University of Florida

TITAN: A 3-D Deterministic Radiation Transport Code

TITAN User Manual Version 1.05

Ce Yi & Alireza Haghighat

Contents

Chapter 1 - Introduction

TITAN is a deterministic radiation transport simulation code in 3-D Cartesian geometry. TITAN numerically solves the time-independent first order transport equation (Linear Boltzmann Equation) using a hybrid Discrete Ordinate (Sn) and Ray-tracing method (Refs. 1-6).

Two transport solvers, an Sn Solver and a ray-tracing solver, are integrated in the TITAN code. Both solvers work on the coarse mesh level in Cartesian geometry. Generally, a TITAN problem model contains more than one coarse mesh. This allows users to apply different solvers to different coarse mesh. This feature can be useful for problems containing a large region of low scattering medium. In such regions, the Sn method (Refs. 7-8) requires finer angular and spatial meshing and becomes less efficient. TITAN's ray-tracing solver is more efficient to solve the transport equation in such regions. The ray-tracing solver is essentially a 3-D Method of Characteristics (Ref. 9-10) solver, only it applies to an individual coarse mesh, instead of the whole spatial domain. Currently the ray-solver applies only on coarse mesh with one material region, and the total cross-section of the material should be close to zero to qualify as 'low scattering' medium. For a multi-region regular coarse mesh, the Sn solver should be used.

TITAN is originally designed to solve radiation transport problems for medical physics applications, where large air regions are very common. It has been applied on a series of SPECT (Single Photon Emission Computed Tomography) models to simulate the projection images (Ref. 2). TITAN can also be used in nuclear engineering application for both shielding and criticality calculations. It has been benchmarked on a number of OECD/NEA benchmark problems, including the C5G7 mox (Ref. 1), Kobayahsi (Ref. 6), and the 3-D parameter space (Ref. 3).

The code, about 20,000 lines at present, is written in FORTRAN 90/95 with some language extensions of object-oriented features (part of the FORTRAN 2003 standard). Object-oriented paradigm is heavily used in the code. Both the Sn solver and the ray-tracing solver are coarse-mesh-oriented. This allows users to apply an individual solver, quadrature set, meshing scheme, and etc. to a given coarse mesh. The subroutines in the code are organized in a kernel-layer structure. The main task for the kernel is to complete a transport sweep within a coarse mesh, in one direction of a quadrature set, and for one energy group. Outer layer subroutines complete the tasks, such as system transport sweep and source iteration loop, by calling the inner layer subroutines.

TITAN is active in development. Some features of the code include:

- Integrated S_N and ray-tracing solvers.
- Shared scattering source kernel allowing arbitrary order anisotropic scattering.

- Backward ray-tracing.
- Block-oriented data structure allowing localized quadrature sets and solvers.
- Layered code structure.
- Level-symmetric and P_N-T_N quadrature sets.
- Incorporation of three ordinate splitting techniques (rectangular, local P_N-T_N, and circular)
- Fast and memory-efficient spatial and angular projections on the interfaces of coarse meshes by using sparse projection matrix.
- 'Frontline-style' interface flux handling.
- An efficient algorithm for calculation of the scattering source and the within-group scattering with a modified scattering kernel.
- A binary I/O library to visualize and post-process data with TECPLOT.
- Extra Sweep technique with the fictitious quadrature technique for calculations of angular fluxes along arbitrary directions.

A parallel version using MPI is also available. Currently the parallel version only does angular decomposition. It uses the same input deck as the serial version. The number of cpu is specified in the 'mpirun' command line. TITAN will distribute the 'angular sweep' tasks evenly to the allocated processors.

Chapter 2 - Theory

2.1 Multi-Block Framework Overview

To numerically solve the LBE with a deterministic method, discretization schemes are required in the energy, angular and spatial domains. Once the discretization grid is built in the phase space, one can evaluate the angular flux on each node by sweeping the grid in a specific order repeatedly via an iteration scheme (e.g., the source iteration scheme) until solution convergence is achieved.

The hybrid method is built on a multi-block spatial meshing scheme, which is also used in the PENTRAN code (Ref. 11). The meshing scheme divides the whole problem model into coarse meshes (blocks) in the Cartesian geometry. And each coarse mesh is further filled with uniform fine meshes or characteristic rays depending on which solver is assigned to the coarse mesh. Figure 2-1 shows the multi-block framework of the hybrid approach.

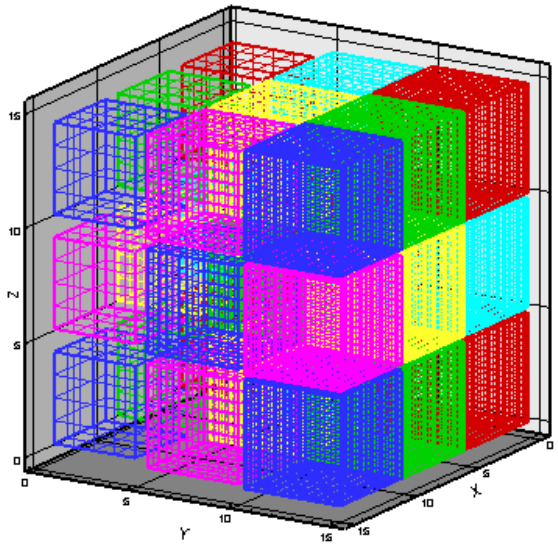


Figure 2-1. Coarse mesh/fine mesh meshing scheme.

The multi-block framework leads to an important feature of the hybrid code: both the S_N and characteristics solvers are coarse-mesh-oriented. They are designed to solve the transport equation on the scope of a coarse mesh. A coarse mesh can be considered as a relatively independent coding unit with its own spatial discretization grid (fine meshes or characteristic rays) and angular discretization grid (quadrature set). Users can assign either solver to each coarse mesh.

In the following sections, we provide the formulations for the block-oriented S_N and characteristics solvers, and demonstrate the two solvers on the multi-block framework. We

also discuss the angular quadrature sets used in the TITAN code along with the ordinate splitting technique.

2.2 Discrete Ordinates Formulations

Here, we apply the multigroup theory (Ref. 7) to discretize the LBE in the energy domain.

$$
\begin{aligned}
&(\mu\frac{\partial}{\partial x}+\eta\frac{\partial}{\partial y}+\xi\frac{\partial}{\partial z})\psi_g(x,y,z,\mu,\varphi)+\sigma_g(x,y,z)\psi_g(x,y,z,\mu,\varphi)= \\
&\sum_{g'=1}^{G}\sum_{l=0}^{L}(2l+1)\sigma_{s,g'\to g,l}(x,y,z)\{P_l(\mu)\phi_{g',l}(x,y,z)+2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu)\cdot \\
&[\phi^k_{C\,g',l}(x,y,z)\cos(k\varphi)+\phi^k_{S\,g',l}(x,y,z)\sin(k\varphi)]\}+ \\
&\frac{\chi_g}{k_o}\sum_{g'=1}^{G}\nu\sigma_{f\,g'}(x,y,z)\phi_{g',0}(x,y,z)\ \text{or}\ \mathrm{S_g^{fix}}(x,y,z,\mu,\varphi)
\end{aligned}
\tag{2-1}
$$

Where, μ, η, and ξ are the x, y and z direction cosines for the discrete ordinates, θ, φ are the polar and azimuthal angles, respectively. (μ, φ) or (μ, η, ξ) specifies a discrete ordinate, where $\mu=\cos(\theta)$, $\eta=\sin(\theta)\cos(\varphi)$, $\xi=\sin(\theta)\sin(\varphi)$. $P_l(\mu)$ is the ℓ^{th} Legendre polynomial (for l =*1*, L where L is Legendre expansion order). And $P_l^k(\mu)$ is the ℓ^{th}, k^{th} associated Legendre polynomial, $\psi_g(x,y,z,\mu,\varphi)$ is the group g angular flux (for g=*1*, G, where G is the number of groups) at the position of (x,y,z) and in the direction of (μ,φ). $\phi_{g',l}$ is the ℓ^{th} Legendre scalar flux moment for group g'. $\phi^k_{C\,g',l}(x,y,z)$ is ℓ^{th}, k^{th} cosine associated Legendre scalar flux moment for group g', and $\phi^k_{S\,g',l}(x,y,z)$ is ℓ^{th}, k^{th} sine associated Legendre scalar flux moment for group g'at the position of (x,y,z). These flux moments are defined as:

$$\phi_{g',l}(x,y,z)=\int_{-1}^{1}\frac{d\mu'}{2}P_l(\mu')\int_0^{2\pi}\frac{d\varphi'}{2\pi}\psi_{g'}(x,y,z,\mu',\varphi') \tag{2-2}$$

$$\phi^k_{C\,g',l}(x,y,z)=\int_{-1}^{1}\frac{d\mu'}{2}P_l^k(\mu')\int_0^{2\pi}\frac{d\varphi'}{2\pi}\cos(k\varphi')\psi_{g'}(x,y,z,\mu',\varphi') \tag{2-3}$$

$$\phi^k_{S\,g',l}(x,y,z)=\int_{-1}^{1}\frac{d\mu'}{2}P_l^k(\mu')\int_0^{2\pi}\frac{d\varphi'}{2\pi}\sin(k\varphi')\psi_{g'}(x,y,z,\mu',\varphi') \tag{2-4}$$

And other variables are:

σ_g: total group macroscopic cross section

$\sigma_{sg'\to g}$: ℓ^{th} moment of the macroscopic differential scattering cross section from $g'\to g$.

χ_g: group fission spectrum

k_0: criticality eigenvalue

$\upsilon\sigma_{fg}$: group fission production

$S_g^{fix}(x,y,z,\mu,\varphi)$: external source on the position of *(x,y,z)* and in the direction of (μ,φ)

We can make several observations on Eq. 2-1. First, obviously it accomplishes the discretization in the energy domain by utilizing the multigroup theory. As a result, $\psi(\vec{r},E,\hat{\Omega})$ becomes $\psi_g(x,y,z,\mu,\varphi)$. Secondly in the angular domain, no further discretization is required, since we solve for the angular flux in a number of discrete directions of (μ_n,φ_n) $n=1,N$, where *N* is the total number of directions. The discrete directions are carefully chosen by the quadrature set so that we can conserve the integral quantities such as scalar fluxes. Thirdly, the most challenging term is the scattering term, in which we convert the integrations over energy and angular domain into numerical summations for energy groups and Legendre expansion terms. Derivations of the scattering kernel are given in Appendix A. It is important to note that in Eq. 2-1, the scattering kernel, as well as the fission term, does not explicitly depend on the angular flux, but on the flux moments. The relationships between the angular flux and the flux moments are defined by Eqs. 2-2 to 2-4. Finally the streaming term becomes a differential term in Cartesian geometry. In order to numerically evaluate the differentials, differencing scheme is required in the S_N method.

2.3 Source Iteration Process

Since the terms on the right hand side of Eq. 2-1, including scattering term, fission term and fix-source term, are not explicitly dependent on the angular flux, we can further simplify Eq. 2-1 by combining all the source terms into one source term.

$$(\mu\frac{\partial}{\partial x}+\eta\frac{\partial}{\partial y}+\xi\frac{\partial}{\partial z})\psi_g(x,y,z,\mu,\varphi)+\sigma_g(x,y,z)\psi_g(x,y,z,\mu,\varphi)=Q_g(x,y,z,\mu,\varphi) \quad (2\text{-}5)$$

where $Q_g=S_{scattering}+S_{fission}$ or S_{fix}. $S_{scattering}$, $S_{fission}$ and S_{fix} represent the three terms on the right hand side of Eq. 2-1 respectively. Eq. 2-5 can be viewed as a numerical iteration equation, which usually is called 'source iteration' scheme (SI).[2] In this iteration process, Q_g is calculated from previous iteration results. Therefore, we can solve Eq. 2-5 for the angular flux by taking Q_g as a constant. Flux moments can be evaluated by Eqs. 2-2 to 2-4 with the latest angular flux, then we can use the flux moments to update Q_g for the next iteration. This process is repeated until the *0'th* flux moment is converged under some convergence criterion. The iteration process for each group (*g*) can be illustrated as follows:

Step 1: Solve Eq. 2-5 for angular flux $\psi_g(x,y,z,\mu,\varphi)$.

Step 2: Evaluate flux moments based on Eqs. 2-2 to 2-4.

Step 3: Update the scattering source.

Step 4: Repeat the process from Step 1, until $\max(|\frac{\phi_g^{(i)} - \phi_g^{(i-1)}}{\phi_g^{(i-1)}}|) \le tolerence$.

In Step 1, ψ_g is calculated for every fine mesh along a given direction, which is referred to as 'one direction sweep'. After sweeps for every direction are completed, flux moments can be updated in Step 2. The group iteration (*g=1, G*) needs to repeat only once for fixed source problems with only down-scattering, because the scattering source for the current group only depends on the converged upper group flux moments. The summation over groups in the scattering term can be reduced to $\sum_{g'=1}^{g-1}$ instead of $\sum_{g'=1}^{G}$, However, for problems with up-scattering, an outer iteration is required since the scattering source is coupled with lower energy groups. For eigenvalue problems, another outer loop is necessary so that the fission source and *k-effective* can be updated in between two successive outer iterations.

2.4 Differencing Scheme

From Eq. 2-1 to Eq. 2-5, we are finally one step away to numerically solving the LBE, which is the evaluation of the differencing (streaming) term in Eq. 2-5 by various differencing schemes. As shown in Figures 2-2, Eq. 2-5 applies on a spatial domain of a fine mesh with the sizes of $\Delta x, \Delta y$ and Δz on three axes.

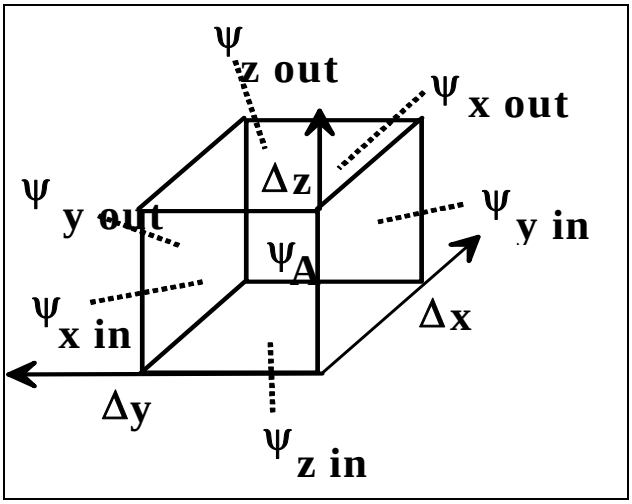


Figure 2-2. Differencing scheme on one fine mesh.

Here, we solve for the average flux on the fine mesh.

$$\psi_{gijk}^{(n)} = \frac{1}{V_{ijk}} \int_{\Delta x} dx \int_{\Delta y} dy \int_{\Delta z} dz \psi_g(x, y, z, u_n, \varphi_n) \qquad (2\text{-}6)$$

Where i, j, k are the fine mesh indices, g is the group index, and n is the direction index. $V_{ijk} = \Delta x \Delta y \Delta z$ is the volume of the fine mesh. Now, we can finally complete the discretizations on all three domains in the phase space. To calculate $\psi_{gijk}^{(n)}$, we integrate Eq. 2-5 over the fine mesh volume V_{ijk}.

$$\begin{aligned}
&\mu_n \int_0^{\Delta y} dy \int_0^{\Delta z} dz \left[\psi_g^{(n)}(\Delta x, y, z) - \psi_g^{(n)}(0, y, z) \right] \\
&+\eta_n \int_0^{\Delta x} dx \int_0^{\Delta z} dz \left[\psi_g^{(n)}(x, \Delta y, z) - \psi_g^{(n)}(x, 0, z) \right] \\
&+\xi_n \int_0^{\Delta x} dx \int_0^{\Delta y} dy \left[\psi_g^{(n)}(x, y, \Delta z) - \psi_g^{(n)}(x, y, 0) \right] \\
&+\sigma_{ijk} \int_0^{\Delta x} dx \int_0^{\Delta y} dy \int_0^{\Delta z} dz \psi_g^{(n)}(x, y, z) = \int_0^{\Delta x} dx \int_0^{\Delta y} dy \int_0^{\Delta z} dz Q_g^{(n)}(x, y, z)
\end{aligned} \tag{2-7}$$

We assume cross sections are constant inside the fine mesh. In a similar way as Eq. 2-6, we define the fluxes on the three incoming boundaries and the three outgoing boundaries as:

$$\begin{aligned}
\psi_{x\text{ in}} &= \frac{1}{\Delta y \Delta z} \int_{\Delta y} dy \int_{\Delta z} dz \psi_g^{(n)}(0, y, z) \\
\psi_{x\text{ out}} &= \frac{1}{\Delta y \Delta z} \int_{\Delta y} dy \int_{\Delta z} dz \psi_g^{(n)}(\Delta x, y, z) \\
\psi_{y\text{ in}} &= \frac{1}{\Delta x \Delta z} \int_{\Delta x} dx \int_{\Delta z} dz \psi_g^{(n)}(x, 0, z) \\
\psi_{y\text{ out}} &= \frac{1}{\Delta x \Delta z} \int_{\Delta x} dx \int_{\Delta z} dz \psi_g^{(n)}(x, \Delta y, z) \\
\psi_{z\text{ in}} &= \frac{1}{\Delta x \Delta y} \int_{\Delta x} dx \int_{\Delta y} dy \psi_g^{(n)}(x, y, 0) \\
\psi_{z\text{ out}} &= \frac{1}{\Delta x \Delta y} \int_{\Delta x} dx \int_{\Delta y} dy \psi_g^{(n)}(x, y, \Delta z)
\end{aligned} \tag{2-8}$$

And the angular source for the fine mesh can be defined as:

$$Q_{gijk}^{(n)} = \frac{1}{V_{ijk}} \int_{\Delta x} dx \int_{\Delta y} dy \int_{\Delta z} dz Q_g(x, y, z, u_n, \varphi_n) \tag{2-9}$$

We can divide both sides of Eq. 2-7 by V_{ijk}, then substitute Eqs. 2-6, 2-8 and 2-9 into Eq. 2-7, and obtain Eq. 2-10.

$$\frac{\mu_n}{\Delta x}(\psi_{x\text{ out}} - \psi_{x\text{ in}}) + \frac{\eta_n}{\Delta y}(\psi_{y\text{ out}} - \psi_{y\text{ in}}) + \frac{\xi_n}{\Delta z}(\psi_{z\text{ out}} - \psi_{z\text{ in}}) + \sigma_{ijk}\psi_{gijk}^{(n)} = Q_{gijk}^{(n)} \tag{2-10}$$

In Eq. 2-10, the three incoming fluxes ($\psi_{x\,\mathrm{in}}$, $\psi_{\mathrm{y\,in}}$ and $\psi_{\mathrm{z\,in}}$) can be obtained from the fine-mesh boundary conditions at the three incoming surfaces. Therefore, to calculate $\psi_{gijk}^{(n)}$ and the three outgoing fluxes, we need three additional equations, which are provided by the differencing scheme. One of the simplest schemes is the linear diamond (LD) differencing (Ref. 7) expressed by:

$$\begin{aligned}
\psi_{\mathrm{x\,out}} &= 2\psi_{\mathrm{x\,in}} - \psi_{gijk}^{(n)} \\
\psi_{\mathrm{y\,out}} &= 2\psi_{\mathrm{y\,in}} - \psi_{gijk}^{(n)} \\
\psi_{\mathrm{z\,out}} &= 2\psi_{\mathrm{z\,in}} - \psi_{gijk}^{(n)}
\end{aligned} \tag{2-11}$$

When moving in positive directions (as shown in Figure 2-2), we may eliminate the outgoing fluxes in Eq. 2-10 by using Eq. 2-11 to obtain Eq. 2-12.

$$\psi_{gijk}^{(n)} = \frac{\frac{2\mu_n}{\Delta x}\psi_{x\,\mathrm{in}} + \frac{2\eta_n}{\Delta y}\psi_{\mathrm{y\,in}} + \frac{2\xi_n}{\Delta z}\psi_{\mathrm{z\,in}} + Q_{gijk}^{(n)}}{\sigma_{ijk} + \frac{2\mu_n}{\Delta x} + \frac{2\eta_n}{\Delta y} + \frac{2\xi_n}{\Delta z}} \tag{2-12}$$

The original LBE (Eq. 2-1) reduces to a set of linear equations of Eqs. 2-11 and 2-12. Note that the incoming surfaces change for different directions. The fine mesh sweeping order is decided by the octant number of the direction. The same principle is also applied to coarse meshes: we always try to calculate the outgoing fluxes by solving the LBE based on the incoming fluxes. In this sweeping process, the outgoing fluxes will be the incoming flux for the next adjacent fine/coarse mesh along the direction. If the incoming or outgoing boundaries of the fine/coarse mesh are aligned with the model boundaries, model boundary conditions are applied. However, for the coarse mesh sweep, flux projections are required on the interface of two adjacent coarse meshes if the two coarse meshes use different spatial and angular discrtetization grids. The projection techniques are discussed in Section 2.9.

In Eq. 2-12, the terms of $\frac{\mu_n}{\Delta x}$, $\frac{\eta_n}{\Delta y}$ and $\frac{\xi_n}{\Delta z}$ are always positive, since we always sweep fine meshes along the direction defined by the direction cosines (μ_n, η_n, ξ_n), i.e., μ_n and Δx, either both are positive, or both are negative. The incoming fluxes, $Q_{gijk}^{(n)}$ and σ_{ijk} are positive with their physical meaning. As a result, $\psi_{gijk}^{(n)}$ is always positive. However, the outgoing fluxes calculated by Eq. 2-11 of the linear diamond differencing scheme could be negative, which conflicts with its physical meaning. In order to avoid negative fluxes, flux zero fix-up is usually applied in the diamond differencing scheme. Furthermore, the diamond differencing scheme introduces artificial oscillations in certain conditions (Ref. 12). For this reason, and to facilitate increasing accuracy with adaptive differencing, more advanced differencing schemes (Refs. 13-14), such as DTW (Ref. 15), EDW (Ref. 16), and EDI (Ref.

17) are implemented in the PENTRAN code. Currently, the diamond and DTW differencing schemes are applied in the TITAN code.

2.5 Characteristics Formulations

Now we further discuss the formulations for the MOC used in the TITAN code. MOC solves the transport equation for the angular flux along characteristic rays with region-wise discretization grid (i.e. coarse mesh) in the spatial domain. Since a region can be any shape, MOC has the ability to treat the geometry of a model exactly. Similar to the coarse/fine mesh sweep process in the S_N method, in the MOC, we still calculate the outgoing flux based on the incoming flux for each region, and the outgoing flux will be the incoming flux for the next adjacent region. In the angular domain, we perform this sweeping process for a number of directions chosen by a quadrature set. Within one region, we assume constant cross sections and calculate the average flux for the region by filling the region with characteristic rays along the directions in a quadrature set. Figure 2-3 shows the parallel characteristic rays along direction *n* in a square region *i*.

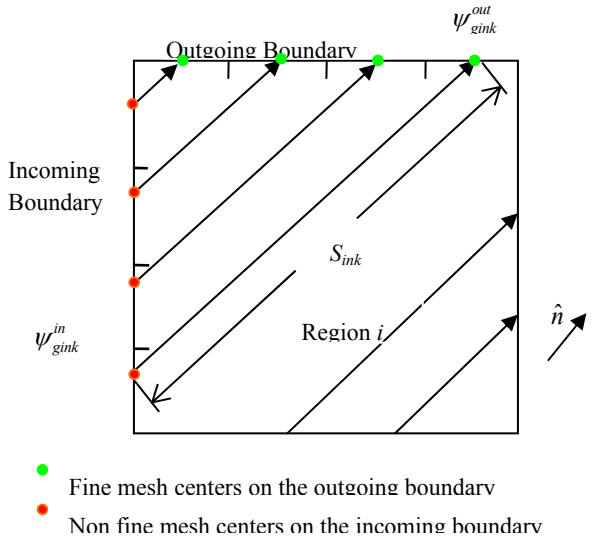


Figure 2-3. Schematic of characteristic rays in a coarse mesh using the characteristics method.

For a given ray of *k* with a path length of s_{ink}, we solve the transport equation for $\psi_{gink}(l)\ \ 0 \le l \le s_{ink}$, which is the angular flux for group *g*, along direction *n*, at position *l* along ray *k* in region *i*. We denote $\psi^{in}_{gink} = \psi_{gink}(0)$ and $\psi^{out}_{gink} = \psi_{gink}(s_{ink})$. The transport equation along ray *k* can be written as:

$$\hat{\Omega}_n \cdot \nabla \psi_{gink}(l) + \sigma_{gi}\psi_{gink}(l) = Q_{gin} \qquad (2\text{-}13)$$

Where $Q_{gin} = S_{scattering} + S_{fission}$ or S_{fix} is the total angular source in region *i* along direction *n* for group *g*. We assume a constant angular source for each ray in region *i* along direction *n*. The streaming term in Eq. 2-13 can be viewed as flux gradient's projection along direction *n*, which is the directional derivative of the angular flux. Therefore, Eq. 2-13 can be rewritten as:

$$\frac{d\psi_{gink}(l)}{dl} + \sigma_{gi}\psi_{gink}(l) = Q_{gin} \tag{2-14}$$

Where, *l* is the path length. Eq. 2-14 can be solved analytically if we know the incoming flux $\psi^{in}_{gink} = \psi_{gink}(0)$ as a boundary condition.

$$\psi_{gink}(l) = \psi^{in}_{gink} e^{-\sigma_{gi}l} + \frac{Q_{gin}}{\sigma_{gi}}(1 - e^{-\sigma_{gi}l}) \tag{2-15}$$

The outgoing flux can be calculated as follows.

$$\psi^{out}_{gink} = \psi_{gink}(s_{ink}) = \psi^{in}_{gink} e^{-\sigma_{gi}s_{ink}} + \frac{Q_{gin}}{\sigma_{gi}}(1 - e^{-\sigma_{gi}s_{ink}}) \tag{2-16}$$

In order to calculate the average angular flux in region *i*, first we use Eqs. 2-15 and 2-16 to evaluate the average angular flux for each parallel ray along direction *n,* which is given by:

$$\begin{aligned}\overline{\psi}_{gink} &= \frac{1}{s_{ink}}\int_0^{s_{ink}} dl\psi_{gink}(l) = \frac{1}{s_{ink}}\int_0^{s_{ink}} dl\left[\psi^{in}_{gink} e^{-\sigma_{gi}l} + \frac{Q_{gin}}{\sigma_{gi}}\left(1 - e^{-\sigma_{gi}l}\right)\right] \\ &= \frac{Q_{gin}}{\sigma_{gi}} + \frac{\psi^{in}_{gink} - \psi^{out}_{gink}}{s_{ink}\sigma_{gi}} = \frac{Q_{gin}}{\sigma_{gi}} + \frac{\Delta_{gink}}{s_{ink}\sigma_{gi}}\end{aligned} \tag{2-17}$$

Where $\Delta_{gink} = \psi^{in}_{gink} - \psi^{out}_{gink}$. Then, we evaluate the average angular flux for region *i* by summation of average angular fluxes for all the parallel rays along direction *n*, with a weighting factor of $\delta V_{ink} = \delta A_{ink} s_{ink}$, where δA_{ink} is the width (in 2-D) or the cross sectional area (in 3-D) which ray (*i,n,k*) represents. The average angular flux along direction *n* is expressed by:

$$\overline{\psi}_{gin} = \frac{\sum_k \overline{\psi}_{gin} \cdot (\delta A_{ink} s_{ink})}{\sum_k (\delta A_{ink} s_{ink})} = \frac{\sum_k (\delta A_{ink} s_{ink}) \cdot \left(\frac{Q_{gin}}{\sigma_{gi}} + \frac{\Delta_{gink}}{s_{ink}\sigma_{gi}}\right)}{\sum_k (\delta A_{ink} s_{ink})} = \frac{Q_{gin}}{\sigma_{gi}} + \frac{\sum_k (\delta A_{ink} \Delta_{ink})}{\sigma_{gi} \sum_k (\delta A_{ink} s_{ink})} \tag{2-18}$$

Note that the volume (in 3-D) or the area (2-D) for region *i* can be represented as $V_i \approx \sum_k \delta V_{ink} = \sum_k \delta A_{ink} s_{ink}$, if δA_{ink} is small enough. Since δA_{ink} represents the distance between two adjacent parallel rays, denser rays are required to cover region *i* as δA_{ink} decreases. Therefore, in order to get an accurate region-averaged angular flux with Eq. 2-18, two conditions are necessary:

- Region *i* is small, or flux changes slowly over the region.
- Rays are dense enough to cover the region.

Note that similar conditions are required in the S_N method in the sense of spatial domain discretization approach. Generally, in the S_N method finer meshes are required to get a more accurate flux distribution.

The source iteration scheme can be applied to the MOC similarly as in the S_N method. Eqs. 2-16 and 2-18, as Eqs. 2-11 and 2-12 in the S_N method, are the fundamental equations for Step 1 (the 'sweep' process) in the source iteration scheme, except that the fine-mesh-averaged angular flux in the S_N method becomes region-averaged angular flux in the MOC.

2.6 Block-Oriented Characteristics Solver

The block-oriented characteristics solver is different from the general MOC approach, in the sense that we only apply the solver on an individual block within the multi-block framework. For a characteristics coarse mesh, we build uniform fine meshing on the boundaries, and draw the characteristic rays from the fine mesh centers along quadrature directions. We consider the characteristics coarse mesh as one region. And the coarse mesh space is covered with characteristic rays. The boundary fluxes with uniform fine meshing grid are used to communicate with adjacent blocks, since coarse meshes are coupled on their interfaces in the sweep process.

2.6.1 Backward Ray-Tracing Procedure

Figure 2-4 shows a typical coarse mesh with 5×5 fine meshes on the 6 surfaces. Note that fine meshing is only applied on the surfaces of a coarse mesh to which the characteristics solver is assigned. The same coarse-mesh volume could be divided into $5 \times 5 \times 5$ fine meshes if the S_N solver is assigned.

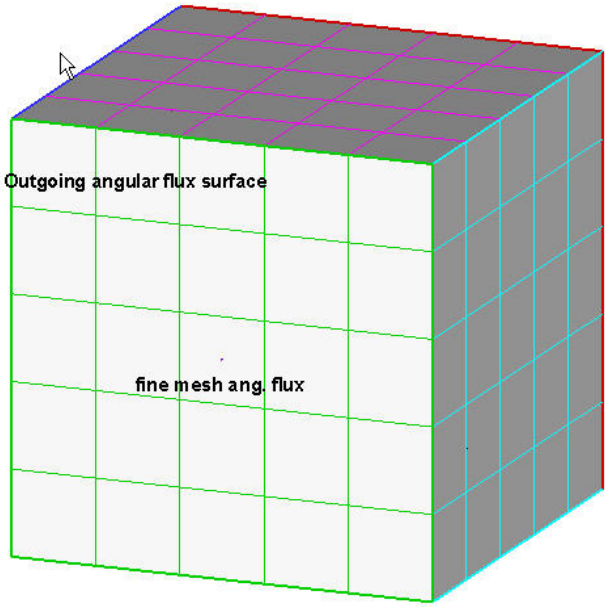


Figure 2-4. A coarse mesh with characteristics solver assigned.

Now we can demonstrate how we set up rays in a coarse mesh shown in Figure 2-4. In the 'sweep' process, our goal is to calculate the outgoing flux based on the incoming flux. In Figure 2-4, the front surface becomes one of the three outgoing surfaces for the directions in four of eight octants in a quadrature set. For the other four octants, it becomes one of the three incoming surfaces. For demonstration purposes, we assume the front surface in Figure 2-4 is one of the outgoing surfaces. Now we need to calculate the outgoing angular flux for each fine mesh on the surface for each direction in the four octants. Figure 2-5 shows the characteristic rays associated with the center fine mesh on the front surface.

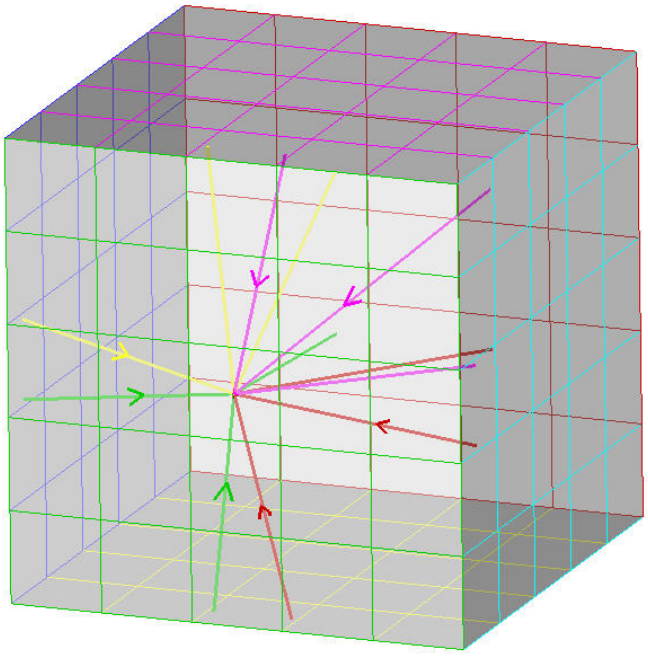


Figure 2-5. Characteristic rays for one fine mesh on one outgoing surface.

As shown in Figure 2-5, we draw 12 rays backward from the center of one fine mesh (located on the front surface) to the incoming surfaces across the coarse mesh. The four different color rays in Figure 2-5 represent the directions in four octants. Since the intersection positions are not necessarily at the centers of fine meshes on the incoming boundary, an interpolation scheme is required to calculate the incoming fluxes at the intersection positions based on the known incoming fluxes at the fine-mesh centers. Here, we consider an S_4 quadrature set which provides three directions per octant. For directions in 4 of the 8 octants, the front surface is one of the three outgoing surfaces. Therefore, 12 rays for each fine mesh on the front surfaces are required. The overall characteristic ray density to cover the coarse mesh depends on both the fine mesh grid densities on the outgoing boundaries and the number of directions in the quadrature set. Figure 2-3 also illustrates the characteristic ray drawing procedure in 2-D. The green dots on the outgoing boundary in Figure 2-3 are located on the centers of the fine meshes. While the red dots, which represent the intersection points on the incoming boundary, are off-centered.

2.6.2 Advantage of Backward Ray-Tracing

In the characteristic ray drawing procedure, we could choose a 'forward' approach: drawing the characteristic rays from the fine mesh centers on the incoming boundary to the outgoing boundary. The outgoing boundary will experience rays intersecting its fine meshes in a scattered manner. After the outgoing angular fluxes are calculated, an interpolation procedure is required to project the scattered outgoing flux onto the fine mesh centers.

In a ray drawing procedure, we can always choose a fine mesh center, either on the incoming boundary or on the outgoing boundary, as one node of each characteristic ray to avoid interpolations on that boundary. The other node of the ray will be scattered onto the other boundary, on which interpolations are required regardless since we are interested in the fluxes only on the centers of the fine mesh grid. An interpolation procedure on the incoming boundary needs to evaluate the angular flux at the incoming node of each characteristic ray based on the known incoming fluxes at the structured fine mesh centers. On the other hand, an interpolation procedure on the outgoing boundary needs to evaluate the outgoing flux at the center of each fine mesh based on the calculated fluxes at the scattered outgoing nodes of the rays. The difference between the two choices is: on the incoming boundary, the interpolation procedure is carried on from structured data points (incoming fluxes on the fine mesh centers) to scattered data points (incoming fluxes for the rays), while on the outgoing boundary, the procedure is carried on from scattered data points (outgoing fluxes from the rays) to structured data points (outgoing fluxes on the fine mesh centers).

In the block-oriented characteristics approach, we choose to fix the interpolations on the incoming boundary, because it is numerically more accurate and efficient to interpolate scattered points from structured points than the other way around. For interpolations on the outgoing boundary, the scattered outgoing nodes of the rays are the known base points. These scattered points could be too few, or too badly non-uniformly scattered on the boundary, to complete a relatively accurate interpolation to evaluate the flux on the center of every fine mesh. For interpolations on the incoming boundary, the structured, uniformly distributed fine mesh center fluxes are the known data points. Four closest fine mesh centers to any scattered point can always be found to complete a bi-liner interpolation. Clearly an interpolation procedure on the incoming boundary is a better choice. The backward ray-tracing facilitates the integration of the block-oriented solvers.

2.6.3 Ray Tracer

In order to calculate the outgoing flux by using Eq. 2-16, we need to evaluate the incoming flux, which is located on the other end of the rays on the incoming surfaces. The incoming flux is known from the boundary conditions if the incoming surface is part of the model boundaries, or from the outgoing flux for the adjacent coarse mesh in the coarse mesh sweep process. We assume these fine-mesh-averaged incoming angular fluxes are located on the center of each fine mesh on the incoming surface. However, the intersection point on the incoming surface is not necessarily on the center of a fine mesh. Therefore, we need to determine the intersection position of the ray with the incoming surface, and to evaluate the flux at the intersection point by some interpolation method from the fine-mesh-centered incoming flux array.

In a MOC code, a ray tracer subroutine is required to calculate the intersection point of a ray with a surface. The coordinates of the points along a ray can be defined as:

$$x = x_0 + t \cdot \mu$$
$$y = y_0 + t \cdot \eta \qquad (2\text{-}19)$$
$$z = z_0 + t \cdot \xi$$

Where (x_0, y_0, z_0) is the starting point of the ray, t is path length along the ray, and (μ, η, ξ) are the direction cosines. We can substitute Eq. 2-19 into a region boundary surface function to evaluate the coordinates of the intersection points of the ray with that surface and the path length t (i.e., s_{ink} in Eqs. 2-16 and 2-18). In the MOC, it can be very expensive, in terms of computer memory, to store the geometry information if the number of rays and the number of regions are very large. For this reason, 3-D MOC could be prohibitive for a large model. The block-oriented characteristics solver considers the whole coarse mesh as one region. Therefore, for Eq. 2-19, the region boundaries become the coarse mesh surfaces. Because the characteristics solver is designed for solving the transport equation in a low scattering medium, across which we can expect that the angular flux along the ray does not change significantly, it is possible to use a relatively large region (i.e. a coarse mesh) for a flat-source MOC formulation.

2.6.4 Interpolation on the Incoming Surface

Based on the positions of the intersection points of rays on the incoming surface of a coarse mesh, we can further evaluate the averaged flux for each fine mesh by interpolation. As shown in Figure 2-6, points *A*, *B*, *C*, and *D* denote the closest 4 neighbors to point *P*, which is the intersection point of a characteristic ray across one incoming boundary. We need to evaluate the angular flux at point *P* based on the fluxes at the 4 neighboring points.

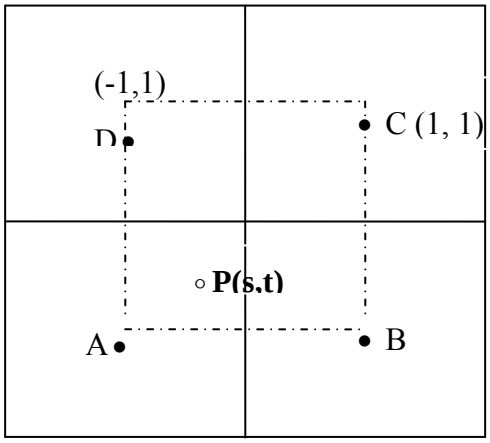


Figure 2-6. Bilinear interpolation for the incoming flux.

For simplification, we assume the coordinates for the 4 neighbors and point *P* are *A(-1,-1)*, *B(1, -1)*, *C(1, 1)*, *D(-1, 1)* and *P(s, t)*, where *s*, *t* are evaluated by the ray tracer. Note

that the actual positions of the fine mesh centers and point *P* are projected into the coordinates shown in Figure 2-6, in which *A*, *B*, *C*, *D* and *P* are located at *(-1,-1)*, *(1,-1)*, *(1,1)*, *(-1,1)* and *(s, t)* for the interpolation. Two interpolation techniques are applied in the TITAN code. Either of them can be used to estimate the incoming flux at point *P*.

- closest neighbor.

ψ_P is equal to the angular flux at the closest neighbor. For example, in Figure 2-6 ψ_P will be equal to the ψ_A under the closest neighbor approach.

- bilinear interpolation.

A bilinear interpolation formulation is applied:

$$\psi(s,t) = \psi(-1,-1)\frac{(1-s)(1-t)}{4} + \psi(-1,+1)\frac{(1-s)(1+t)}{4} + \psi(+1,-1)\frac{(1+s)(1-t)}{4} + \psi(+1,+1)\frac{(1+s)(1+t)}{4} \tag{2-20}$$

Where $\psi(-1,-1) = \psi_A$, $\psi(1,-1) = \psi_B$, $\psi(+1,+1) = \psi_C$, $\psi(-1,+1) = \psi_B$, and $\psi(s,t) = \psi_P$. The truncation error indicates the bilinear approach is a second order interpolation. And it should be more accurate than the first approach, which is a first order interpolation. However, we should note that these point-wise angular fluxes are actually averaged values: fine-mesh-centered fluxes (ψ_A, ψ_B, ψ_C, and ψ_D) are the averaged fluxes on the fine meshes, and the ray intersection-point flux (ψ_P) is the averaged flux on the cross sectional area (δA_{ink} in Eq. 2-18) of the volume the ray represents. An assumption is made that the averaged flux happens at the center of the fine mesh, or at point *P* of the ray cross section area. This assumption is reasonable if the fine mesh is small. Therefore, our ray solver may require a relatively finer meshing on the coarse mesh surfaces, which leads to denser rays in the coarse mesh and longer computer time and memory requirements. On the other hand, if the fine mesh is relatively large, the closest neighbor interpolation scheme is not necessarily less accurate than the advanced bilinear interpolation. The most suitable interpolation scheme could depend on the problem and its modeling. By default, the bilinear interpolation scheme is used in the TITAN code.

In the characteristics solver, the cross sectional area represented by each ray (defined in Eq. 2-18) can by calculated by the following formulation:

$$\delta A_{i,j} = S_{i,j} \times \cos(\theta) \tag{2-21}$$

Where $S_{i,j}$ is the fine mesh area on the outgoing boundary, and θ is the angle between the ray direction and the direction normal to the boundary. Even with a uniform fine meshing applied on the surfaces of a coarse mesh for the characteristics solver, rays are not

necessarily distributed uniformly within the coarse mesh volume, because rays along a certain direction can form different angles with the normal directions of the three incoming surfaces of the coarse mesh. Non-uniform ray distribution could lead to the requirement of denser rays and/or smaller coarse meshes to maintain accuracy of the bi-linear interpolation.

2.7 Quadrature Set

We discussed the formulations for the S_N and characteristics solver, respectively. Our focus has been on the Step 1 of the source iteration scheme, which is to solve the transport equation for the angular flux. For Steps 2 and 3, the formulations are fundamentally the same for both solvers because of the following similarities between two methods:

- Calculate the angular flux, although with different formulations.
- Apply the same energy and angular domain discretization approaches.
- Use the source iteration scheme.

The major difference between the two methods is the discretization method in spatial domain. Both block-oriented solvers share the same goal to calculate the outgoing angular fluxes for a block. However, they complete the task with different formulations of the original LBE. Now we can further demonstrate Step 3 of the source iteration scheme. In both methods, we denote the source term in Eq. 2-5 or Eq. 2-15 by:

$$Q = S_{scattering} + S_{fission} \text{ or } S_{fix} \tag{2-22}$$

For simplification, we omit the index for energy group, direction, and fine mesh (S_N) or region (MOC). In Eq. 2-22, S_{fix} is known as external source. $S_{scattering}$ and $S_{fission}$ can be evaluated from flux moments calculated from the results of the previous iteration.

$$\begin{aligned} S_{scattering}^{(i)} = \sum_{g'=1}^{G}\sum_{l=0}^{L}(2l+1)\sigma_{s,g'\to g,l,x}\{P_l(\mu_n)\phi_{g',l,x}^{(i-1)} + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n)\cdot \\ [\phi_{C\,g',l,x}^{k,(i-1)}\cos(k\varphi_n) + \phi_{S\,g',l}^{k,(i-1)}\sin(k\varphi_n)]\} \end{aligned} \tag{2-23}$$

Where *i* is the iteration index, *g* is the energy group index, l and *k* are the Legendre expansion indices, (μ_n, φ_n) specifies direction *n* in the quadrature set, $\phi_{g',l,x}^{(i-1)}$, $\phi_{C\,g',l,x}^{k,(i-1)}$, and $\phi_{S\,g',l}^{k,(i-1)}$ are the flux moments calculated from the last iteration, which is indexed by *i-1* here, and *x* is the fine mesh index in the S_N formulation, or the region index in the MOC formulation.

The scattering kernel defined by Eq. 2-23 can be expanded to an arbitrary Legendre order if the same order of cross section data is provided. The isotropic fission source and the *k-effective* can be evaluated by Eqs. 2-24 and 2-25 from an outer iteration.

$$S_{fission}^{(j)} = \frac{\chi_g}{k^{(i-1)}} \sum_{g'=1}^{G} \nu\sigma_{f\ g',x} \phi_{g',0,x}^{(j-1)} \tag{2-24}$$

$$k^{(j)} = k^{(j-1)} \cdot \frac{< Q_{fission}^{(j)} >}{< Q_{fission}^{(j-1)} >} \tag{2-25}$$

Where <> denotes the integration over the entire phase space. Note that *j* is the outer iteration index, while in Eq. 2-23 *i* is the inner iteration index. Scattering source is updated after one sweep is completed for each group, while the fission source is updated only after all groups are converged based on the previous fission source.

Equations 2-23 and 2-24 are the formulations for Step 3 in the source iteration scheme. For Step 2, we use a quadrature set to evaluate the integral over angular domain defined in Eqs. 2-2 to 2-4 for flux moments.

$$\begin{aligned} \phi_l &= \frac{1}{8} \sum_{n=1}^{N} w_n \psi_n P_l(\mu_n) \\ \phi_{C,l}^k &= \frac{1}{8} \sum_{n=1}^{N} w_n \psi_n P_l^k(\mu_n) \cos(k\varphi_n) \\ \phi_{S,l}^k &= \frac{1}{8} \sum_{n=1}^{N} w_n \psi_n P_l^k(\mu_n) \sin(k\varphi_n) \end{aligned} \tag{2-26}$$

Here, for simplification, we drop the indices for energy group and fine mesh or region. Direction *n* can be specified by (μ_n, φ_n) where $-1 \le \mu_n \le 1$, $0 \le \varphi_n < 2\pi$, or (μ_n, η_n, ξ_n) where $-1 \le \mu_n, \eta_n, \xi_n \le 1$, $\mu_n^2 + \eta_n^2 + \xi_n^2 = 1$. In order to preserve symmetries, a quadrature set only specifies directions in the first octant ($0 \le \mu_n, \eta_n, \xi_n \le 1$), directions in the other octants can by acquired by changing the signs of μ_n, η_n, and/or ξ_n. For example, $(-\mu_n, -\eta_n, -\xi_n)$ specifies the opposite direction corresponding to direction (μ_n, η_n, ξ_n) in another octant. Direction (μ_n, η_n, ξ_n) and all its seven corresponding directions in other octants have the same weight (w_n). Usually, we keep the total weight for all directions in one octant equal to one. These directions and the associated weights (w_n) are carefully chosen by a quadrature set, so that we can accurately evaluate the moments of direction cosines and the flux moments defined by Eq. 2-26. Other concerns related to the physics of the problems can affect the choice of the directions too. Further discussions are given in Appendix B. Currently, in the TITAN code, we have two types of quadrature sets available: the level-symmetric quadrature (Ref. 6) and the Legendre-Chebyshev quadrature (Ref. 19)

2.7.1 Level-symmetric Quadrature

Figure 2-7 shows a level-symmetric quadrature with an order of 10 (S_{10}). We use a point on the unit sphere to represent a direction. The *xyz* coordinates of the point are the three

direction cosines of the direction. These directions are ordered with a 'triangle shape' formation. To generate a quadrature set, we need to find the direction cosines and the weights for all the directions.

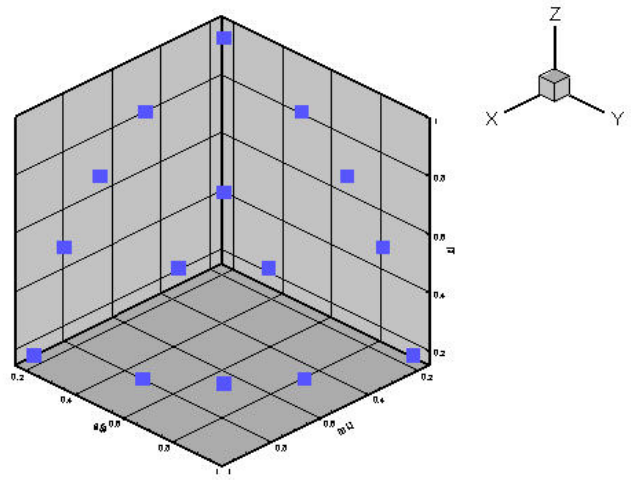


Figure 2-7. Schematic of the S_{10} level-symmetric quadrature set in one octant.

S_{10} specifies 15 directions in the first octant on 5 levels. Directions in the other seven octants are chosen to be symmetric to the directions in the first octant. Therefore, the total number of directions on the unit sphere is $15\times 8=120$ for all 8 octants. Generally, for a level-symmetric quadrature with an order of N, we can calculate the number of levels L, and total number of directions M in the first octant by:

$$L=\frac{N}{2},\ \mathrm{M}=\frac{\mathrm{N}\times(\mathrm{N}+2)}{8} \tag{2-27}$$

To keep a symmetric layout of the directions, N is always chosen from even numbers. The level-symmetric quadrature set is widely used in the S_N codes for its rotation invariance property and preservation of moments. Rotation invariance keeps the quadrature directions unchanged after 90 degree rotation along any axis. In other words, if (μ_n,η_n,ξ_n) is one direction in the first octant of the quadrature set, any combinations of μ_n, η_n, and ξ_n, such as (μ_n,ξ_n,η_n) or (ξ_n,η_n,μ_n), are also defined in the first octant of the quadrature set. Note that rotation invariance is different from octant symmetry of the directions, where $(\pm\mu_n,\pm\xi_n,\pm\eta_n)$ defines the eight symmetric directions in the eight octants. Rotation invariance is very desirable in many real problems to keep the symmetry, especially when reflective boundary conditions are applied. However, it also places a strict constraint on the choice of the quadrature directions. The symmetry condition requires μ_i,η_j,ξ_k for $1\le i,j,k\le\frac{N}{2}$ following the same sequence.

$$\mu_i = \eta_j = \xi_k \text{ for } \quad i, j, k = 1, 2, \cdots, N/2$$
$$\mu_i = \mu_1^{\,2} + C(i-1) \tag{2-28}$$
$$C = \frac{2(1-3\mu_1^{\,2})}{N-2}$$

In Eq. 2-28, only μ_1 is free of choice. The remaining degrees of freedom on direction weights are used to conserve the odd and even moments of μ, η, and ξ.[10]

$$\sum_{m=1}^{M} w_m = 1.0$$
$$\sum_{m=1}^{M} w_m \mu_m^n = \sum_{m=1}^{M} w_m \eta_m^n = \sum_{m=1}^{M} w_m \xi_m^n = 0 \quad \text{for n odd} \tag{2-29}$$
$$\sum_{m=1}^{M} w_m \mu_m^n = \sum_{m=1}^{M} w_m \eta_m^n = \sum_{m=1}^{M} w_m \xi_m^n = \frac{1}{n+1} \quad \text{for n even}, n \le L$$

The directions and their associated weights can be calculated by Eqs. 2-28 and 2-29. Level-symmetric quadrature only can conserve moments to an order of maximum *L=N/2* because of the symmetry condition. Another disadvantage of level-symmetric quadrature is that Eqs. 2-28 and 2-29 lead to negative weights if *N* is greater than 20. Negative weights are not physical. Therefore, they cannot be used. This means that the order of Level-Symmetric quadrature is limited to 20.

2.7.2 Legendre-Chebyshev Quadrature

The Legendre-Chebyshev quadrature, also called P_N-T_N quadrature, aims to conserve moments to a maximum order without the constraints of the symmetry condition. Figure 2-8 shows a P_N-T_N S_{10} quadrature layout.

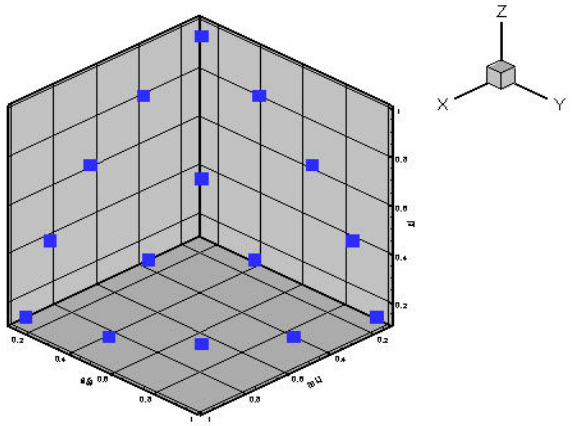


Figure 2-8. P_N-T_N quadrature of order 10.

The Legendre-Chebyshev quadrature conserves moments to the order of *2L-1*, instead of *L* in the level-symmetric quadrature set (*L=N/2*), at the cost of lack of rotation invariance. Moments in Eq. 2-28 cannot be conserved strictly in the P_N-T_N quadrature. Note that Figures

2-7 and 2-8 share a similar triangle-shaped direction layout on the unit sphere, because Eq 2-27 still holds in the P_N-T_N quadrature. The direction weights are positive definite in the P_N-T_N quadrature. Therefore, unlike the level-symmetric quadrature set, the P_N-T_N quadrature order is unlimited mathematically, except for the limitation of computer memory limitation.

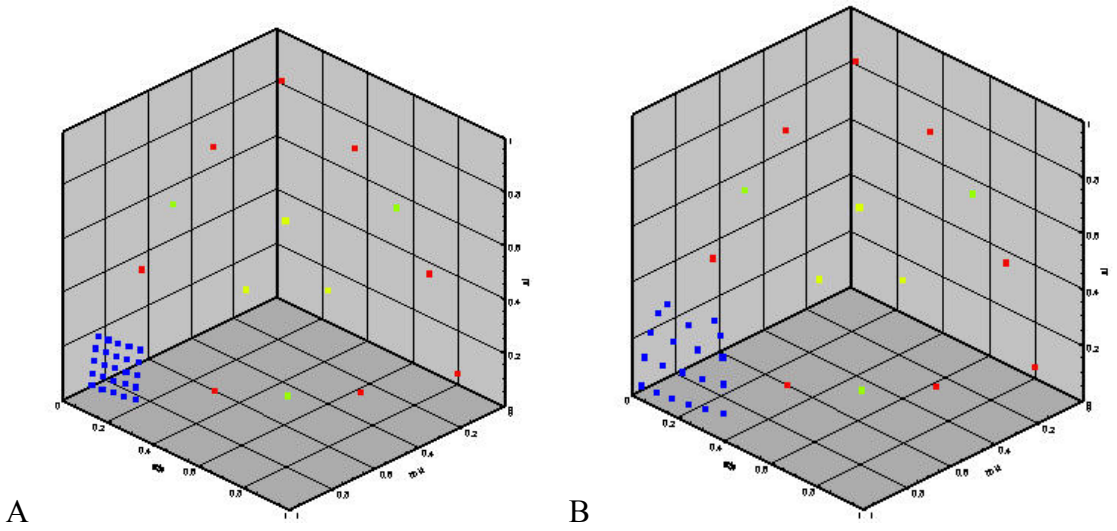
We have derived the procedure on how to build the P_N-T_N quadrature on the unit sphere. Based on the procedure, it can be shown that the P_N-T_N quadrature is the best choice in mathematically conserving higher moments. We also have proved the positivity of weights in P_N-T_N quadrature. Details of the above derivations are given in Appendix B. To build a P_N-T_N quadrature set, it is required to find the roots of an even order Legendre polynomial. These roots are used as level positions of the quadrature. A modified Newton's method is applied. Details of the algorithm also are given in Appendix B.

2.8 Ordinate Splitting

Ordinate splitting is a technique associated with a quadrature set (Ref. 20). A selected direction in a quadrature set can be further split into a number of directions. The total weight of the split directions is equal to the weight of the original direction in the quadrature. We apply the ordinate splitting techniques to solve problems with highly peaked angular-dependent flux and/or source.

2.8.1 Rectangular and Pn-Tn Spliting

Figure 2-9 depicts the two splitting directions for one direction of an S_{10} quadrature set. Note that ordinate splitting technique is independent of choice of quadrature set type or order, and can be applied to as many directions as necessary.



A B

Figure 2-9. Ordinate splitting technique. A) Rectangular splitting. B) P_N-T_N splitting.

In the rectangular splitting technique, the split directions are uniformly distributed within a box-shape region centered at the original quadrature direction. In the TITAN code, the size of the box can be defined by users. The total number of splitting directions can be calculated from the user-specified splitting order with Eq. 2-30.

$$s = (2l-1)^2 \tag{2-30}$$

Where s is the total number of splitting directions, l is the splitting order. Figure 2-9A shows the 25 split directions for a rectangular splitting with an order of 3. All the splitting directions are equal-weighted, defined as $w_s = \frac{1}{s} w_n$, where w_n is the weight of the original direction, which remains in the quadrature set after splitting with a reduced weight.

The rectangular-shaped layout of the split direction may not be efficient in conserving the moments. We developed the Legendre-Chebyshev (P_N-T_N) splitting technique based on the regional angular refinement (RAR) technique.[26] In the P_N-T_N splitting, the original direction can be associated with a local area on the unit sphere surface centered on the original direction. And the range of the area can be decided by users as in the rectangular splitting. The technique projects the directions in the first octant of a regular P_N-T_N quadrature set with an order of $2l$ (l is the splitting order), into the local area. For a regular P_N-T_N quadrature, usually there is only one direction on the top level as shown in Figure 2-8. For the local P_N-T_N quadrature fitted in the splitting technique, users can specify the number of directions on the top level. The number of directions on the following levels increases by one from the previous level, as for a general P_N-T_N quadrature. Therefore, the total number of split directions can be calculated by:

$$s = \frac{(2t + l - 1) \cdot l}{2} \tag{2-31}$$

Where t is user-specified number of directions on the top level, and l is the splitting order. The weights of the split directions are calculated in the same way as a general P_N-T_N quadrature, except that we normalize the total weight to the original direction weight, instead of unity as in a general P_N-T_N quadrature. The split direction weights is calculated by Eq. 2-32.

$$w_s = w_n \cdot w_{S\_P} \cdot w_{S\_T} \tag{2-32}$$

Where w_n is the original weight of the splitting direction, w_{S_P} and w_{S_T} are the level weight and the Chebyshev weight, respectively for one split direction in the local P_N-T_N quadrature. Note that unlike the rectangular splitting, the original splitting direction is dropped off after splitting in the P_N-T_N splitting technique. However, the split directions could be more 'uniformly' distributed within the splitting region than the rectangular splitting,

since it is formed ‘uniformly’ on a sphere surface instead of a rectangular region, and also the P_N-T_N quadrature conserves integrations more accurately than an equal-weighting formulation.

2.8.2 Circular Splitting

Circular Ordinate Splitting (COS) technique is originally developed to simulate the SPECT collimator blurring effect (Ref. 4) as shown in Figure 2-10.

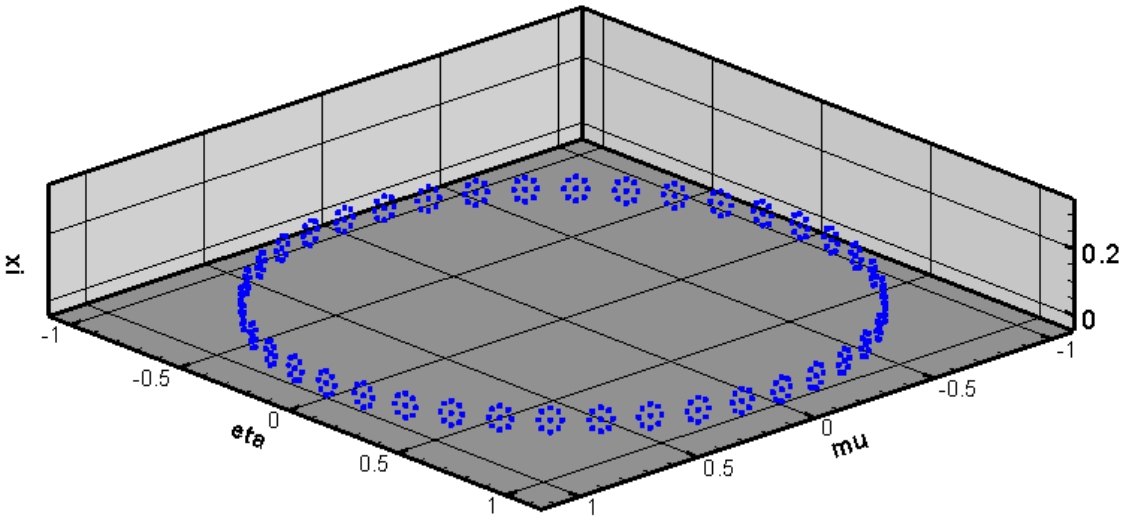


Figure 2-10 Circular ordinate splitting in a fictitious quadrature set (See section 2.10)

In the COS, the split directions are located along a circle centered on the original projection direction. The radius of the circle is calculated based on the SPECT collimator acceptance angle. We use quaternions to mathematically describe rotations and calculate the directional cosines for each direction. By averaging the angular fluxes over the original and split directions, we can simulate part of the projection image blurring effects due to the collimator acceptance angle. Note that particle transport within the collimator is not simulated in the deterministic calculation. In other words, we assume all particles travelling outside the acceptance angle will be absorbed in the collimation septa. As with other ordinate splitting techniques, the number of split directions in the COS is determined by the splitting order. The splitting order is the number of directions on one circle, however, a user can consider more than one circle, and therefore achieve a more refined angular representation.

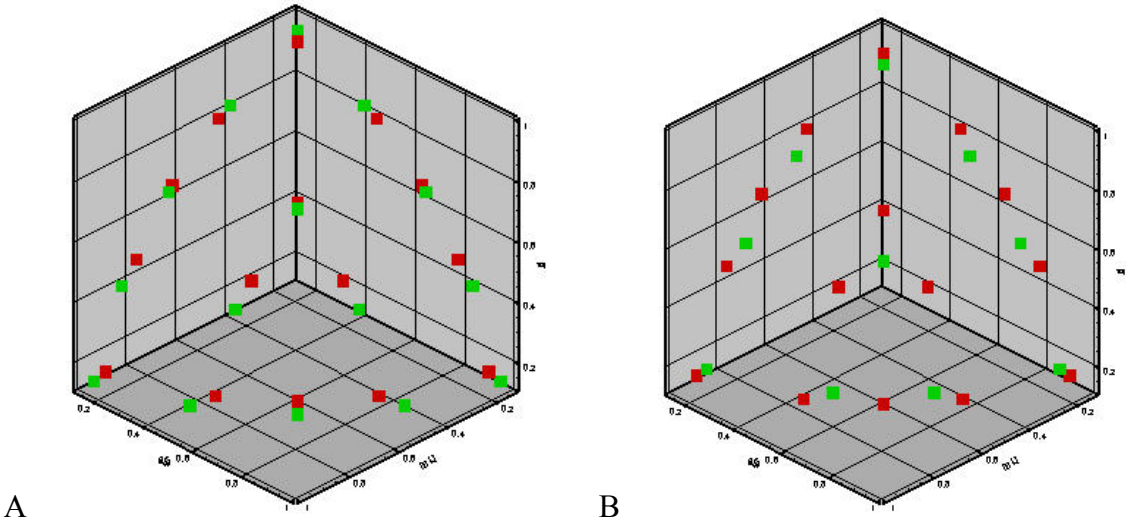
2.9 Projections on the Interface of Coarse Meshes

The TITAN code is built on the multi-block framework with the source iteration scheme. Both the block-oriented S_N and characteristics solvers can apply an individual quadrature set and fine-meshing scheme on each coarse mesh. Transport calculations can benefit from the multi-block framework, which provides users more options on the choices of discretization grids in different regions of a problem model. However, the benefits are not free in term of computational cost. In Step 1 of the source iteration scheme, while sweeping

across the interface of two coarse meshes, we need to project the angular flux on the interface from one frame to the other, if the two coarse meshes use different quadrature sets and/or fine-meshing schemes. Therefore, angular and spatial projection techniques are developed to transfer the interface angular fluxes in the coarse-mesh-level sweep process.

2.9.1 Angular Projection

Angular projection is triggered by the two adjacent coarse meshes with different quadrature sets. Figure 2-11 shows the layout of directions in two quadrature sets.



A B

Figure 2-11. Angular projection. A) Level-symmetric S_{10} (red) to P_N-T_N S_{10} (green). B) S_{10} to S_8.

Figure 2-11A compares the directions for the level-symmetric and P_N-T_N quadrature sets of order 10. Figure 2-11B presents a more general situation of angular projection: from a higher order quadrature to a lower order quadrature, or vice versa. In general, an angular projection from quadrature *P* to quadrature *Q* is used to evaluate the angular fluxes for the directions in quadrature *Q* for each fine mesh on the interface, based on the angular fluxes from quadrature *P*. For each direction Ω_n in quadrature *Q*, we search for the closest three neighboring directions in quadrature *P* to Ω_n. The angular flux for Ω_n can be calculated by a $\frac{1}{\theta^m}$ weighting scheme, where *m* is a positive integer, and θ is the angle between Ω_n and one neighbor direction in quadrature *P*. Note that θ also represents the shortest distance between Ω_n and its neighbor on the surface of a unit sphere. As shown in Figure 2-12, P_1, P_2, and P_3 are the three closest neighbors in quadrature *P* to Ω_n in quadrature *Q*.

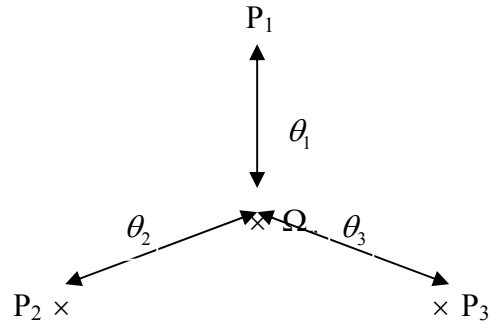


Figure 2-12. Theta weighting scheme in angular domain.

If we consider that the distances between Ω_n and the three closest neighbors are θ_1, θ_2, and θ_3, respectively, then the angular flux at Ω_n can be written as:

$$\psi_{Q_j}^{(m)} = \begin{cases} \psi_{A_i}, & if \quad \theta_i = \min(\theta_1, \theta_2, \theta_3) \le 10^{-4} \\ \dfrac{1}{f^{(m)}} \cdot \left(\dfrac{\psi_{A_1}}{\theta_1^m} + \dfrac{\psi_{A_2}}{\theta_2^m} + \dfrac{\psi_{A_3}}{\theta_3^m} \right) & otherwise \end{cases} \tag{2-33}$$

Where $f^{(m)}$ is the *m'th* normalization factor and defined as $f^{(m)} = \dfrac{1}{\theta_1^m} + \dfrac{1}{\theta_2^m} + \dfrac{1}{\theta_3^m}$. Note that we set the angular flux at Ω_n equal to the closest neighbors, if the minimum distance is less or equal than 10^{-4} radians.

The *0'th* moment (scalar flux) and the first moment (flux current) of the angular flux have to be conserved after an angular projection. Therefore, we need to maintain:

$$\phi = \sum_{i=1}^{N} w_{P_i} \psi_{P_i} = \sum_{j=1}^{M} w_{Q_j} \psi_{Q_j} \tag{2-34}$$

$$J = \sum_{i=1}^{N} w_{P_i} \mu_{P_i} \psi_{P_i} = \sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j} \tag{2-35}$$

Where, N and M are the total number of directions in one octant in quadratures *P* and *Q*, respectively. μ_{P_i} is the cosine of the angle between the interface normal direction and direction *i* in quadrature *P*. μ_{Q_j} is the cosine of the angle between the interface normal direction and direction *j* in quadrature *Q*. And *w*'s are the direction weights. Note that the total weights are set to one for both quadrature sets ($\sum_{i=1}^{N} w_{P_i} = \sum_{j=1}^{M} w_{Q_j} = 1$). In order to evaluate

$\psi_j^{(Q)}$, while conserving the scalar flux and the current, we assume ψ_{Q_j} is a linear combination of $\psi_{Q_j}^{(1)}$ and $\psi_{Q_j}^{(2)}$.

$$\psi_{Q_j} = \alpha \cdot \psi_{Q_j}^{(1)} + \beta \cdot \psi_{Q_j}^{(2)} \tag{2-36}$$

Where, $\psi_{Q_j}^{(1)}$ and $\psi_{Q_j}^{(2)}$ are calculated with Eq. 3-1 with *m=1, 2*, respectively. And α and β are the linear coefficients, which can be evaluated by substituting Eq. 3-4 into Eqs. 3-2 and 3-3.

$$\alpha = \frac{J \cdot \sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(2)} - \phi \cdot \sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(1)}}{(\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(1)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(2)}) - (\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(2)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(1)})} \tag{2-37}$$

$$\beta = \frac{\phi \cdot \sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(2)} - J \cdot \sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(1)}}{(\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(1)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(2)}) - (\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(2)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(1)})} \tag{2-38}$$

Once $\psi_{Q_j}^{(1)}, \psi_{Q_j}^{(2)}$, α, and β are evaluated by Eqs. 3-1, 3-5, and 3-6, ψ_{Q_j} can be calculated by Eq. 3-4. Under this angular projection scheme, the scalar flux and the first flux moment remains the same for each fine mesh on the interface before and after the projection. It is also possible to conserve higher moments at additional computational cost. We can always introduce higher order weighting schemes with Eq. 3-1 (e.g. $\frac{1}{\theta^3}$, $\frac{1}{\theta^4}$), then more terms and coefficients can be added in Eq. 3-4. In order to calculate the linear combination coefficients (α, β, γ etc.), higher moment conservation equations can be introduced besides Eqs. 3-2 and 3-3. Although the scattering source term defined by Eq. 2-23 is calculated with all flux moments up to the order of *L*, generally it is not necessary to conserve flux moments with an order higher than one on the interface, since only the *0'th* and first moments carry physical meanings (scalar flux and flux current), other than just a mathematical term.

In the TITAN code, we also apply a negative fix-up rule to keep the positivity of angular fluxes by relaxing the *0'th* and/or the first moment conservation rule if necessary. The angular projection can be used with any type of the quadrature set. It is also compatible with the ordinate splitting technique. In order to perform a relatively efficient angular projection, it is recommended that both projecting and projected quadrature sets have at least three directions per octant (i.e. at least S_4). If there is only one direction in one octant (i.e. S_2), the direction can be considered as three directions with the same position and only one-third of the original weight, so the above angular projection procedure still can be performed without any modifications.

2.9.2 Spatial Projection

Spatial projection is triggered if the fine-meshing schemes mismatch on the interface of two adjacent coarse meshes. Figure 2-13 shows a projection situation between a 3x3 meshing scheme and a 2x2 meshing scheme.

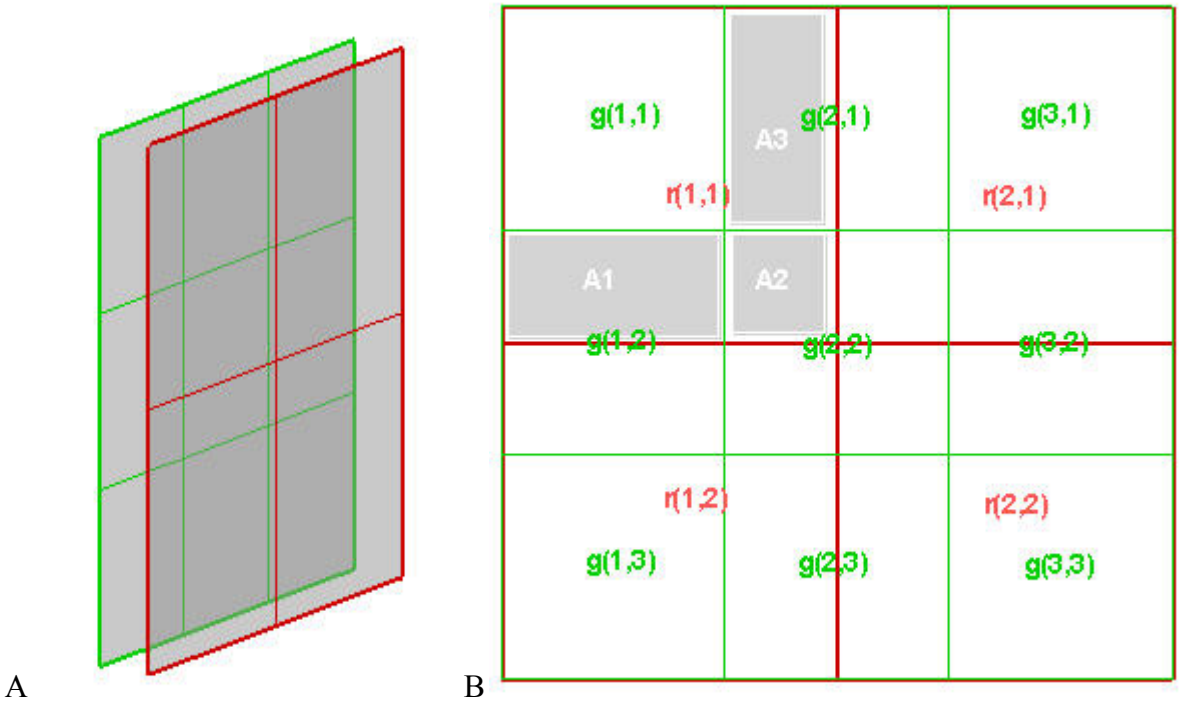


Figure 2-13. Mismatched fine-meshing schemes on the interface of two adjacent coarse meshes. A) 3-D layout. B) 2-D layout.

In Figure 2-13B, we denote the 3x3 fine meshes on the green surface as *g(1,1), g(2,1) ... g(3,3)*, the 2x2 fine meshes on the red surface as *r(1,1), r(2,1) ... r(2,2)*. The average angular fluxes on these fine meshes can be referred to as $\psi_{(g)}(1,1) \to \psi_{(g)}(3,3)$ and $\psi_{(r)}(1,1) \to \psi_{(r)}(2,2)$.

Assuming a green-to-red projection, we need to calculate $\psi_{(r)}(1,1) \to \psi_{(r)}(2,2)$ based on $\psi_{(g)}(1,1) \to \psi_{(g)}(3,3)$ by an area weighting scheme. Here, we only demonstrate how to calculate the angular flux on fine mesh *r(1,1)*. The rest of the red meshes can be evaluated based on the same approach.

$$\psi_r(1,1) = \frac{\psi_g(1,1)\cdot A_g(1,1) + \psi_g(1,2)\cdot A_1 + \psi_g(2,2)\cdot A_2 + \psi_g(2,1)\cdot A_3}{A_g(1,1) + A_1 + A_2 + A_3} \qquad (2\text{-}39)$$
$$= f_g(1,1)\cdot\psi_{(g)}(1,1) + f_g(1,2)\cdot\psi_{(g)}(1,2) + f_g(2,2)\cdot\psi_{(g)}(2,2) + f_g(2,1)\cdot\psi_{(g)}(2,1)$$

Where A_1, A_2, and A_3 are the shade areas in Figure 3-3B. $A_g(1,1)$ is the area of fine mesh *g(1,1)*. Since fine meshes are uniformly distributed on either surface, we can denote

$A_g(1,1) = A_g$. Note that $A_r = A_g(1,1) + A_1 + A_2 + A_3$ is the area of fine mesh *r(1,1)*. Therefore, the factor $f_{(g)}$ can be denoted as:

$$f_{(g)}(1,1) = \frac{A_g}{A_r},\quad f_{(g)}(1,2) = \frac{A_1}{A_r},\quad f_{(g)}(2,2) = \frac{A_2}{A_r},\quad f_{(g)}(2,1) = \frac{A_3}{A_r} \tag{2-40}$$

If we assume a red-to-green projection, $\psi_{(g)}(1,1) \to \psi_{(g)}(3,3)$ will be evaluated based on $\psi_{(r)}(1,1) \to \psi_{(r)}(2,2)$. The same area weighting scheme can be applied:

$$\begin{aligned} &\psi_{(g)}(1,1) = \psi_{(r)}(1,1) \\ &\psi_{(g)}(2,1) = \psi_{(r)}(1,1)\cdot\frac{A_3}{A_g} + \psi_{(r)}(2,1)\cdot\frac{A_g - A_3}{A_g} \end{aligned} \tag{2-41}$$

The area weighting scheme can conserve the angular flux for each fine mesh, assuming a flat flux distribution within fine meshes. Therefore, the total angular flux over the entire interface is conserved automatically. The post re-normalization process described in the angular projection is not necessary in spatial projection. In the TITAN code, we separate the 2-D projection to two single 1-D projections in order to reduce computation cost. For example, a 2-D $3\times 8 \to 6\times 4$ projection can be separated as a $3 \to 6$ projection along *x* axis, and an $8 \to 4$ projection along *y* axis, because *x* and *y* projections are actually independent of each other. Generally, a projection pair, $n \to m$ and $m \to n$, require $2\times n\times m$ memory units to store the geometry meshing factors ($f_{(g)}$, $f_{(r)}$). However, since most of the factors are zeros, we store only the non-zero factors with a sparse matrix for each projection pair. Note that the factors in an $n \to m$ projection remain the same whether they are applied in an *x* or *y* axis projection.

2.9.3 Projection Matrix

Both angular and spatial projections could be expensive in the source iteration scheme, because for every iteration, they are performed whenever the 'sweep' processes cross the interface of two coarse meshes with different angular or spatial frame. If both projections are required on an interface, we perform the angular projection first, then the spatial projection. A projection from coarse mesh *A* to coarse mesh *B* on the interface can be described as

$$\psi_B = P_{AB}\psi_A \tag{2-42}$$

Where P_{AB} is a projection matrix, which stores all the necessary geometry information on the interface. Since projection matrices are independent of angular fluxes, they can be calculated and stored before the sweep process starts.

2.10 Fictitious Quadrature

We introduce a special kind of problems that the TITAN code can be applied: the particle transport problem within a digital medical phantom. To solve a regular transport

problem, modeling of the problem is required as one of the initial tasks. And a meshing scheme need to be carefully chosen based on the physics of the problem. While in a digital phantom, the source and material distributions are stored in the format of voxel values as activity (source) and material attenuation coefficients. Therefore, it is a natural choice to consider one voxel as one fine mesh in the initial modeling task. In the TITAN code, a module is developed to process the digital phantom binary files and automatically generate the meshing scheme. Furthermore, since transport calculations for medical phantoms often involve the simulations of radiation projection images, we developed the fictitious quadrature technique to calculate the angular fluxes for specific directions of interest that may not be available in a regular quadrature set. The performance of the technique is tested in a digital heart phantom benchmark.

2.10.1 Extra Sweep

In the TITAN code, multiple quadrature sets can be used in one problem model. A regular quadrature is built based on the criteria of conservation of flux moments. Fictitious quadrature is designed differently from the regular type of quadrature in that its purpose is to calculate only the angular fluxes for certain directions, not to conserve the flux moments. Therefore, it can not be used in a regular sweep process since the scattering source and flux moments cannot be properly handled. However, it can be used after the source iteration process is complete with the converged flux moments.

Generally, in a transport problem, users' major concern is the scalar flux distribution and/or *k-eff*. However, in some cases, the angular fluxes in the directions of interest need to be evaluated. Since the directions are not necessarily included in the problem quadrature sets, the angular fluxes in these directions usually cannot directly be calculated by the sweep process with a regular quadrature set. In the TITAN code, we can define the directions of interest in a fictitious quadrature set, which is used with an extra sweep process only after the source iteration process is converged with the regular quadrature set(s). The converged flux moments are used to evaluate the scattering source in the extra sweep with the fictitious quadrature.

$$S_{scattering}^{(e.s.)} = \sum_{g'=1}^{G}\sum_{l=0}^{L}(2l+1)\sigma_{s,g'\to g,l}\{P_l(\mu_n^{(fic)})\cdot\phi_{g',l}^{(con)} + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n^{(fic)})\cdot \left[\varphi_{C,g',l}^{k,(con)}\cdot\cos(k\varphi_n^{(fic)}) + \varphi_{S,g',l}^{k,(con)}\cdot\sin(k\varphi_n^{(fic)})\right] \tag{2-43}$$

Where, upper script *(e.s)* stands for *extra sweep, (fic)* for *fictitious*, *(con)* for *converged*. $\phi_{g',l}^{(con)}$, $\phi_{C,g',l}^{k,(con)}$, and $\phi_{S,g',l}^{k,(con)}$ are the converged ℓ^{th} order *regular, cosine* and *sine* flux moments. And $(\mu_n^{(fic)}, \varphi_n^{(fic)})$ specifies a direction in the fictitious quadrature set.

Equation 6-1 is similar to Eq. 2-23, except that we use the converged flux moments after the source iteration process instead of the flux moments from last iteration. And the polar and azimuthal angles refer to a direction in the fictitious quadrature set. The fixed

source or the fission source can be evaluated the same way as in a regular sweep process. After the total source is estimated, we can use the extra sweep process to evaluate the angular fluxes in the directions of the fictitious quadrature.

One also could choose some other methods based on the calculated angular fluxes in the quadrature directions to evaluate the angular fluxes of interest. For example, the angular projection technique in Chapter 3 can be applied with some modifications. We have tried this approach in the TITAN code. Another method could be to apply the Legendre expansion of the angular flux based on the converged flux moments. One potential problem with these two approaches is that their efficiencies are subject to the accuracy of the angular fluxes in the directions of a regular quadrature set. Usually a convergence criterion is set on the scalar flux in the source iteration scheme. The accuracy of the angular fluxes or higher moments is not always granted. And further mathematical manipulations on the angular fluxes or higher moments could introduce more secondary inaccuracies. One advantage of the fictitious quadrature technique over the secondary approaches is that the angular fluxes of interest are calculated directly from a sweep process. And since the sweep process can be considered as a simulation procedure to the physical particle transport phenomenon in certain directions, some physics of the model along the interested directions (e.g. fixed source and scattering) are taken into account in the evaluation process. Thereby, the extra sweep with the fictitious quadrature has more potential to provide an accurate estimation on the interested angular fluxes.

2.10.2 Implementation of Fictitious Quadrature

It is straightforward to implement the fictitious quadrature technique, since all the formulations used in a regular sweep can be applied in the extra sweep. However, due to the special design of the fictitious quadrature, some modifications on the regular sweep are required to effectively complete an extra sweep.

The extra sweep starts upon the completion of the source iteration process. The fictitious quadrature is built as an initialization task before the source iteration starts. Fictitious quadrature sets can be treated as a regular user-defined quadrature set in the initialization process, except that any direction regardless of its octant can be defined in the quadrature input file, and these directions can be arbitrarily chosen. Note that in a regular user-defined quadrature set, only directions in the first octant are defined, and directions in other octants are determined by symmetry. As a result, the extra sweep is performed only along specific directions defined in the first octant. The extra sweep procedure can be illustrated by Figure 2-13.

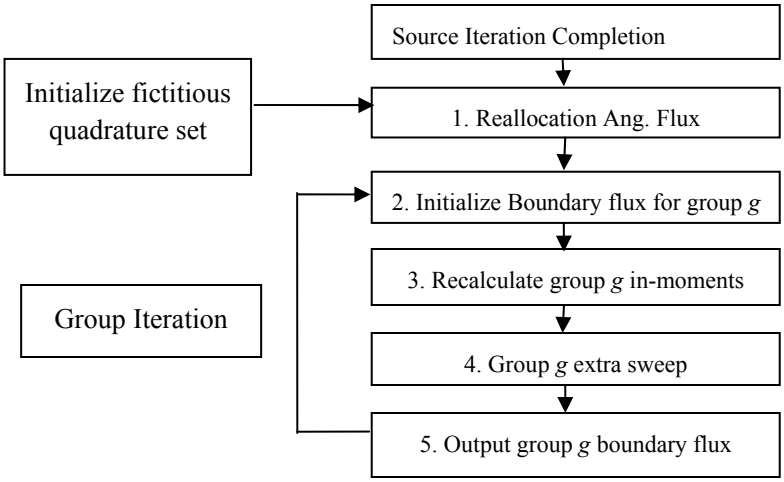


Figure 2-14. Extra sweep procedure with fictitious quadrature.

As shown in Figure 2-14, we start the extra sweep by reallocating the angular flux array based on the fictitious quadrature set. Since the values of angular fluxes in the regular quadrature sets will be lost after the memory reallocation, any task which requires the calculated angular fluxes need to be completed before the extra sweep. At the beginning of the sweep for group *g*, we allocate a new array for the boundary angular fluxes, which will be deallocated after the group *g* sweep. The original boundary fluxes calculated from regular sweep remain untouched during the extra sweep, because an angular projection from the regular quadrature to the fictitious quadrature could be employed on the boundaries if reflective boundary condition is used. We apply the same scattering-in moment approach discussed in Chapter 5 in the extra sweep as well. Note that the scattering-in moments are calculated based on the converged flux moments from regular sweeps, and they are only used for evaluation of the scattering source in an extra sweep. Also note that the step to calculate flux moments in a regular sweep is removed in the extra sweep procedure.

We developed a new set of subroutines to complete the extra sweep. Most of these new routines are on layer 3 or 4, including the angular projection module, the coarse mesh sweep routine, and the differencing scheme routine. Although these subroutines share the similar tasks as their counterparts in the regular sweep, some modifications are required due to the following concerns:

- Iteration structure.
- Direction singularity.
- Solver compatibility.

The iteration architecture in a regular sweep for group *g* is built on the following order (from outer to inner): Octant loop, coarse mesh loop, direction loop, fine mesh loop.

However the characteristics of the fictitious quadrature require that the extra sweep to follow a different order: direction loop, coarse mesh loop, fine mesh loop. This structure change affects most of routines on layer 3 and 4, since all the directions in the same octant are handled as a group in the regular sweep, while in an extra sweep, each direction need to be treated individually. For example, the coarse mesh or fine mesh sweep order is assigned individually for each direction instead by octant. Another modification is made to allow negative directional coordinates in the user-defined fictitious quadrature set.

A regular quadrature set usually avoids directions along an axis or perpendicular to an axis. Zero directional cosine or sine occurs for these directions. This singularity could cause some potential problems in the sweep process. For example, in the differencing scheme discussed in Chapter 2, normally a small perturbation in one boundary incoming angular flux can cast some effect on all the three outgoing fluxes, since the three components of the incoming angular flux along x, y and z axes are all positive definite or all zeros. For a singular direction, however, this is not always true. For example, an incoming angular flux along the x axis only has only one positive x component. Therefore, while calculating the outgoing fluxes, a differencing scheme need to take measures to treat a singular incoming angular flux.

Unfortunately, singular directions often happen to be the interested directions in a fictitious quadrature set. A series of modifications have been made to keep the extra sweep subroutines singularity safe, including the differencing scheme, the fine mesh sweep procedure, and the angular projection routine.

The two-solver structure of the TITAN code causes another dimensional difficulty in the implementation of the fictitious quadrature set. The technique is originally designed for the S_N solver only. Later the compatibility to the characteristics solver is achieved.

Chapter 3 - Code Structure

The fundamental structure of the TITAN code is built on the four steps of the Source Iteration (SI) scheme with the multi-block framework. And the S_N and characteristics solver kernels are integrated in Step 1, in which we apply the 'sweep' process to solve the LBE for angular fluxes. 'Sweep' is a process to calculate the outgoing flux from the incoming flux for a coarse mesh, a fine mesh (S_N), or a region (characteristics) by simulating the particle transport along certain directions. The fine mesh/region averaged angular fluxes are updated during the process. In Step 2, we evaluate the flux moments based on the angular flux calculated in Step 1 by a numerical quadrature set, then use the flux moments to update the source in Step 3 for next iteration. The iteration process continues until fluxes are converged based on a convergence criterion.

In this chapter, first we introduce the overall block structure of the code. Then, we further discuss the transport calculation block, with some details of several key subroutines. Finally, the front-line style sweep process is presented.

3.1 Block Structure

The TITAN code is composed of three major blocks: input, processing, and output. The input block loads the input decks to initialize the model material and the fixed source distribution, meshing scheme, and some control variables. The processing block performs the transport calculation. And the output block handles the calculation results. In this section, we introduce the input and output blocks. The processing block is discussed in the next section.

The input files include the cross-section data file, and a block-structured input deck, to setup some control variables such as quadrature sets and solvers for each coarse mesh. By default, the output block writes up the material number, the source intensity and the calculated scalar flux for each fine mesh into a TECPLOT-format binary data file. The data in this file is organized by coarse meshes. Each data point/fine mesh is composed of an array of values: *xyz* coordinates of the center of the fine mesh, material number and fixed source intensity in the fine mesh, and the average scalar flux for each energy group. Comparing to the ASCII format of the TECPLOT data file, the binary file is smaller in size and faster to load by TECPLOT for various plotting. As an option, the output block can also prepare the input deck for the PENTRAN code.

3.2 Processing Block

The subroutines in the processing block can be roughly arranged in four levels. The lower level routines are called only by the immediate upper level routines. The top level (0^{th} level) routines choose the corresponding module for different types of problems (shielding or criticality). The first level routines setup the source iteration schemes for all energy groups. The second level routines complete one system sweep for all the directions in the quadrature

sets for one group. The third level routines only handle one sweep for all the directions in one octant for one coarse mesh and one group. Finally on the forth level, we apply the S_N or MOC formulations discussed in Chapter 2 to calculate the angular flux in one fine mesh (S_N) or one region (characteristics). Figure 3-1 shows the major subroutines within the four-level code structure. In the following sections, we further discuss some of the routines on each level.

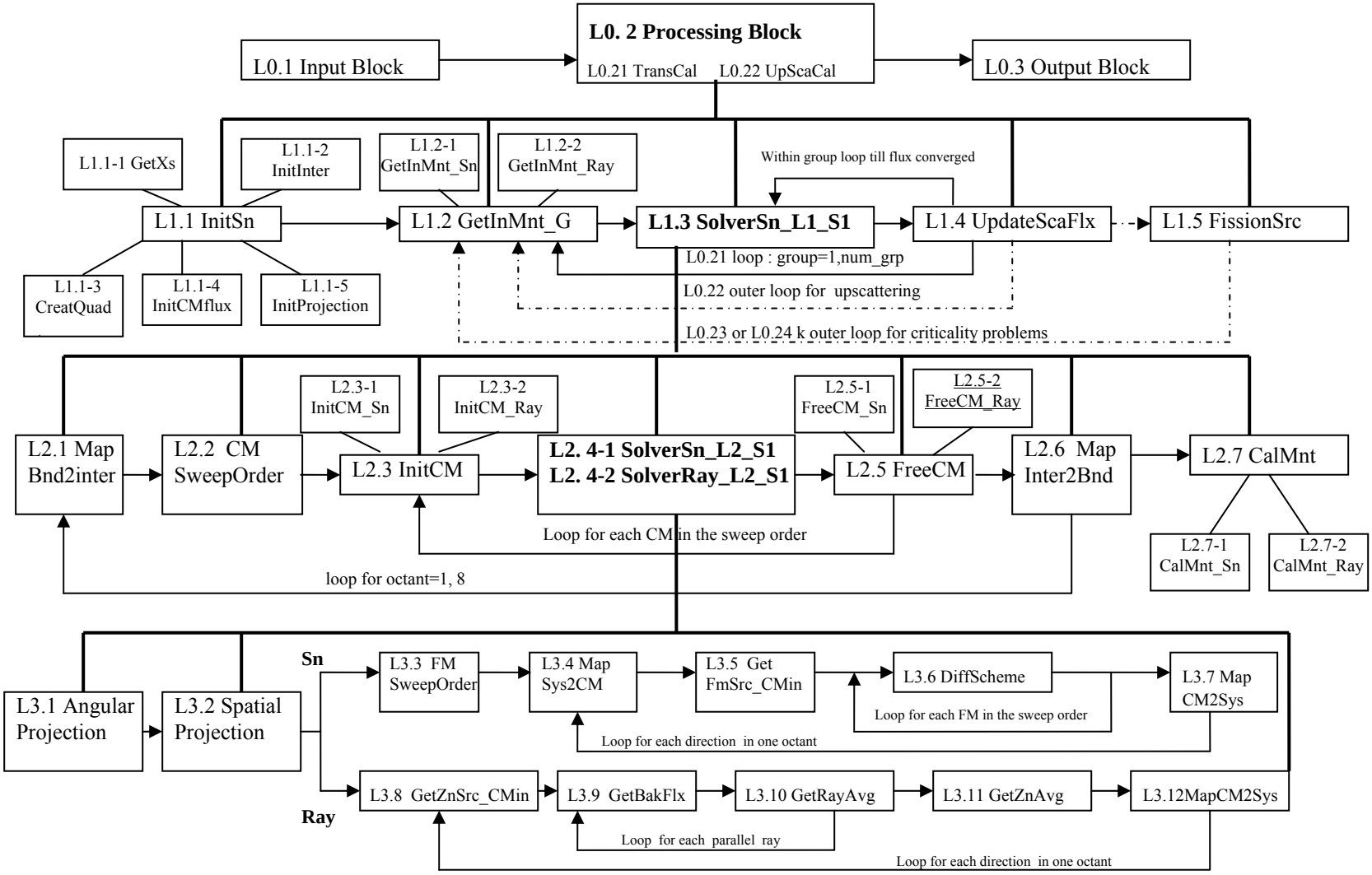


Figure 3-1. Code structure flowchart.

On the top level, TITAN has a simple three-block structure: input block, processing block, and output block. In the processing block, four kernel subroutines are available for different types of problems:

L0.21 TransCal: fixed source problem with only down scattering.

L0.22 UpScaCal: fixed source problem with upscattering.

L0.23 Ksearch: criticality problem with only down scattering.

L0.24 Ksearch_up: criticality problem with upscattering.

Based on some parameters from the input block, we choose one of the four subroutines to perform the transport calculation. *TransCal* provides the fundamental loop structure of the source iteration scheme. Here, we assume that the source iteration scheme starts from the energy group loop. The other three subroutines require one (*L0.22* and *L0.23*) or two (*L0.24*) additional outer loops besides the fundamental source iteration scheme loop structure (*L0.21*). They are designed for problems with upscattering and/or criticality problems.

3.2.1 First Level Routines: Source Iteration Scheme

The flowchart on the first level demonstrates the structure of the processing block. The subroutines on this level can be illustrated in the following pseudo-code.

```
!! Pseudocode: processing block (TransCal, UpScaCal, Ksearch, Ksearch_up)
Call InitSn
Loop outer_k                          ! k loop(power iteration) if eigenvalue problem
    Loop outer_g                      ! outer_g loop if upscattering presents
        For g=1, num_group            ! group loop
            call GetInMnt_G(g)
            while (flux not converged)    ! within group loop
                call SolverSN_L1_S1(g)
                call UpdateScaFlx(g)
            end within group loop
        end group loop
 end outer_g loop                     ! if upscattering presents
   call  FissionSrc                   ! if k loop presents
End outer_k loop
```

Figure 3-2. Pseudo-code of the source iteration scheme.

Subroutine *L1.1 InitSn* is designed to complete the initialization works before the transport calculation starts. This initialization includes loading cross section data, allocating memory for

interface fluxes, angular fluxes, and flux moments, and initialization of the quadrature sets and projection matrices.

Subroutine *L1.2 GetInMnt_G* is called at the beginning of each group loop. And it has only one input argument: group index *g*. *GetInMnt_G(g)* calculates the flux moment summation for all other groups other than group *g*, which we call scattering-in-moments, or in-moments. In-moments are used to efficiently calculate the scattering source, which is performed in Step 3 of the source iteration scheme. By applying the in-moments, we can rewrite Eq. 2-23 by switching the group and Legendre order expansion.

$$
\begin{aligned}
S_{scattering}^{(i)} &= \sum_{g'=1}^{G}\sum_{l=0}^{L}(2l+1)\sigma_{s,g'\to g,l,x}\{P_l(\mu_n)\phi_{g',l,x}^{(i-1)} + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n)\cdot \\
&\quad [\phi_{C\ g',l,x}^{k,(i-1)}\cos(k\varphi_n) + \phi_{S\ g',l}^{k,(i-1)}\sin(k\varphi_n)]\} \\
&= \sum_{l=0}^{L}(2l+1)\{P_l(\mu_n)[\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{g',l,x}^{(i-1)} + \sigma_{s,g\to g,l,x}\phi_{g,l,x}^{(i-1)}] + \\
&\quad 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n)\cos(k\varphi_n)\cdot[\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{C\ g',l,x}^{k,(i-1)} + \sigma_{s,g\to g,l,x}\phi_{C\ g,l,x}^{k,(i-1)}] + \\
&\quad 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n)\cdot\sin(k\varphi_n)\cdot[\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{S\ g',l}^{k,(i-1)} + \sigma_{s,g\to g,l,x}\phi_{S\ g,l}^{k,(i-1)}]\}
\end{aligned}
\tag{3-1}
$$

In Eq. 3-1, the terms of $\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{g',l,x}^{(i-1)}$, $\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{C\ g',l,x}^{k,(i-1)}$, and $\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{S\ g',l}^{k,(i-1)}$ are defined as zero in-moments, cosine in-moments and sine in-moments. Mathematically, this formulation seems more complicated than Eq. 2-23. However, it is more efficient to evaluate scattering source. The in-moments can be pre-calculated before the within-group starts, since they are independent of group *g* moments, which are the only changing moment terms between the within-group loops. Therefore, once the in-moments are pre-calculated by the subroutine *GetInMnt_G*, the summation process over all groups inside the within-group loop reduces to a two-term summation: in-moments plus the group *g* moments.

Inside the subroutine *GetInMnt_G*, we calculate the in-moments for all the coarse meshes. If the characteristics solver is assigned to a coarse mesh, Subroutine *L1.2-2 GetInMnt_ray* is called to calculate the in-moments for each region in the coarse mesh. Otherwise, *L1.2-1 GetInMnt_Sn* is called to calculate the in-moments for each fine mesh within the coarse mesh.

Subroutine *L1.3 Solver_Sn_L1* is the kernel subroutine on this level, which completes one system sweep for a given group *g*. Its structure is illustrated on the next level. Subroutine *L1.4 UpdateScaFlx* is used to calculate the scalar fluxes for the current iteration, and evaluate the maximum difference from the previous iteration. *Solver_Sn_L1* and *UpdateFlx* are the two major

subroutines of the within-group loop. They are repeatedly called until the maximum scalar flux difference between two interations satisfies the user-defined convergence criterion.

L1.5 FissionSrc is called at the end of each *k-effective* loop (power iteration) to update the fission source and the *k-effective* for the next power iteration. The fission source is considered as an isotropic fixed source for all the other inner loops (within-group loop and upscattering loop)**.** Fission source is evaluated for each fine mesh. Then, the *k-effective* is calculated by using Eq. 2-25. More advanced formulas derived from power iteration acceleration techniques can be investigated and applied within the scope of this subroutine.

3.2.2 Second Level Routines: Sweeping on Coarse Mesh Level

The subroutines on this level are called by the kernel subroutine *SolverSN_L1_S1* of the first level. Two inner loops, octant loop and coarse mesh loop are constructed in *SolverSN_L1_S1*. Its structure can be illustrated in the following pseudo code.

```
!! Pseudocode: SolverSn_L1_S1 (group)    !group: energy group index
For octant=1,  8                          ! octant loop
    call MapBnd2inter(octant,group)
    call SweepOrder_cm(octant)
    for cm_ijk in the sweeping order     !coarse mesh loop
        if (MOC solver is assigned to cm_ijk)
            call InitCmRay(cm_ijk)
            call SolverRay_L2_S1(cm_ijk, octant, group)
            call FreeCmRay(cm_ijk)
        else
            call InitCmSn(cm_ijk)
            call SolverSn_L2_S1(cm_ijk, octant, group)
            call FreeCmSn (cm_ijk)
        endif
    end cm loop
   call MapInter2Bnd(octant,group)
end octant loop
call CalMnt(group)
```

Figure 3-3. Pseudo-code of the coarse mesh sweep process.

Subroutines *L2.4-1 SolverRay_L2_S1* and *L2.4-2 SolverSn_L2_S1* are the kernel subroutines, which complete the sweep process within the scope of one coarse mesh for directions in one octant and for a given group by using either the characteristics solver or the S_N solver. The detail structures of the two subroutines are illustrated in the next section.

Subroutines *L2.1 MapBnd2inter* and *L2.6 MapInter2Bnd* are used in the sweep process on the system level. The sweep process starts from the three incoming boundaries of the model for the directions in a given octant, and ends at the three outgoing boundaries. At the incoming surfaces, model boundary conditions need to be applied. And if the outgoing surfaces are reflective or albedo boundaries, the outgoing angular fluxes need to be reflected back as incoming fluxes for directions in another octant. Therefore, at the beginning of the system sweep

process, *MapBnd2inter* is called to map the incoming system boundary conditions to a system interface flux array, while at the end of the sweep process, *MapInter2Bnd* is called to map the system interface flux back to the model boundary.

Subroutine *L2.2 SweepOrder_CM* initializes the coarse mesh sweep order for directions in a given octant before the coarse mesh loop starts. Subroutines *L2.3 InitCM* and *L2.5 FreeCM* are designed to allocate and free memory for the interface flux array within one coarse mesh. More details about the interface flux array will be discussed later. Both *InitCM* and *FreeCM* have two versions corresponding to the characteristics and S_N solver kernel.

Subroutine *L2.7 CalMnt* is called after the system sweep completes. The subroutine is used to evaluate the flux moments (source iteration scheme: Step 2) based on the angular fluxes calculated by the system sweep (source iteration scheme: Step 1).

3.2.3 Third Level Routines: Sweeping on Fine Mesh Level

Two sets of routines are built on this lowest level for the characteristics and S_N solvers, respectively. Both calculate angular fluxes within the scope of one coarse mesh, one octant, and one group. Their structures can be illustrated by the following pseudo code.

```
!! Pseudocode: SolverSn_L2_S1 (cm_ijk, octant, group)
call Projection_H0 (cm_ijk , octant)       ! angular projection
call Projection_D0 (cm_ijk , octant)       ! spatial projection
call SweepOrder_fm(cm_ijk , octant)
For direc=1,  num_direc                    ! direction loop within one octant
    call MapSys2CM(cm_ijk , direc)
    call GetFmSrc_CMin(cm_ijk, octant, direc, group)
    for fm_ijk in the sweeping order     !fine mesh loop
        call DiffScheme
    end fine mesh loop
    call MapCM2Sys(cm_ijk , direct)
end direction loop

!! Pseudocode: SolverRay_L2_S1 (cm_ijk, octant, group)
call Projection_H0 (cm_ijk , octant)       ! angular projection
call Projection_D0 (cm_ijk , octant)       ! spatial projection
For direc=1,  num_direc                    ! direction loop within one octant
    call GetZnSrc_CMin(cm_ijk, octant, direc, group)
    for each parallel ray                  ! ray loop
        call GetBakFlx
        call GetRayAvg
    end ray loop
    call GetZnAvg
    call MapCM2Sys(cm_ijk , direct)
end direction loop
```

Figure 3-4. Pseudo-code of the fine mesh sweep process.

Subroutines *L3.1 Projection_H0* and *L3.2 Projection_D0* complete angular and spatial projection procedures. The two subroutines, called within *SolverSn_L2_S1* and *Solver_Ray_L2_S1*, remap the incoming flux array onto the same frame (in the angular domain and spatial domain) as the current coarse mesh by the projection techniques. Note that here angular projection is performed first if both projections are required.

For the S_N solver, Subroutine *L3.3 SweepOrder_fm* initializes the fine mesh sweep order for the following fine mesh loop. *L3.4 MapSys2CM* and *L3.7 MapCM2Sys* are similar to their counterparts, *L2.1* and *L2.7*, on the second level. However, here we need to map between the system interface flux array and the coarse mesh interface array, instead of between the model boundaries and the system interface flux array.

Subroutine *L3.5 GetFmSrc_CMin* calculates the total source term for each fine mesh before the fine mesh loop starts. Within the fine mesh loop, *L3.6 DiffScheme* is called to calculate the outgoing flux and fine-mesh-averaged flux based on the incoming flux by a differencing scheme. The diamond-differencing and direction-theta-weighted differencing[19] schemes are implemented. Other differencing schemes can be added into this subroutine.

The characteristics subroutine set is similar to the S_N set with a two-level loop structure: direction loop and parallel ray loop, instead of fine mesh loop in the S_N solver. *L3.8 GetZnSrc_CMin*, as its counterpart *L3.5* for the S_N solver, calculates the total source term for each zone, instead of each fine mesh. For each parallel ray, *L3.9 GetBakFlx* evaluates the incoming flux by the bilinear interpolation scheme. *L3.10 GetRayAvg* calculates the average angular flux for the current ray. After all the parallel ray average fluxes are updated, *L3.11 GetZnAvg* is used to calculate the average flux for the zone/coarse mesh. And the coarse mesh outgoing flux is mapped back onto the system interface flux array.

3.3 Data Structure and Initialization Subroutines

The 4-level code flowchart, as outlined in the previous section, is built on the data structure, which organizes of the data arrays, such as angular fluxes and flux moments. In the TITAN code, a number of derived data types are defined by applying the paradigm of object-oriented programming (OOP). These user-defined data objects, such as coarse mesh object, quadrature object, and projection objects, are initialized in subroutine *L1.1 InitSn* at the beginning of transport calculation. In recent years, OOP has already evolved into one standard paradigm for modern coding language for computer applications. While FORTRAN 90/95, designed mainly for scientific computing, generally is not considered as an object-based language. However, FORTRAN 90/95 does provide some tools and language extensions to allow users to utilize some concepts of OOP. And the OOP support is further enhanced in the new FORTRAN 2003 standard.

In the TITAN code, coarse mesh is treated as a relatively independent object, within which a number of parameters, arrays, and sub-object are defined. Among these parameters are *Solver_ID*, *Quad_ID*, *Mat_matrix*, *Src_matrix*, and angular flux and flux moment sub-objects.

Solver_ID and *Quad_ID* specify the solver and quadrature set for the coarse mesh, respectively. *Mat_matrix* and *Src_matrix* are the material and source distributions within the coarse mesh, respectively. And the angular flux and moments for the coarse mesh are defined as sub-objects for each group and octant. They are initialized in subroutine *L1.1-4 InitCMflux*.

Quadrature set is another essential object, which contains the direction cosine values and the weights associated with the directions for each direction in one octant. *L1.1-3 CreatQuad* generates all the quadrature sets with ordinate splitting used in the model. For the level-symmetric quadrature, direction cosines and weights are preset for quadrature order from 2 to 20. For the P_N-T_N quadrature set, since the quadrature order is not limited to 20 as level-symmetric quadrature, directions cosines and weights are pre-calculated by a polynomial root-finding subroutine. After one S_N or P_N-T_N quadrature is created, another subroutine is called to build up the splitting ordinates on top of the regular quadrature set.

As described by Eq. 2-43, the projection matrix should be pre-calculated in both spatial and angular domain. In the spatial domain, *L1.1-5 InitProjection* scans all the coarse mesh interfaces and analyzes all the projections on the interfaces of coarse meshes. Since a 2-D projection is defined by two separated 1-D projections, only a $3 \to 5$ projection matrix is necessary for a projection of $3 \times 3 \to 5 \times 5$. The 2-D projection matrix is built implicitly by the 1-D component projection matrix. Furthermore, 1-D projection matrix is always stored in pair, e.g. $3 \to 5$ and $5 \to 3$, because they always happen together on the same coarse mesh interface depending the sweeping direction. Note that since the same projection could happen in a number of interfaces, it is not necessary to build one projection matrix for every coarse mesh interface. In such case, only one projection matrix is stored to reduce the memory cost. And a projection ID is assigned to each coarse mesh interface to specify the associated projection matrix. The angular projection matrix is built in a similar way, but with a subroutine to find the three closest neighbor directions in one quadrature set to every direction in the other quadrature set. Afterwards, the three neighboring direction indices and the distance weights are stored in an angular projection matrix.

3.4 Coarse and Fine Mesh Interface Flux Handling

In the sweeping process, the fine-mesh interface flux propagates along the sweep direction. Instead of storing all the interface fluxes for each fine mesh, we only store the fluxes on the propagation frontline. As shown in Figure 3-5, for a 2-D coarse mesh with 4 by 4 fine meshes, two one dimensional interface arrays, *Inter_x(:)* and *Inter_y(:)*, can be allocated to store the frontline interface flux, both with a size of 4.

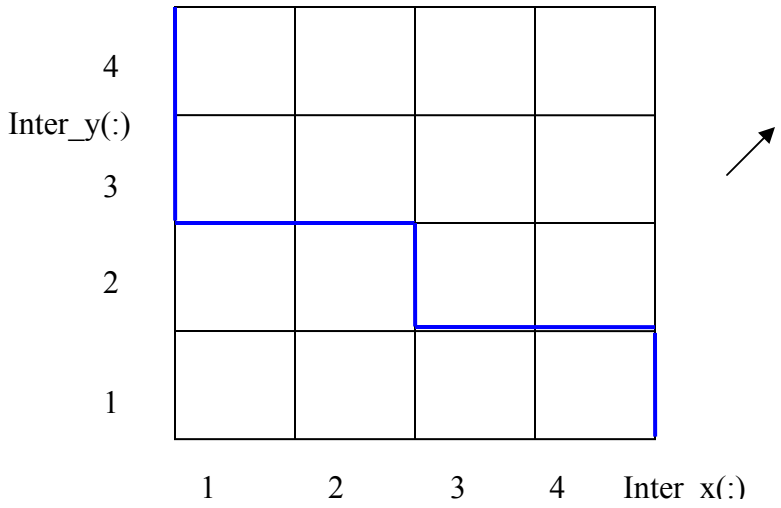


Figure 3-5. Frontline interface flux handling.

At the beginning of the direction *n* sweep process, *Inter_x* and *Inter_y* are assigned to the incoming fluxes at the bottom and left boundary, respectively. This task is completed by subroutine *L3.3 MapSys2CM*. The sweep process starts from *FM (1,1)* by using *Inter_y(1)* and *Inter_x(1)* as incoming fluxes. After the average flux for *FM(1,1)* is updated, we assign the outgoing flux for *FM(1,1)* back into *Inter_y(1)* and *Inter_x(1)*. And the rest of elements of *Inter_x* and *Inter_y* remain the same. Therefore, for *FM(1,2)*, *Inter_x(1)* and *Inter_y(2)* become the incoming fluxes. Generally speaking, for *FM(m,n)*, *Inter_x(m)* and *Inter_y(n)* always store the incoming fluxes before the sweep begins, and the outgoing fluxes afterwards. For example, after the sweep process updates the fluxes for the first 6 fine meshes, the blue line becomes the propagation frontline. At this point, *Inter_x* stores the interface fluxes on the horizontal lines along the blue front line, while *Inter_y* stores all the interface flux on the vertical lines. After all the fine meshes are processed, *Inter_x* and *Inter_y* store the outgoing fluxes for the coarse mesh at the top and right boundaries, respectively.

The front-line approach to handle the fine-mesh interface fluxes can be extended to the sweep process in a 3-D coarse mesh. We use three 2-dimentional arrays to store the interface fluxes: *Inter_xy(:,:)*, *Inter_xz(:,:)*, and *Inter_yz(:,:)*, instead of *Inter_x(:) and Inter_y(:)* in a 2-D coarse mesh. The front-line shown in Figure 4-2 becomes 'front-surface' in 3-D along *x*, *y* and *z* axes.

The front-line approach is memory-efficient compared to the straightforward process to store the interface fluxes for all the fine meshes. Under this approach, only the interface fluxes on the marching front-line are stored. For the case shown in Figure 4-2, the frontline approach only requires 8 memory units, while 40 memory units are necessary otherwise. For a 3-D coarse mesh with $i \times j \times k$ fine meshes, a total of $i \times j \times (k+1) + i \times (j+1) \times k + (i+1) \times j \times k$ memory

units are required if all the interface fluxes are stored. While the front-line approach only requires $i \times j + i \times k + j \times k$ memory units. Another benefit of the frontline approach is to avoid 'memory jumps' for the fine mesh incoming fluxes during the sweep process. As shown in Figure 4-2, the interface flux arrays, *Inter_x(:) and Inter_y(:)*, are always accessed sequentially as the frontline marches forward, which is much more efficient than 'memory jumps', especially when handling large size arrays.

The same approach can be applied on the coarse mesh sweep process, in which a coarse mesh is considered as the finest unit. However, each element of the interface flux array becomes another array, or an object, instead of a scalar value as in the fine mesh sweep process. Here we use another set of object arrays, called system interface arrays *Inter_xy_cm(:,:), Inter_xz_cm(:,:), and Inter_yz_cm(:,:)*, which are similar to *Inter_xy(:,:), Inter_xz(:,:), and Inter_yz(:,:).* They can be considered as an array of arrays, or an array of objects on the system level, which means each element in *Inter_xy_cm(:,:)* is another array, instead of a scalar value as in a regular array. *Inter_xy_cm(:,:)* represents the front-line coarse mesh fluxes on the *xy* plane in the global sweep process, as *Inter_xy(:,:)* represents the front-line fine mesh fluxes in a coarse mesh sweep process. The system interface arrays are initialized by Subroutine *L1.1-2 InitInter*, and connected to coarse mesh interface flux arrays by subroutines *L3.3 MapSys2CM* and *L3.7 MapCM2Sys*, which performs two mapping actions:

- Mapping one system array element to the corresponding coarse mesh interface array as the coarse mesh incoming flux before the fine mesh sweep process starts.
- Mapping the coarse mesh interface array back onto the system array element afterwards as the outgoing flux.

Chapter 4 – I/O Structure

TITAN requires two input files: an input deck to define model geometry, transport parameters etc; and a cross section data file. The input deck file name can be specified at command line by using '-n' option. By default, the file name is 'titan.inp'. The cross section data file name is specified in the input deck with keyword 'xsfile'.

4.1 Input Structure

TITAN input deck uses a 'section-keyword' free style of format. The input deck contains several sections, and each section accepts a number of keywords. Several keywords can be specified in one line, and the entries for one keyword can be spread out to several lines. An input deck can be prepared manually or by the PENMSHXP utility code (Ref. 21).

4.1.1 General Parameters

The first section is to define some general parameters.

```
#0  /general transport parameter quadrature
/acceptable keywords: nquad, tquad, oquad, splitq(multi), ncmesh, numsrc, nummat, numgrp
/ nquad: number of quadratures.
/ tquad: type of quadratures, NumOfEntry=nquad (0=LevelSym, 1=LegendreCheby)
/ oquad: order of quadratures, NumOfEntry=nquad (even number)
/ splitq: ordinate splitting setup
/   splitq=QuadId, #Splitting, SplitDirIDs, SplitOrders, SplitType, Topnums, alpha
/   QuadId=1, 2,..or nquad, #Splitting=num of splitting directions
/   #Splitting= total num of splitting directions in QuadID
/   SplitDirIDs=splitting direction IDs in the base quadrature,NumOfEntry=#Splitting
/   SplitOrder=splitting order for each splitting direction,NumOfEntry=#Splitting
/   SplitType: 0=rectangular, 1=Pn-Tn, 2=circular, NumOfEntry=#Splitting
/   Topnum: rectangular: unused; Pn-Tn: Num of dirs on top level; circular: # of circles.
/  NumOfEntry=#Splitting
/   alpha: rectangular/Pn-Tn: unused (angular range for future version); circular: derail angle.
/   NumOfEntry=#Splitting
/ ncmesh: # of coarse mesh along x, y ,z . NumOfEntry=3
/ numsrc: # of sources . NumOfEntry=1
/ nummat: # of materials . NumOfEntry=1
/ numgrp: # of groups . NumOfEntry=1

        ncmesh=3  3  3
        numsrc=1
        nummat=3
        numgrp=1
        nquad=1
        tquad=1
        oquad=60:
```

On the first line of the above example, '#0' marks Section 0. The following lines are comment lines, which are marked with '/'. The keywords used in this example are explained as follows.

'ncmesh' is the number of coarse meshes along x, y, and z axis. Three entries are required for this keyword. In this case, the model contains 3x3x3 coarse meshes.

'numsrc' is the number of coarse meshes containing fixed sources. In this example, one of the 3x3x3 coarse meshes contains fixed source.

'nummat' is the number of materials in the model

'numgrp' is the number of energy groups

'nquad' is the number of quadrature set used in the model. In this example, only one quadrature is uniformly used for every coarse mesh.

'tquad' is the type of quadrature sets. The number of entries equals to 'nquad'. Two types of quadrature set are available. Type 0 is the level symmetric quadrature set. Type 1 is the Pn-Tn quadrature set.

'oquad' is the order of the quadrature sets. The number of entries equals to 'nquad'.

Another keyword available for this section, but not included in the example, is 'splitq', which is used to specify ordinate splitting parameters.

4.1.2 Geometry Parameters

Some model geometry parameters and coarse mesh properties are defined in this section

```
#1  /Section 1 : Geometry setup
/acceptable keywords: dcpara, xcmbnd, ycmbnd, zcmbnd, cmxfin, cmyfin, cmzfin,
cmsolv,cmdiff,cmquad
/ xcmbnd, ycmbnd, zcmbnd: x,y,z coarse mesh boundaries, NumOfEntry=ncmesh+1
/ cmxfin, cmyfin, cmzfin: fine mesh number along x,y,z for each coarse mesh,
NumOfEntry=TotNumOfCM
/ cmsolv: solver ID for each corase mesh, 0=Sn, 1=Characteristics
/ cmdiff: Differencing Scheme ID for each corase mesh, 1=DD with fixup, 2=DTW
/ cmquad: Quadrature ID for each coarse mesh, available values=1,2,... or nquad

xcmbnd= 0.00000E+00  5.00000E-02  9.50000E-01  1.00000E+00
ycmbnd= 0.00000E+00  5.00000E-02  9.50000E-01  1.00000E+00
zcmbnd= 0.00000E+00  2.50000E-01  4.75000E+00  5.00000E+00
cmxfin=6 18 2R6 18 2R6 18 2R6 18 2R6 18 2R6 18 2R6 18 2R6 18 2R6 18 6
cmyfin=3R6 3R18 6R6 3R18 6R6 3R18 3R6
cmzfin=9R6 9R72 9R6
cmsolv=27R0
cmdiff=27R1
cmquad=27R1
```

This section is marked by '#1'.

‘xcmbnd’ specifies the coarse mesh boundary positions along x axis. The number of entries equals to the number of coarse mesh along x axis (defined in #0 section by ‘ncmesh’) plus one. The unit of entries is centimeter

‘ycmbnd’ specifies the coarse mesh boundary positions along y axis. The number of entries equals to the number of coarse mesh along y axis (defined in #0 section by ‘ncmesh’) plus one.

‘zcmbnd’ specifies the coarse mesh boundary positions along y axis. The number of entries equals to the number of coarse mesh along z axis (defined in #0 section by ‘ncmesh’) plus one.

‘cmxfin’ specifies number of fine meshes along x for all the coarse meshes. The number of entries equals to the number of coarse meshes. Coarse meshes are ordered in a way similar to storing a 3-dimension array in FORTRAN. The 3 dimensions are x, y and z. For example, for a coarse mesh array cm(3,3,3), the first coarse mesh is cm(1,1,1), the second one is cm(2,1,1), and so on. Note that FIDO characters ‘R’ is used in this example. ‘mRn’ means ‘repeat number *n* for *m* times’. If a ray-tracing solver is assigned to a coarse mesh, the fine-meshing is built only on the coarse mesh boundaries. See Section 2.6 for details.

‘cmyfin’ specifies number of fine meshes along y axis for all coarse meshes.

‘cmzfin’ specifies number of fine meshes along z axis for all coarse meshes.

‘cmsolv’ specifies the solver for each coarse meshes. Currently two solvers are available: Solver 0 is the Sn solver, and Solver 1 is the ray-tracing solver.

‘cmdiff’ specifies the differencing scheme used for each coarse mesh. Currently two types of differencing scheme are available. Type 1 is zero-fixed-up diamond, and Type 2 is Directional Theta Weighted.

‘cmquad’ specifies the quadrature set used for each coarse mesh. Quadrature sets are numbered from 1 to ‘nquad’. Each coarse mesh is assigned a quadrature set ID.

4.1.3 Material Distribution

This section, marked by ‘#2’, specifies the material distribution for each coarse mesh. Each material is assigned a material ID from 1 to ‘nummat’ according to the order of its appearance in the cross section data file.

```
#2  /Section 2 : Fine mesh mat. number
/ cmmatn=CM#, mat # for each fine mesh in this CM
cmmatn=1
216R3
cmmatn=2
648R1
cmmatn=3
216R1
cmmatn=4
648R1
cmmatn=5
1944R1
cmmatn=6
648R1
cmmatn=7
216R1
cmmatn=8
648R1
cmmatn=9
216R1
cmmatn=10
2592R1
cmmatn=11
7776R1
cmmatn=12
2592R1
cmmatn=13
7776R1
cmmatn=14
23328R2
cmmatn=15
7776R1
cmmatn=16
2592R1
cmmatn=17
7776R1
cmmatn=18
2592R1
cmmatn=19
216R1
cmmatn=20
648R1
cmmatn=21
216R1
cmmatn=22
648R1
cmmatn=23
1944R1
cmmatn=24
648R1
cmmatn=25
216R1
cmmatn=26
648R1
cmmatn=27
216R1
```

The first entry of ‘cmmatn’ is the coarse mesh number, followed by a sequence of material IDs for each fine mesh (Sn solver) or region (ray-tracing solver) in this coarse mesh. If a ray-tracing solver is assigned a coarse mesh, only one material region is allowed currently. The ordering of fine meshes in a coarse mesh follows the same convention as the coarse mesh ordering. For a coarse mesh with 6x6x6 fine meshes, the material IDs for each fine mesh will be stored in a three dimensional array: fm(6,6,6), where fm(1,1,1) will be the first fine mesh, fm(2,1,1) will be the second, and so on.

Another keyword available in this section is ‘fmsize’. Similar to ‘cmmatn’, the first entry for ‘fmsize’ is the coarse mesh number, followed by a sequence of entries of fine mesh size along x axis, then a sequence of entries of fine mesh size along y axis, then along z axis. This keyword is optional and used to specify non-uniform fine size along an axis. If this keyword is present, fine mesh size is uniform along any axis.

4.1.4 Source Distribution

This section, marked by ‘#3’, specifies the source distribution for any coarse mesh, where a source is present.

```
#3  /Section 3 : src distribution
/acceptable keywords: srcloc, srcmag, srcspm, srcdis (multi), keffin
/ srcloc: source location (CM number where source located), NumOfEntry=numsrc
/ srcmag: source strength (srcmag*srcdis=src density in that fm, #/cm3-sec),
 / NumOfEntry=numsrc
/ srcspm: source spectrum (srcmag*srcdis*srcspm=src density for a group),
NumOfEntry=numsrc*numgrp
/ srcdis: source spatial distribution  srcdis=Src#, FineMeshDist
/ keffin: initial Keff guess, NumOfEntry=1, (more entries reserved for future Keff accelaration

srcloc=1
srcmag=2.16000E+02
srcspm=1.000E+00
srcdis=1
216R4.62963E-03
```

‘srcloc’ specifies the coarse meshes where a fixed source is present. The number of entries equals to ‘numsrc’ defined in Section 0.

‘srcmag’ specifies the source magnitude for each source. The number of entries equals to ‘numsrc’.

‘srcspm’ specifies the source spectrum. The number of entries equals to ‘numsrc’ multiplied by ‘numgrp’.

‘srcdis’ specifies the source distribution in a coarse mesh. Similar to ‘cmmatn’, the first entry is source ID number, followed by a sequence of source density for each fine mesh. For example, srcmag*srcdis(3,3,3)*srcspm(1) is the source density for fine mesh(3,3,3) and for Group 1. Currently, fixed sources can only be defined in a Sn solver coarse mesh.

Another keyword, ‘keffin’, is to specify the initial k-effective. If ‘keffin’ is present, the above keywords are ineffective.

4.1.5 Cross Section Parameter

This section, marked by ‘#5’, specifies the cross section data parameters.

```
#4  /Section 4 : xs data
/acceptable keywords: xsname, xstype, numcmt, xstihm, xstihs,xstiht, legord, legoxs,xstchi
/ xsname: cross section file name (charater entry)
/ xstype: 0=(2l+1) is not pre-multiplied; 1=(2l+1) pre-multiplied
/ numcmt: number of comment lines in between material xs block
/ xstihm: xs table total length, (SigmaTot@Column 3 always
/   xstihm=3+numgrp:  SigmaSelfScatter@4 , downscattering only
/   xstihm=3+(2*numgrp-1):  SigmaSelfScatter@(3+numgrp) , upperscattering

xsname=ps.xs
legord=0  legoxs=0
xstype=1
xstihm=4
xstchi=3R1.0000E+00
numcmt=1
```

‘xsname’ specifies the cross section data file name. Currently, TITAN uses an ASCII row format xs data file, same as the format used by PENTRAN (type 0 and 1). The following is a sample template for a 3-group library with down-scattering only.

```
/material 1 P0
/sig-a    vsig-f    sig-t     sig-s scattering matrix
                              1->1
                              2->2       1->2
                              3->3       2->3     1->3
  /material 1 P1
/sig-a    vsig-f    sig-t     sig-s scattering matrix
                              1->1
                              2->2       1->2
                              3->3       2->3     1->3

…….
```

For every Pn order of each material, a data section is used. Sections are separated by comment line(s). The number of comment lines between sections is specified by keyword ‘numcmt’. The number of rows in each section equals to the number of group. For a P0 section, the first three columns are macro absorption cross section, nu*sigma_fission, and total cross section. For the higher Pn order data, the first three columns are not used, but they should be filled with zeros. The rest of the columns are the scattering matrix. For downscattering only data, the group self scattering is located in Column 4.

The following is a sample template for a 3-group library with upscattering.

```
/material 1 P0
/sig-a    vsig-f    sig-t     sig-s scattering matrix
                              3->1       2->1     1->1
                                         3->2     2->2       1->2
                                                  3->3       2->3     1->3

…….
```

For upscattering xs data, the group self-scattering xs is located in column 3+ ‘numgrp’.

‘legord’ specifies the transport calculation Pn order, and ‘legoxs’ specifies the Pn order of the cross section file. ‘legord’ should not be greater than ‘legoxs’.

‘xstype’ specifies the xs type. In a Type 0 xs file, (*2l+1*) is not factored in the scattering matrix for Pn order greater than 0 sections, while in a Type 1 xs file, (*2l+1*) is pre-multiplied.

‘xstihm’ specifies the total number of columns in the xs file. TITAN uses this entry to decide along with ‘numgrp’ to decide if ‘upscattering’ data is present in the xs file.

‘xstiht’ and ‘xstihs’ are optional, and used to specify the column number of the total xs and the self-scattering xs.

‘xstchi’ specifies the fission chi data. The number of entries equals to ‘numgrp’ multiplied by ‘nummat’.

‘numcmt’ specifies the number of comment lines between sections in the xs file.

4.1.6 Boundary Condition

This section, marked by ‘#5’, specifies the boundary conditions and some iteration control parameters.

```
#5  /Section 5 : boundary cond. and tol.
/acceptable keywords: tolinn,tolout,maxinn,maxout,xminus,xpluss,yminus,ypluss,zminus,zpluss
/ tolinn: inner iteration (within-group) tolerance, negative value: adjustable for keff loop
/ tolout: outer iteration (keff loop) tolerance, negative value: adjustable for keff loop
/ maxinn: maxium inner iteration number, negative value: adjustable for keff loop
/ maxout: maxium outer iteration number, negative value: adjustable for keff loop
/ xminus,xpluss,yminus,ypluss,zminus,zpluss : Boundary conditions at -x,+x, -y,+y, -z,+z
/   =0: vaccum;  =1 albedos for each group: reflective
xminus=0
xpluss=0
yminus=0
ypluss=0
zminus=0
zpluss=0
tolinn= 1.00000E-03
tolout= 1.00000E-05
maxout=10
maxinn=150
```

Two boundary conditions are supported: vacuum and reflective (albedo) on the six surfaces of a problem model.

‘xminus’ specifies the boundary condition on the ‘x-‘ surface. ‘xminus=0’ specifies a vacuum boundary, and ‘xminus=1’ specifies a reflective boundary, followed by a sequence of albedos for each group.

‘xpluss’, ‘yminus’, ‘ypluss’, ‘zminus’, and ‘zpluss’ are the boundary conditions for the other five surfaces.

‘tolinn’ is inner iteration (within-group) tolerance, and ‘tolout’ is outer iteration (k-effective) tolerance

‘maxout’ is the limit of outer iteration number, and ‘maxinn’ is the limit of inner iteration number.

4.1.7 SPECT Section

This section, marked by ‘#10’, is optional, and is used for simulation of SPECT projection images.

```
#10  /Section 10 (Optional) : SPECT
/acceptable keywords: numang,iniang,endang, vecaxs, posaxs, radius, sptcir,detsiz
/ numang: number of projection angles
/ iniang, endang: rotation starting and endding angles in degree
/ vecaxs: rotation axis vector, NumOfEntry=3
/ posaxs: position of rotation vector, NumOfEntry=3
/ radius: rotation radius
/ sptcir: circular splitting for collimators
/   =splitting order(num of dir on one circle), number of circles, collimator angle
/ detsiz: detector size along x and y, in cm and pixels
/   =x-size(cm), y-size(cm), #OfPix along x, #ofPix along y

numang=180
iniang=-9.00000E+01
endang= 2.70000E+02
vecaxs= 0.00000E+00   0.00000E+00   1.00000E+00
posaxs= 2.00000E+01   2.00000E+01   2.00000E+01
radius= 3.00000E+01
sptcir=3  1 1.15350E-02
detsiz= 4.00000E+01   4.00000E+01  128  128
```

‘numang’ specifies the number of projection angles.

;iniang’ is the rotation starting angle in the unit of degree.

‘endang’ is the rotation end angle in the unit of degree. The interval between ‘iniang’ and ‘endang’ will be uniformly divided into ‘numang’ angles.

‘vexaxs’ is a vector to specify the rotation axis.

‘posaxs’ is the origin of ‘vexaxs’ vector

‘radius’ is the rotation radius

'sptcir' is optional, and used to specify the circular ordinate splitting parameters. There are three entries for this keyword. The first entry is the number of directions on one circle; the second entry is the number of circles. The third entry is the radius of the outermost circle in 'radian', and it can be interpreted as the collimator acceptance angle. For details of SPECT simulation, see Ref. 4

This section is designed for SPECT simulation, but it basically asks TITAN to calculate angular fluxes on the model boundary for any given directions. Therefore, it can be applied to any problem as if to 'take pictures' of the model from any angle.

4.2 Output Files

The following table lists the main output files of TITAN.

File name	File Description
read.log	Input processing log
prbname_solver.log	Transport calculation log
Prbname_mix.plt	TECPLOT binary data file, contains all the material, source and calculated flux distributions
Prbname_mix.mcr	TECPLOT macro file, used in TECPLOT to generate various plots
Prbname_quad.dat	ASCII file, quadrature set information

Most of ASICC output files are self-explained. By default, TITAN only output the scalar flux distribution for all groups into a TECPLOT binary data file. TECPLOT is used to open the binary data file, and make various plots of the calculated results. If TECPLOT is not available, users can use '-flx' option to dump the fluxes into a series of ASCII file, named prbname<grp#>.flx. Users can use their preferred utility software to plot the data in these ASCII files. PENMSHXP also can be used to post-processing the .flx files. Users can also use '-d' option in the command line to dump all the flux moments into a binary file. This binary file can be used in a continuous run with the '-c' command line option.

4.3 Command Line Options

TITAN can take a number of command options as listed in following table

Option	Arguments	Description
-i	Folder name	Specifies the input deck and xs data file directory, default is the current directory
-n	Input deck file name	Specifies input deck file name, default is titan.inp

-h	N/A	Displays a 'help' screen
-adj	N/A	Adjoint calculation
-flx	N/A	Outputs .flx files (scalar flux distribution)
-d	N/A	Dumps the flux binary in a binary file named 'prbname.mnt'
-c	N/A	Specifies a continuous run, 'prbname.mnt' should be present in the current directory.

'-adj' option specifies an adjoint calculation. TITAN automatically flips multi-group xs data and transposes the scattering matrix. But it is users' responsibility to reverse the source spectrum specified in the keyword of 'srcspc'. And users should aware that the calculated group fluxes are flipped as well. i.e. The last group flux actually is the first group flux. TITAN is designed this way to increase the users' awareness of an adjoint calculation. Users can use PENMSHXP to flip the flux back into the right order.

APPENDIX A – Scattering Kernel in Linear Boltzmann Equation

Introduction

In the discretized form of the linear Boltzmann equation (Eq. 2-1), the scattering kernel is the most complicated term. In this appendix, we will prove the following formulation:

$$\int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \sigma_s(\vec{r}, E' \to E, \hat{\Omega}' \to \hat{\Omega}) \psi(\vec{r}, E', \hat{\Omega}') =$$
$$\sum_{g'=1}^{G} \sum_{l=1}^{\infty} (2l+1) \sigma_{s,g' \to g,l}(\vec{r}) \{ P_l(\mu) \phi_{g',l}(\vec{r}) + 2 \sum_{k=1}^{l} \frac{(l-k)!}{(l+k)!} P_l^k(\mu) \cdot \qquad \text{(A-1)}$$
$$[\phi_{C,g',l}^k(\vec{r}) \cos(k\varphi) + \phi_{S,g',l}^k(\vec{r}) \sin(k\varphi)] \}$$

In Eq. A-1, the discretization in energy domain can be easily separated with the discretization in the angular domain. The energy and spatial dependency of the scattering source on the left hand side is represented by the flux moment terms ($\phi_{g',l}^k(\vec{r})$, $\phi_{C,g',l}^k(\vec{r})$ and $\phi_{S,g',l}^k(\vec{r})$) on the right hand side. Since the $\int_0^\infty dE' \to \sum_{g'=1}^{G}$ conversion can be achieved straightforwardly by the multigroup approximation, here our main focus is on the conversion of $\int_{4\pi} d\hat{\Omega}' \to \sum_{l'=1}^{\infty}$. For simplicity, we drop the energy group index (*g'* and *g*) and spatial dependency ($\vec{r}$) in the flux moment terms and the cross section moment term. Furthermore, instead of an infinitive Legendre expansion order, we assume a maxim expansion order of *L*. With above simplifications, we can rewrite the formulation to be proved:

$$\int_{4\pi} d\hat{\Omega}' \sigma_s(\hat{\Omega}' \to \hat{\Omega}) \psi(\hat{\Omega}') \approx$$
$$\sum_{l=1}^{L} (2l+1) \sigma_{s,l} \{ P_l(\mu) \phi_l + 2 \sum_{k=1}^{l} \frac{(l-k)!}{(l+k)!} P_l^k(\mu) [\phi_{C,l}^k \cos(k\varphi) + \phi_{S,l}^k \sin(k\varphi)] \} \qquad \text{(A-2)}$$

From now on, we also use the following denotations:

$$\hat{\Omega} \to (\theta, \varphi) \to (\mu, \varphi), \text{ and } \hat{\Omega}' \to (\theta', \varphi') \to (\mu', \varphi') \qquad \text{(A-3)}$$

Where θ is the polar angle with *x* axis, φ is azimuthal angle on the *y*-*z* plane, and $\mu = \cos(\theta)$, $\mu' = \cos(\theta')$. The integration over the unit sphere becomes $\int_{4\pi} d\hat{\Omega}' = \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu = 4\pi$. In some references, for simplicity one can also use

$\int_{4\pi} d\hat{\Omega}' = \int_0^{2\pi} \frac{d\varphi}{2\pi} \int_{-1}^{+1} \frac{d\mu}{2} = 1$. However, we found it is not necessary to make such assumption, and it could cause some confusion in the spherical harmonic expansion. So here we still respect the mathematical fact that the overall solid angle is 4π. Note that with or without this assumption, the formulation of Eq. A-2 should remain the same.

In order to prove Eq. A-2, we need to expand the angular flux and the cross section into a series of Legendre polynomials in the angular domain, respectively. In this appendix, we provide such an expansion for both the angular flux and cross section. By substitute the two expansion series into the left hand of Eq. A-1, we can evaluate the new terms, and finally prove the scattering kernel formulation.

Spherical Harmonic Expansion of the Angular Flux

In this section, we also demonstrate how and why the cosine and sine flux moments are defined. A smooth function defined on the surface of a unit sphere, such as the angular flux $\psi(\hat{\Omega}') = \psi(\mu', \varphi')$, can be expanded by the spherical harmonic function.

$$\psi(\hat{\Omega}') = \psi(\mu', \varphi') = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m Y_n^m(\mu', \varphi') \tag{A-4}$$

The general form of the spherical harmonic function $Y_n^m(\mu', \varphi')$ is defined by:

$$Y_n^m(\mu', \varphi') = \sqrt{\frac{(2n+1)}{4\pi} \cdot \frac{(n-m)!}{(n+m)!}} \cdot P_n^m(\mu') \cdot e^{im\varphi'} \tag{A-5}$$

Where $P_n^m(\mu')$ is the associated Legendre polynomial. The coefficient a_n^m is given by:

$$\begin{aligned} a_n^m &= \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu \, \psi(\mu, \varphi) \overline{Y}_n^m(\mu, \varphi) \\ &= \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu \, \psi(\mu, \varphi) \sqrt{\frac{(2n+1)}{4\pi} \cdot \frac{(n-m)!}{(n+m)!}} \cdot P_n^m(\mu) \cdot e^{-im\varphi} \end{aligned} \tag{A-6}$$

Where $\overline{Y}_n^m(\mu, \varphi)$ is the complex conjugate of $Y_n^m(\mu, \varphi)$.

The angular flux expansion defined by Eq. A-4 should be a real value. So we expect the imaginary part of Eq. A-4 is zero. In order to prove this, we rewrite Eq. A-4 as following:

$$\psi(\mu', \varphi') = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m Y_n^m(\mu', \varphi') = \sum_{n=0}^{\infty} \{a_n^0 Y_n^0(\mu', \varphi') + \sum_{m=1}^{n} [a_n^m Y_n^m(\mu', \varphi') + a_n^{-m} Y_n^{-m}(\mu', \varphi')]\} \tag{A-7}$$

Based on Eq. A.5, we have:

$$Y_n^0(\mu',\varphi') = \sqrt{\frac{2n+1}{4\pi}} P_n(\mu') \tag{A-8}$$

By applying the following identity of the spherical harmonic function,

$$Y_n^{-m}(\mu',\varphi') = (-1)^m \overline{Y}_n^m(\mu',\varphi') \,, \tag{A-9}$$

The coefficient a_n^{-m} can be evaluated as:

$$\begin{aligned}
a_n^{-m} &= \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu \, \psi(\mu,\varphi) \overline{Y}_n^{-m}(\mu,\varphi) \\
&= \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu \, \psi(\mu,\varphi) \cdot \sqrt{\frac{(2n+1)}{4\pi} \cdot \frac{(n+m)!}{(n-m)!}} \cdot P_n^{-m}(\mu) \cdot e^{im\varphi} \\
&= \sqrt{\frac{(2n+1)}{4\pi} \cdot \frac{(n+m)!}{(n-m)!}} \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu \, \psi(\mu,\varphi) \cdot (-1)^{\mathrm{m}} \frac{(n-m)!}{(n+m)!} P_n^m(\mu) \cdot e^{im\varphi} \\
&= (-1)^{\mathrm{m}} \cdot \frac{(n-m)!}{(n+m)!} \cdot \sqrt{\frac{(2n+1)}{4\pi} \cdot \frac{(n+m)!}{(n-m)!}} \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu \, \psi(\mu,\varphi) P_n^m(\mu) \cdot e^{im\varphi} \\
&= (-1)^{\mathrm{m}} \cdot \sqrt{\frac{(2n+1)}{4\pi} \cdot \frac{(n-m)!}{(n+m)!}} \int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu \, \psi(\mu,\varphi) P_n^m(\mu) \cdot \overline{e^{-im\varphi}} \\
&= (-1)^m \overline{a_n^m}
\end{aligned} \tag{A-10}$$

Note in Eq. A-10, we also apply the following identity of the associated Legendre polynomial.[49]

$$P_n^{-m}(\mu) = (-1)^{\mathrm{m}} \frac{(n-m)!}{(n+m)!} P_n^m(\mu) \tag{A-11}$$

According Eqs. A-9 and A-10, the last term in Eq. A-7 can be rewritten as:

$$a_n^{-m} Y_n^{-m}(\mu',\varphi') = (-1)^m \overline{a}_n^m \cdot (-1)^m \overline{Y}_n^m(\mu',\varphi') = \overline{a}_n^m \cdot \overline{Y}_n^m(\mu',\varphi') = \overline{(a_n^m Y_n^m(\mu',\varphi')} \tag{A-12}$$

We substitute Eq. A-12 back to Eq. A-7,

$$\begin{aligned}
\psi(\mu',\varphi') &= \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m Y_n^m(\mu',\varphi') = \sum_{n=0}^{\infty} \{a_n^0 Y_n^0(\mu',\varphi') + \sum_{m=1}^{n} [a_n^m Y_n^m(\mu',\varphi') + a_n^{-m} Y_n^{-m}(\mu',\varphi')]\} \\
&= \sum_{n=0}^{\infty} \{a_n^0 Y_n^0(\mu',\varphi') + \sum_{m=1}^{n} [a_n^m Y_n^m(\mu',\varphi') + \overline{(a_n^m Y_n^m(\mu',\varphi')}]\} \\
&= \sum_{n=0}^{\infty} \{a_n^0 Y_n^0(\mu',\varphi') + 2\sum_{m=1}^{n} \mathrm{Re}[a_n^m Y_n^m(\mu',\varphi')]\}
\end{aligned} \tag{A-13}$$

Here we denote the real part of $a_n^m Y_n^m(\mu',\varphi')$ as $\mathrm{Re}[a_n^m Y_n^m(\mu',\varphi')]$. As we expected, the angular flux is always a real value according Eq. A-13. Now we can further calculate the two terms in Eq. A-13 based on Eqs. A-5 and A-6. The second term is:

$$\begin{aligned}
&\mathrm{Re}[a_n^m Y_n^m(\mu',\varphi')] \\
&= \mathrm{Re}[\{\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi) \sqrt{\frac{(2n+1)}{4\pi}\cdot\frac{(n-m)!}{(n+m)!}}\cdot P_n^m(\mu)\cdot(\cos(m\varphi)-i\sin(m\varphi))\}\cdot \\
&\{\sqrt{\frac{(2n+1)}{4\pi}\cdot\frac{(n-m)!}{(n+m)!}}\cdot P_n^m(\mu')(\cos(m\varphi')+i\sin(m\varphi'))\}] \\
&= \frac{(2n+1)}{4\pi}\cdot\frac{(n-m)!}{(n+m)!} P_n^m(\mu')\cos(m\varphi')\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi)\cdot P_n^m(\mu)\cdot\cos(m\varphi) + \\
&\frac{(2n+1)}{4\pi}\cdot\frac{(n-m)!}{(n+m)!} P_n^m(\mu')\sin(m\varphi')\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi)\cdot P_n^m(\mu)\cdot\sin(m\varphi)
\end{aligned} \tag{A-14}$$

And the first term is:

$$\begin{aligned}
a_n^0 Y_n^0(\mu',\varphi') &= \{\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi)\sqrt{\frac{(2n+1)}{4\pi}}\cdot P_n(\mu)\}\cdot\{\sqrt{\frac{2n+1}{4\pi}} P_n(\mu')\} \\
&= \frac{(2n+1)}{4\pi} P_n(\mu')\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi)\cdot P_n(\mu)
\end{aligned} \tag{A-15}$$

If we define the regular flux moment, cosine moment and sine moment as follows.

$$\phi_n = \frac{1}{4\pi}\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi)\cdot P_n(\mu), \tag{A-16}$$

$$\phi_{C,n}^m = \frac{1}{4\pi}\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi)\cdot P_n^m(\mu)\cdot\cos(m\varphi), \tag{A-17}$$

$$\phi_{S,n}^m = \frac{1}{4\pi}\int_0^{2\pi} d\varphi \int_{-1}^{+1} d\mu\, \psi(\mu,\varphi)\cdot P_n^m(\mu)\cdot\sin(m\varphi), \tag{A-18}$$

We can rewrite Eqs. A-14 and A-15 as follows.

$$\mathrm{Re}[a_n^m Y_n^m(\mu',\varphi')] = (2n+1)\cdot\frac{(n-m)!}{(n+m)!}[P_n^m(\mu')\cos(m\varphi')\phi_{C,n}^m + P_n^m(\mu')\sin(m\varphi')\phi_{S,n}^m] \tag{A-19}$$

$$a_n^0 Y_n^0(\mu',\varphi') = (2n+1) P_n(\mu')\phi_n \tag{A-20}$$

By substituting Eqs. A-19 and A-20 into Eq. A-13, finally we derive the expansion formulation for the angular flux.

$$\psi(\mu',\varphi') = \sum_{n=0}^{\infty}\{a_n^0 Y_n^0(\mu',\varphi') + 2\sum_{m=1}^{n}\mathrm{Re}[a_n^m Y_n^m(\mu',\varphi')]\}$$
$$= \sum_{n=0}^{\infty}(2n+1)\{P_n(\mu')\phi_n + 2\sum_{m=1}^{n}\frac{(n-m)!}{(n+m)!}[P_n^m(\mu')\cos(m\varphi')\phi_{C,n}^m + P_n^m(\mu')\sin(m\varphi')\phi_{S,n}^m]\} \quad \text{(A-21)}$$

One may notice that Eq. A-21 looks similar to Eq. A-4, which is the formulation we need to prove. However, further derivations are still required to reach Eq. A-4. After the integration, μ' and φ' disappear on the right hand side of Eq. A-4. And only μ and φ dependencies are left. At this point, Eq. A-21 is only a function of μ' and φ'. Here we intentionally use *n* and *m* as the index, so that we can distinguish them with *l* and *k,* which we will use in the next section while expanding the cross section term.

The flux moment formulations, Eqs. A-16 to A-18, are equivalent to Eqs. 2-2 to 2-4 we discussed in Chapter 2. Note a 4π factor is used in these formulations.

Scattering Cross Section Expansion and the Spherical Harmonic Addition Theorem

The cross section term in Eq. A-2 can be written as follows.

$$\sigma_s(\hat{\Omega}' \rightarrow \hat{\Omega}) = \sigma_s(\hat{\Omega}'\cdot\hat{\Omega}) = \sigma_s(\mu_0) \quad \text{(A-22)}$$

Since the cross section only depends on the scattering angle. With the notations in Eq. A-3, we can derive the formulation for $\mu_0 = \hat{\Omega}'\cdot\hat{\Omega}$.

$$\hat{\Omega}' = \cos(\theta')\vec{i} + \sin(\theta')\cos(\varphi')\vec{j} + \sin(\theta')\sin(\varphi')\vec{k} \quad \text{(A-23)}$$

$$\hat{\Omega} = \cos(\theta)\vec{i} + \sin(\theta)\cos(\varphi)\vec{j} + \sin(\theta)\sin(\varphi)\vec{k} \quad \text{(A-24)}$$

$$\mu_0 = \hat{\Omega}'\cdot\hat{\Omega} = \cos(\theta)\cos(\theta') + \sin(\theta)\sin(\theta')\cos(\varphi - \varphi') \quad \text{(A-25)}$$

With Eq. A-25, we can apply the spherical harmonic addition theorem.

$$P_l(\mu_0) = P_l(u)P_l(u') + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu)P_l^k(\mu')[\cos(k\varphi)\cos(k\varphi') + \sin(k\varphi)\sin(k\varphi')] \quad \text{(A-26)}$$

Now we can expand Eq. A-22 with the Legendre polynomial.

$$\sigma_s(\mu_0) = \sum_{l=0}^{\infty}\frac{2l+1}{4\pi}\sigma_{s,l}P_l(\mu_0)$$
$$= \sum_{l=0}^{\infty}\frac{2l+1}{4\pi}\sigma_{s,l}\{P_l(u)P_l(u') + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu)P_l^k(\mu')\cdot \quad \text{(A-27)}$$
$$[\cos(k\varphi)\cos(k\varphi') + \sin(k\varphi)\sin(k\varphi')]\}$$

Note we use the 4π factor in Eq. A-27, because usually we assume $\sigma_{s,0}$ is the total scattering cross section. So in case of isotropic scattering, the differential cross section becomes $\sigma_s(\mu_0) = \frac{\sigma_s}{4\pi}$.

Formulation of the Scattering Kernel

So far we have expanded the angular flux with the spherical harmonic function, and the scattering cross section with the Legendre polynomial. In this section, we multiply the two terms together and complete the angular integration. Eventually Eq. A-2 is derived.

We begin with rewriting the two expansion formulations (Eqs. A-21 and A-27) and limiting the expansion order to *L*.

$$\psi(\mu',\varphi') = \sum_{n=0}^{L}(2n+1)\{P_n(\mu')\phi_n + 2\sum_{m=1}^{n}\frac{(n-m)!}{(n+m)!}P_n^m(\mu')[\cos(m\varphi')\phi_{C,n}^m + \sin(m\varphi')\phi_{S,n}^m]\} \quad \text{(A-28)}$$

$$\sigma_s(\mu_0) = \sum_{l=0}^{L}\frac{2l+1}{4\pi}\sigma_{s,l}\{P_l(u)P_l(u') + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu)P_l^k(\mu')[\cos(k\varphi)\cos(k\varphi') + \sin(k\varphi)\sin(k\varphi')]\} \quad \text{(A-29)}$$

When we evaluate $\int_0^{2\pi}d\varphi'\int_{-1}^{+1}d\mu'\psi(\mu',\varphi')\cdot\sigma_s(\mu\cdot\mu')$ using Eqs. A-28 and A-29, all the μ and φ terms can be moved out the integration, and obviously a lot of multiplication terms will appear. Most of the terms become zero. Among the zero terms, some of them are erased by the orthogonal property of Legendre polynomials, others are scratched off by the facts that:

$$\int_0^{2\pi}d\varphi'\cos(m\varphi') = 0 \text{ and } \int_0^{2\pi}d\varphi'\sin(m\varphi') = 0 \text{ for } m=1, 2\ldots \quad \text{(A-30)}$$

We will identify these terms step by step. Here, we refer to the term $P_n(\mu')\phi_n$ in Eq. A-28, and the term $P_l(u)P_l(u')$ in Eq. A-29 as '*the first part*' of the respective equation, and the summation term over *m* or *k* in both equations as *'the second part'*. Now we can apply the orthogonal property of the regular Legendre polynomials.

$$P_l(\mu)\phi_n\int_0^{2\pi}d\varphi'\int_{-1}^{+1}d\mu'P_n(\mu')\cdot P_l(\mu') = P_l(\mu)\phi_n\cdot 2\pi\frac{2\delta_{n,l}}{2l+1} = P_l(\mu)\phi_l\frac{4\pi\delta_{n,l}}{2l+1} \quad \text{(A-31)}$$

Where $\delta_{n,l} = \begin{cases} 1 & l = n \\ 0 & otherwise \end{cases}$

Therefore, all *the first part* multiplication terms become zeros except for those *n=l.* Now we consider *the first part* of Eq. A-28 multiplied by *the second part* of Eq. A-29 (the summation term over *m*). One can observe that these terms become zeros because of Eq. A-30. Similarly, the terms, acquired by multiplying *the second part* of Eq. A-28 with *the first part* of Eq. A-29, become zeros as well.

So far the terms we have not covered are the multiplications of *the second parts* from both Eqs. A-28 and A-29. A common mistake one might make is to assume $\int_{-1}^{+1} d\mu' P_l^k(\mu') P_n^m(u') = C \cdot \delta_{l,n}\delta_{k,m}$. The assumption is very convenient here. Unfortunately, such strict orthogonal relationship for the associated Legendre polynomials can not hold for arbitrary *l*, *k*, *n*, and *m*. However, a relaxed version is always true.[49]

$$\int_{-1}^{+1} d\mu' P_l^m(\mu') P_n^m(\mu') = \frac{2}{2l+1} \cdot \frac{(l+m)!}{(l-m)!}\delta_{l,n} \tag{A-32}$$

In order to apply Eq. A-32, we need to notice the facts that:

$$\int_0^{2\pi} d\varphi' \cos(m\varphi')\cos(k\varphi') = \int_0^{2\pi} d\varphi' \sin(m\varphi')\sin(k\varphi') = \begin{cases} \pi & k = m \\ 0 & otherwise \end{cases} \quad m,k=1, 2\ldots \tag{A-33}$$

$$\int_0^{2\pi} d\varphi' \cos(m\varphi')\sin(k\varphi') = \int_0^{2\pi} d\varphi' \sin(m\varphi')\cos(k\varphi') = 0 \qquad \text{for } m,k=1, 2\ldots \tag{A-34}$$

By using Eqs. A-33 and A-34, we are able to remove all the terms except the terms of $\cos(k\varphi')\cos(m\varphi')$ and $\sin(k\varphi')\sin(m\varphi')$ with *k=m*. Then, we can apply Eq. A-32 on all the remaining terms. In the end, we can conclude that only the terms with *k=m and l=n* will survive among all the second part multiplication terms.

Based on the above explanations, we can write the scattering kernel with all the remaining terms by combining Eqs. A-31 to A-34. Finally, we have proved Eq. A-2.

$$\begin{aligned} &\int_{4\pi} d\hat{\Omega}' \sigma_s(\hat{\Omega}' \to \hat{\Omega})\psi(\hat{\Omega}') \\ &\approx \sum_{l=1}^{L} \frac{(2l+1)^2}{4\pi}\sigma_{s,l}\{P_l(\mu)\phi_l \cdot \frac{4\pi}{2l+1} + 4\sum_{k=1}^{l}[(\frac{(l-k)!}{(l+k)!})^2 \cdot \frac{2}{2l+1} \cdot \frac{(l+k)!}{(l-k)!} \cdot \pi \cdot \\ &P_l^k(\mu)[\phi_{C,l}^k \cos(k\varphi) + \phi_{S,l}^k \sin(k\varphi)]\} \\ &= \sum_{l=1}^{L}(2l+1)\sigma_{s,l}\{P_l(\mu)\phi_l + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu)[\phi_{C,l}^k \cos(k\varphi) + \phi_{S,l}^k \sin(k\varphi)]\} \end{aligned} \tag{A-35}$$

Summary

The energy dependency and its integration can be introduced back into Eq. A-35. And we acquire the multigroup form of the scattering kernel. In the TITAN code, we apply the scattering-in moment form by switching the summation over the group and Legendre order (Eq. 4-1). The switching seems meaningless mathematically. However, it can generate significant benefits in the coding practice. Further discussions on the scattering-in moment form are already given in Chapter 4.

In Eq. A-35, the direction (μ,φ), which is the particle moving direction after a scattering reaction, is not required to be one of the directions in a quadrature set, although this happens to be true in the sweep process with a regular quadrature set. Mathematically, (μ,φ) can be an arbitrary direction in Eq. A-35. We take advantage of this fact in the fictitious quadrature technique we developed in Chapter 6, and also the ordinate splitting technique in Chapter 2. It is not evident to claim that the scattering source evaluated by Eq. A-35 on regular quadrature directions has a higher accuracy than on an arbitrary direction. Nevertheless, the flux moments are always calculated with a regular quadrature set to conserve the integrations in Eqs. A-16 to A-18.

APPENDIX B - Numerical Quadrature on Unit Sphere Surface

Introduction

In the process of solving the linear Boltzmann equation, flux moments need to be evaluated in order to calculate the angular-dependent scattering source term. Flux moment (Eqs. 2-2 to 2-4), by its mathematical nature, is nothing but an integration of a function defined on a unit sphere surface. The function is the angular flux multiplied by a corresponding regular or associated Legendre polynomial. Flux moments become angular independent after the integration over the surface of a unit sphere. The exact distribution of the angular flux on the unit sphere is unknown. However, we can evaluate function values of the angular flux by the sweep process at a given number of points ('discrete ordinates') on the unit sphere. Positions and associated weights of these points are prescribed by a quadrature set. Then, the flux moments can be simply calculated by a summation of the function values multiplied the associated weights.

Quadrature is a simple but powerful numerical integration technique. For example, a Gaussian quadrature with an order of N, can acquire the exact value of the integration of any polynomial up to order of $2N-1$ defined within $[-1, +1]$. In our case, the integration domain is the surface of a unit sphere. Thereby, we need to build a quadrature to evaluate a double integration. Mathematically, a good quadrature of a given order always tends to conserve the integration to the highest order. However, the property of symmetry of a quadrature generally plays a significant role in a physical problem. For example, in a problem with reflective boundaries, we obviously hope all reflected directions of a given direction are also in the quadrature set. Therefore, we often build a quadrature on the balance between keeping symmetry and conserving higher order integration. For example, the level-symmetric quadrature with an order of N can conserve moments only up to the Nth order, but with an excellent symmetry property of rotation invariance. The Legendre-Chebyshev quadrature can conserve moments up to the $2N-1$, but rotation invariance is slightly disturbed.

In this appendix, we prove that the Legendre-Chebyshev quadrature is the best choice in regards to conserving higher moments. Through the discussion of the procedure, hopefully we can cast some insights on how a quadrature is built on the balance of simple mathematics and physics for transport calculations.

General Quadrature Theorem

The popular Gaussian quadrature is built on the orthogonal Legendre polynomial, which is defined on $[-1, +1]$ with a weighting function $w(x)=1$. In general, we can

consider $\{\varphi_n(x) \mid n \geq 0\}$ as the orthogonal polynomials defined on *(a, b)* with a weighting function of $w(x) \geq 0$ for $a < x < b$. According to the orthogonality property, we have:

$$\int_a^b w(x)\varphi_n(x)\varphi_m(x)dx = \begin{cases} 0 & m \neq n \\ \gamma_n & m = n \end{cases} \quad \text{(B-1)}$$

Where $\gamma_n = \int_a^b w(x)[\varphi_n(x)]^2 dx$. We also denote that $\varphi_n(x) = A_n x^n + \cdots\cdots$ and $a_n = \frac{A_{n+1}}{A_n}$.

And the integral of a function *f(x)* can be represented by an *n'th* quadrature formula:

$$I(f) = \int_a^b w(x)f(x)dx \cong \sum_{j-1}^{n} w_{j,n} f(x_{j,n}) = I_n(f) \quad \text{(B-2)}$$

For a given number of nodes, we choose the node positions $\{x_{j,n}\}$ and weights $\{w_{j,n}\}$ in hoping that we can conserve Eq. B-2 as accurate as possible for any *f(x)*. Mathematically, if we assume *f(x)* is a polynomial, this means that the positions and weights of the nodes can hold the integration exactly as the true value to the highest order of the polynomial. In this sense, the nodes and weights can by calculated with Theorem B-1, which is the fundamental guide for building the Legendre-Chebyshev quadrature.

Theorem B-1:

For each $n \geq 1$, there is unique numerical integration formula of degree of precision 2n-1, Assuming f(x) is 2n times continuously differentiable on [a , b], the formula for $I_n(f)$ and its error is given by

$$\int_a^b w(x)f(x)dx = \sum_{j=1}^{n} w_j f(x_j) + \frac{\gamma_n}{A_n^2(2n)!} f^{(2n)}(\eta) \quad \text{(B-3)}$$

For some $a < \eta < b$. The nodes $\{x_j\}$ are the zeros of $\varphi_n(x)$, and the weights $\{w_j\}$ are given by:

$$w_j = \frac{-a_n\gamma_n}{\varphi_n'(x_j)\varphi_{n+1}(x_j)} \qquad j = 1, \ldots, n \quad \text{(B-4)}$$

Legendre-Chebyshev Quadrature on Unit Sphere

Theorem B-1 lays the foundation for building a quadrature set for one-dimensional integration. In order to apply the theorem for a function defined on a unit sphere, we need

to separate the two-dimensional integration of the angular flux into two one-dimensional integrations.

In general, we consider $f(\mu,\varphi)$ is a real smooth function defined on a unit sphere surface, where μ, $-1 \le \mu \le 1$, is the cosine of the polar angle, and φ, $-\pi \le \varphi \le +\pi$ is the azimuthal angle. We need to estimate:

$$I = \int_{4\pi} d\Omega\, f(\mu,\varphi) = \int_{-1}^{+1} d\mu \int_{0}^{2\pi} d\varphi\, f(\mu,\varphi) \tag{B-5}$$

First we define a function of $g(\mu)$:

$$g(\mu) = \int_{0}^{2\pi} d\varphi\, f(\mu,\varphi) \tag{B-6}$$

$$I = \int_{4\pi} d\Omega\, f(\mu,\varphi) = \int_{-1}^{+1} d\mu\, g(\mu) \tag{B-7}$$

The integration defined by Eq. B-7 can be estimated by a Gaussian quadrature, since the weighting function is $w(x) = 1$. Based on Theorem B-1, we choose the quadrature nodes $\{\mu_i\}$ as the roots of the *N'th* Legendre polynomial.

$$P_N(\mu_i) = 0 \tag{B-8}$$

Note we usually choose N as an even integer, so that the roots are symmetrically distributed on the axis. The weights $\{w_i\}$ can be calculated by Eq. B-4. Next we need to determine the function values of $\{g(\mu_i)\}$. $g(\mu_i)$ itself is an integration over a unit circle defined by Eq. B-6. And it can be estimated by another quadrature, in which we still prefer that the quadrature nodes are symmetrically distributed on the four quadrant of a unit circle. Thereby, we separate the integration defined by Eq. B-6 into two parts:

$$g(\mu_i) = \int_{0}^{2\pi} d\varphi\, f(\mu_i,\varphi) = \int_{0}^{\pi} d\varphi\, f(\mu_i,\varphi) + \int_{\pi}^{2\pi} d\varphi\, f(\mu_i,\varphi) \tag{B-9}$$

Now we can consider only the integration over the first half of the unit circle, since nodes on the other half of the circle are decided by symmetry. We denote $g(\varphi) = f(\mu_i,\varphi)$ and $\eta = \cos(\varphi)$. The first part of Eq. B-9 can be rewritten as:

$$\int_{0}^{\pi} d\varphi\, f(\mu_i,\varphi) = \int_{0}^{\pi} d\varphi\, \mathrm{g}(\varphi) = \int_{-1}^{+1} \frac{d\eta}{\sqrt{1-\eta^2}}\, \mathrm{g}(\arccos(\eta)) = \int_{-1}^{+1} \frac{d\eta}{\sqrt{1-\eta^2}}\, \mathrm{h}(\eta) \tag{B-10}$$

Note here $d\varphi = d\arccos(\eta) = \dfrac{-d\eta}{\sqrt{1-\eta^2}}$. And we denote $h(\eta) = g(\arccos(\eta))$.

In Eq. B-10, $w(\eta) = \dfrac{1}{\sqrt{1-\eta^2}}$ is the weighting function for Chebyshev polynomial $T_n(x) = \cos(n \cdot \arccos(x))$. Thereby, we are required to choose the Chebyshev quadrature to evaluate the integration defined by B-10, so that we can precisely estimate the integration if $h(\eta)$ is a polynomial up to the order of *2n-1*. Usually, we choose an even integer for *n*, because we can keep the symmetry on the top half of the unit circle. Figure B-1 shows the roots of $T_4(x)$ on the unit circle.

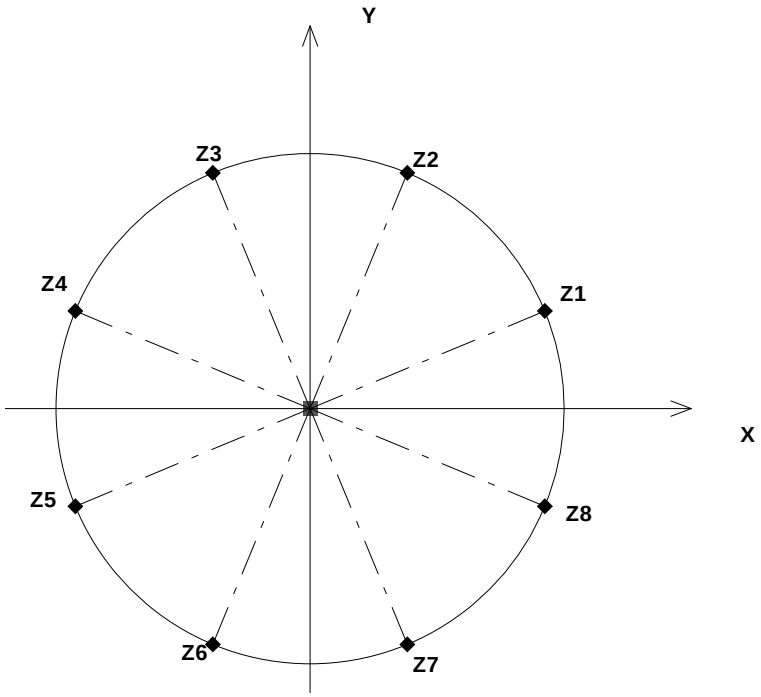


Figure B-1. Chebyshev roots (N =4) on a unit circle.

The *x* coordinates of *Z1-Z4* are the roots of $T_4(x)$. For an even order Chebyshev polynomial, *Z1* and *Z2* are symmetric to *Z3* and *Z4* respectively. *Z5-Z8* are intentionally selected to keep symmetry. As a result, *Z1-Z8* are symmetrically distributed over the four quadrants. Furthermore, the Chebyshev roots are uniformly located on the unit circle, and they are equally weighted by Eq. B-4.

By combining Eqs. B-7 and B-10, the Legendre-Chebyshev quadrature can be built on a unit sphere. However, some physical concerns on symmetry still need to be addressed. Normally, we require the directions in one octant form a 'triangle-shaped' ordering as shown in Figure 2-8 in Chapter 2. And all directions in the other seven octants are decided by symmetry. The 'triangle-shaped' distribution is required to keep the property of 'rotation invariance'. For example, in the level-symmetric quadrature, number of directions per level increases by one from one level to the next. And the choice of the polar axis (*x*, *y*, or *z*) does not affect the distribution of the directions

because the directions are perfectly symmetrical. In the Legendre-Chebyshev quadrature, we cannot keep this 'perfect symmetry' because its priority is to conserve higher moments over rotation invariance. However, we can still keep some 'slightly disturbed symmetry' of rotation invariance by employing the same 'triangle-shaped' direction ordering.

The procedure to build a Legendre-Chebyshev S_{10} quadrature in the first octant can be explained as follows: We choose the five positive roots of $P_{10}(x)$ as the level positions. There is only one direction on the top level. And its position on the circle is decided by the positive root of $T_2(x)$. On the second level, the two positive roots of $T_4(x)$ become the quadrature node positions. The third level node positions are chosen by the three roots of $T_6(x)$, and so on. On the bottom level, five directions are to be defined, which are the positive roots of $T_{10}(x)$. These five level nodes form a triangle-shaped distribution in the first octant. The final layout of the nodes has a quite similar look as the level symmetry quadrature of S_{10}. Figure 2-10A shows the difference of direction distribution between the level-symmetric and Legendre-Chebyshev quadrature with an order of 10.

Newton's Method to Find $P_n(x)$ Roots

In the Legendre-Chebyshev quadrature, the roots of Legendre and Chebyshev polynomials are essential to locate the positions of the quadrature nodes. Chebyshev roots are easy to find since they are uniformly distributed on the unit circle as shown in Figure B-1.

$$T_n(x) = \cos(n \cdot \arccos(x)) = 0 \Rightarrow x_i = \cos\left[\frac{2i-1}{2n}\pi\right] \Rightarrow \varphi_i = \frac{2i-1}{2n}\pi \qquad \text{(B-11)}$$

For a Legendre polynomial $f(x)=P_N(x)$, we apply a variant of Newton's method to find all the positive zeros $\{x_i\}$ in an increasing order as follows.

Step 1: Set initial guess $x_g=0$ for the first (smallest) positive root x_1.

Step 2: For $i=1, 2, \ldots, N$, repeat step 3-5, where N, an even integer, is the polynomial rank.

Step 3: Use Newton's method to find root x_i.

Step 4: Set $f(x) = \dfrac{f(x)}{(x - x_i)}$.

Step 5: Set initial guess $x_g = x_i$ for next root x_{i+1}.

Step 6: Stop

In Step 3 of the above algorithm, the polynomial $f(x)$ and its derivative can be defined as follows.

$$f(x) = \frac{P_N(x)}{\prod_{m=1}^{i-1}(x - x_m)} \tag{B-12}$$

$$f'(x) = \frac{d}{dx}\left(\frac{P_N(x)}{\prod_{m=1}^{i-1}(x-x_i)}\right) = \frac{dP_N(x)}{dx}\left(\frac{1}{\prod_{m=1}^{i-1}(x-x_i)}\right) - \frac{P_N(x)}{\prod_{m=1}^{i-1}(x-x_i)}\left(\sum_{m=1}^{i-1}\frac{1}{x-x_i}\right)$$
$$= \frac{\frac{dP_N(x)}{dx}}{P_N(x)} f(x) - f(x)\left(\sum_{m=1}^{i-1}\frac{1}{x-x_i}\right) \tag{B-13}$$

Then we can apply the following iterative formulation of Newton's method to find root x_i

$$x_i = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{P_N(x_i)}{\frac{dP_N(x_i)}{dx} - P_N(x_i)\left(\sum_{m=1}^{i-1}\frac{1}{x-x_i}\right)} \tag{B-14}$$

In Eq. B-14, $P_N(x)$ and $P_N^{'}(x)$ can be estimated by the recurrence relations of Legendre polynomial defined in Eqs. B-15 and B-16.

$$(n+1)P_{n+1}(x) - (2n+1)xP_n(x) + nP_{n-1}(x) = 0 \tag{B-15}$$

$$(1-x^2)P_n^{'}(x) = -nxP_n(x) + nP_{n-1}(x) = (n+1)xP_n(x) - (n+1)P_{n+1}(x) \tag{B-16}$$

So far we have set up the layout of the directions on the unit sphere by finding roots of *$P_n(x)$* and *$T_n(x)$*. We will further discuss the node weights in the next section.

Positivity of Weights

Another physical concern is the positivity of the node weights. Level-symmetric quadrature is limited to the order of 20, because negative weights occur beyond order 20. In the Legendre-Chebyshev quadrature, the weight for node *i* is calculated by the product of polar weight (level weight) and azimuthal weight.

$$w_i = w_P \cdot w_T \tag{B-17}$$

Both the polar weight w_p and azimuthal weight w_T are calculated by Eq. B-4 with Legendre and Chebyshev polynomials, respectively. First we evaluate the terms in Eq. B-4 for azimuthal weights by applying some Chebyshev polynomial properties.

$$A_n = 2^{n-1} \Rightarrow a_n = \frac{A_{n+1}}{A_n} = 2 \quad \text{and} \quad \gamma_n = \frac{1}{2}\pi \tag{B-18}$$

$$T_n'(x_i) = \frac{(-1)^{i+1} n}{\sin(\varphi_i)} \text{ , and } T_{n+1}(x_i) = (-1)^i \sin(\varphi_i) \tag{B-19}$$

We can substitute Eqs. B-18 and B-19 into Eq. B-4.

$$w_T = \frac{-a_n \gamma_n}{T_n'(x_i) T_{n+1}(x_i)} = \frac{\pi}{n} \tag{B-20}$$

So the Chebyshev nodes are equally weighted. In the TITAN code, we normalize the azimuthal weights on the same level to one. So we simply use normalized weights.

$$w_T = \frac{1}{n}, \tag{B-21}$$

Where *n* is level number. Next we can evaluate the level weights by applying some properties of Legendre polynomial given in Eq. B-22.

$$A_n = \frac{(2n)!}{2^n (n!)^2} \Rightarrow a_n = \frac{A_{n+1}}{A_n} = \frac{[2(n+1)]}{2^{n+1}[(n+1)!]^2} \cdot \frac{(2n)!}{2^n (n!)^2} = \frac{2n+1}{n+1} \text{ and } \gamma_n = \frac{2}{2n+1} \tag{B-22}$$

By substituting Eq. B-22 into Eq. B-4, and applying the recurrence property of Eq. B-16, we can rewrite Eq. B-4 as follows.

$$w_T = \frac{-a_n \gamma_n}{P_n'(x_i) P_{n+1}(x_i)} = -\frac{2}{(n+1) P_n'(x_i) P_{n+1}(x_i)} = \frac{2(1-x_i^2)}{(n+1)^2 [P_{n+1}(x_i)]^2} \tag{B-23}$$

Note in deriving Eq. B-23, we also apply $P_n(x_i) = 0$. Since $0 < x_i < 1$, w_T defined by Eq. B-23 is positive definite. Therefore, unlike the level-symmetric quadrature, the Legendre-Chebyshev quadrature weights are always positive. Furthermore, we can prove that the sum of the weights $\sum_{i=1}^{n} w_i = 2$, because of the following identity of Legendre polynomial.

$$\sum_{i=1}^{n} \frac{1-x_i^2}{(n+1)^2 [P_{n+1}(x_i)]^2} = 1 \tag{B-24}$$

In the Legendre-Chebyshev quadrature, we always choose *n* as an even integer. The roots and weights are symmetrical regarding to *x=0.* We can apply Eqs. B-17, B-21 and B-24 to calculate the total weight for all directions in the first octant.

$$\sum_i w_i = \sum_{n=1}^{N/2} w_n^P \sum_{k=1}^{n} w_k^T = \sum_{n=1}^{N/2} w_n^P \sum_{k=1}^{n} \frac{1}{n} = \sum_{n=1}^{N/2} w_n^P = 1 \tag{B-25}$$

As the level-symmetric quadrature, all the directions in other octants are determined by applying symmetry to the ones in the first octant. We can conclude that the sum of the Legendre-Chebyshev quadrature weights in one octant is equal to one as in the level-symmetric quadrature.

Conclusions

We have proved two very desirable properties of the Legendre-Chebyshev quadrature for transport calculations. First, it can conserve integration up to *2N-1* order. Second, the weights are always positive for any order of the quadrature. However, we do lose some symmetry of rotation invariance. On the other hand, the level symmetry quadrature keeps the perfect symmetry of rotation invariance at the cost of only *Nth* order accuracy and an order limitation of 20. These two quadrature types reflect the trade-off while pursuing mathematical accuracy and physical symmetry.

In the TITAN code, a quadrature set can be further biased by physical concerns. We can apply the ordinate splitting technique (Chapter 2) on some directions with more 'physical importance'. We also developed the fictitious quadrature technique (Chapter 5), which is designed for calculating the angular fluxes in the directions with more 'physical interests'.

LIST OF REFERENCES

1. CE YI, "Hybrid Discrete Ordinates and Characteristics Method for Solving the Linear Boltzmann Equation," PhD Thesis, University of Florida (2007).
2. C. YI and A. HAGHIGHAT, "A Hybrid Block-Oriented Discrete Ordinates and Characteristics Method Algorithm for Solving Linear Boltzmann Equation" *Proc. Int.Conf. M&C* Monterey, CA (2007*)*.
3. C. YI and A. HAGHIGHAT, "Accuracy of TITAN Based on a New OECD-NEA Benchmark over a Range in Parameter Space", Submitted and accepted to *International Topical Meeting on Mathematics & Computation Methods, and Reactor Physics (*2009*)*
4. C. YI and A. HAGHIGHAT, "Hybrid Discrete Ordinate and Ray-tracing with Fictitious Quadrature for Simulation of SPECT", Submitted and accepted to *International Topical Meeting on Mathematics & Computation Methods, and Reactor Physics (*2009*)*
5. C. Yi and A. HAGHIGHAT, "Parallel Performance of a Hybrid Discrete Ordinate and Characteristics Algorithm," AMERICAN NUCLEAR SOCIETY *TRANSACTIONS*, VOL. 98 (2008).
6. C. YI and A. HAGHIGHAT "A 3-D Block-Oriented Hybrid Discrete Ordinates and Characteristics Method" *Submitted to Nuclear Science and Engineering (accepted for publication) (*2009*)*
7. E. E. LEWIS and W. F. MILLER, *Computational Method of Neutron Transport*, John Wiley & Sons, New York (1984).
8. B. G. CARLSON and K.D. LATHROP, "Discrete Ordinates Angular Quadrature of the Neutron Transport Equation," LA-3186, Los Alamos National Laboratory (1965).
9. J. R. ASKEW, "A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries," AEEW-M1108, United Kingdom Atomic Energy Authority (UKAEA), Winfrith (1972).
10. M. D. BROUGH and C.T. CHUDLEY, "Characteristic Ray Solution of the Transport Equation," *Advances in Nuclear Science and Technology Yearbook* (1980).
11. G. E. SJODEN and A. HAGHIGHAT, "PENTRAN: Parallel Environment Neutral-particle TRANsport S_N in 3-D Cartesian Geometry - User Guide Version 9.30c," University of Florida (2004).
12. B. PETROVIC and A. HAGHIGHAT, "Analysis of Inherent Oscillations in Multidimensional S_N Solutions of the Neutron Transport Equation," *Nucl. Sci. Eng.*, **124**, 31 (1996).
13. A. M. KIRK, "On the Propagation of Rays in Discrete Ordinates," *Nucl. Sci. Eng.*, **132**, 155 (1999).
14. K. D. LATHROP, "Spatial Differencing of the Transport Equation: Positivity vs. Accuracy," *J. Comput. Phys.*, **4,** 475 (1969).
15. W. RHOADES and W. ENGLE, "A New Weighted Difference Formulation for Discrete Ordinates Calculations," *Trans. Am. Nucl. Soc.*, **27**, 776 (1977).

16. B. PETROVIC and A. HAGHIGHAT, "New Directional Theta-Weighted S_N Differencing Scheme," *Trans. Am. Nucl. Soc.*, **73**, 195 (1995).
17. G. E. SJODEN and A. HAGHIGHAT, "The Exponential Directional Weighted (EDW) Differencing Scheme in 3-D Cartesian Geometry," *Proc. Int. Conf. on Mathematical Methods and Supercomputing for Nuclear Applications (M&C 1997)*, Saratoga Springs, NY, American Nuclear Society (1997).
18. G. E. SJODEN, "An Efficient Exponential Directional Iterative Differencing Scheme for 3-D S_N Computations in XYZ Geometry," *Nucl. Sci. Eng.*, **155**, 179 (2007).
19. B. G. CARLSON, "Transport Theory: Discrete Ordinates Quadrature over the Unit Sphere," LA-4554, Los Alamos National Laboratory (1970).
20. G. LONGONI and A. HAGHIGHAT, "Development of the Regional Angular Refinement and Its Application to the CT-Scan Device," *Trans. Am. Nucl. Soc.*, **86,** 246 (2002).
21. C. YI, "PENMSH XP manual: A Mesh Generator to Build PENTRAN Input Deck with Compatibility to PENMSH," University of Florida (2007).