = 1.0 - fm
        intensities = []
        for l in range(self.group_count):
            intensities.append(
                ri * (  rj * (  rk * (rm * self.get_matrix_element(l, i, j, k, m, pw, bw, sw)    + fm *
self.get_matrix_element(l, i, j, k, mp, pw, bw, sw))
                              + fk * (rm * self.get_matrix_element(l, i, j, kp, m, pw, bw, sw)   + fm *
self.get_matrix_element(l, i, j, kp, mp, pw, bw, sw))) \
                    + fj * (  rk * (rm * self.get_matrix_element(l, i, jp, k, m, pw, bw, sw)   + fm *
self.get_matrix_element(l, i , jp, k, mp, pw, bw, sw))
                              + fk * (rm * self.get_matrix_element(l, i, jp, kp, m, pw, bw, sw)  + fm *
self.get_matrix_element(l, i , jp, kp, mp, pw, bw, sw)))) \
                + fi * (rj * (  rk * (rm * self.get_matrix_element(l, ip, j, k, m, pw, bw, sw)   + fm *
self.get_matrix_element(l, ip, j, k, mp, pw, bw, sw))
                              + fk * (rm * self.get_matrix_element(l, ip, j, kp, m, pw, bw, sw)  + fm *
self.get_matrix_element(l, ip, j, kp, mp, pw, bw, sw))) \
                    + fj * (  rk * (rm * self.get_matrix_element(l, ip, jp, k, m, pw, bw, sw)  + fm *
self.get_matrix_element(l, ip, jp, k, mp, pw, bw, sw))
                              + fk * (rm * self.get_matrix_element(l, ip, jp, kp, m, pw, bw, sw) + fm *
self.get_matrix_element(l, ip, jp, kp, mp, pw, bw, sw))))
            )
            # convert from log-intensity to intensity
            if abs(intensities[-1]) > 1e-20:
                intensities[-1] = max(0.0, np.exp(intensities[-1]))
        return intensities
```

parseutils.py

```
import struct
import numpy as np

def read_float32_ndarray_dim(f, dims, dimidx, array):
    if dimidx == len(dims) - 1:
        # parse and append data along this last dimension
        for n in range(dims[dimidx]):
            array[n] = read_float32(f)
    else:
        for n in range(dims[dimidx]):
            read_float32_ndarray_dim(f, dims, dimidx+1, array[n])
```

```
def read_float32_ndarray(f, dims):
    array = np.ndarray(shape=dims, dtype=float)
    read_float32_ndarray_dim(f, dims, 0, array)
    return array

def read_float32(f):
    data = f.read(4)
    return struct.unpack("f",data)[0]

def read_float32_array(f, count):
    result = []
    for n in range(count):
        result.append(read_float32(f))
    return result

def read_int32(f):
    data = f.read(4)
    return struct.unpack("i",data)[0]

def read_int32_array(f, count):
    result = []
    for n in range(count):
        result.append(read_int32(f))
    return result

def read_string(f, charcount):
    data = f.read(charcount)
    data_format = "{0}s".format(charcount)
    return struct.unpack(data_format,data)[0]

def read_string_array(f, charcount, stringcount):
    result = []
    for n in range(stringcount):
        result.append(read_string(f, charcount))
    return result
```

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Steven Horne	6634	smhorne@sandia.gov
Dean Mitchell	6630	djmitch@sandia.gov
Michael Enghauser	6634	mwengha@sandia.gov
Sean O'Brien	6634	sobrien@sandia.gov
Technical Library	01177	libref@sandia.gov

This page left blank

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.