HYBRID DISCRETE ORDINATES AND CHARACTERISTICS METHOD FOR SOLVING THE LINEAR BOLTZMANN EQUATION

By

CE YI

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2007

To my Mom, Dad and Brother

ACKNOWLEDGMENTS

I thank my advisor Dr. Alireza Haghighat for his instructions and guidance. Without his support, the accomplishment of this research work would have not been possible. And I am grateful to Dr. Glenn Sjoden for his insightful suggestions. His PENTRAN code manual has served as a constant source of knowledge throughout my studies. I also wish to express my gratitude to other committee members, Dr. David Gilland, Dr. John Wagner, Dr. Jayadeep Gopalakrishnan, and Dr. Shari Moskow, for their help and support.

I would like to gratefully acknowledge Mr. Benoit Dionne, Mr. Mike Wenner, and other members of the transport theory group at University of Florida for their support, especially Benoit for his understanding of this work. The discussions with him on various topics inspired me finding ways to improve the performance of my transport code.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

HYBRID DISCRETE ORDINATES AND CHARACTERISTICS METHOD FOR SOLVING THE LINEAR BOLTZMANN EQUATION

By

Ce Yi

August 2007

Chair: Alireza Haghighat
Major: Nuclear Engineering Sciences

With the ability of computer hardware and software increasing rapidly, deterministic methods to solve the linear Boltzmann equation (LBE) have attracted some attention for computational applications in both the nuclear engineering and medical physics fields. Among various deterministic methods, the discrete ordinates method (S_N) and the method of characteristics (MOC) are two of the most widely used methods. The S_N method is the traditional approach to solve the LBE for its stability and efficiency. While the MOC has some advantages in treating complicated geometries. However, in 3-D problems requiring a dense discretization grid in phase space (i.e., a large number of spatial meshes, directions, or energy groups), both methods could suffer from the need for large amounts of memory and computation time.

In our study, we developed a new hybrid algorithm by combing the two methods into one code, TITAN. The hybrid approach is specifically designed for application to problems containing low scattering regions. A new serial 3-D time-independent transport code has been developed. Under the hybrid approach, the preferred method can be applied in different regions (blocks) within the same problem model. Since the characteristics method is numerically more efficient in low scattering media, the hybrid approach uses a block-oriented characteristics solver in low scattering regions, and a block-oriented S_N solver in the remainder of the physical model.

In the TITAN code, a physical problem model is divided into a number of coarse meshes (blocks) in Cartesian geometry. Either the characteristics solver or the S_N solver can be chosen to solve the LBE within a coarse mesh. A coarse mesh can be filled with fine meshes or characteristic rays depending on the solver assigned to the coarse mesh. Furthermore, with its object-oriented programming paradigm and layered code structure, TITAN allows different individual spatial meshing schemes and angular quadrature sets for each coarse mesh. Two quadrature types (level-symmetric and Legendre-Chebyshev quadrature) along with the ordinate splitting techniques (rectangular splitting and P_N-T_N splitting) are implemented. In the S_N solver, we apply a memory-efficient 'front-line' style paradigm to handle the fine mesh interface fluxes. In the characteristics solver, we have developed a novel 'backward' ray-tracing approach, in which a bi-linear interpolation procedure is used on the incoming boundaries of a coarse mesh. A CPU-efficient scattering kernel is shared in both solvers within the source iteration scheme. Angular and spatial projection techniques are developed to transfer the angular fluxes on the interfaces of coarse meshes with different discretization grids.

The performance of the hybrid algorithm is tested in a number of benchmark problems in both nuclear engineering and medical physics fields. Among them are the Kobayashi benchmark problems and a computational tomography (CT) device model. We also developed an extra sweep procedure with the fictitious quadrature technique to calculate angular fluxes along directions of interest. The technique is applied in a single photon emission computed tomography (SPECT) phantom model to simulate the SPECT projection images. The accuracy and efficiency of the TITAN code are demonstrated in these benchmarks along with its scalability. A modified version of the characteristics solver is integrated in the PENTRAN code and tested within the parallel engine of PENTRAN. The limitations on the hybrid algorithm are also studied.

CHAPTER 1
INTRODUCTION

Overview

The linear Boltzmann equation (LBE) (also called neutron transport equation) describes the behavior of neutral particles in a system (e.g. a nuclear reactor, a radiological medical device). LBE is derived based on the physics of particle balance in a phase space composed of energy, spatial and angular domains. By solving the LBE, we can acquire some insights into the characteristics of the system. In this work, we developed a hybrid transport algorithm to solve the LBE, specifically for application to problems containing regions of low scattering. A new deterministic transport code (TITAN) has been developed based on the new hybrid approach. The code, over 16,000 lines at present, is written in FORTRAN 95 with some language extensions of object-oriented features (part of the FORTRAN 2003 standard). TITAN is benchmarked for several problems.

Linear Boltzmann Equation

The original Boltzmann equation is derived for molecular dynamics of sufficiently dilute gas, in which only binary interaction is considered.[1]

$$(\frac{\partial}{\partial t} + \vec{v} \cdot \nabla + \frac{\vec{F}}{m}\frac{\partial}{\partial v}) f(\vec{r},\vec{v},t) = (\frac{\partial f}{\partial t})_{collision} \tag{1-1}$$

Where,

$\vec{v}$ = the velocity of gas molecules.
$\vec{r}$ = the position of gas molecules.
$f(\vec{r},\vec{v},t)d\vec{r}d\vec{v}$ = the expected number of gas molecules in phase space $d\vec{r}d\vec{v}$.
$\vec{F}$ = external force on molecules.
m = mass of molecules.

Since only binary collision is considered, the collision term on the right side of Eq.1-1 can be written as:

$$(\frac{\partial f}{\partial t})_{collision} = \int d\vec{v}_1 \int d\hat{\Omega} |\vec{v}_1 - \vec{v}| \cdot \sigma(\hat{\Omega}, |\vec{v}_1 - \vec{v}|) \cdot [f(\vec{r}, \vec{v}_1', t) f(\vec{r}, \vec{v}', t) - f(\vec{r}, \vec{v}_1, t) f(\vec{r}, \vec{v}, t)] \quad (1\text{-}2)$$

Where $\vec{v}_1'$ and $\vec{v}'$ are the velocities prior to collision, $\sigma(\hat{\Omega}, |\vec{v}_1 - \vec{v}|)$ represents the probability of two molecular collision, and the $f(\vec{r}, \vec{v}_1', t) f(\vec{r}, \vec{v}', t)$ and $f(\vec{r}, \vec{v}_1, t) f(\vec{r}, \vec{v}, t)$ terms represent the gain and loss of molecules in the phase space, respectively. Note that the gain and loss terms are quadratic. Thereby, the original Boltzmann equation (Eq. 1-1) is nonlinear. In order to solve the equation numerically, one has to linearize the equation first, which could introduce some system errors to the physics of the real problem to be solved. Usually Monte Carlo approach is used to solve gas dynamics problems without solving the equation directly.

Fortunately, the neutron, or in general, neutral particle transport phenomenon, can be simulated with the linear form of the Boltzmann equation. The linear form is valid, because neutron-neutron interactions are negligible compared to neutron-nucleus interactions. For example, in a typical reactor, the neutron number density is usually ~15 orders of magnitude less than the number density of surrounding medium. In such systems, only the neutron-nucleus interaction is considered, and the medium remains unchanged within the time scope of neutron transport. This assumption reduces the collision term on the right side of Eq.1-1 from quadratic to a linear term. Further simplification can be achieved by assuming $\vec{F} = 0$, which is true in most situations, because neutrons or neutral particles are not affected by electric or magnetic field, and gravitational force is negligible because of the negligible weight neutrons and zero weight of gamma rays. With these simplifications to Eq.1-1, the neutron transport equation, or the linear Boltzmann equation (LBE), can be written as:

$$\frac{1}{v}\frac{\partial\psi(\vec{r},E,\hat{\Omega},t)}{\partial t}+\sigma_t(\vec{r},E)\psi(\vec{r},E,\hat{\Omega},t)+\hat{\Omega}\cdot\nabla\psi(\vec{r},E,\hat{\Omega},t)=$$
$$\int_0^\infty dE'\int_{4\pi}d\hat{\Omega}'\sigma_s(\vec{r},E'\to E,\hat{\Omega}'\to\hat{\Omega})\psi(\vec{r},E',\hat{\Omega}',t)+ \tag{1-3}$$
$$\frac{\chi}{4\pi}\int_0^\infty dE'\int_{4\pi}d\hat{\Omega}'\upsilon\sigma_f(\vec{r},E')\psi(\vec{r},E',\hat{\Omega}',t)+S(\vec{r},E,\hat{\Omega},t)$$

Where v is the speed of neutron, $\psi(\vec{r},E,\hat{\Omega},t)=v\cdot n(\vec{r},E,\hat{\Omega},t)$ is the angular flux, $n(\vec{r},E,\hat{\Omega},t)$ is expected number of neutrons in the phase space of $d\vec{r}dEd\hat{\Omega}$, σ_t, σ_s and σ_f are total, scattering and fission cross sections of the nuclei in the medium, respectively, $\chi(E)$ is the fission spectrum, and $S(\vec{r},E,\hat{\Omega},t)$ is the independent source. The time-independent linear Boltzmann equation can be written as:[2, 3]

$$\sigma_t(\vec{r},E)\psi(\vec{r},E,\hat{\Omega})+\hat{\Omega}\cdot\nabla\psi(\vec{r},E,\hat{\Omega})=$$
$$\int_0^\infty dE'\int_{4\pi}d\hat{\Omega}'\sigma_s(\vec{r},E'\to E,\hat{\Omega}'\to\hat{\Omega})\psi(\vec{r},E',\hat{\Omega}')+ \tag{1-4}$$
$$\frac{\chi}{4\pi}\int_0^\infty dE'\int_{4\pi}d\hat{\Omega}'\upsilon\sigma_f(\vec{r},E')\psi(\vec{r},E',\hat{\Omega}',t)+S(\vec{r},E,\hat{\Omega})$$

Equation 1-4 is the fundamental equation we are to solve with our code. It represents two basic types of problems. In operator form, they are:

- Fixed source problem: $H\psi=S_{fix}$
- Eigenvalue problem: $H\psi=\frac{1}{k}F\psi$

Where the transport operator H and the fission operator F are defined as:

$$H=\hat{\Omega}\cdot\nabla+\sigma_t(\vec{r},E)-\int_0^E dE'\int_{4\pi}d\hat{\Omega}'\sigma_s(\vec{r},E'\to E,\hat{\Omega}'\to\hat{\Omega}) \tag{1-5}$$

$$F=\frac{\chi(E)}{4\pi}\int_0^\infty dE'\int_{4\pi}d\hat{\Omega}'\nu\sigma_f(\vec{r},E') \tag{1-6}$$

And k is the eigenvalue of the system. The fixed source problem is also referred to as *shielding problem*, and the eigenvalue problem often is called *criticality problem* in the area of reactor physics.

Numerical Methods to Solve the LBE

In the past fifty years, numerous numerical methods have been developed to solve the transport equation. Two of the most widely used methods are:

- Discrete ordinates method (S_N).
- Method of characteristics (MOC).

These methods are often referred to as *deterministic* methods, as opposed to the *Monte Carlo* method, in the sense that they are to solve the LBE or its derived formulations directly by numerical methods. To numerically solve a differential equation, it is required to discretize the equation in its phase space. In the LBE, the angular flux $\psi(\vec{r}, E, \hat{\Omega})$ is defined in a phase space composed of three domains: spatial, energy and angular domain. In deterministic methods, generally, the energy domain variable is discretized using the multigroup approximation.[4] The angular domain variables are discretized using the numerical quadrature technique.[2] And in the spatial domain, different methods may take their individual approaches in various geometry systems. For example, one can divide space into structured or unstructured meshes (S_N) with finite differencing or finite element approach, or arbitrary-shaped material regions (MOC).

Discrete Ordinates Method

The S_N method was first introduced by Carlson into the nuclear engineering field in 1958.[5] It has been one of the dominant deterministic methods for its efficiency and numerical stability. In the S_N method, Eq. 1-4 is the fundamental equation to solve. And the angular flux is only calculated in a number of discrete directions. In other words, if we consider the angular flux as a

function defined on the surface of a unit sphere, the S_N method evaluates function values at discrete points on the surface, which are carefully chosen by a quadrature set in order to conserve the flux moments. In the spatial domain, numerical differencing schemes are required in the S_N method to evaluate the streaming term.

Method of Characteristics (MOC)

Recently with the advancements in computing hardware, MOC has drawn more and more attentions in both the nuclear engineering and medical physics communities.[6, 7] A number of 2D/3D MOC codes [8, 9] have been developed for reactor physics and medical applications. Among advantages of the MOC, its ability to treat arbitrary geometrical bodies is an attractive feature, especially for medical applications, in which the Monte Carlo approach is still dominant. MOC usually uses the same quadrature technique as the S_N method to accomplish angular discretization. It solves the LBE along parallel straight lines (referred to as the characteristic rays) instead of discretized meshes as in S_N method. The angular flux along a characteristic ray can be described by the formulation of the integral transport equation:

$$\begin{aligned}\psi(\vec{r},E,\hat{\Omega}) &= \frac{\chi(E)}{4\pi}\int_0^R dl\int_0^\infty dE'\upsilon\sigma_f(\vec{r}-l\hat{\Omega},E')\psi(\vec{r}-l\hat{\Omega},E',\hat{\Omega})e^{-\tau_E(\vec{r},\vec{r}-l\hat{\Omega})} \\ &+ \int_0^R dl\int_0^\infty dE'\int_{4\pi} d\hat{\Omega}'\sigma_s(\vec{r}-l\hat{\Omega},E'\to E,\hat{\Omega}'\to\hat{\Omega})\psi(\vec{r}-l\hat{\Omega},E',\hat{\Omega}')e^{-\tau_E(\vec{r},\vec{r}-l\hat{\Omega})} \\ &+ \int_0^R dlS_{fix}(\vec{r}-l\hat{\Omega},E,\hat{\Omega})e^{-\tau_E(\vec{r},\vec{r}-l\hat{\Omega})} + \psi(\vec{r}-R\hat{\Omega},E,\hat{\Omega})e^{-\tau_E(\vec{r},\vec{r}-l\hat{\Omega})}\end{aligned} \tag{1-7}$$

Where $\tau_E(\vec{r},\vec{r}-l\hat{\Omega}) = \int_0^l dl'\sigma_t(\vec{r}-l'\hat{\Omega},E)$ is the optical path length along the characteristic ray for particles with energy of *E*. Figure 1-1 illustrates the terms in Eq. 1-6, which is the fundamental formulation for the MOC.

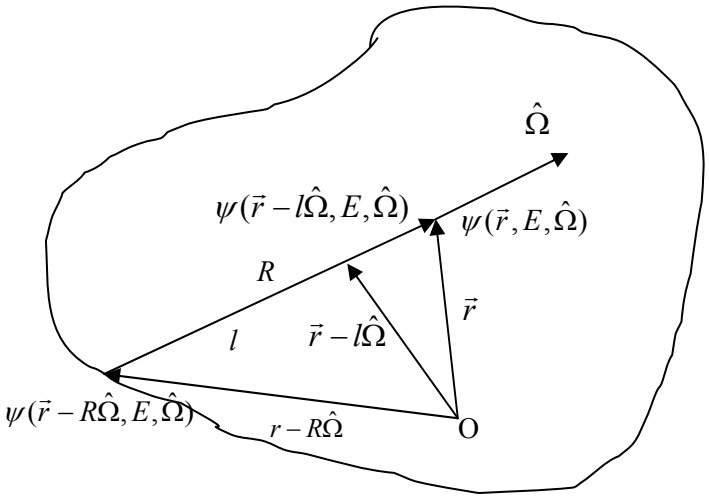


Figure 1-1. Angular flux formulation of the integral transport equation.

The streaming term in the LBE disappears in Eq. 1-6 because of the integration over the characteristic ray. Therefore, as one benefit, differencing schemes are not required in the MOC. However, MOC requires a sufficient number of rays in order to adequately cover the spatial domain. The main disadvantage of the MOC is the need of a large amount of memory to store the geometry information for the characteristic rays. Since the 3-D MOC could be very expensive,[10] some synthesis methods, coupled 2-D MOC with 1-D nodal/transport method,[11] have been developed based on the fact that in most reactor system, flux profile changes relatively slowly along *z* axis, comparing to rapidly changing profile over the *x-y* plane.

Ray-Effects in Low Scattering Region

One numerical difficulty for the deterministic methods is the so-called ray-effects,[12] especially in the S_N method with structured meshing in a low scattering medium, where the uncollided flux is dominant. As the distance between a localized source and a region of interest increases, the number of discrete ordinates that intersect each distant spatial mesh is reduced, resulting in unphysical oscillations of the scalar flux. Generally, the meshing and the quadrature

set in the S_N method should remain consistent. Otherwise, in a system where spatial and angular domains are tightly coupled, the mismatch between discretization grids in the two domains may cause the ray-effects.

The ray-effects can be alleviated naturally by increasing isotropic scattering or fission, since fission is always considered isotropic and an isotropic influence tends to "flatten" the flux in the angular domain. However, the ray-effects become worse in low scattering medium or a highly absorbing medium, where the flux is usually highly angular dependent. Therefore, particle transport problems in low-scattering media often present a difficulty for deterministic methods.

Hybrid Approach

Both the S_N and characteristics methods have been studied intensively, and utilized into many codes. The goal of this work is to solve the LBE efficiently by taking a hybrid S_N and characteristics approach for problems containing low scattering regions. Such problems are very common in medical physics applications and in many shielding problems.

Both methods numerically solve the LBE by discretizing the angular flux in the spatial, angular and energy domains. However, they solve different formulations of the LBE, which in return leads to different spatial discretization approaches. In the general S_N method, the resolution and accuracy of flux distribution depends on the mesh size and the differencing scheme. In the characteristics method, the resolution of flux distribution depends on the sizes of flat source regions. And the accuracy of the flux for each region relies on the densities of characteristic rays. Although the two methods use different discretization methods in the spatial domain, the same discretization approaches (energy group and discrete quadrature set) can be used in both methods in the energy and angular domains. Therefore, it is possible to combine both methods into one code.

The S_N method and the MOC are two of most efficient techniques to solve the LBE. However, in 3-D problems requiring a dense grid in phase space discretization (i.e. a large number of spatial meshes, directions, or energy groups), both techniques could suffer from the need for large amounts of memory and computation time. In this work, we developed a new transport code (TITAN) with a hybrid discrete ordinates and characteristic method, specifically for application to problems containing regions of low scattering. In this hybrid approach, different methods can be applied to solve the LBE for a given spatial block (coarse mesh) in a physical model. The hybrid approach can take advantages of both methods by applying the preferred method in different regions (blocks) based on the problem physics. Since the characteristics method is numerically more efficient in low scattering media, the hybrid approach uses a block-oriented characteristics solver in low scattering regions, and uses a block-oriented S_N solver in the remainder of the physical model.

CHAPTER 2
THEORY AND ALGORITHMS

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.[13] 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.

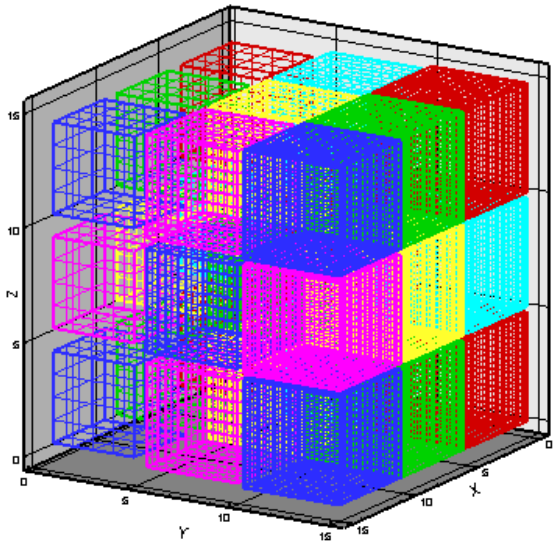


Figure 2-1. Coarse mesh/fine mesh meshing scheme.

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.

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.

Discrete Ordinates Formulations

Here, we apply the multigroup theory[4] to discretize the LBE in the energy domain. And we rewrite Eq. 1-5 in the Cartesian geometry as:[3]

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

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:

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

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

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

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
$S^{fix}_g(x,y,z,\mu,\varphi)$: 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, if we compare Eqs. 1-5 and 2-1, 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 in Eq. 1-5 becomes a differential term in Cartesian geometry. In order to numerically evaluate the differentials, differencing scheme is required in the S_N method.

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.

Differencing Scheme

From Eq. 1-5 to 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.[14] 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.

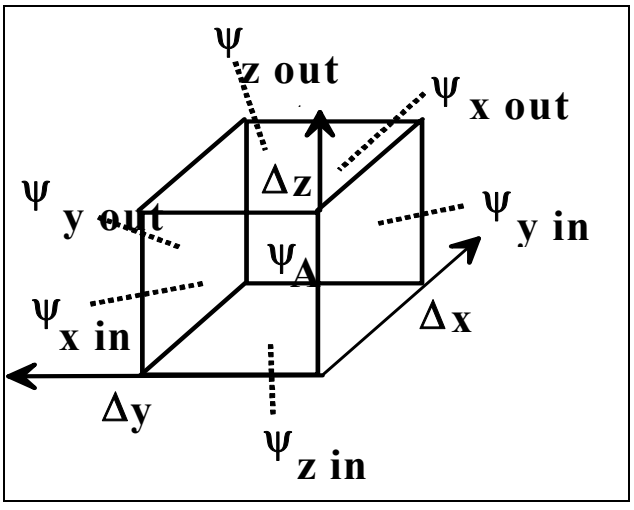


Figure 2-2. Differencing scheme on one fine mesh.[15]

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

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

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.

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

In Eq. 2-10, the three incoming fluxes ($\psi_{x\,\text{in}}$, $\psi_{\text{y in}}$ and $\psi_{\text{z in}}$) 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 expressed by:

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

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.

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

The original LBE (Eq. 1-5) finally 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 Chapter 3.

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[14] is usually applied in the diamond differencing scheme. Furthermore, the diamond differencing scheme introduces artificial oscillations in certain contiditions.[16] For this reason, and to facilitate increasing accuracy with adaptive differencing, more advanced differencing schemes[17, 18], such as DTW[19], EDW[20], and EDI[21] are implemented in the PENTRAN code. Currently, the diamond and DTW differencing schemes are applied in the TITAN code.

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*.

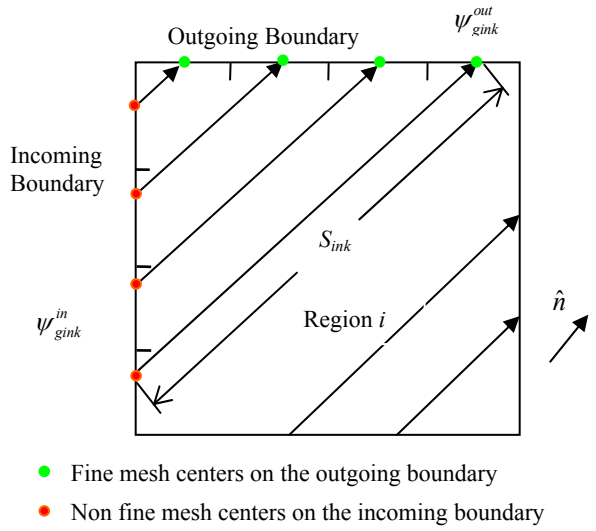


Figure 2-3. Schematic of characteristic rays in a coarse mesh using the characteristics method.

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 $\psi_{gink}^{in} = \psi_{gink}(0)$ and $\psi_{gink}^{out} = \psi_{gink}(s_{ink})$. The transport equation along ray *k* can be written as:

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

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.

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.

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 $5\times5\times5$ fine meshes if the S_N solver is assigned.

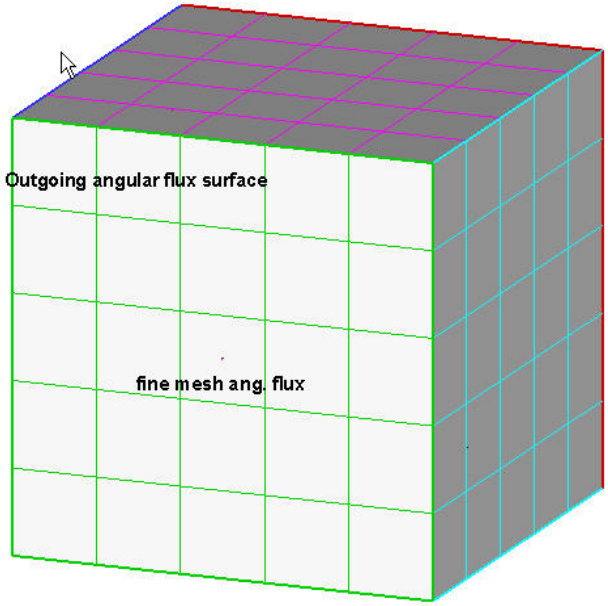


Figure 2-4. A coarse mesh with characteristics solver assigned.

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.

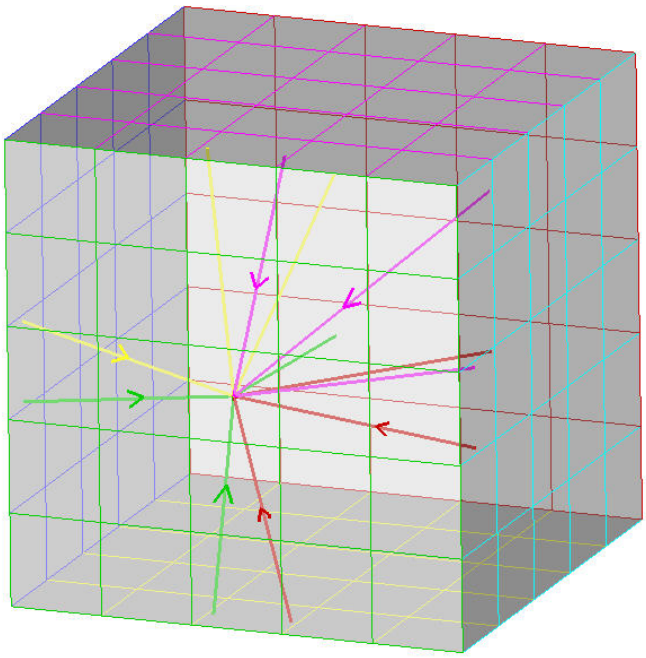


Figure 2-5. Characteristic rays for one fine mesh on one outgoing surface.

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.

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.

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:

$$\begin{aligned} x &= x_0 + t \cdot \mu \\ y &= y_0 + t \cdot \eta \\ z &= z_0 + t \cdot \xi \end{aligned} \tag{2-19}$$

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.

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.

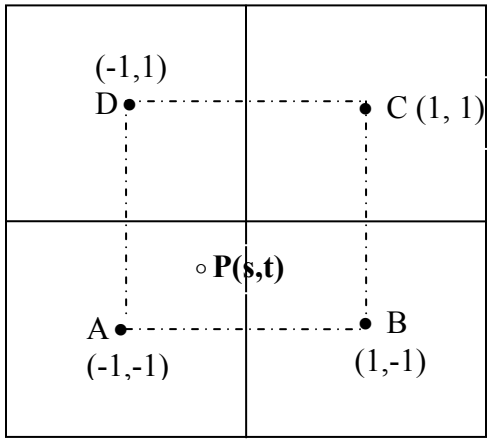


Figure 2-6. Bilinear interpolation for the incoming flux.

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:[22]

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

  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.

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.

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

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, $\phi^{(i-1)}_{g',l,x}$, $\phi^{k,(i-1)}_{C\ g',l,x}$, and $\phi^{k,(i-1)}_{S\ g',l}$ 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.

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

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

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[5] and the Legendre-Chebyshev quadrature.[23]

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.

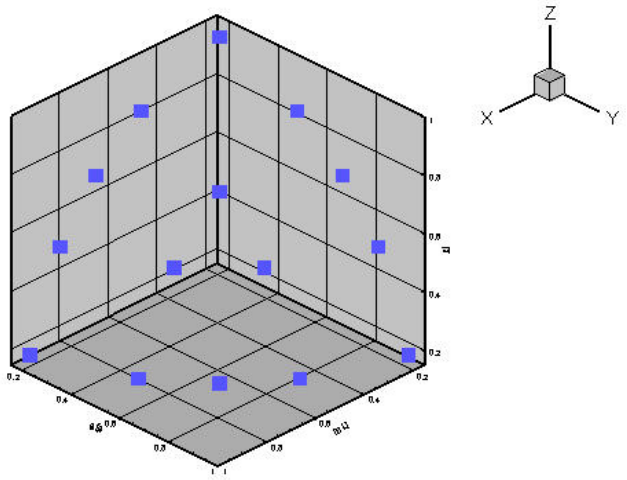


Figure 2-7. Schematic of the S_{10} level-symmetric quadrature set in one octant.

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.

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

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]

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

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.

Legendre-Chebyshev Quadrature

The Legendre-Chebyshev quadrature,[23] 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.

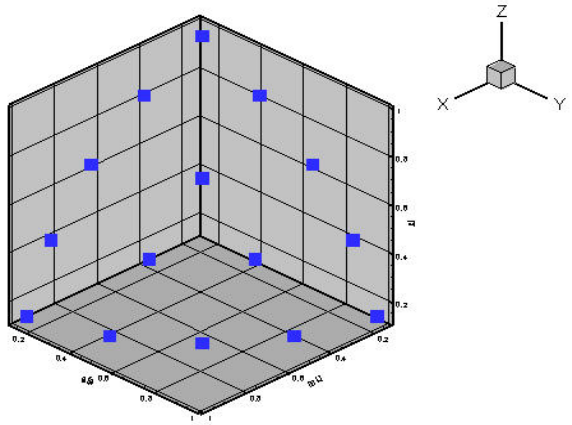


Figure 2-8. P_N-T_N quadrature of order 10.

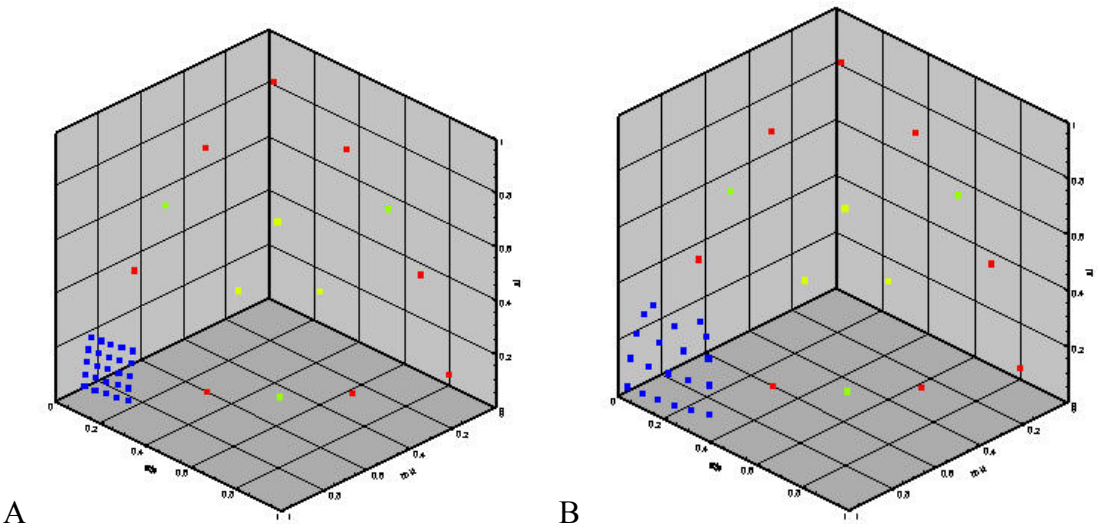
The Legendre-Chebyshev quadrature conserves moments to the order of $2L$-1, instead of L in the level-symmetric quadrature set ($L=N/2$), at the cost of lack of rotation invariance. Moments in Eq. 2-28 cannot be conserved strictly in the P_N-T_N quadrature.[24] Note that Figures 2-7 and 2-8 share a similar 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.

Rectangular and P_N-T_N Ordinate Splitting

Ordinate splitting is a technique associated with a quadrature set.[25] 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. Two splitting methods, rectangular splitting and P_N-T_N splitting are available in the TITAN code. 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. In Chapter 5, we will use the splitting techniques on one benchmark problem.

At the end of this chapter, we quote a comment on different deterministic methodologies by Weinberg and Wigner.[27] The comment was made about half a century ago, yet even today, it provides us some insights on this matter.

At present, with so much of the practical work of reactor design being done with large digital computers, the arguments in favor of one method of approximation rather than another tend to center around the question of how well suited the method is for digital computers. Actually, as the computers become larger, the choice between methods becomes less and less clear: any method which converges will do if the computer is large enough. This viewpoint certainly has practical merit; however, convenience for a digital computer is hardly a substitute for intrinsic mathematical beauty or physical relevance. In this respect the spherical harmonics method is perhaps most satisfying; its first order is identical with diffusion theory, and its higher orders show the deviations from diffusion theory very clearly.

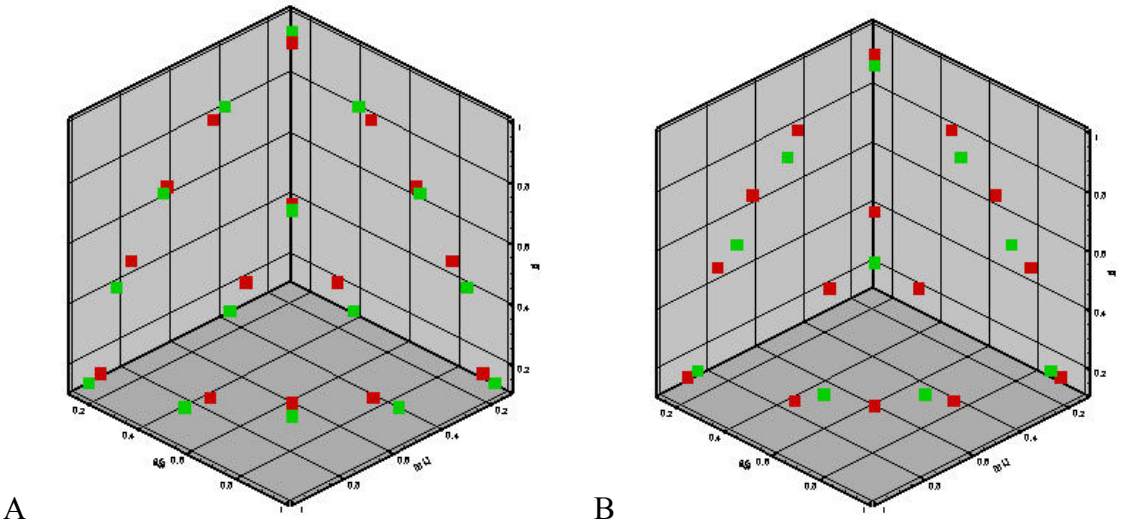
Alvin M. Weinberg & Eugene P. Wigner, 1958

CHAPTER 3
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.

Angular Projection

Angular projection is triggered by the two adjacent coarse meshes with different quadrature sets. Figure 3-1 shows the layout of directions in two quadrature sets.



A B

Figure 3-1. Angular projection. A) Level-symmetric S_{10} (red) to P_N-T_N S_{10} (green). B) S_{10} to S_8.

Figure 3-1A compares the directions for the level-symmetric and P_N-T_N quadrature sets of order 10. Figure 3-1B 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 3-2, P_1, P_2, and P_3 are the three closest neighbors in quadrature *P* to Ω_n in quadrature *Q*.

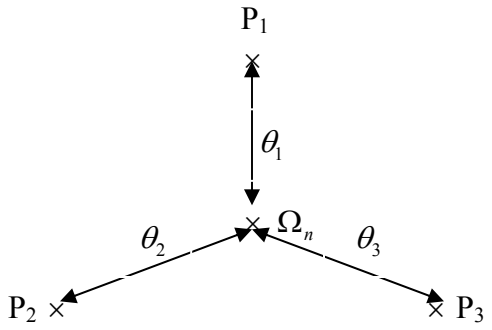


Figure 3-2. Theta weighting scheme in angular domain.

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:

$$\psi_{Q_j}^{(m)} = \begin{cases} \psi_{A_i}, & \text{if} \quad \theta_i = \min(\theta_1, \theta_2, \theta_3) \le 10^{-4} \\ \dfrac{1}{f^{(m)}} \cdot \left(\dfrac{\psi_{A_1}}{\theta_1^m} + \dfrac{\psi_{A_2}}{\theta_2^m} + \dfrac{\psi_{A_3}}{\theta_3^m}\right) & \text{otherwise} \end{cases} \tag{3-1}$$

Where $f^{(m)}$ is the *m'th* normalization factor and defined as $f^{(m)} = \frac{1}{\theta_1^m} + \frac{1}{\theta_2^m} + \frac{1}{\theta_3^m}$. 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:

$$\phi = \sum_{i=1}^{N} w_{P_i} \psi_{P_i} = \sum_{j=1}^{M} w_{Q_j} \psi_{Q_j} \tag{3-2}$$

$$J = \sum_{i=1}^{N} w_{P_i} \mu_{P_i} \psi_{P_i} = \sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j} \tag{3-3}$$

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 $\psi_{Q_j}^{(2)}$.

$$\psi_{Q_j} = \alpha \cdot \psi_{Q_j}^{(1)} + \beta \cdot \psi_{Q_j}^{(2)} \tag{3-4}$$

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.

$$\alpha = \frac{J \cdot \sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(2)} - \phi \cdot \sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(1)}}{(\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(1)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(2)}) - (\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(2)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(1)})} \tag{3-5}$$

$$\beta = \frac{\phi \cdot \sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(2)} - J \cdot \sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(1)}}{(\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(1)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(2)}) - (\sum_{j=1}^{M} w_{Q_j} \mu_{Q_j} \psi_{Q_j}^{(2)}) \cdot (\sum_{j=1}^{M} w_{Q_j} \psi_{Q_j}^{(1)})} \tag{3-6}$$

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.

Spatial Projection

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

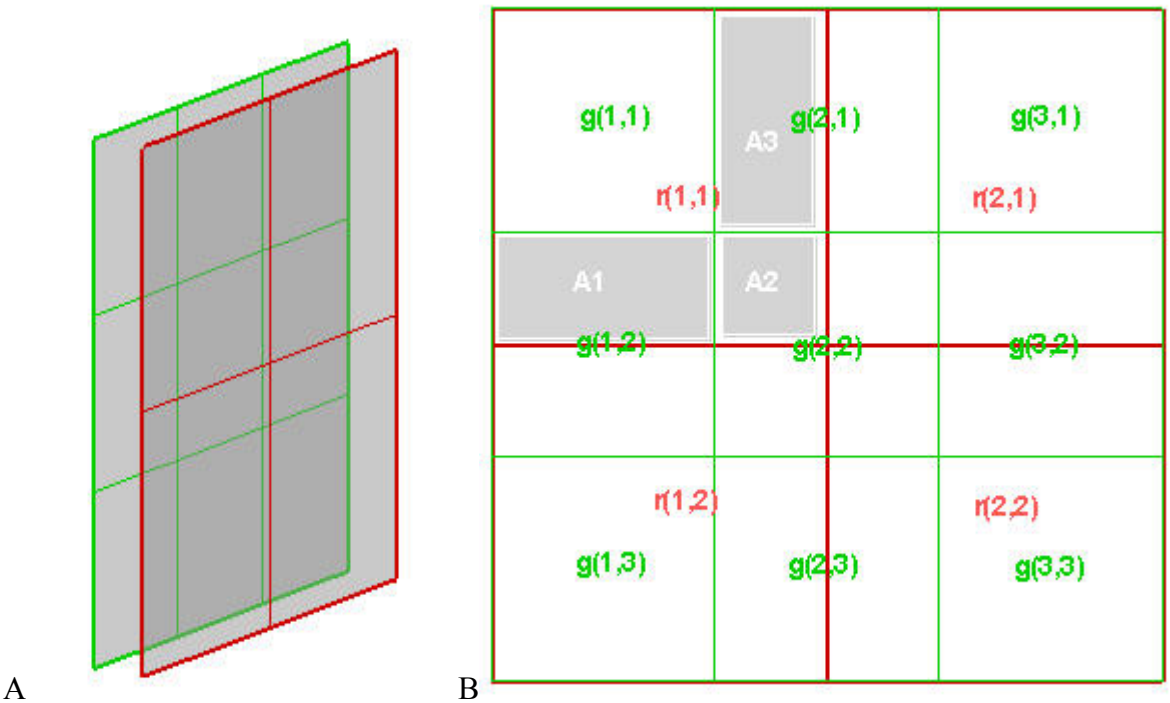


Figure 3-3. Mismatched fine-meshing schemes on the interface of two adjacent coarse meshes. A) 3-D layout. B) 2-D layout.

In Figure 3-3B, 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 $\psi_{(g)}(1,1) \rightarrow \psi_{(g)}(3,3)$ and $\psi_{(r)}(1,1) \rightarrow \psi_{(r)}(2,2)$.

Assuming a green-to-red projection, we need to calculate $\psi_{(r)}(1,1) \rightarrow \psi_{(r)}(2,2)$ based on $\psi_{(g)}(1,1) \rightarrow \psi_{(g)}(3,3)$ 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.

$$\begin{aligned}\psi_r(1,1) &= \frac{\psi_g(1,1)\cdot A_g(1,1) + \psi_g(1,2)\cdot A_1 + \psi_g(2,2)\cdot A_2 + \psi_g(2,1)\cdot A_3}{A_g(1,1) + A_1 + A_2 + A_3} \\ &= f_g(1,1)\cdot\psi_{(g)}(1,1) + f_g(1,2)\cdot\psi_{(g)}(1,2) + f_g(2,2)\cdot\psi_{(g)}(2,2) + f_g(2,1)\cdot\psi_{(g)}(2,1)\end{aligned} \tag{3-7}$$

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:

$$f_{(g)}(1,1) = \frac{A_g}{A_r},\quad f_{(g)}(1,2) = \frac{A_1}{A_r},\quad f_{(g)}(2,2) = \frac{A_2}{A_r},\quad f_{(g)}(2,1) = \frac{A_3}{A_r} \tag{3-8}$$

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:

$$\begin{aligned}&\psi_{(g)}(1,1) = \psi_{(r)}(1,1) \\ &\psi_{(g)}(2,1) = \psi_{(r)}(1,1)\cdot\frac{A_3}{A_g} + \psi_{(r)}(2,1)\cdot\frac{A_g - A_3}{A_g}\end{aligned} \tag{3-9}$$

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.

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

$$\psi_B = P_{AB}\psi_A \tag{3-10}$$

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.

CHAPTER 4
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.

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 decks include the cross-section data file, PENMSH-style input files to build up the model geometry,[28, 29] and a block-structured input file (*bonphora.inp*), 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. More details about TITAN I/O file format are given in Appendix D.

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 4-1 shows the major subroutines within the four-level code structure. In the following sections, we further discuss some of the routines on each level.

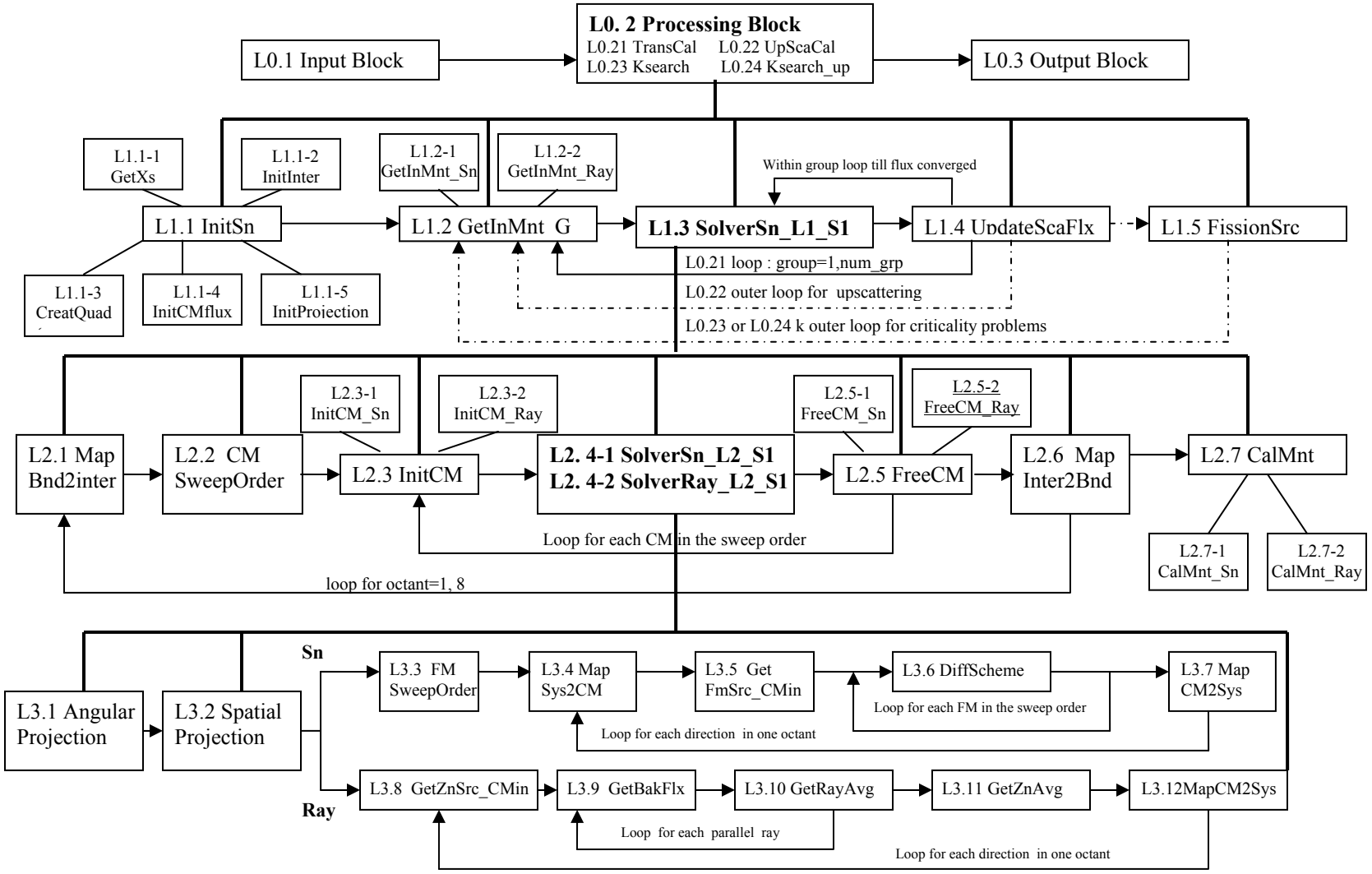


Figure 4-1. Code structure flowchart.

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.

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 4-2. 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.

$$
\begin{aligned}
S_{scattering}^{(i)} &= \sum_{g'=1}^{G}\sum_{l=0}^{L}(2l+1)\sigma_{s,g'\to g,l,x}\{P_l(\mu_n)\phi_{g',l,x}^{(i-1)} + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n)\cdot \\
&\quad [\phi_{C\,g',l,x}^{k,(i-1)}\cos(k\varphi_n) + \phi_{S\,g',l}^{k,(i-1)}\sin(k\varphi_n)]\} \\
&= \sum_{l=0}^{L}(2l+1)\{P_l(\mu_n)[\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{g',l,x}^{(i-1)} + \sigma_{s,g\to g,l,x}\phi_{g,l,x}^{(i-1)}] + \\
&\quad 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n)\cos(k\varphi_n)\cdot[\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{C\,g',l,x}^{k,(i-1)} + \sigma_{s,g\to g,l,x}\phi_{C\,g,l,x}^{k,(i-1)}] + \\
&\quad 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n)\cdot\sin(k\varphi_n)\cdot[\sum_{\substack{g'=1\\ g'\neq g}}^{G}\sigma_{s,g'\to g,l,x}\phi_{S\,g',l}^{k,(i-1)} + \sigma_{s,g\to g,l,x}\phi_{S\,g,l}^{k,(i-1)}]\}
\end{aligned}
\tag{4-1}
$$

In Eq. 4-1, 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.

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 4-3. 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).

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 4-4. 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.

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.

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 4-2, 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.

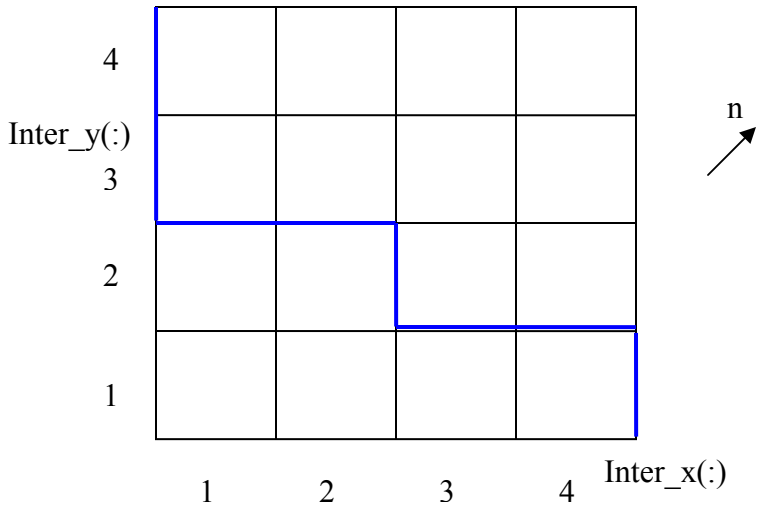


Figure 4-5. Frontline interface flux handling.

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 5
BENCHMARKING

We carefully chose a number of benchmark problems to test the performance of the TITAN code:

- A uniform medium and fixed source problem, to test the S_N solver.
- A simplified CT model, to test the hybrid approach with the ordinates splitting technique.
- The Kobayashi benchmark, to test both the S_N and hybrid formulations.
- The C5G7 MOX benchmark, to test eigenvalue problems.

These benchmark problems are used to examine different aspects of the code. In this chapter, we present the results of the TITAN code on these benchmark problems, and provide some analysis on the results.

Benchmark 1 - A Uniform Medium and Source Problem

This benchmark is a test problem designed to examine the accuracy of the S_N solver of the hybrid algorithm. A 15x15x15 cm^3 water cube is divided into 3x3x3 coarse meshes of size of 5x5x5 cm. Each coarse mesh is divided by 5x5x5 fine meshes. The entire model, as shown in Figure 5-1, is composed of 15x15x15 fine meshes in 27 coarse meshes. The fine mesh size is 1x1x1 cm^3. The vacuum boundary condition is applied on all the six surfaces of the water box. The cross section data is extracted from the SAILOR-96 library by the GIP code.[30] We only use the first 3 neutron group cross section data from the SAILOR-96 47-group structure. Both P_0 and P_3 cross section data are tested. A fixed source is uniformly distributed in the water with a uniform source spectrum.

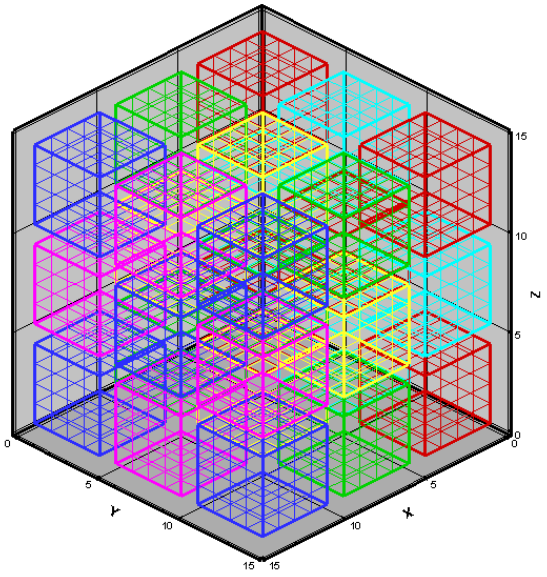
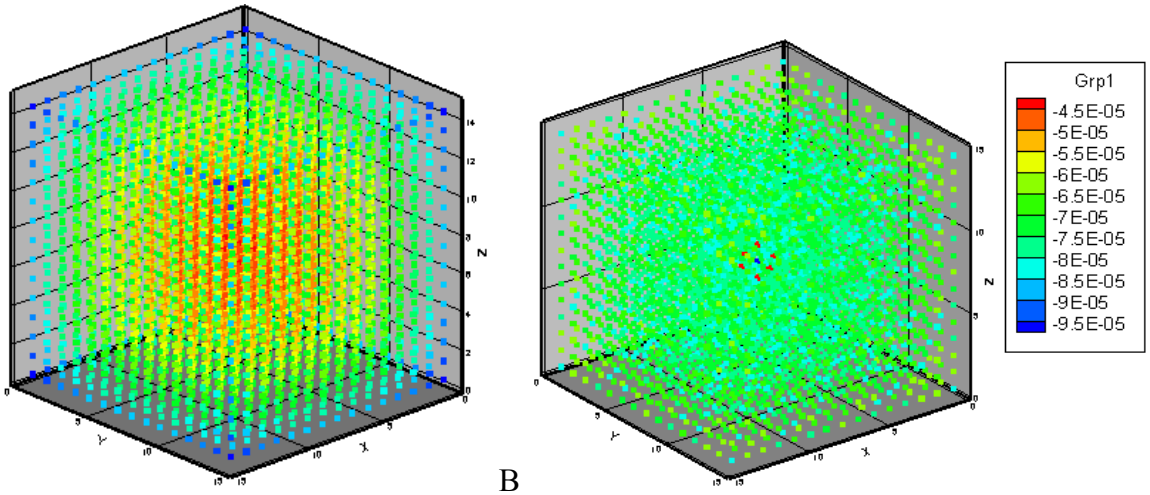


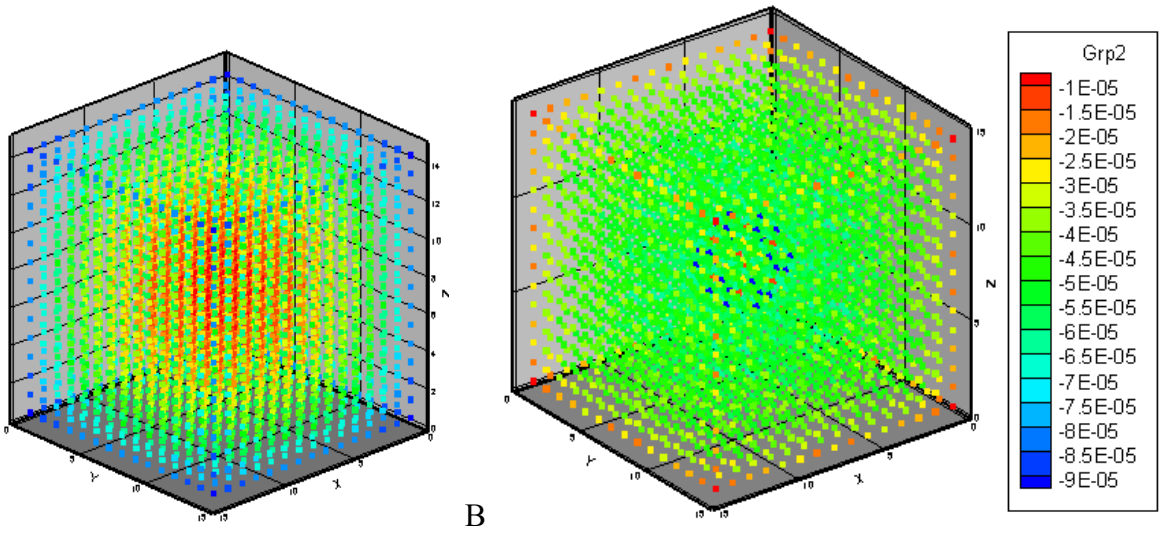
Figure 5-1. Uniform medium and source test model.

We ran this model with an S_6 quadrature set. As a reference, we also simulated the problem with the PENTRAN code with the same setup (without acceleration, and with diamond-differencing scheme only). The calculated scalar fluxes and the relative difference with PENTRAN for the 3 groups are shown in Figures 5-2, 5-3 and 5-4.



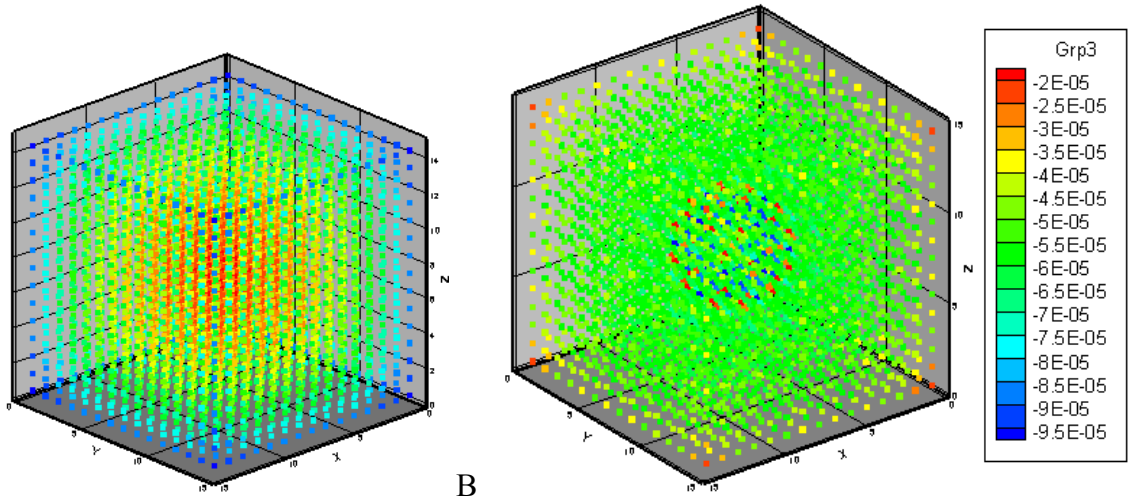
A B

Figure 5-2. Group 1 calculation result. A) Flux. B) Relative difference with PENTRAN.



B

Figure 5-3. Group 2 calculation result. A) Flux. B) Relative difference with PENTRAN.



B

Figure 5-4. Group 3 calculation result. A) Flux. B) Relative difference with PENTRAN.

As shown in Figures 5-2 to 5-4, TITAN yields the same solution as PENTRAN since the relative difference (magnitude order of 10^{-5}) is less than the flux tolerance (10^{-4}). It is also worth noting that relative difference is symmetric, and the larger difference generally occurs around the corners and edges of the water box, where the scalar fluxes are lower than the center. A test on code scalability and stability is also performed on a similar problem, in which we keep the same

fine mesh size, but only one coarse mesh for the whole box. TITAN provides the same solution on the derived model with the similar memory requirement and running time.

As the first testing problem, this benchmark demonstrates that the basic algorithms in the S_N solver are correct. The simple setup of this model is designed to eliminate possible complicated numerical effects on the S_N solver. For example, no spatial or angular projections are required in this model, since no mismatch exists between coarse meshes in either spatial or angular domain. As a result, the convergence speed for this model is relatively fast (within seconds), with only 5 or 6 within-group loops required for all the three groups.

Benchmark 2 - A Simplified CT Model

A simplified computational tomography (CT) device model is built to test the hybrid methodology and algorithm. A general CT device is shown in Figure 5-5.

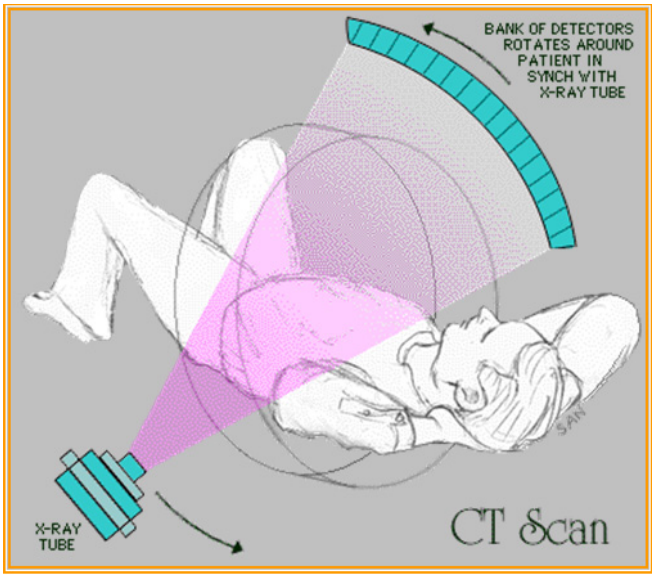


Figure 5-5. Computational tomography (CT) scan device.

In a general CT device, the directional gamma rays emitted from the X-ray tube (source) enter the human body (target) on the center. Some of the gamma particles could be scattered or absorbed in the target. The uncollided gamma particles, carrying some information about the

attenuation coefficients on different parts of the target, can be recorded by the detector array on the other side to form a projection image. Projections from different angles, acquired by rotating the source and detector array, can be used to reconstruct the target cross section image. In our simplified CT model, we only consider a center slice of a CT device without the target. A 2-D meshing plot of the simplified CT model is shown in Figure 5-6.

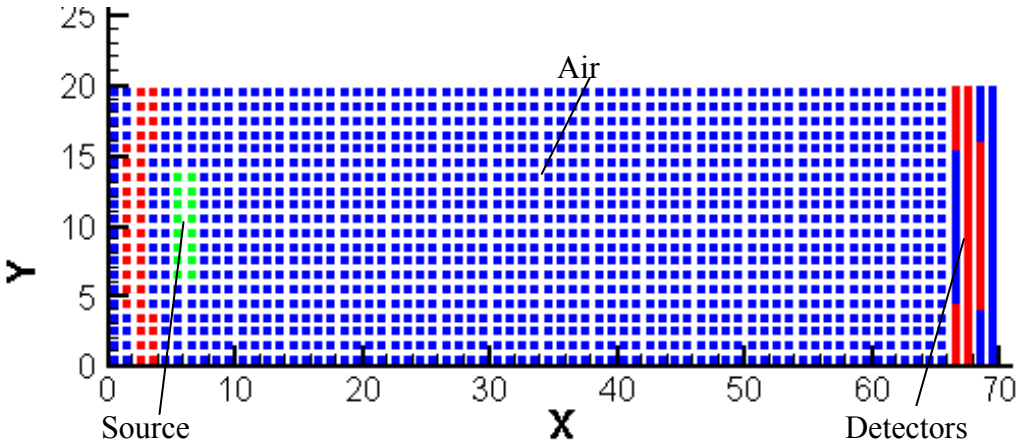


Figure 5-6. A simplified CT model.

In the simplified CT model, the photon source and an array of detectors are located on the left and right side of a slice of the whole CT device, respectively, and the target object is removed from the center. Our goal is to calculate the scalar fluxes of the 20 fine meshes along y direction in the detector region (i.e., red region at the right hand side of Figure 5-6). The relatively large air region between the source and detector usually causes serious ray-effects when the S_N method is used. In order to overcome the ray-effects, The S_N algorithm requires finer discretization grids in both spatial and angular domains. Alternatively, a process called 'smearing' can be used to resolve the discretization grid mismatch in spatial and angular domain by carefully choosing the mesh size along the discrete ordinates. In this test, we use the ordinate splitting technique as a ray-effect remedy. And the TITAN solutions with different solvers are compared with the MCNP5 reference calculation.[31]

Monte Carlo Model Description

Figure 5-7 shows the geometry for the Monte Carlo MCNP5 model, which is built exactly as the deterministic model shown in Figure 5-6.

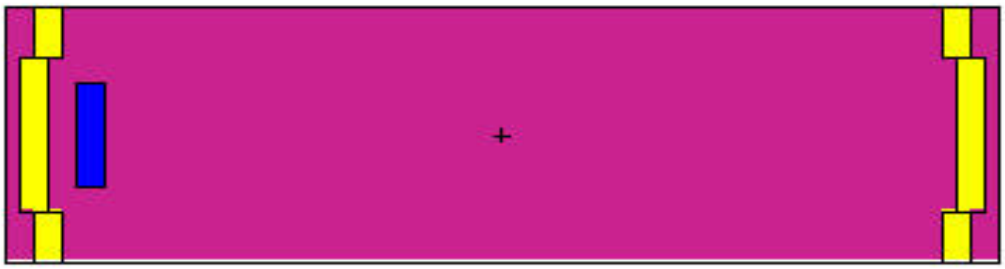


Figure 5-7. MCNP model of the simplified CT device.

We use MCNP5 code in multigroup mode,[32] so that we can apply the same cross section data as used in the deterministic calculations. A mesh tally is used to evaluate the 20 fine-mesh fluxes in the detector region.

Deterministic Model Description

Figure 5-8 shows the S_N solver model, which is composed of 7 coarse meshes with 14,000 fine meshes.

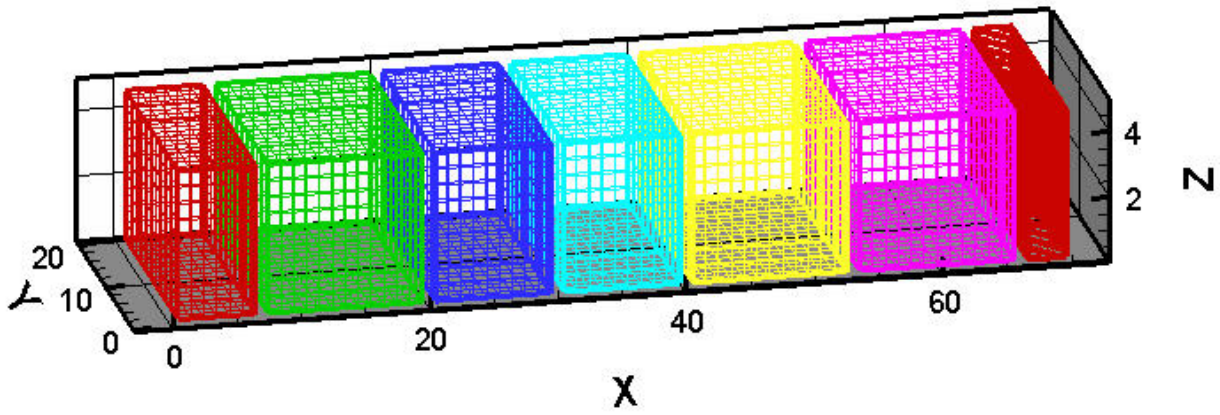


Figure 5-8. S_N solver meshing scheme for the CT model.

Here, we use five coarse meshes in the air region to resolve the ray effect. The average fluxes for the 20 detector fine meshes are extracted after the calculation.

Figure 5-9 shows the hybrid solver model with 3 coarse meshes and 3,000 fine meshes.

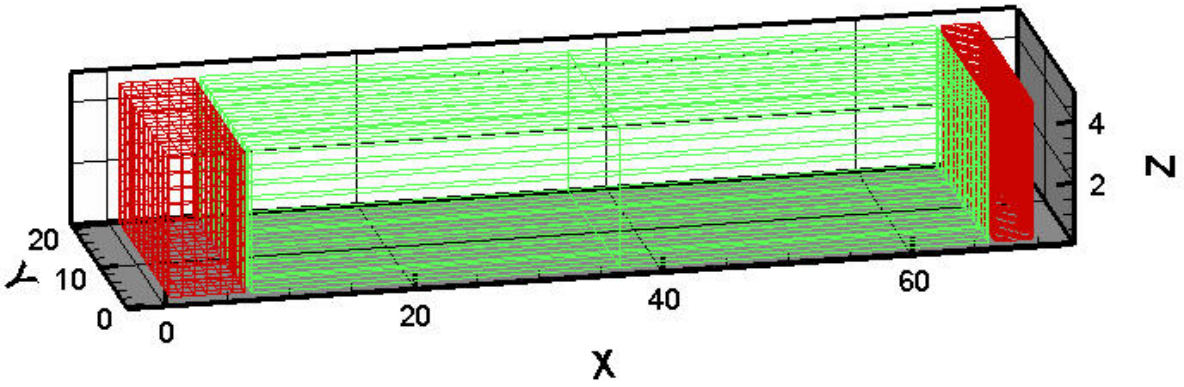


Figure 5-9. Hybrid model meshing for the CT model.

In the hybrid model, we apply characteristics solver in the air region (coarse mesh #2), and the S_N solver in both the source and detector regions (coarse meshes #1&3). The number of fine meshes in the hybrid model is much less than the one in the S_N model.

Comparison and Analysis of Results

A number of cases are tested for the simplified CT problem. In the first set of cases (Cases 2 and 3), we apply the S_N Solver only to solve the problem, and try to alleviate the ray effect by increasing the S_N order. Due to the relatively large distance between the source and the detectors, and the relatively small size of the detector fine mesh, very high order of quadrature set is required to eliminate the ray-effects if no other ray effect remedy techniques are applied. This approach to reduce the ray-effect is not efficient, because the memory requirement is roughly proportional to square of the S_N order. Figure 5-10 shows the results for an S_{100} case and an S_{200} case compared with the MCNP reference case.

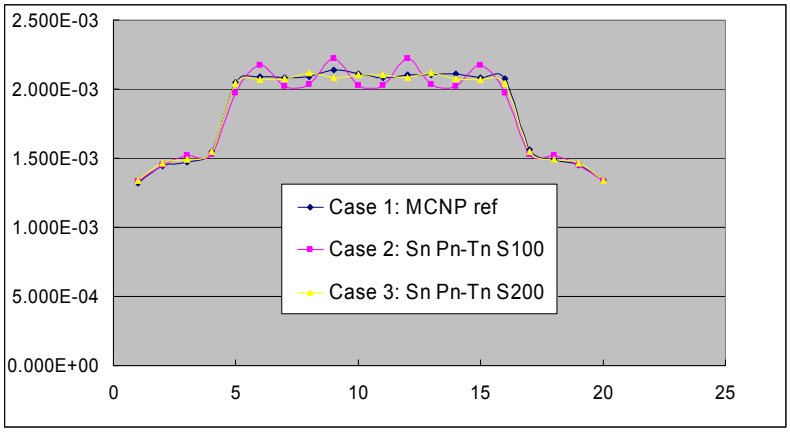
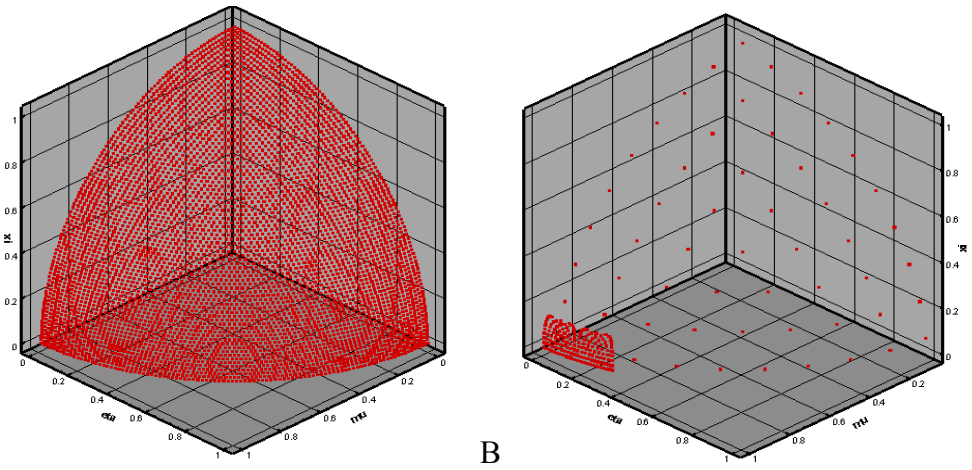


Figure 5-10. S_N simulation results without ordinate splitting.

The ray-effect is obvious in Case 2 with S_{100}. Note that in most real problems, S_N order usually can not reach as high as 100 due to the memory limitation. However, since this simplified model is relatively small with about 14,000 fine meshes, and one group cross section structure, we are able to apply an S_{200} P_N-T_N quadrature set (shown in Figure 5-11A) for Case 2, in which the ray-effects are significantly reduced.



A B

Figure 5-11. Quadrature sets used in the CT benchmark. A) P_N-T_N S_{200}. B) Biased P_N-T_N S_{20}.

In the second set of test cases (Cases 4 and 5), the ordinate splitting technique is applied as a remedy for elimination of ray-effects. In this model, obviously particles streaming along the directions close to *x* axis will contribute the most for the detector fluxes. Therefore, we use a P_N-

T_N S_{20} quadrature set with the local P_N-T_N splitting technique on two directions close to the *x* axis. both with a splitting order of 11 as shown in Figure 5-11B. The hybrid approach is tested in Case 5. Figure 5-12 shows the results for the S_N solver case and the hybrid case, both compared with the MCNP reference case.

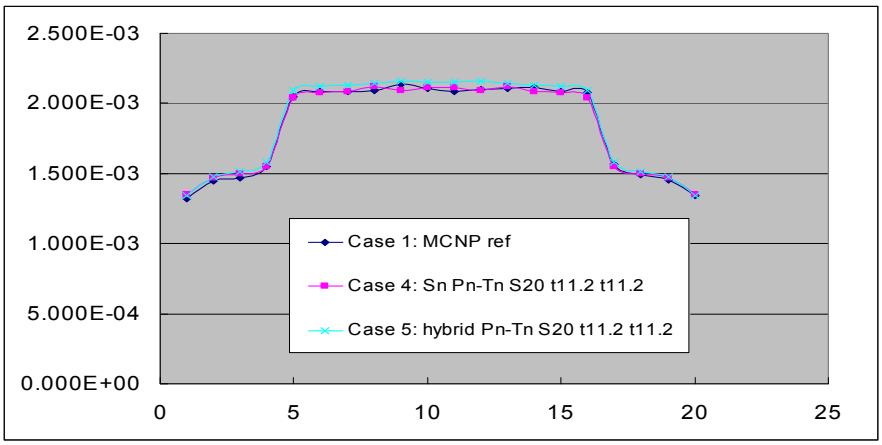


Figure 5-12. Hybrid and S_N simulation results with ordinate splitting.

Both cases show a good agreement with the MCNP reference case without ray-effects. It is worth noting that in the hybrid model, as discussed in the last section, the number of fine meshes is reduced by a factor of ~5 comparing to the S_N model. The run times and error norms as compared to the MCNP reference case are presented in Table 5-1.

Table 5-1. CT model run time and error norm comparison with the MCNP reference case.

Case number	Descriptions	Run Time (sec)	Run Time Comparison	Err 2-norm[1]	Err inf-norm[2]
1	MCNP ref, nps=2e8, rel.err. <0.01	3510	1.0	0.000E+00	0.00%
2	S_N P_N-T_N S_{100} (10,200)*	441.3	7.9	2.182E-02	5.86%
3	S_N P_N-T_N S_{200} (40,400)*	1755.8	2.0	2.655E-03	2.41%
4	S_N P_N-T_N S_{20} t11.2 t11.2 (207)*	71.4 sec	49.1	2.820E-03	2.09%
5	Hybrid P_N-T_N S_{20} t11.2 t11.2 (207)*	14.1 sec	248.9	7.510E-03	3.28%

[1] Error 2-norm measures the overall error for the 20 points
[2] Error inf-norm represents the maxim local relative error
* Total number of directions

For the MCNP reference case, we use 200 million particles to yield a relative flux error of less than 1% for all 20 meshes. Here, we use the infinity-norm and 2-norm to measure the maxim local relative error and the overall error for the 20 points respectively. All the deterministic cases show a good agreement with the Monte Carlo reference case and with less computation time. The hybrid approach (Case 5) is about 5 times faster than the S_N solver only case (Case 4), since in the hybrid model, we use about 5 time less fine meshes than in the S_N model. This benchmark demonstrates that for problems with a large region of low scattering medium, the hybrid approach can achieve the same level of accuracy as the S_N method with much fewer fine meshes and thereby significantly lower computation cost.

Benchmark 3 - Kobayashi 3-D Problems with Void Ducts

This benchmark consists of three problems with simple geometries and void regions.[33] Furthermore, each problem includes two cases: zero-scattering and 50% scattering. We tested all the three problems with the zero-scattering case. And each problem model is composed of three regions:

Region 1: Source (no scattering).

Region 2: Void.

Region 3: Pure absorber.

We present the calculation results of our code and the comparison with the analytical solution provided by the benchmark. Note we use uniform meshing for all the three problems: each coarse mesh with a size of 10x10x10 cm^3, and each fine mesh with a size of 1x1x1 cm^3. And the point-wise fluxes in the benchmark are compared with the averaged fluxes calculated over corresponding coarse mesh.

Problem 1: Shield with Square Void

As shown in Figure 5-13, this box-in-box problem is composed of three cubes: 10x10x10 cm^3 source box in the corner, 50x50x50 cm^3 air box, and 100x100x100 cm^3 pure absorber box.

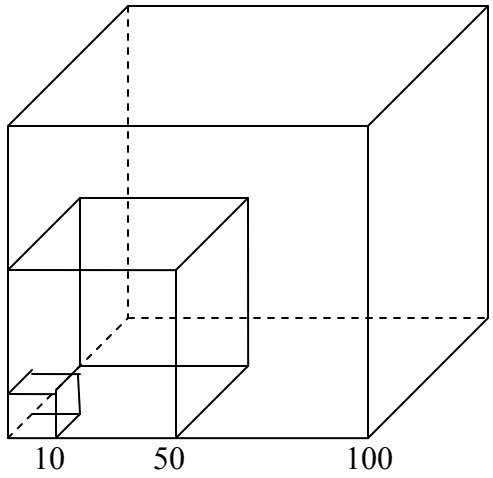


Figure 5-13. Kobayashi Problem 1 box-in-box layout.

We consider three cases:

Case 1: MOC solver applied in Region 2 (void), Regions 1 and 3 with S_N solver.

Case 2: MOC solver in Region 2&3 (void and pure absorber). S_N solver in Region 1.

Case 3: S_N solver in all three regions.

Tables 5-2 to 5-4 compare the results of Case 1 with different quadrature sets for the three point sets. We also calculate the ratios to analytical solutions.

Table 5-2. Kobayashi Problem 1 Point A set flux results for Case 1.

Point 1A	Analytical	Case 1 (S_{24})	Ratio	Case 1 (S_{30})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981	5.94414E+00	0.9979
5,15,,5	1.37185E+00	1.44872E+00	1.0560	1.44446E+00	1.0529
5,25,5	5.00871E-01	5.01333E-01	1.0009	5.00703E-01	0.9997
5,35,5	2.52429E-01	2.48688E-01	0.9852	2.49114E-01	0.9869
5,45,5	1.50260E-01	1.45821E-01	0.9705	1.46590E-01	0.9756
5,55,5	5.95286E-02	6.16731E-02	1.0360	6.21947E-02	1.0448
5,65,5	1.52283E-02	1.56001E-02	1.0244	1.56733E-02	1.0292
5,75,5	4.17689E-03	4.26493E-03	1.0211	4.16728E-03	0.9977
5,85,5	1.18533E-03	1.16145E-03	0.9799	1.18505E-03	0.9998
5,95,5	3.46846E-04	3.13078E-04	0.9026	3.45040E-04	0.9948
ErrNorm (Err2Norm Err1Norm)		1.8232E-02	9.736%	6.0117E-03	5.293%

Table 5-3. Kobayashi Problem 1 Point B set flux results for Case 1.

Point 1B	Analytical	Case 1 (S_{24})	Ratio	Case 1 (S_{30})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981	5.94414E+00	0.9979
15,15,15	4.70754E-01	4.81175E-01	1.0221	4.79594E-01	1.0188
25,25,25	1.69968E-01	1.70050E-01	1.0005	1.70665E-01	1.0041
35,35,35	8.68334E-02	8.73159E-02	1.0056	8.67251E-02	0.9988
45,45,45	5.25132E-02	5.12734E-02	0.9764	5.29735E-02	1.0088
55,55,55	1.33378E-02	1.08504E-02	0.8135	1.04048E-02	0.7801
65,65,65	1.45867E-03	1.50095E-03	1.0290	1.29943E-03	0.8908
75,75,75	1.75364E-04	1.99741E-04	1.1390	1.78873E-04	1.0200
85,85,85	2.24607E-05	2.42707E-05	1.0806	2.55221E-05	1.1363
95,95,95	3.01032E-06	2.67405E-06	0.8883	3.53673E-06	1.1749
ErrNorm (Err2Norm Err1Norm)		9.0705E-02	18.649%	1.3184E-01	21.990%

Table 5-4. Kobayashi Problem 1 Point C set flux results for Case 1.

Point 1C	Analytical	Case 1 (S_{24})	Ratio	Case 1 (S_{30})	Ratio
5,55,5	5.95286E-02	6.29408E-02	1.0573	6.18784E-02	1.0395
15,55,5	5.50247E-02	6.00183E-02	1.0908	5.95864E-02	1.0829
25,55,5	4.80754E-02	5.14090E-02	1.0693	5.16984E-02	1.0754
35,55,5	3.96765E-02	4.24917E-02	1.0710	4.33243E-02	1.0919
45,55,5	3.16366E-02	3.44892E-02	1.0902	3.48761E-02	1.1024
55,55,5	2.35303E-02	2.15000E-02	0.9137	2.14425E-02	0.9113
65,55,5	5.83721E-03	6.37570E-03	1.0923	6.26243E-03	1.0728
75,55,5	1.56731E-03	1.59919E-03	1.0203	1.66064E-03	1.0595
85,55,5	4.53113E-04	4.36921E-04	0.9643	4.82881E-04	1.0657
95,55,5	1.37079E-04	1.46529E-04	1.0689	1.41297E-04	1.0308
ErrNorm (Err2Norm Err1Norm)		4.7278E-02	9.225%	4.9872E-02	10.240%

In Table 5-3, point (55, 55, 55) has a relative error of 20%, which is largest error among all points, because it is located in the coarse mesh on the interface between the absorber region and the air region. The transport solver may encounter difficulties in resolving the highly angular dependent flux on the interface. Another difficult point (95, 95, 95) is located on the far corner away from the source, where the ray-effect may be severer than the regions closer to the source. The S_{30} case shows no significant improvement as compared to the S_{24} case, which may indicate that we need to apply finer meshes to take advantage of a higher order quadrature set. Tables 5-5 to 5-7 compare the results for Case 2 with an S_{24} quadrature set for the three point sets.

Table 5-5. Kobayashi Problem 1 Point A set flux results for Case 2.

Point 1A	Analytical	Case 2 (S_{24})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981
5,15,,5	1.37185E+00	1.44872E+00	1.0560
5,25,5	5.00871E-01	5.01333E-01	1.0009
5,35,5	2.52429E-01	2.48688E-01	0.9852
5,45,5	1.50260E-01	1.45821E-01	0.9705
5,55,5	5.95286E-02	6.12631E-02	1.0291
5,65,5	1.52283E-02	1.55573E-02	1.0216
5,75,5	4.17689E-03	4.25971E-03	1.0198
5,85,5	1.18533E-03	1.16837E-03	0.9857
5,95,5	3.46846E-04	3.18352E-04	0.9178
ErrNorm (Err2Norm Err1Norm)		1.3822E-02	8.215%

Table 5-6. Kobayashi Problem 1 Point B set flux results for Case 2.

Point 1B	Analytical	Case 2 (S_{24})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981
15,15,15	4.70754E-01	4.81175E-01	1.0221
25,25,25	1.69968E-01	1.70050E-01	1.0005
35,35,35	8.68334E-02	8.73159E-02	1.0056
45,45,45	5.25132E-02	5.12734E-02	0.9764
55,55,55	1.33378E-02	1.06986E-02	0.8021
65,65,65	1.45867E-03	1.45221E-03	0.9956
75,75,75	1.75364E-04	1.90111E-04	1.0841
85,85,85	2.24607E-05	2.29690E-05	1.0226
95,95,95	3.01032E-06	2.51840E-06	0.8366
ErrNorm (Err2Norm Err1Norm)		1.0662E-01	19.787%

Table 5-7. Kobayashi Problem 1 Point C set flux results for Case 2.

Point 1C	Analytical	Case 2 (S_{24})	Ratio
5,55,5	5.95286E-02	6.24240E-02	1.0486
15,55,5	5.50247E-02	5.97140E-02	1.0852
25,55,5	4.80754E-02	5.11524E-02	1.0640
35,55,5	3.96765E-02	4.22937E-02	1.0660
45,55,5	3.16366E-02	3.43198E-02	1.0848
55,55,5	2.35303E-02	2.13553E-02	0.9076
65,55,5	5.83721E-03	6.35753E-03	1.0891
75,55,5	1.56731E-03	1.59983E-03	1.0207
85,55,5	4.53113E-04	4.42516E-04	0.9766
95,55,5	1.37079E-04	1.45681E-04	1.0628
ErrNorm (Err2Norm Err1Norm)		4.3423E-02	9.243%

Tables 5-8 to 5-10 compare the results of Case 3 for different quadrature sets along different lines to analytical solutions.

Table 5-8. Kobayashi Problem 1 Point A set flux results for Case 3.

Point 1A	Analytical	Case 3 (S_{24})	Ratio	Case 3 (S_{34})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981	5.94319E+00	0.9978
5,15,,5	1.37185E+00	1.44622E+00	1.0542	1.44694E+00	1.0547
5,25,5	5.00871E-01	5.02432E-01	1.0031	5.03886E-01	1.0060
5,35,5	2.52429E-01	2.50261E-01	0.9914	2.51595E-01	0.9967
5,45,5	1.50260E-01	1.47601E-01	0.9823	1.48909E-01	0.9910
5,55,5	5.95286E-02	6.23020E-02	1.0466	6.32288E-02	1.0622
5,65,5	1.52283E-02	1.58269E-02	1.0393	1.60293E-02	1.0526
5,75,5	4.17689E-03	4.31608E-03	1.0333	4.29219E-03	1.0276
5,85,5	1.18533E-03	1.16330E-03	0.9814	1.20356E-03	1.0154
5,95,5	3.46846E-04	3.15751E-04	0.9103	3.54708E-04	1.0227
ErrNorm (Err2Norm Err1Norm)		1.7566E-02	8.965%	1.0191E-02	6.216%

Table 5-9. Kobayashi Problem 1 Point B set flux results for Case 3.

Point 1B	Analytical	Case 3 (S_{24})	Ratio	Case 3 (S_{34})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981	5.94319E+00	0.9978
15,15,15	4.70754E-01	4.80788E-01	1.0213	4.78621E-01	1.0167
25,25,25	1.69968E-01	1.70059E-01	1.0005	1.71342E-01	1.0081
35,35,35	8.68334E-02	8.75903E-02	1.0087	8.73758E-02	1.0062
45,45,45	5.25132E-02	5.12572E-02	0.9761	5.24423E-02	0.9986
55,55,55	1.33378E-02	1.09012E-02	0.8173	1.09080E-02	0.8178
65,65,65	1.45867E-03	1.52321E-03	1.0442	1.37740E-03	0.9443
75,75,75	1.75364E-04	2.04277E-04	1.1649	1.66625E-04	0.9502
85,85,85	2.24607E-05	2.50787E-05	1.1166	2.04334E-05	0.9097
95,95,95	3.01032E-06	2.80145E-06	0.9306	2.73143E-06	0.9074
ErrNorm (Err2Norm Err1Norm)		8.9359E-02	18.268%	7.6500E-02	18.217%

Table 5-10. Kobayashi Problem 1 Point C set flux results for Case 3.

Point 1C	Analytical	Case 3 (S_{24})	Ratio	Case 3 (S_{34})	Ratio
5,55,5	5.95286E-02	6.28676E-02	1.0561	6.35577E-02	1.0677
15,55,5	5.50247E-02	5.94963E-02	1.0813	5.94504E-02	1.0804
25,55,5	4.80754E-02	5.16771E-02	1.0749	5.19020E-02	1.0796
35,55,5	3.96765E-02	4.25678E-02	1.0729	4.31326E-02	1.0871
45,55,5	3.16366E-02	3.45271E-02	1.0914	3.47598E-02	1.0987
55,55,5	2.35303E-02	2.14367E-02	0.9110	2.15454E-02	0.9156
65,55,5	5.83721E-03	6.35281E-03	1.0883	6.21321E-03	1.0644
75,55,5	1.56731E-03	1.58707E-03	1.0126	1.64677E-03	1.0507
85,55,5	4.53113E-04	4.34709E-04	0.9594	4.74924E-04	1.0481
95,55,5	1.37079E-04	1.47770E-04	1.0780	1.45363E-04	1.0604
ErrNorm (Err2Norm Err1Norm)		4.8256E-02	9.137%	4.9324E-02	9.872%

The above results for Cases 2 and 3 show a similar agreement with the analytical solution as Case 1, with a largest relative error about 20% on the interface point. Unlike the previous CT model benchmark, the three point sets in this benchmark cover most of the difficult positions

throughout the model, while in the CT model, we are only interested in the detector region with a high resolution. It seems that the hybrid approach is more desirable in problems like the previous benchmark. For this benchmark, the hybrid algorithm performs roughly as efficient as the S_N method for the 1 million mesh model. However, the computation costs are different for the three cases as listed in Table 5-11.

Table 5-11. CPU time and memory requirement for S_N and hybrid methods (1 million meshes and S_{24} model).

Case #	Solver			CPU time (sec)	Memory (Gigabyte)
	Reg. 1 (source)	Reg. 2 (air)	Reg. 3 (absorber)		
1	S_N	MOC	S_N	690	2.7
2	S_N	MOC	MOC	267	4.3
3	S_N	S_N	S_N	753	2.5

The characteristics solver is faster, but requires more memory to store the geometry information. The S_N solver is slower, but has a less memory requirement. The tradeoff between memory and CPU time is always a coding concern, which is reflected in this problem. The CPU time for Case 3 is reduced by a factor of ~2.8, however, requires about 1.7 time more memory. It seems that Case 3 is preferred if memory requirement is affordable and/or the speed is the major concern for the user. For simplicity, in the following Problem 2 and 3 calculations, only the S_N solver results are provided.

Problem 2: Shield with Void Duct

Figure 5-14 shows the first z level of the problem layout. The blue region is the source region, the green region is the void duct, and the rest of the model is filled with a pure absorber, which is Region 3.

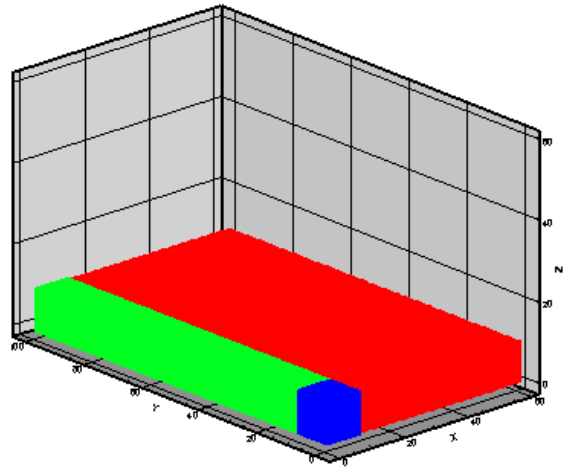


Figure 5-14. Kobayashi Problem 2 first *z* level model layout.

The S_N solver calculation results are listed in Tables 5-12 and 5-13.

Table 5-12. Kobayashi Problem 2 Point A set flux results for Case 3.

Point 2A	Analytical	Case 3 (S_{24})	Ratio	Case 3 (S_{30})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981	5.94414E+00	0.9979
5,15,,5	1.37185E+00	1.44797E+00	1.0555	1.44793E+00	1.0555
5,25,5	5.00871E-01	5.03502E-01	1.0053	5.04210E-01	1.0067
5,35,5	2.52429E-01	2.51243E-01	0.9953	2.51994E-01	0.9983
5,45,5	1.50260E-01	1.48549E-01	0.9886	1.49214E-01	0.9930
5,55,5	9.91726E-02	9.71016E-02	0.9791	9.80078E-02	0.9883
5,65,5	7.01791E-02	6.79254E-02	0.9679	6.90949E-02	0.9846
5,75,5	5.22062E-02	5.17088E-02	0.9905	5.03751E-02	0.9649
5,85,5	4.03188E-02	3.91599E-02	0.9713	3.91877E-02	0.9719
5,95,5	3.20574E-02	2.85735E-02	0.8913	3.20167E-02	0.9987
ErrNorm (Err2Norm Err1Norm)		2.0340E-02	10.868%	5.4047E-03	5.546%

Table 5-13. Kobayashi Problem 2 Point B set flux results for Case 3.

Point 2B	Analytical	Case 3 (S_{24})	Ratio	case S_N (S_{30})	Ratio
5,95,5	3.20574E-02	2.85735E-02	0.8913	3.20167E-02	0.9987
15,95,5	1.70541E-03	8.85805E-04*	0.5194	1.49781E-03*	0.8783
25,95,5	1.40557E-04	1.79639E-04	1.2781	1.53422E-04	1.0915
35,95,5	3.27058E-05	3.17893E-05	0.9720	3.39511E-05	1.0381
45,95,5	1.08505E-05	9.20428E-06	0.8483	1.12324E-05	1.0352
55,95,5	4.14132E-06	4.72351E-06	1.1406	4.32799E-06	1.0451
ErrNorm (Err2Norm Err1Norm)		9.6633E-01	48.059%	3.0605E-02	12.173%

* Results are calculated by averaging the corresponding fine mesh(s), instead of coarse mesh.

Our calculation shows a good agreement with the analytical solution on most of points, except point (15 95 5), which is located on the far side interface between Regions 2 and 3.

Problem 3: Shield with Dogleg Void Duct

Figure 5-15 shows the layout of the void duct in the model. The rest of the model is filled with pure absorber. The S_N calculation results are listed in Tables 5-14 to 5-16.

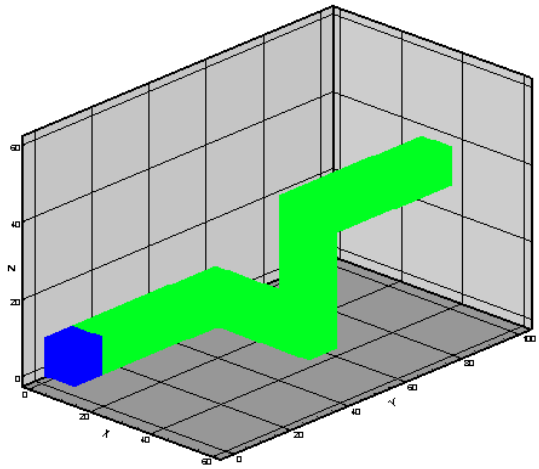


Figure 5-15. Kobayashi Problem 3 void duct layout.

Table 5-14. Kobayashi Problem 3 Point A set flux results for Case 3.

Point 3A	Analytical	Case 3 (S_{24})	Ratio	Case 3 (S_{30})	Ratio
5,5,5	5.95659E+00	5.94515E+00	0.9981	5.94414E+00	0.9998
5,15,,5	1.37185E+00	1.44797E+00	1.0555	1.44793E+00	1.0000
5,25,5	5.00871E-01	5.03502E-01	1.0053	5.04210E-01	1.0014
5,35,5	2.52429E-01	2.51243E-01	0.9953	2.51994E-01	1.0030
5,45,5	1.50260E-01	1.48549E-01	0.9886	1.49214E-01	1.0045
5,55,5	9.91726E-02	9.71016E-02	0.9791	9.80078E-02	1.0093
5,65,5	4.22623E-02	4.37756E-02	1.0358	4.46212E-02	1.0193
5,75,5	1.14703E-02	1.20425E-02	1.0499	1.17776E-02	0.9780
5,85,5	3.24662E-03	3.34282E-03	1.0296	3.32867E-03	0.9958
5,95,5	9.48324E-04	8.95157E-04	0.9439	9.94322E-04	1.1108
ErrNorm (Err2Norm Err1Norm)		1.1213E-02	5.606%	9.2256E-03	5.582%

Table 5-15. Kobayashi Problem 3 Point B set flux results for Case 3.

Point 3B	Analytical	Case 3 (S_{24})	Ratio	Case 3 (s_{30})	Ratio
5,55,5	9.91726E-02	9.71016E-02	0.9791	9.80078E-02	0.9883
15,55,5	2.45041E-02	2.66812E-02*	1.0888	2.61306E-02*	1.0664
25,55,5	4.54447E-03	4.84126E-03	1.0653	4.91017E-03	1.0805
35,55,5	1.42960E-03	1.46750E-03	1.0265	1.48483E-03	1.0386
45,55,5	2.64846E-04	3.00417E-04*	1.1343	2.88298E-04*	1.0885
55,55,5	9.14210E-05	9.58897E-05	1.0489	9.55481E-05	1.0451
ErrNorm (Err2Norm Err1Norm)		2.7730E-02	13.431%	1.9429E-02	8.855%

* Results are calculated by averaging the corresponding fine mesh(s), instead of coarse mesh.

Table 5-16. Kobayashi Problem 3 Point C set flux results for Case 3.

Point 3C	Analytical	Case 3 (S_{24})	Ratio	Case 3 (S_{30})	ratio
5,95,35	3.27058E-05	3.46102E-05	1.0582	3.16989E-05	0.9692
15,95,35	2.68415E-05	3.04241E-05	1.1335	2.88384E-05	1.0744
25,95,35	1.70019E-05	1.61464E-05	0.9497	1.86621E-05	1.0976
35,95,35	3.37981E-05	2.62570E-05	0.7769	2.38136E-05	0.7046
45,95,35	6.04893E-06	5.30795E-06*	0.8775	4.85885E-06*	0.8033
55,95,35	3.36460E-06	3.43148E-06	1.0199	4.00289E-06	1.1897
ErrNorm (Err2Norm Err1Norm)		1.2205E-01	22.312%	2.7493E-01	29.542%

* Results are calculated by averaging the corresponding fine mesh(s), instead of coarse mesh.

Problem 3 seems to be the most difficult one among the three Kobayashi problems, since particles tend to streaming along the dogleg void duct. As expected, the worst point (45, 95, 35) is located on the interface of the far end of the duct.

Analysis of Results

For the three problems, our calculation results show a relatively good agreement with the analytical solutions for most of the points. The characteristics solver also provides similar results as the S_N solver for problem 1. Figures 5.16-18 show the normalized flux calculation results for the S_N solver for the three problems (P_N-T_N S_{24} for Problem 1, P_N-T_N S_{30} for Problems 2&3).

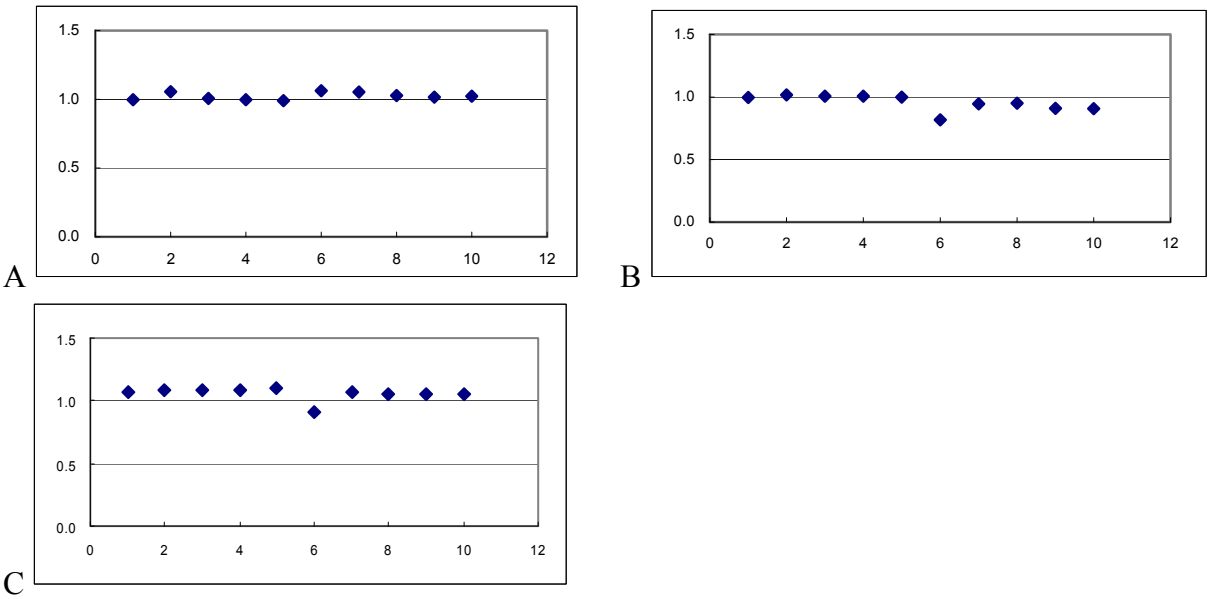


Figure 5-16. Relative fluxes for Kobayashi Problem 1. A) Point set A. B) Point set B. C) Point set C

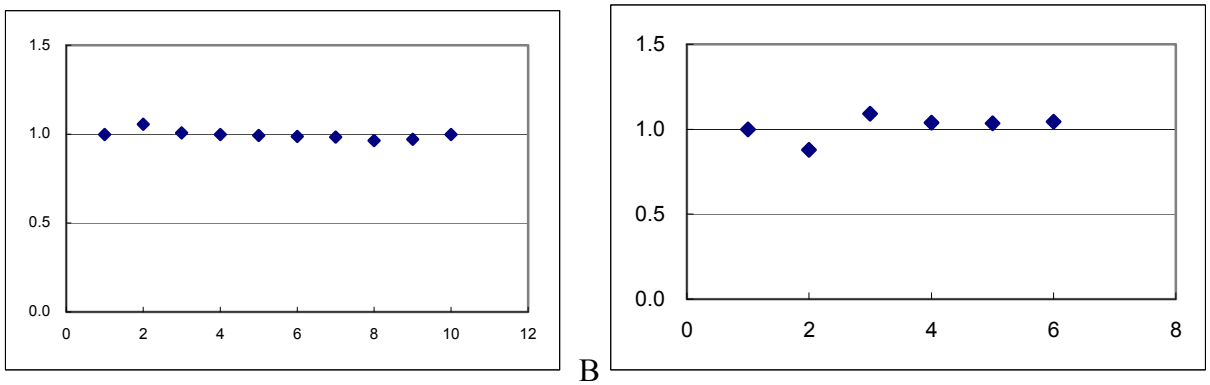


Figure 5-17. Relative fluxes for Kobayashi Problem 2. A) Point set A. B) Point set B.

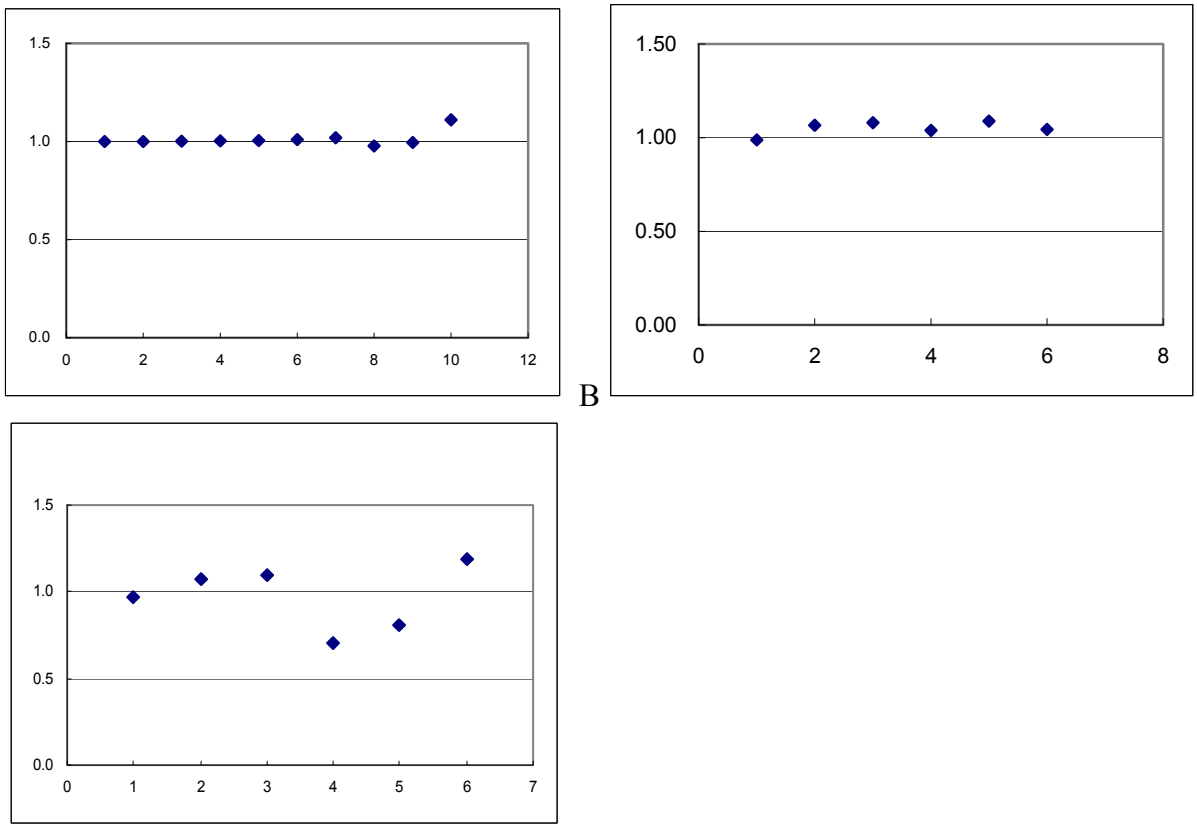


Figure 5-18. Relative fluxes for Kobayashi Problem 3. A) Point set A. B) Point set B. C) Point set C

Figures 5-16 to 5-18 show that points with relatively large errors typically occur on the interface between the void region and the pure absorber region due to the highly directional particle streaming on the interface. Since no scattering exists in the model and the source is

located in the corner, ray-effects could be very severe in this 3D model. Therefore, it is difficult for an S_N code without any ray-effect remedies to calculate all the point sets with only one calculation.[34]

Benchmark 4 - 3-D C5G7 MOX Fuel Assembly Benchmark

We tested the *k-effective* calculation ability of the TITAN code on the extended 3-D C5G7 MOX benchmark.[35, 36] TITAN categorizes transport problems into four types: fixed source problems with only down-scattering, fixed source with up-scattering, criticality with down-scattering, and criticality with up-scattering. Details on the four kernels are discussed in Chapter 4. This benchmark falls in the fourth category with the reflective boundary condition, which is numerically the most difficult type. The size of the model also presents a challenge for a serial non-lattice transport code as TITAN.

Model Description

The C5G7 MOX reactor is a proposed design for this benchmark, which has 2 by 2 assemblies (2 MOX assemblies and 2 UO2 assemblies). Each assembly is composed of 17x17 fuel pins. And the four fuel assemblies are surrounded by moderator as shown in Figure 5-19.

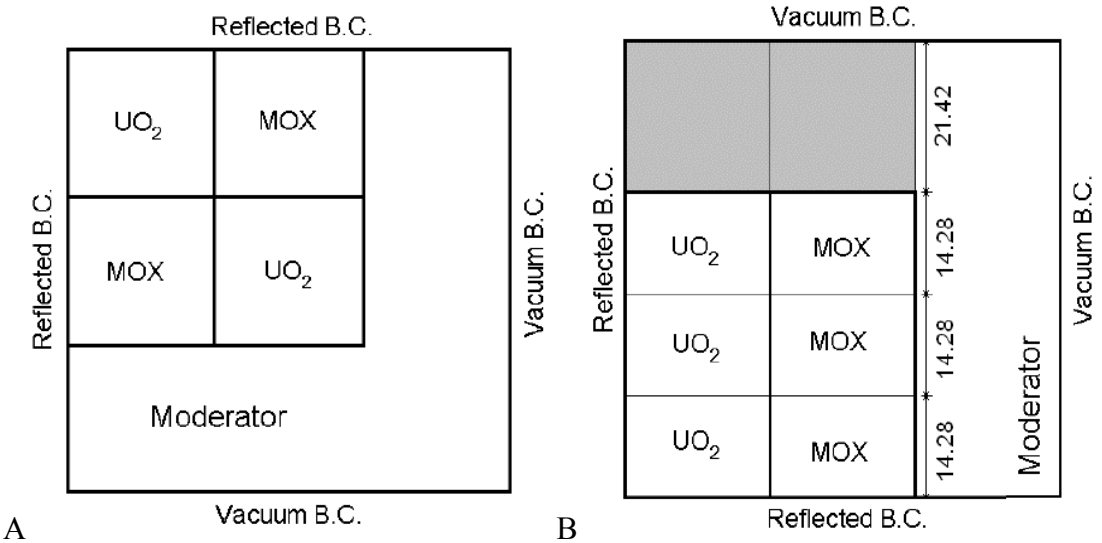


Figure 5-19. C5G7 MOX reactor layout. A) *x-y* plane. B) Unrodded configuration.

Axially the fuel region of the reactor can be divided equally into three segments as shown in Figure 5-18B. And control rods can be inserted into different depth of the core. Three control rod configurations are used in the extended version of this benchmark:[36]

- Unrodded.
- Rodded A.
- Rodded B.

In the unrodded case, the control rods only reach the moderator region on the top of the core (grey area in Figure 5-18B). In the other two cases, control rods in the MOX and UO2 assemblies reach different positions in the core.

Several models with different disctetization grids are tested. Only the S_N solver is used in the calculations. The finest grid model we used has about 3 million meshes (12 *z* levels) with a S_{10} quadrature set. This model requires ~1.8Gig memory. Based on the calculation results, the *k-effective* is relatively insensitive to the grid size, although the pin-power distribution does improve slightly with finer discretization grid. Figure 5-20 shows the meshing scheme for the 2x2 fuel assemblies and an individual fuel pin.

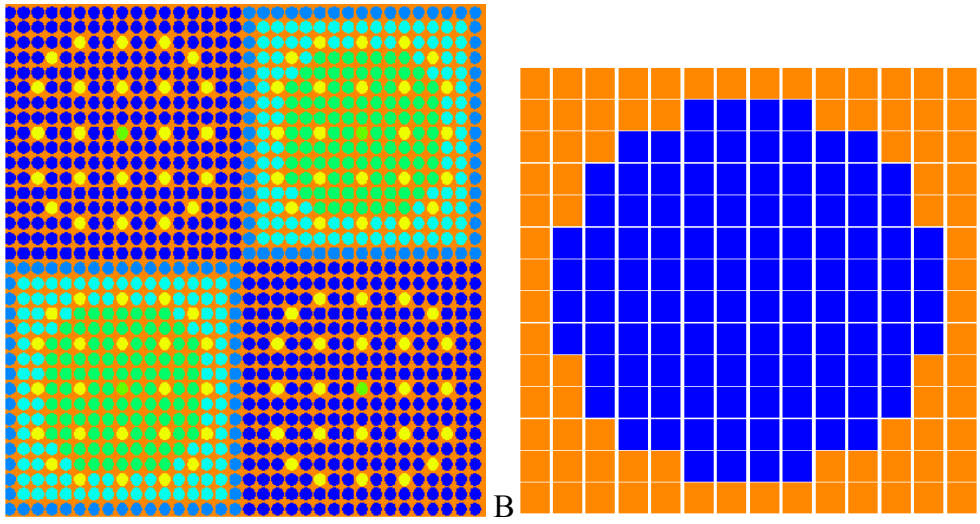


Figure 5-20. 3-D C5G7 MOX model. A) Four fuel assemblies. B) Fuel pin.

We use 14x14 fine meshes to represent each fuel pin in this four z-level model, which leads to a fine mesh size: 0.09x0.09x14.28 cm^3. The mesh size along *z* axis is much larger than *x-y* size, because finer meshing is required to represent the round shape of the fuel pin in the Cartesian geometry. A minimum four z-levels are required to represent the different control rod configurations. It is necessary to add more z-levels to resolve the axial flux shape because of different control rod configurations. However, the tests indicate that *k-effective* is more sensitive to the *x-y* size than the *z* mesh size. Here we only reported the four z-level S_6 model calculation results due to our computation resource limitation. The model has about one million fine meshes. Note that the multigroup cross section data and the reference solutions (acquired by Monte Carlo calculations) are provided with the benchmark.

Pin Power Calculation Results

The Monte Carlo reference solution provides the pin power distribution for the three slices in the reactor core region. In the TITAN model, each fuel pin is composed of 14x14 fine meshes. Since the power is proportional to the fission rate, a special subroutine is developed to evaluate the pin power by summating the fission rates for all the 14x14 fine meshes and for all the seven energy groups. Then, the output can be imported to the EXCEL template provided with the benchmark specification. The differences between user calculation results and the reference are automatically evaluated by the template. The pin power results calculated by TITAN for the unrodded case are compared with the reference solution in Table 5-17.

Table 5-17. Pin power calculation results for the unrodded case

	Z Slice #1	Z Slice #1	Z Slice #2	Z Slice #2	Z Slice #3	Z Slice #3	Overall	Overall
Specific Pin Power Data	Ref.	User	Ref.	User	Ref.	User	Ref.	User
Maximum Pin Power	1.108	1.148	0.882	0.884	0.491	0.449	2.481	2.481
Percent Error (associated 68% MC)	0.090	3.563	0.100	0.244	0.130	-8.449	0.060	0.007
Distribution Percent Error Results								
Maximum Error (associated 68% MC)	0.220	4.673	0.320	1.803	0.130	8.449	0.192	1.395
AVG Error	0.164	3.340	0.183	0.421	0.245	7.069	0.109	0.268
RMS Error	0.171	3.381	0.190	0.536	0.255	7.096	0.114	0.354
MRE Error	0.062	1.496	0.055	0.140	0.042	1.445	0.093	0.200
Number of Accurate Fuel Pin Powers								
Number of Fuel Pins Within 68% MC	371	0	371	146	371	0	371	147
Number of Fuel Pins Within 95% MC	518	0	518	278	518	0	518	257
Number of Fuel Pins Within 99% MC	540	0	540	334	540	0	540	336
Number of Fuel Pins Within 99.9% MC	544	0	544	387	544	0	544	398
Total Number of Fuel Pins	545	545	545	545	545	545	545	545
Average Pin Power In Each Assembly								
UO2-1 Power	219.04	226.70	174.24	173.79	97.93	90.54	491.21	491.03
MOX Power	94.53	97.38	75.25	75.10	42.92	39.95	212.70	212.44
UO2-2 Power	62.12	64.55	49.45	49.65	27.82	25.89	139.39	140.09
UO2-1 Power Percent Error	0.082	3.498	0.073	-0.258	0.055	-7.554	0.123	-0.038
MOX Power Percent Error	0.061	3.017	0.054	-0.193	0.041	-6.911	0.092	-0.122
UO2-2 Power Percent Error	0.043	3.920	0.038	0.404	0.029	-6.936	0.065	0.506

The format of Table 5-17 is provided by the benchmark template. In the unrodded case, control rods are inserted to the moderator region on the top of the reactor core. The TITAN results show a relatively good agreement with the reference solution for the overall pin power distribution (power summation of the three axial segments). However, large differences exist if we compare different segments, especially Slices #1 and #3. The error could be attributed to the large mesh size along the *z* axis and the lower order of the quadrature set. Similar error pattern also occurs in the rodded A and B cases as provided in Tables 5-18 and 5-19.

Table 5-18. Pin power calculation results for the rodded A case.

	Z Slice #1	Z Slice #1	Z Slice #2	Z Slice #2	Z Slice #3	Z Slice #3	Overall	Overall
Specific Pin Power Data	Ref.	User	Ref.	Uer	Ref.	User	Ref.	User
Maximum Pin Power	1.197	1.211	0.832	0.826	0.304	0.321	2.253	2.274
Percent Error (associated 68% MC)	0.080	1.145	0.100	-0.696	0.200	5.518	0.059	0.919
Distribution Percent Error Results								
Maximum Error (associated 68% MC)	0.100	1.625	0.250	1.877	0.330	7.044	0.149	1.701
AVG Error	0.157	0.691	0.180	0.760	0.260	3.922	0.108	0.714
RMS Error	0.163	0.819	0.186	0.860	0.266	4.251	0.111	0.803
MRE Error	0.066	0.388	0.056	0.297	0.037	0.582	0.094	0.690
Number of Accurate Fuel Pin Powers								
Number of Fuel Pins Within 68% MC	371	87	371	60	371	11	371	14
Number of Fuel Pins Within 95% MC	518	163	518	113	518	13	518	43
Number of Fuel Pins Within 99% MC	540	200	540	160	540	18	540	70
Number of Fuel Pins Within 99.9% MC	544	237	544	216	544	25	544	104
Total Number of Fuel Pins	545	545	545	545	545	545	545	545
Average Pin Power In Each Assembly								
UO2-1 Power	237.41	240.06	167.51	165.79	56.26	58.89	461.18	464.74
MOX Power	104.48	104.67	78.01	77.42	39.23	38.27	221.71	220.36
UO2-2 Power	69.80	70.51	53.39	53.18	28.21	26.85	151.39	150.54
UO2-1 Power Percent Error	0.087	1.118	0.071	-1.029	0.040	4.674	0.119	0.772
MOX Power Percent Error	0.065	0.182	0.056	-0.747	0.040	-2.447	0.094	-0.610
UO2-2 Power Percent Error	0.047	1.012	0.040	-0.382	0.029	-4.817	0.068	-0.565

In the rodded A case, control rods are inserted to the Slice 3 in one assembly. Slice 3 is the top slice in the reactor core region, which has the least power contribution among the 3 slices. Slice 3 has the largest percentage error. The maximum error associated 68% Monte Carlo reference is about 7% for Slice 3, while it is about 2% for the other two slices. The overall assembly power errors are less than 1% for both UO2 and MOX assembly.

Table 5-19. Pin power calculation results for the rodded B case.

	Z Slice #1	Z Slice #1	Z Slice #2	Z Slice #2	Z Slice #3	Z Slice #3	Overall	Overall
Specific Pin Power Data	Ref.	User	Ref.	Uer	Ref.	User	Ref.	User
Maximum Pin Power	1.200	1.167	0.554	0.585	0.217	0.205	1.835	1.818
Percent Error (associated 68% MC)	0.090	-2.779	0.150	5.603	0.240	-5.657	0.083	-0.890
Distribution Percent Error Results								
Maximum Error (associated 68% MC)	0.090	3.438	0.140	6.689	0.220	15.558	0.071	1.715
AVG Error	0.146	1.313	0.181	2.713	0.285	4.407	0.105	0.710
RMS Error	0.150	1.557	0.184	3.231	0.290	5.714	0.108	0.823
MRE Error	0.073	0.916	0.055	0.971	0.034	0.577	0.098	0.727
Number of Accurate Fuel Pin Powers								
Number of Fuel Pins Within 68% MC	371	36	371	4	371	31	371	37
Number of Fuel Pins Within 95% MC	518	75	518	6	518	51	518	78
Number of Fuel Pins Within 99% MC	540	93	540	20	540	70	540	109
Number of Fuel Pins Within 99.9% MC	544	121	544	35	544	88	544	137
Total Number of Fuel Pins	545	545	545	545	545	545	545	545
Average Pin Power In Each Assembly								
UO2-1 Power	247.75	241.84	106.56	112.38	41.12	37.45	395.43	391.67
MOX Power	125.78	124.15	81.41	82.93	29.42	30.42	236.62	237.50
UO2-2 Power	91.64	91.97	65.02	66.40	30.68	30.95	187.34	189.33
UO2-1 Power Percent Error	0.091	-2.385	0.056	5.460	0.035	-8.924	0.112	-0.951
MOX Power Percent Error	0.073	-1.300	0.058	1.875	0.034	3.379	0.100	0.374
UO2-2 Power Percent Error	0.055	0.364	0.046	2.124	0.032	0.899	0.078	1.062

The axial flux profile becomes more and more difficult to resolve from the unrodded case to the rodded B case, as the control rods insert deeper in the reactor core with different configurations. As a result, one can observe the overall pin power accuracy worsens slightly from Table 5-17 to 5-19. This is expected, considering we only use the minimum 4 z-levels model with the diamond differencing scheme, and a relatively lower quadrature order.

Eigenvalue Comparison

The eigenvalues for the three cases are listed in Table 5-20. The tolerance used in TITAN input file for *keff* is *1.0E-05*. Similar to the pin power error, the *keff* error increases as the control rods insert further into the core.

Table 5-20. Eigenvalues for three cases of C5G7 MOX benchmark problems.

Case	Ref.	% Error (68% MC)	User	Difference (pcm).
Unrodded	1.143080	0.0026	1.13911	169
Rodded A	1.128060	0.0027	1.12600	206
Rodded B	1.077770	0.0028	1.07415	362

Analysis of Results

Since the S_N solver in the TITAN code is designed only for Cartesian geometry. We had to use an 'unusual' meshing scheme in this benchmark: the *z* mesh size is about 158 times larger than the *x* or *y* mesh size. Such imbalanced meshing could be valid only for problems in which the axial flux changes very slowly comparing with radical flux profile. Our computer hardware limitation is another reason why we use a reduced meshing scheme (about 1 million fine meshes). TITAN is still serial code. And we need to fit the whole problem onto one machine. It takes about 10 hours to run the 4 z-level S_6 model on an AMD Opteron 242 CPU (1.6MHz, 1024k cache) with about 323M memory requirement. The calculation result is reasonable considering the meshing scheme we used.

Specially designed lattice transport codes for reactor neutronics could be more efficient for this benchmark. However, TITAN has the potential to increase the efficiency in eigenvalue problems with some power iteration acceleration techniques implemented. Figure 5-21 shows the eigenvalue convergence pattern for the rodded A case.

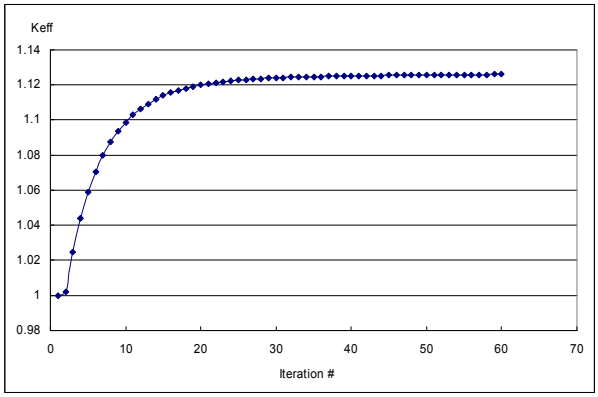


Figure 5-21. Eigenvalue convergence pattern for the rodded A configuration.

The rodded A case takes more than 2,000 inner iterations and about 60 outer iterations. As shown in Figure 5-20, the *k-effective* converged relatively fast for the first 20 iterations without any power acceleration technique applied. The convergence rate is much slower for the rest of the iterations, although this pattern is generally expected. The output indicates that some iterations are wasted to converge fluxes with the un-converged fission source, especially with up-scattering present. We took some intuitive measures in the code to improve the pattern, including using adaptive flux convergence criterion, adaptive inner loop and outer loop iteration number limitations, and Aitken extrapolation method.[37] I also combined the up-scattering loop and the power loop into one loop at the beginning, and separated them toward the end. These measures are optional in TITAN (Appendix D). And they can improve the convergence rate in certain situations.

CHAPTER 6
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.

Extra Sweep with Fictitious Quadrature

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.

$$S_{scattering}^{(e.s.)} = \sum_{g'=1}^{G}\sum_{l=0}^{L}(2l+1)\sigma_{s,g'\to g,l}\{P_l(\mu_n^{(fic)})\cdot\phi_{g',l}^{(con)} + 2\sum_{k=1}^{l}\frac{(l-k)!}{(l+k)!}P_l^k(\mu_n^{(fic)})\cdot \left[\varphi_{C,g',l}^{k,(con)}\cdot\cos(k\varphi_n^{(fic)}) + \varphi_{S,g',l}^{k,(con)}\cdot\sin(k\varphi_n^{(fic)})\right] \tag{6-1}$$

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.

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.

Extra Sweep Procedure

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 6-1.

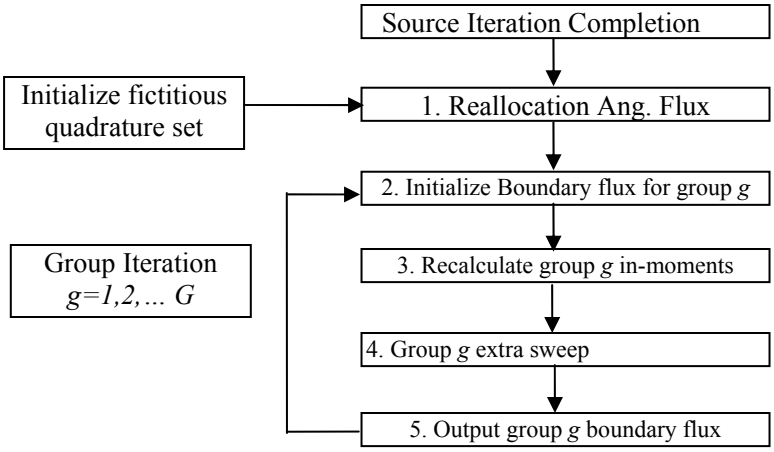


Figure 6-1. Extra sweep procedure with fictitious quadrature.

As shown in Figure 6-1, 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.

Implementation Concerns

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.

Iteration structure

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.

Direction singularity

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.

Solver compatibility

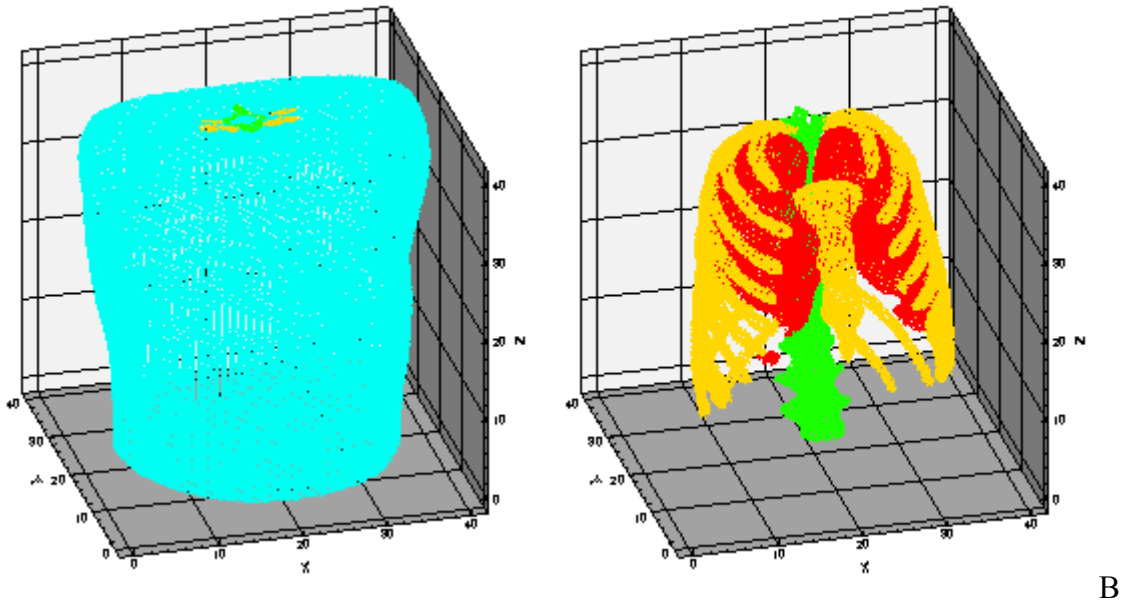
The two-solver structure of the TITAN code causes another dimensional difficulty in the implementation of the fictitious quadrature set. The technique is originally designed for the S_N solver only. Later the compatibility to the characteristics solver is achieved.

Heart Phantom Benchmark

Originally, we developed the fictitious quadrature technique to calculate the boundary angular fluxes for a single photon emission computed tomography (SPECT) benchmark. In SPECT, a small amount of photon radiation source is deposited in the target organ with some nuclear medicine intake. The source distribution in the organ can be reconstructed with the projection images. The 3-D source distribution image can be used to diagnose some malfunctions in the organ. Dozens of projection images from different angles are required to reconstruct the source distribution to achieve a good resolution. In medical physics, SPECT simulation usually is performed with the Monte Carlo approach. In this benchmark, our goal is to simulate the projection images of a body phantom with a deterministic transport calculation.

Model Description

We applied the TITAN code on a digital heart phantom generated by the NCAT code.[38] NCAT can provide various cardiac-torso phantoms with support of the heart motion. Users can specify the amount of activity deposited in different organs. The phantom we use in this benchmark is the first frame of the heart motion cycle. The phantom contains two binary file: the attenuation file and the activity file. The attenuation file records the linear attenuation coefficients for each voxel. And the radiation activity in each voxel is stored in the activity file.



A B

Figure 6-2. Heart phantom model. A) Torso. B) Organs.

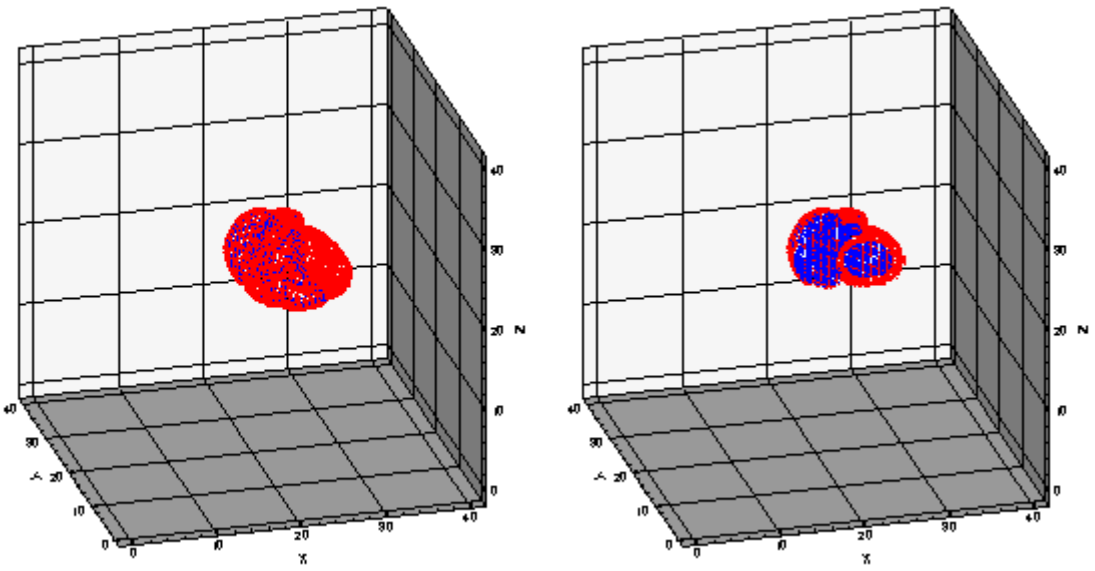
Figure 6-2 shows the material distribution of the phantom. The size of the phantom is 40x40x40 cm^3 with 128x128x128 voxels. The voxel size is 0.3125x0.3125x0.3125 cm^3. In the deterministic transport calculation, we consider the whole phantom as one coarse mesh with 128x128x128 fine meshes. The model has about 2 million fine meshes, and each fine mesh represents a voxel in the phantom. The body is surrounded by air. Most of the organs including the heart are considered the same material, except for the lungs and the bones. A total of five materials are used in this model as listed in Table 6-1. The material densities and linear

attenuation coefficients are given in the output files of the NCAT code. The TITAN code also can process the phantom attenuation binary file, and automatically generate a material list based on the different attenuation values in the file.

Table 6-1. Materials list in the heart phantom model.

Mat. Number	Mat. name	Density (g/cm^3)	Linear Attenuation Coefficients (1/pixel) or (0.3125cm/pixel)
1	Air	1.00E-06	
2	Body (Muscle)	1.02	0.0469
3	Dry Spine Bone	1.22	0.0520
4	Dry Rib Bone	1.79	0.0653
5	Lung	0.30	0.0135

Figure 6-3 shows the source activity distribution. The radiation source is deposited only in the heart, with 75 unit activity in the myocardium (heart muscular substance) and 2 unit activity in the heart chambers (blood pool).



A B

Figure 6-3. Activity distribution in the phantom model. A) Heart. B) Heart cross section view.

Photon Cross Section for the Phantom Model

The CEPXS package[39] is used to generate the cross section data for this benchmark. The group structure is decided based on the gamma decay energy of Tc-99m (~140kev), which is

widely used in the area of cardiac nuclear medicine. We also assume a typical 7% energy resolution for the *NaI* detectors used in the SPECT camera. Therefore, the width of the first group is about 10keV with a mid-range energy of 140keV. The rest of the group structure can be arbitrary chosen, since only the angular flux for the first group is required in this benchmark. In Table 6-2, we present a four-group structure with only down scattering.

Table 6-2. Group structure of cross section data for the heart phantom benchmark.

Group Number	Upper Energy Bound (keV)	Lower Energy Bound (keV)
1	145	135
2	135	100
3	100	50
4	50	1

An ideal SPECT camera takes projection images only from the uncollided photons. Therefore, only the first group angular fluxes on the boundaries are required to simulate the projections images. To deliver the cross section data, CEPXS also requires the weight percentage for each element in a mixture and its density. For the five materials listed in Table 6-1, the body and lung materials (mat. # 2 and 5) can be considered as water. And we assume the bone materials are composed of 22% water and 78% calcium. Detailed material compositions and densities are provided in Table 6-3.

Table 6-3. Material densities and compositions used in CEPXS.

Mat. #	Name	Density (g/cm^3)	Composition (element : weight fraction)
1	Air	1.00E-06	N : 0.78 O: 0.22
2	Body (Muscle)	1.02	H : 0.111 O: 0.889
3	Dry Spine Bone	1.22	H: 0.024 O: 0.196 Ca: 0.78
4	Dry Rib Bone	1.79	H: 0.024 O: 0.196 Ca: 0.78
5	Lung	0.30	H : 0.111 O: 0.889

Cross section data sets with Legendre order of 0 and 3 are generated based on the group structure (Table 6-2) and mixture composition (Table 6-3). Note that deterministic calculation results for the lower groups carry some information about the phantom. They might be useful to improve the quality of the reconstructed phantom image.

Performance of Fictitious Quadrature Technique

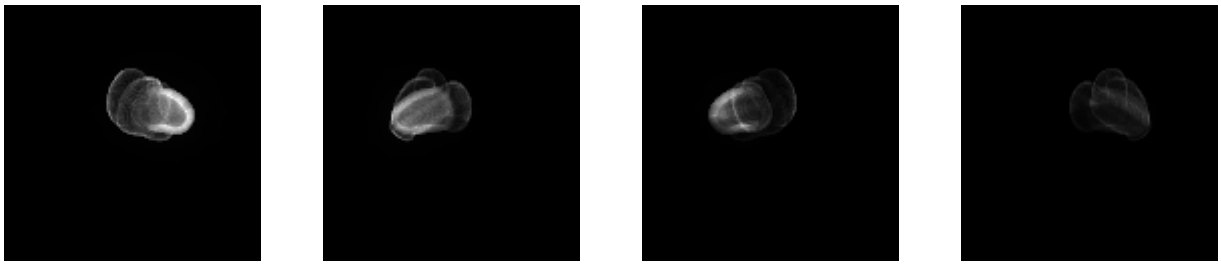
We demonstrate the TITAN code's performance on this benchmark by simulating the four projection images along the directions normal to the four boundaries parallel to the *z* axis. Four directions are defined in the fictitious quadrature specification file:

Table 6-4. Directions in the fictitious quadrature set for the heart phantom benchmark.

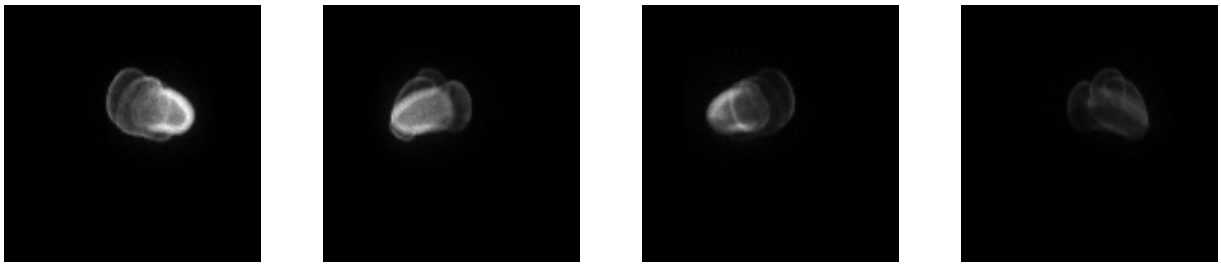
Direction Number	μ	η	ξ	Description
1	1.0	0	0	Normal to the left boundary
2	-1.0	0	0	Normal to the right boundary
3	0	1.0	0	Normal to the back boundary
4	0	-1.0	0	Normal to the front boundary

The four directions in Table 6-4 are singularity directions. The angular fluxes along the four directions on the corresponding model boundaries are computed with an extra sweep after the source iteration process is completed. Assuming a perfect 128x128 collimator array adjacent to the body (i.e. all other photons are blocked except those along the interested directions), the angular flux distribution can be used to simulate the projection images taken by the SPECT camera. More directions can be added in the fictitious quadrature to simulate projection images from other angles. Since the phantom model has 128x128x128 fine meshes, all the four simulated images have 128x128 pixels. We simulated several cases with different S_N orders (S_8 and S_{10}) order and P_N order (P_0 and P_3) with the S_N solver. The output images are similar. We also performed a Monte Carlo reference calculation with the SIMIND code.[40] SIMIND is a Monte Carlo code used in the nuclear medicine discipline to generate SPECT projection images.

SIMIND uses about 8 minutes (2 min/projection) on a 1.5GHz Pentium 4 processor. While it takes about 4 minutes for TITAN to compute the boundary angular fluxes for the first group on an AMD Opteron 242 CPU (1.6MHz, 1024k cache), which is about twice faster than the Pentium CPU. Figure 6-4 compares the globally normalized images calculated by TITAN (S_8 and P_0) and SIMIND. And Figure 6-5 compares the individually normalized images.

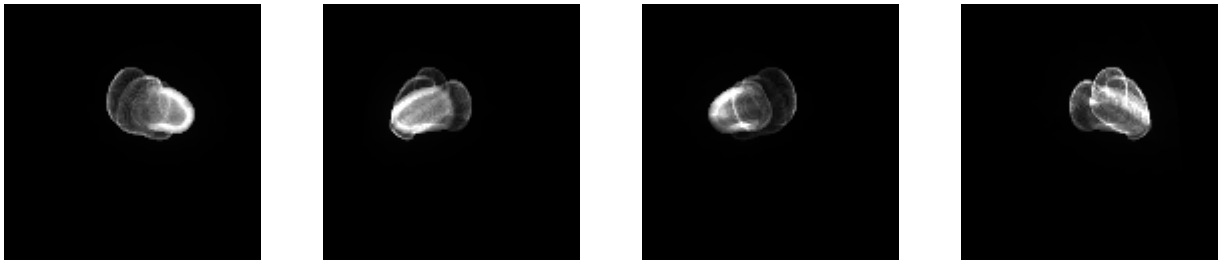


A1) TITAN: front A2) TITAN: left A3) TITAN: back A4) TITAN: right

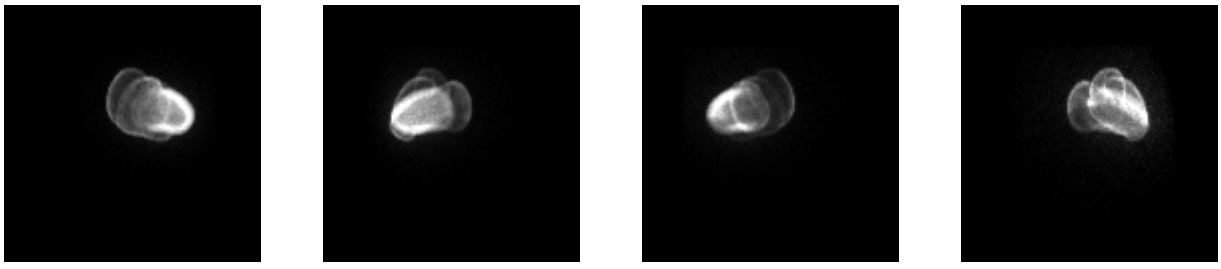


B1) SIMIND: front B2) SIMIND: left B3) SIMIND: back B4) SIMIND: right

Figure 6-4. Globally normalized projection images calculated by TITAN and SIMIND.



A1) TITAN: front A2) TITAN: left A3) TITAN: back A4) TITAN: right



B1) SIMIND: front B2) SIMIND: left B3) SIMIND: back B4) SIMIND: right

Figure 6-5. Individually normalized projection images calculated by TITAN and SIMIND.

In Figure 6-4, the four images (front, left, back, and right) are normalized together. It provides a valid intensity comparison between the four images, among which the right projection is the weakest, since it has the longest distance to the heart. In Figure 6-5, the four images are normalized individually. It shows a clearer view on the difference between SIMIND and TITAN calculation results. By visual comparison, it seems that the TITAN computed images have a higher contrast ratio. For a better understanding the amount difference between the results, Table 6-5 provides the overall differences for the voxels above 90% intensity, which are mostly located in the heart region.

Table 6-5. TITAN calculation errors relative to the SIMIND simulation.

Images	Max. Error	2-norm Error
Front	18.89%	3.711E-03
Left	11.29%	1.349E-03
Back	41.92%	6.882E-03
Right	40.22%	8.950E-03

As expected, larger differences are observed in the back and right projections that are farther from the heart as compared to left and front projections. Further, the 2-norm of the results is very low, indicating the maximum errors occur at small fraction of voxels. The differences could be attributed to the following:

- In SIMIND simulation, we specified a parallel collimator and NaI detector. The effects, including particle reaction in collimator septa and detector efficiency, are not considered in the TITAN code.
- SIMIND uses an equal number of particles (i.e., 767,555) to generate all the four projection images, while they are located at significantly different distances from the hearth. Hence, it is expected that the back and right images exhibit larger relative errors. In order to resolve this important issue, it is essential to determine the pixel-wise statistical uncertainty map in SIMIND.
- TITAN uses the group cross section file generated by CEPXS. While the continuous energy cross section data built in SIMIND is tuned to the human body materials and SPECT simulation. Some errors could be due to the cross section data.

Particle transport problems for SPECT traditionally are simulated by the Monte Carlo approach. Although it is still difficult to perform a strict comparison with the Monte Carlo simulation by SIMIND due to the reasons discussed in the previous section, the preliminary results of the TITAN calculation show a reasonably good agreement with the reference. One potential advantage of deterministic method over the Monte Carlo approach is the reduced computation time when simulation of a large number of projection images is required. In a SIMIND simulation, the CPU time is proportional to the number of projection images. While the computational cost for TITAN is mostly dedicated to the calculation of the flux moments. After the flux moments are converged, an extra sweep can compute a projection image with much less cost. Furthermore, flux moments can be stored after the transport calculation. And projection image simulations for different angles can be processed in parallel using the same stored flux moments. Therefore, TITAN could be much faster for simulation to a large number of projection images.

The usage of the fictitious quadrature is not limited to SPECT simulations. The technique is a relatively reliable approach to evaluate the angular fluxes in interested directions. However, currently extra sweep with fictitious quadrature can be applied only for problems with vacuum boundary condition. And although multiple regular quadrature sets can be defined in TITAN, only one fictitious quadrature is allowed in one problem model.

CHAPTER 7
PENTRAN INTEGRATION AND LIMITATION STUDIES OF THE CHARATERISTICS SOLVER

The coarse mesh/fine mesh scheme in the multi-block framework of the TITAN code is the same as the one used in the PENTRAN code. The block-oriented S_N solver and characteristics solver are developed based on the meshing scheme. We incorporated a modified version of the characteristics solver into the parallel engine of the PENTRAN code. In this chapter, the implementation of characteristics solver into PENTRAN is discussed. The performance of the integrated characteristics solver is tested on the simplified CT model benchmark with different parallel decomposition schemes. Finally, the limitations of the characteristics solver in TITAN are examined.

Implementation of the Characteristics Solver in PENTRAN

The data structure difference between PENTRAN and TITAN leads to some modifications on the characteristics solver in order to complete the integration. PENTRAN's data structure is tuned to the parallel environment. The major data arrays, including angular flux, flux moment, etc., are allocated locally. Since TITAN is still serial code, one major challenge is to seamlessly integrate the serial characteristics solver into the parallel engine.

In PENTRAN, based on the number of fine meshes within a coarse mesh, a memory-tuning procedure is used to group the coarse meshes into two categories: medium and large coarse meshes. While TITAN's object-oriented programming paradigm allows each coarse mesh to be treated individually. The structure of the angular flux array is built on the loop architecture of the source iteration scheme. The dimensions for energy group, coarse mesh, direction octant in the angular flux array are treated as parent objects of the fine mesh flux. PENTRAN also stores all the boundary fluxes for each fine mesh, and the boundary flux for each coarse mesh is stored implicitly with the fine mesh boundary arrays. In TITAN, both coarse mesh and fine mesh

boundary fluxes are treated explicitly by the frontline style sweep procedure, and only the front line fluxes are dynamically stored. Some differences of the memory structure between the two codes are listed in Table 7-1.

Table 7-1. Memory structure differences between PENTAN and TITAN

Array name	PENTRAN	TITAN
Angular flux	Two category, locally	Coarse mesh individually
Flux moment	Two category, locally	Coarse mesh individually
Fine mesh boundary flux	Stored	Not stored, front-line style sweep
Coarse mesh boundary flux	Stored	Not stored, front-line style sweep

We decided to keep the memory structure untouched in PENTRAN while integrating the characteristics solver. Thereby, instead of reallocating arrays, new arrays are allocated in PENTRAN when it is necessary, and de-allocated when they are not needed any more. Table 7-2 compares the characteristics solver in the modified version of PENTRAN (PENTRAN-CM) and TITAN.

Table 7-2. Comparison of the characteristics solver in PENTAN-CM and TITAN

	PENTRAN-CM	TITAN
Ray-tracing	On the fly	Pre-calculated
Geometry information	Not stored	Stored
Bilinear interpolation	Employed	Employed
Coarse mesh material	Void	Void, low-scattering medium, pure absorber
Projection compatibility	Not completely compatible with Taylor projection	Compatible with angular and spatial projection

In the TITAN code, the ray-tracing along the quadrature directions are performed as an initial task. And the calculated geometry information, such as intersection points, path lengths, and bilinear interpolation weights, are stored and can be accessed directly in the sweep process. Depending on the meshing and quadrature set, a relatively large amount of memory is required to store the geometry information. At the cost of memory, the characteristics solver can sweep the coarse meshes much faster. In the PENTRAN-CM code, the geometry information is not stored. The ray-tracing procedure is performed on the fly within every sweep. This approach is CPU-

intensive. However, it reduces the memory requirement. This approach is also suitable to the PENTRAN's coarse mesh data structure, thereby, requiring minimal programming changes. For compatibility reasons, currently, the characteristics solver in PENTRAN-CM can only be used in void regions. Note that PENTRAN is fully parallelized in the three domains (energy, angle, and space) of the phase space,[15] while TITAN is a serial code. However, in the PENTRAN-CM implementation, we take full advantage of the parallel engine, such that the characteristics solver module can be distributed to different processors to complete the assigned tasks. The individual tasks for each processor can be transport calculations for a subset of energy groups, octants, and/or coarse meshes specified by a decomposition scheme.[15]

Benchmarking of PENTRAN-CM

We tested the performance of the characteristics solver in PENTRAN-CM using the simplified CT benchmark discussed in Chapter 5. Some measures are taken in meshing, cross section and quadrature set, so that we can make a fair and valid comparison within the PENTRAN parallel engine.

Meshing, Cross Section and Quadrature Set

We recall that two meshing schemes are used in the original benchmark: the 7-coarse-mesh model (for the S_N solver shown in Figure 5-8) and the 3 coarse mesh model (for the hybrid solver shown in Figure 5-9). Both models are tested in this PENTRAN-CM benchmarking. A two-group cross section data file is used to test the parallel decomposition in the energy domain. The one-group data in the original benchmark is listed in Tables 7-3.

Table 7-3. One group cross section used in the CT benchmark with TITAN.

Material #	Σ_a	$\upsilon\Sigma_f$	Σ_t	Σ_s
1 (air)	4.77840E-09	0.0E+00	7.16860E-07	5.94460E-07
2 (source)	4.77840E-09	0.0E+00	7.16860E-07	5.94460E-07
3 (detector)	2.03430E-02	0.0E+00	3.88343E-01	2.60387E-01
4 (Water)	7.96423E-04	0.0E+00	1.48783E-01	1.23481E-01

Materials #1 and #2 are the same material. They are represented separately because the problem can be modeled more easily this way. Material #4 is the characteristics coarse mesh material. By changing the group constants of material #4, we can further examine the scattering ratio limitation of the characteristics solver. Table 7-4 lists the two-group cross section data used in PENTRAN-CM for group parallel decomposition.

Table 7-4. Two group cross section used in the CT benchmark with PENTRAN-CM.

Mat #	Grp #	Σ_a	$\upsilon\Sigma_f$	Σ_t	$\Sigma_{s,1\to g}$	$\Sigma_{s,2\to g}$
1	1	4.77840E-09	0.0E+00	7.16860E-07	5.94460E-07	0.0E+00
1	2	3.17640E-08	0.0E+00	8.79200E-07	7.75720E-07	1.17630E-07
2	1	4.77840E-09	0.0E+00	7.16860E-07	5.94460E-07	0.0E+00
2	2	3.17640E-08	0.0E+00	8.79200E-07	7.75720E-07	1.17630E-07
3	1	2.03430E-02	0.0E+00	3.88343E-01	2.60387E-01	0.0E+00
3	2	1.08305E-01	0.0E+00	5.48045E-01	3.78060E-01	1.07613E-01
4	1	7.96423E-04	0.0E+00	1.48783E-01	1.23481E-01	0.0E+00
4	2	5.29416E-03	0.0E+00	1.80640E-01	1.60378E-01	2.45063E-02

The two-group cross section data is mixed by the GIP code with the Sailor96 library with only down-scattering.[30] And the first group constants in Table 7-4 are the same as the one-group cross section data in Table 7-3. Therefore, the first group fluxes at the detectors remain the same regardless of the number of groups. We can compare the detector responses calculated by PENTRAN-CM with the TITAN results.

In the original CT model, we also applied the P_N-T_N ordinate splitting technique. In PENTRAN, only the rectangular ordinate splitting technique is available. In order to use the same quadrature, the P_N-T_N S_{20} quadrature set with two P_N-T_N splitting directions (Figure 5-11) is extracted from the TITAN code, and used as a user-defined quadrature set in PENTRAN. A minor modification in the quadrature routine of PENTRAN is made to process the split directions. In this quadrature set, there are total 207 directions in each octant.

Benchmark Results and Analysis

A number of cases are tested with the characteristics solver in a parallel environment as listed in Table 7-5. Note that in PENTRAN, the characteristics solver is included in PENTRAN's adaptive differencing strategy as Option 5.[15] (i.e. the differencing variable *ndmeth=5*).

Table 7-5 compares the first 10 detector responses calculated by PENTRAN-CM with TITAN. Note that the TITAN results are extracted from Table 5-1 for Case 5. The other 10 detectors are symmetric, thereby, they have the same responses as the first 10 detectors.

Table 7-5. Characteristics solver calculated detector response by PENTRAN-CM and TITAN.

Detector #	Case 1a in Table 7-3	Case 5 in Table 5-1	Difference
1	1.345E-03	1.345E-03	4.20E-07
2	1.474E-03	1.475E-03	3.60E-07
3	1.510E-03	1.510E-03	5.40E-07
4	1.579E-03	1.579E-03	-7.00E-08
5	2.095E-03	2.094E-03	-6.00E-07
6	2.123E-03	2.123E-03	-1.80E-07
7	2.131E-03	2.132E-03	9.50E-07
8	2.146E-03	2.146E-03	-4.50E-07
9	2.155E-03	2.155E-03	-5.90E-07
10	2.152E-03	2.152E-03	-4.00E-07

Table 7-5 shows the difference between the two cases is in the order of 10^{-7}, which is much lower than the scalar flux convergence tolerance 10^{-4}. Therefore, the characteristics solver produces the same calculation results within the machine truncation error in both TITAN and PENTRAN-CM. Table 7-6 compares the CPU time of PENTRAN-CM for a number of cases with different parallelization decomposition schemes. Note that the detector responses cases are almost the same as the results in Table 7-5. This also demonstrates the accuracy of PENTRAN's parallel engine for different parallelization decompositions schemes.

Table 7-6. Characteristics solver performance in PENTRAN parallel environment.

Case #	# of CM	# of CPU	Decomposition factor (decmpv[1])			Differencing Scheme (ndmeth[2])			CPU Time (sec)
			Angular	Group	Spatial	First coarse mesh	Middle coarse mesh	Last coarse mesh	
1a	3	16	8	2	1	-2	5	-2	7.7
1b	7	16	8	2	1	-2	-2	-2	33.3
2a	3	8	8	1	1	-2	5	-2	10.2
2b	7	8	8	1	1	-2	-2	-2	43.5
3	3	12	2	2	3	-2	5	-2	23.0
4a	3	1	1	1	1	-2	5	-2	64.0
4b	7	1	1	1	1	-2	5	-2	330.0
5a	3	1	Serial Run			-2	5	-2	61.4
5b	3	1	Serial Run			-2	-2	-2	323.0

[1] PENTRAN parallel decomposition variable.
[2] PENTRAN differencing scheme variable, *ndmeth=-2* corresponds to the Directional Theta-Weighted scheme, and *ndmeth=-5* corresponds to the characteristics solver.

Cases 1a and 1b use 16 processors with an angular decomposition factor of 8, an energy group decomposition factor of 2, and a spatial decomposition factor of 1. In Case 1a, we use the characteristics solver by setting *ndmeth=5* for coarse mesh #2. Case 1b applies the S_N solver only, and uses a total of 7 coarse meshes in order to overcome the ray-effects. The solutions for both cases are accurate comparing to the solution of Cases 4 and 5 in Table 5-1 respectively (compared in Table 7-5). An acceleration factor of about 4.3 is achieved with the characteristics solver comparing to the S_N solver, which is slightly lower than in TITAN code.

We can draw the same conclusion based on other cases. Cases 2a, 2b, and 3 use 8 and 12 processors respectively. Cases 4a and 4b are parallel runs, although only one processor is used. Case 5a and 5b provide the results for serial version of PENTRAN. It takes about 61.4 second with the characteristics solver, while about 323 seconds for the S_N solver with the refined meshing. This result shows that the characteristics solver is more efficient than the S_N solver in void regions in term of CPU time. In PENTRAN-CM, ray-tracing procedure is performed on the fly. In TITAN, the characteristics solver can be faster than the S_N solver even with the same meshing, since ray-tracing information is pre-calculated and stored.

Investigation on the Limitations of Characteristics Solver

Thus far, we have benchmarked the characteristics solver in TITAN with the CT model and Kobayashi problems. We also integrated the solver into the PENTRAN code, and tested the on-the-fly mode of the solver in a parallel environment. The hybrid approach with the characteristics solver shows an excellent performance on the benchmarks. However, the limitations of the solver and its sensitivity related to meshing and quadrature order are not fully addressed. In this section, we further analyze the characteristics solver based on its memory requirement, factors that affect accuracy, and possible improvement approaches.

Memory Usage

In the storage module of the characteristics solver, we use an array of user-defined type, called GEOSET in the TITAN code, to store the coarse mesh geometry information for the characteristics solver. The size of the GEOSET array equals to the product of the number of fine meshes on the coarse mesh boundaries and the number of directions in the quadrature set for the coarse mesh. Therefore, every characteristic ray in the coarse mesh requires a GEOSET object, which specifies five variables for the ray:

- Fine mesh index i at the incoming boundary (2 byte integer).
- Fine mesh index j at the incoming boundary (2 byte integer).
- Bilinear weight s on the incoming boundary (4 byte real).
- Bilinear weight t on the incoming boundary (4 byte real).
- Track length l of the ray (4 byte real).

These five variables, which are calculated by the ray-tracing routine before the source iteration process starts, represent all the required geometry information for a characteristic ray, if we consider the coarse mesh as one region. The pair (i, j) is used to locate the four fine meshes

on the incoming boundary for the bilinear interpolation procedure. The pair (*s, t*) is the bilinear weights as defined in Eq. 2.20. And *l* is the track length across the coarse mesh.

If we consider a four-byte real number as one memory unit, each GEOSET can be stored with 4 memory units. Note here we store the (*i, j*) pair as 2-byte integers, instead of the regular 4-byte integers. So the pair can be considered as one memory unit. The amount of memory required by the GEOSET can be very large with fine spatial meshing and high order of quadrature set. In certain cases, it can be even larger the S_N solver. For example, for a coarse mesh with 10x10x10 fine meshes and with the same quadrature, the S_N solver requires *1000*× *number of direction* memory units to store the angular flux. While the characteristics solver needs *10x10x6*× *number of direction* × *4* memroy units. The characteristics solver needs about twice amount of memory as the S_N solver. This is demonstrated in Table 5-11 with the Kobayashi benchmark problems.

The bilinear interpolation procedure requires at least *2x2* meshing on a boundary. On the other hand, because we use 2-byte integer to store the fine-mesh index in a GEOSET, the number of boundary fine meshes is limited to *255x255* for the characteristics solver, which is more than enough for most problems. We further discuss the mesh size limitation in the next section.

Limitation on the Spatial Discretization

A deterministic solver does not suffer from the statistical uncertainties as in the Monte Carlo approach. However, since in a deterministic method, the phase space has to be discretized, the solution accuracy is affected by mesh/grid size. Generally speaking, finer grid size (i.e. finer energy group structure, higher order quadrature set, and smaller spatial meshing) should lead to a more accurate solution at a higher computational cost. It is difficult to set up some universal criteria on how to decide the optimistic grid size, since it depends on both the algorithms and the individual problem model. Generally, a good understanding of the physics of the problem can

provide some guidelines in the process of modeling. For example, for a zero-moment S_N solver with the diamond differencing scheme, it is recommended to keep the mesh size under the material mean free path.

2-D meshing on the coarse mesh boundaries

In the characteristics solver, we integrate the transport equation along the characteristic rays. A 2-D meshing scheme is required on the coarse mesh boundaries. Generally, the 2-D meshing scheme is subject to the spatial discretization requirement for a deterministic solver. Furthermore, we need to consider two major factors to determine the mesh size on the coarse mesh boundaries for the characteristics solver.

- Angular flux distribution fluctuation on the coarse mesh boundaries.
- Angular flux resolution requirement on coarse mesh boundaries for the model.

The first factor is introduced with the bilinear interpolation procedure, which assumes a linear angular flux distribution on the local four fine meshes surrounding the intersection point of each ray with the incoming boundary. With a relatively flat incoming boundary flux distribution, larger fine mesh size can be used while preserving the accuracy of the bilinear interpolation. In an S_N coarse mesh, we specify the number of fine meshes (i, j, and k) along x, y, and z axes. In the characteristics solver, we still use the three integers to define the meshing on each boundary. For example, the two x-y boundaries have $i \times j$ fine meshes. With this meshing scheme, the bilinear interpolation can keep consistency on the incoming and outgoing boundaries for directions in different octants. More discussion on the accuracy of the bilinear interpolation was given in Chapter 2. The second factor can be illustrated with the simplified CT model as shown in Figure 7-1.

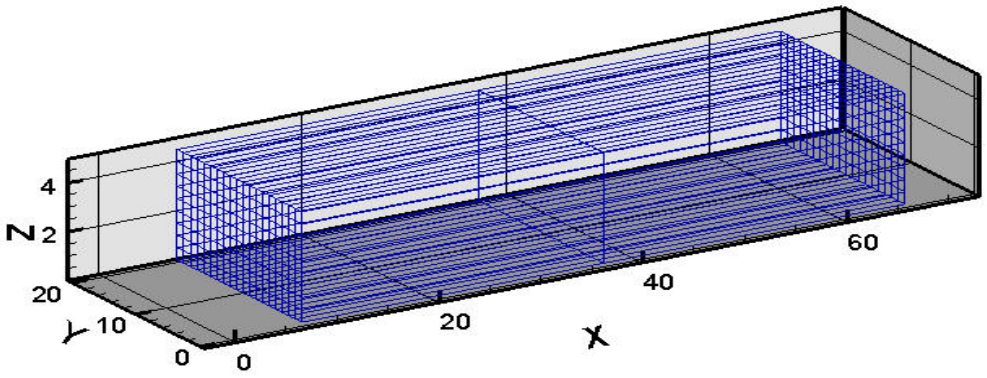


Figure 7-1. Characteristics coarse mesh boundary meshing based on flux resolution requirement.

Figure 7-1 shows the meshing scheme of the second coarse mesh in which the characteristics solver is used. We use 20x10 meshing on the two *y*-*z* boundaries, while only 2x2 meshing is applied on the other four boundaries. Our goal is to calculate the detector responses on the right side of the coarse mesh. Therefore, it is required to apply finer meshing on the *y*-*z* boundaries. We can use much coarser meshing on the other four vacuum boundaries because these boundary fluxes cannot affect the detector responses. Note that here we choose 2x2 meshing, which is the minimum requirement on meshing for the characteristics solver by the bi-linear interpolation procedure.

We also investigated the impact of the 2-D meshing on two *y*-*z* boundaries. The original hybrid model uses 20x10 meshing on *y*-*z* boundaries of coarse mesh #2. Figure 7-2 examines the detector response errors as compared to the reference MCNP case by using different number of *z* fine meshes. Case 1 is the MCNP reference case. In Case 2a to 6a, the characteristics solver is used in coarse mesh #2 with 5, 8, 9, 10, and 12 *z* fine meshes. The error curve moves up closer to the reference solution as increasing the number of *z* fine meshes from 5 to 8. It indicates that the characteristics solver provides more accurate solution with finer discretization grid.

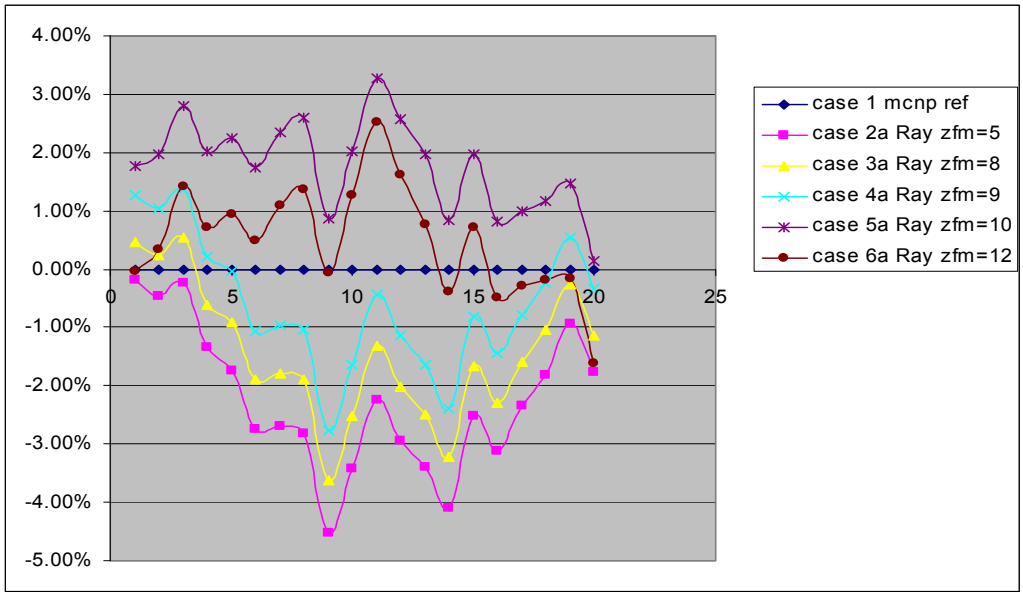


Figure 7-2. Detector response relative errors with different number of *z* fine meshes for the characteristics solver.

Figure 7-3 shows the relative errors for the S_N solver with different *z* meshing on the same coarse mesh. Note that for the S_N solver, the fine mesh size along *x* axis is *1cm*. Case 2b to 5b use the S_N solver in coarse mesh #2 with 5, 8, 9 and 10 *z* fine meshes.

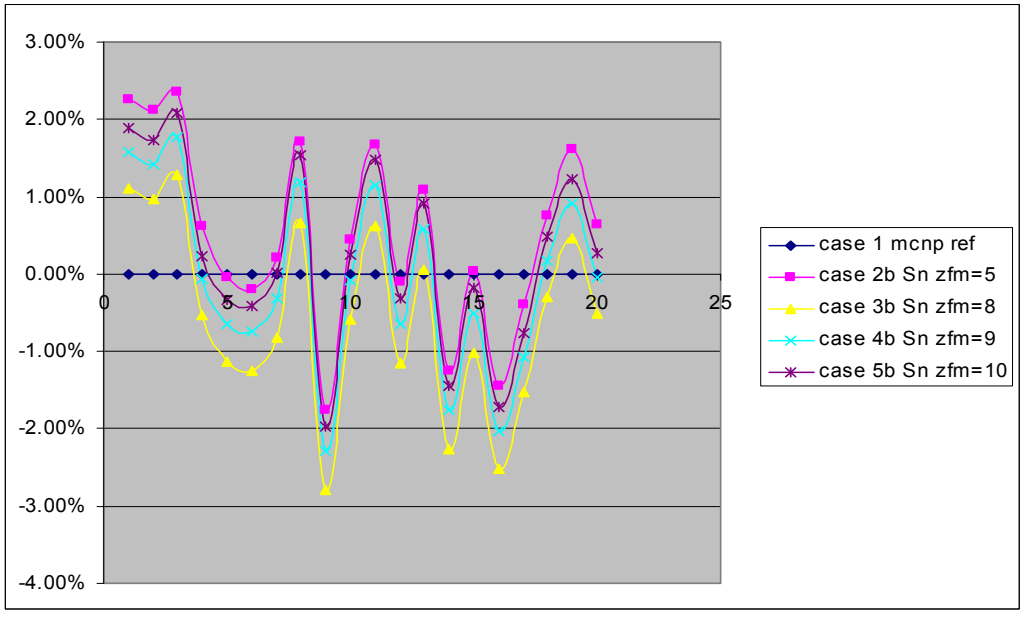


Figure 7-3. Detector response relative errors with different number of *z* fine meshes for the S_N solver.

All the curves in either Figure 7-2 or Figure 7-3 follow a similar trend. One can observe a jump when increasing *zfm=9* (Case 4a) to *zfm=10* (Case 4b) for the characteristics solver (*zfm* is the number of fine mesh along *z*). It seems that the solutions by the characteristics solver is affected by the *z* fine meshing more sensitively than the S_N solver. Table 7-7 provides the maxim percentage and 2-norm errors for all the cases.

Table 7-7. Error comparison with different *z* meshing.

Number of *z* fine meshes	Characteristics solver cases	Error 1-norm	Error 2-norm	S_N solver cases	Error 2-norm	Maxim error
5	Case 2a	1.3103E-02	4.536%	Case 2b	3.3245E-03	2.360%
8	Case 3a	6.7115E-03	3.622%	Case 3b	3.3309E-03	2.786%
9	Case 4a	3.1872E-03	2.771%	Case 4b	2.6947E-03	2.285%
10	Case 5a	7.5098E-03	3.280%	Case 5b	2.8202E-03	2.092%
12	Case 6a	2.1630E-03	2.515%		3.3245E-03	2.360%

Case 5a and Case 5b are the models used in the CT benchmark discussed in Chapter 5. Table 7-7 indicates that one can acquire a relatively accurate solution with different *zfm* numbers around 10, which demonstrates the stability of the hybrid algorithm. We further investigated the effects of *y* mesh size. Figure 7-4 shows the detector response sensitivity to the number of fine meshes along *y* axis (*yfm*) for the S_N and characteristics solver.

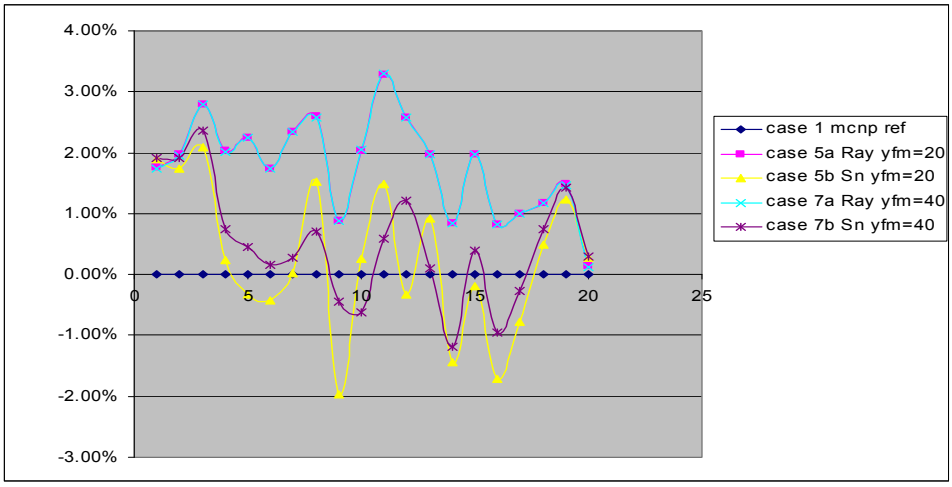


Figure 7-4. Detector response sensitivity to the fine mesh number along *y* axis.

The curves are acquired by using the same *z* fine mesh number (*zfm=10*), but different *y* fine mesh numbers and solvers. Cases 5a and 7a use the characteristics solver with *yfm=20* and *yfm=40*, respectively, While the S_N solver is used in Cases 5b and 7b. The two S_N curves follow a similar trend. And as expected, the solution for Case 7b (*yfm=40*) is more accurate than Case 5b (*yfm=20).* The solutions with *yfm=20* and *yfm=40* are nearly identical for the characteristics solver. This indicates the *yfm*=20 meshing scheme is fine enough for the characteristics solver to evaluate the 20 detector responses.

Coarse mesh size limitation for the characteristics solver

We further investigated the effects of the coarse mesh size on the accuracy of the characteristics solver. Since we consider the coarse mesh as one region, the limitation on the path length of the characteristic rays across the coarse mesh is the major factor in determining the coarse mesh size. The characteristics solver integrates the LBE along the rays with the assumption that the scattering source is constant throughout the coarse mesh in one sweep (flat source region). If the material for the coarse mesh is void or pure absorber, such assumption is valid because the scattering source is always zero. For example, in the CT model, we can use a large coarse mesh size with the characteristics solver in the air region. In materials other than void or pure absorber, the scattering ratio of the material is the major factor on the size limitation of the coarse mesh. With scattering collision increasing, we have to reduce the coarse mesh size to maintain the flat source assumption.

We examined this effect by changing the material cross section data in the CT model. Here, we use the S_N model as the reference, since we already validate the S_N solver on this model. The MCNP model requires much longer CPU time without variance reduction to achieve a good statistical confidence, because it is more difficult for the detectors to score a particle when increasing the total cross section in the 'air' region. For the S_N model, here we use *yfm=20*

and *zfm=5*. Therefore, all the S_N coarse meshes are filled with 1x1x1 cm^3 fine meshes. Note that the S_N solution is not necessarily as accurate as the MCNP reference we used in the original CT benchmark. However, it can provide a valid approximate reference solution for the purpose of this benchmark.

The accuracy of the characteristics solver in void regions is already demonstrated with the original CT benchmark. Here we further examined the performance of the solver in pure absorber regions. Figure 7-5 shows the detector response difference for the S_N and characteristics solvers with pure absorber in the 'air region' (cross sections for material in coarse mesh #2 are: $\sigma_t = 1.48783\text{E-}01\ \sigma_s = 0.0$).

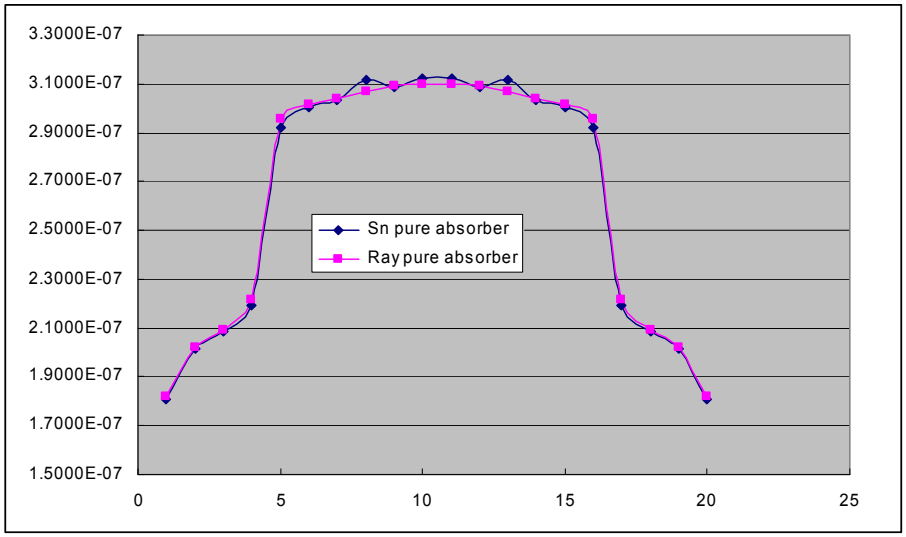


Figure 7-5. Detector response comparison between S_N and characteristics solver in pure absorber media.

The characteristics solution (characteristics coarse mesh meshing scheme: *yfm=20, zfm=5,* and *xfm=2*) shows a relatively close agreement with the S_N solution (maxim difference *1.52%*). This demonstrates that the characteristics solver is accurate in pure absorber media. Figure 7-5 shows that the characteristics solver is less sensitive to the ray-effects, which is also demonstrated in the original CT benchmark for void regions.

We further changed the cross section data with different scattering ratios while fixing $\sigma_t = 1.48783\text{E}-02$. Figure 7-6 shows the difference between the S_N and characteristics solutions for four different scattering ratios (σ_s / σ_t=0.05, 0.08, 0.10, and 0.20). Note here the characteristics coarse mesh size is *59cm* along *x* axis with meshing scheme: *yfm=20, zfm=5,* and *xfm=2*.

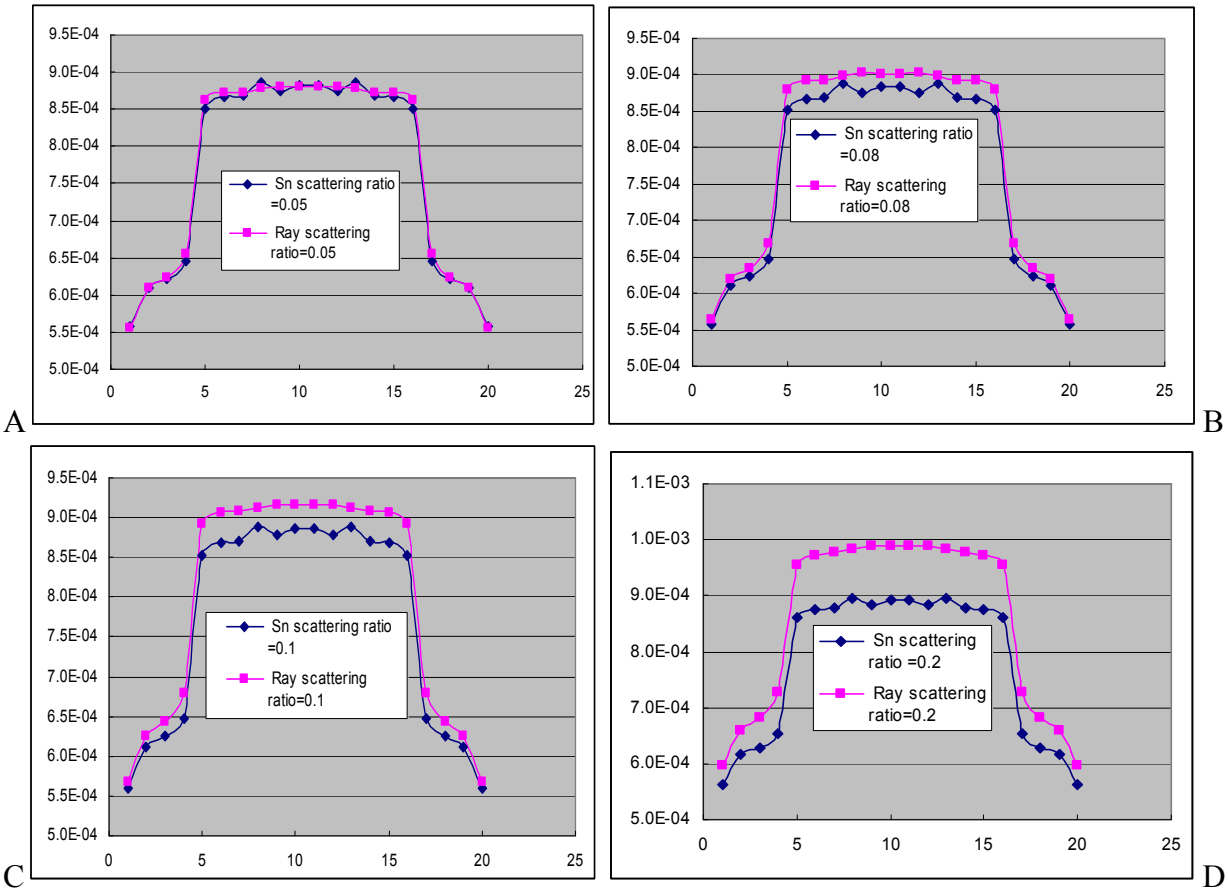


Figure 7-6. Detector response comparison between S_N and characteristics solver in media with different scattering ratio. A) ratio=0.05 B) ratio=0.08 C) ratio=0.10 D) ratio=0.20

By comparing the S_N solutions in Figures 7-6A and 7-6B, one can observe that the detector responses increase very slightly when increasing the scattering ration from 0.01 to 0.2. This is because that the detector responses are mainly dictated by the magnitude of the total cross section, which remains the same for all cases. Figure 7-6 also shows that the characteristics

solver tends to over-estimate the solution with higher scattering ratio. This can be attributed to the flat source assumption in Eq. 2-16, and it can be explained as follows. The scalar flux in the coarse mesh approximately decreases exponentially from the source region to the detector region ($\phi = e^{-\sigma_t x}$). Here the scattering source is the only contributing source term, since no fixed source is present in the characteristics coarse mesh. With the flat source assumption, the scattering source is calculated by multiplying the coarse-mesh averaged flux and the scattering cross sections, and it remains the same in the coarse mesh within each iteration. As a result, the scattering source is over-estimated as x close to the detector region, resulting in the overestimation of the outgoing angular fluxes for the coarse mesh, which leads to a higher detector response. The source term's contribution to the outgoing angular fluxes increases with the scattering ratio. Therefore, Figure 7-6 shows that the detector responses are overestimated further with higher scattering ratios.

In order to correct this overestimation (i.e. allow the flat source assumption to be applicable), it is necessary to decrease the length of the characteristic ray, or decrease the size of the coarse mesh along the axis of interest. Figure 7-7 compares the characteristics results with different coarse mesh sizes of 46 cm, 36 cm, and 32 cm to the S_N reference solution. For this test, we use a scattering ratio of 0.2.

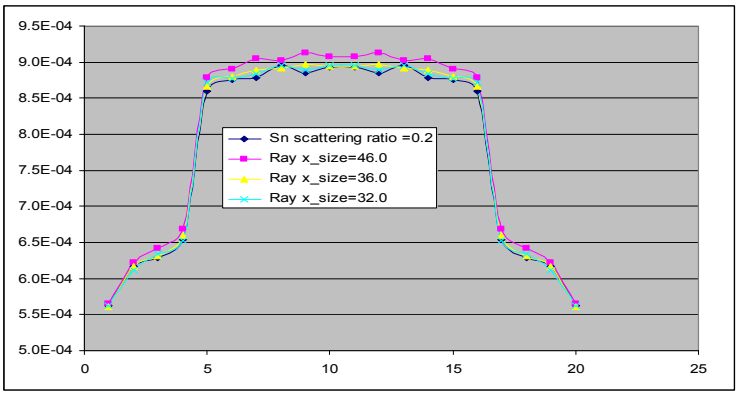


Figure 7-7. Characteristics solutions with different coarse mesh size along x axis.

As expected the characteristics solution approaches to the S_N solution as the coarse mesh size decreases. The relative errors and CPU time for all the characteristics cases in Figures 7-6 and 7-7 are given in Table 7-8.

Table 7-8. Characteristics solution relative difference to S_N solution with different scattering ratios and coarse mesh size.

Case #	Coarse mesh size along *x* (cm, mfp)	Total cross section (cm^{-1})	Scattering cross section	Scattering ratio	Error 2-norm	Error 1-norm	CPU Time Ratio (S_N/Ray)
1	59 (0.87*)	1.48783E-02	7.43915E-04	0.05	1.2761E-03	1.393%	3.13
2	59 (0.87)	1.48783E-02	1.19026E-03	0.08	1.1499E-02	3.336%	3.10
3	59 (0.87)	1.48783E-02	1.48783E-03	0.10	2.6466E-02	4.646%	3.08
4	59 (0.87)	1.48783E-02	2.97566E-03	0.20	1.9858E-01	11.677%	3.12
5	46 (0.68)	1.48783E-02	2.97566E-03	0.20	8.1535E-03	3.131%	1.88
6	36 (0.54)	1.48783E-02	2.97566E-03	0.20	1.1328E-03	1.330%	1.41
7	32 (0.48)	1.48783E-02	2.97566E-03	0.20	8.0144E-04	1.353%	1.28
8	27 (0.40)	1.48783E-02	3.71958E-03	0.25	7.5987E-03	3.026%	1.13
9	22 (0.32)	1.48783E-02	4.46349E-03	0.30	2.5802E-03	2.223%	1.01
10	17 (0.25)	1.48783E-02	5.95132E-03	0.40	1.6210E-03	1.487%	0.91

* Values in the parentheses are in unit of mean free path (mfp).

Based on the results in Table 7-8, we conclude that the characteristics solver can provide an accurate solution by reducing size of the coarse mesh with higher scattering ratio. For the cases with scattering ratio of 0.20 (Cases 4 to 7), the limitation on the distance along *x* is about 36 cm, which is about half of the mean free path for the material (~70 cm). Generally, the accuracy of the solver depends on both the scattering ratio and mean free path of the material. Table 7-8 also indicates that the product of scattering ratio and the mean free path, which is coarse mesh size in unit of scattering mean free path, should be around 0.1 or less. It is recommended that the characteristics solver is used for materials with a scattering ration less than 0.20, because with higher scattering ratio, users need to further refine the coarse mesh size. And The CPU time comparison in Table 7-8 indicates that the characteristics solver generates

less computational benefits as decreasing the mesh size as shown in Cases 7 to 10. In these four cases, we keep the coarse mesh size in unit of scattering mean free path close to 0.1, and the CPU time ratio decreases gradually. As in Case 10, the S_N becomes faster than the characteristics solver.

Possible Improvements and Extendibility of the Characteristics Solver

The meshing scheme on the characteristics coarse mesh boundaries are limited by the bi-linear interpolation. And the size and scattering ratio limitations are due to the flat source assumption. Therefore, we could further study on these two aspects to improve the accuracy of the characteristics solver. The bi-linear interpolation assumes that the average flux happens on the center of a fine mesh. We could develop a new interpolation scheme on the incoming boundaries, which addresses the fact that the point flux actually should be the averaged flux on the fine mesh area or the cross sectional area of the ray. Instead of assuming a flat scattering source throughout the region, we could use some higher scheme, for example, linear source assumption, to represent the source term more accurately. Investigations on these two aspects will continue.

In summary, the characteristics solver is efficient and accurate in void, pure absorber regions. For low-scattering medium with scattering ratio less than 0.20, as a conservative guideline, the size of the characteristics coarse mesh should be equal or less than half of the mean free path. For higher scattering ratio materials, in which the characteristics solver is not recommend, the coarse mesh size should be less than tenth of the scattering mean free path.

CHAPTER 8
CONCLUSIONS AND FUTURE WORK

Conclusions

We developed a hybrid algorithm to solve the LBE for realistic 3-D problems, especially for the problems containing large regions of low scattering media, where the traditional S_N method might become inefficient. A ray-tracing solver is designed and integrated in the TITAN code along with an S_N solver. Both solvers are written under the paradigm of object-oriented programming with the block-oriented feature. And they are built on the framework of a multi-block discretization grid (coarse/fine meshing scheme and block-localized angular quadrature). Both solvers are well-tuned in terms of memory management and CPU efficiency.

The main features of the TITAN code are:

- Integrated S_N and characteristics 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 two ordinate splitting techniques (rectangular and local P_N-T_N) for the two type of quadrature sets.
- 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.

We tested the performance of the TITAN code with a number of benchmark problems. For applications in the field of nuclear engineering, TITAN is used to solve the Kobayashi benchmark, which is a set of difficult shielding problems, and the 3-D C5G7 MOX benchmark, which is a *k-effective* problem without homogenization for a MOX/UO2-fueled reactor with different control rod layouts. For applications in the medical physics field, we tested the code on the CT device model, which is difficult for deterministic codes to solve due to ray-effects, and the SPECT phantom model, in which transport simulation is commonly performed only by the Monte Carlo approach. The fictitious quadrature technique we developed for the SPECT model can be very useful for other medical applications as well. The benchmark results demonstrate not only the accuracy and efficiency of the code, but also the scalability of the code. For example, in the CT model, the memory usage still keeps proportional to the quadrature order while increasing to S_{200}. And in the SPECT model, we are able to use the S_N solver in one coarse-mesh with about two million fine meshes.

Future Work

TITAN provides a code base for future development with its excellent extendibility. There are still several areas where the code can be further enhanced.

Acceleration Techniques

The loop structure of the code is composed of power iteration loop, upper-scattering loop, energy group loop, within-group loop, octant loop, coarse mesh loop, direction loop, and fine mesh/ray loop. Various acceleration techniques can be applied on the power iteration level and the within-group loop. These acceleration techniques aim to accelerate the convergence of the fission source or the within-group scattering source. Generally, they can be applied in both S_N

and MOC. Coarse mesh rebalancing (CMR) and coarse mesh finite difference (CMFD) are widely used acceleration techniques,[41] which accelerate the within-group loop by forcing the particle balance in each coarse mesh for each loop. Another physical approach is the synthetic method,[42] in which one can use some lower order methods like diffusion method to acquire a better estimation of scattering source in-between within-group loops. Some numerical approaches, such as multi-grid method,[43, 44] and pre-conditioned subspace projection iteration method,[45, 46] can also be applied. However, the general numerical iteration techniques usually need to be modified here, since in S_N codes, we usually do not build up the matrix A in a liner iteration system $x=Ax+b$ due to memory limitation.

Currently, there are two source iteration kernels in the TITAN code. The default kernel is the *S1* kernel, in which the flux moments are updated after angular fluxes are calculated within each sweep. While the *S2* kernel subroutines updates the flux moments immediately after the angular flux is calculated for each direction. The relationship between *S2* and *S1* is similar to the one between Gauss-Seidel and Jacobi iteration methods. Numerically, *S2* kernel has a faster convergence speed than the *S1* kernel in most cases without much additional computation cost. However, it could cause numerical instability problems in some extreme cases. And it is not preferable for code parallelization in the angular domain. Therefore, currently we choose the *S1* kernel as the default option. Another set of kernel subroutines can be added with the flux moments updated after each octant is processed. This process is numerically similar to the red-black block or multi-cyclic iteration schemes.

In the future, higher order iteration schemes should be implemented. Krylov subspace projection iteration pre-conditioned by CMFD would be a good acceleration combination. The acceleration subroutines can be inserted into Figure 3-1 around Subroutines L2.7 and/or L3.5.

Parallelization

We can parallelize the TITAN code by using MPI and/or OpenMP. One essential part of code parallelization work is the loop parallelization. In Figure 3-1, we could break up the coarse mesh loop and octant loop into a distributed computing environment by using MPI. OpenMP can be used to parallelize the direction loop in a shared memory environment. Other parallel algorithms can be applied.[47, 48] An MPI and OpenMP hybrid approach can take advantage of the cost-efficient cluster hardware, as well as multi-CPU nodes and dual-core CPUs. Furthermore, Code parallelization can benefit from the multi-block framework, since each coarse mesh in the framework can be treated relatively independently..

Improvements on Characteristics Solver

The TITAN code considers a characteristics coarse mesh as one region, which is sufficient in this work, since the characteristics solver is only designed for low scattering media. Some multi-regions data structures already are in place in the code. A more efficient ray-tracer is required for a multi-region solver.

Other Enhancements

Projection techniques need to be tested in more problems, since the efficiency and accuracy of the projections are essential under the multi-block framework. It is worth noting that the multi-block framework can assemble other types of solvers besides S_N and MOC. For example, some non-Cartesian meshing schemes can be implemented in a coarse mesh with a potential finite element solver.

With above proposed future work, we consider the code still under development. We hope in the future our community can build an online open-source forum for deterministic calculations, where users and developers can freely share source codes and ideas.

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:

$\hat{\Omega} \to (\theta,\varphi) \to (\mu,\varphi)$, and $\hat{\Omega}' \to (\theta',\varphi') \to (\mu',\varphi')$ (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.[49, 50]

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

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

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

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,[49, 51]

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

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:

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

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.

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

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.

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

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

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

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

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

With Eq. A-25, we can apply the spherical harmonic addition theorem.[49]

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

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

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

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:

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

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]

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

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

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

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

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.

Finally, it is worth noting that we choose x axis as the polar axis in all the derivations, which means μ is the cosine between $\hat{\Omega}$ and $\hat{x}$. The choice of polar axis does not alternate the formulation of the scattering kernel. However, the values of some terms in Eq. A-35 are affected by the choice of the polar axis, except for a Level-Symmetric quadrature set, in which all term values remain the same because of the rotation invariance property. In other quadrature types, e.g. the Legendre-Chebyshev quadrature, a number of terms in Eq. A-35 change with different polar axes. For example, if we choose the y as the polar axis instead of x, we can build a relationship between the two systems.

$$\begin{cases} \mu^{(y)} = \sin(\theta^{(x)}) \cdot \cos(\varphi^{(x)}) \\ \varphi^{(y)} = \text{atan2}[\sin(\theta^{(x)}) \cdot \sin(\varphi^{(x)}), \cos(\theta^{(x)})] \end{cases} \tag{A-36}$$

Where *atan2* is the extended inverse tangent function, which is available in most math libraries with various languages. Obviously, Eq. A-36 affects all the terms depending on (μ, φ) in Eq. A-35, including the flux moments, Legendre polynomial values, cosine's and sine's. However, the overall scattering source should remain the same even with all these changed terms, because physically the scattering source should not be affected by the choice of polar axis. Mathematically, one might be able to demonstrate this statement by substituting Eq. A-36 into Eq. A-35 and Eqs. A-16 to A-18. In reality, we can only expand the scattering kernel to a limited order. In the TITAN code, originally we chose the z axis as the polar axis, later We changed it to the x axis. The results are almost the same for the first benchmark problem discussed in Chapter. 5. It would be interesting to further investigate the effect of different choices of polar axis on the scattering kernel.

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 $\{\varphi_n(x) \mid \mathrm{n} \geq 0\}$ 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:

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

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:

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

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,[37] 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*

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

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

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

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, $-1 \leq \mu \leq 1$, is the cosine of the polar angle, and φ, $-\pi \leq \varphi \leq +\pi$ is the azimuthal angle. We need to estimate:

$$I = \int_{4\pi} d\Omega \, f(\mu, \varphi) = \int_{-1}^{+1} d\mu \int_0^{2\pi} d\varphi \, f(\mu, \varphi) \qquad \text{(B-5)}$$

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

$$g(\mu) = \int_0^{2\pi} d\varphi \, f(\mu, \varphi) \qquad \text{(B-6)}$$

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

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.

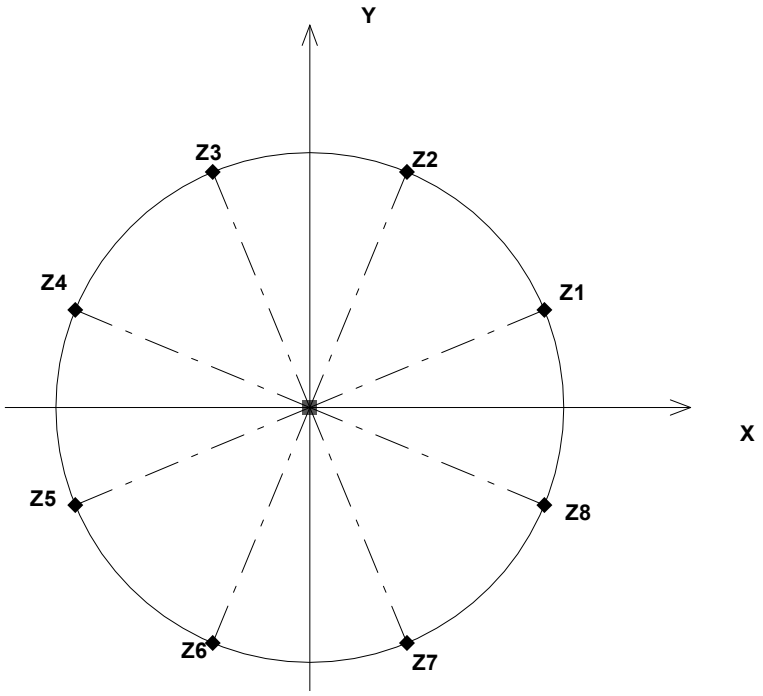


Figure B-1. Chebyshev roots (N =4) on a unit circle.

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 can not 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 level 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.

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

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

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.

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

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.

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

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.

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

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.

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

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.

Conclusion

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'.

APPENDIX C
IS FORTRAN 90/95 BETTER THAN C++ FOR SCIENTIFIC COMPUTING?

On Nov. 18, 2004, the international FORTRAN standards committee (WG5) published the FORTRAN 2003 standard under the identification of ISO/IEC 1539-1:2004(E), which is considered a major revision of the previous FORTRAN 95 standard. Among many new features in the 2003 standard are: derived type enhancements, object-oriented programming (OOP) support, data manipulation enhancements, and interoperability with the C programming language. The standard adopts some features of C++ and other modern languages and moves FORTRAN closer to C++, while trying to keep and enhance the advantages of FORTRAN in scientific computing. Some of the new features, widely applied in other languages, could play an important role in scientific programming.

The performance of scientific computer codes has significantly benefited from the fast-advancing computing technology in terms of processor speed, memory limit, and the concept of parallel computing. More benefits can be obtained with the use of the new FORTRAN 2003 and newer compliers. However, the new language features need to be accepted and utilized by the scientific computing community. Although now no complier can fully support the new standard, a few compiler vendors are working on the implementation of FORTRAN 2003 in their compiler products gradually. Among them are the Intel FORTRAN Compiler (IFC), formerly Compaq Visual FORTRAN compiler (CVF), and Portland Group FORTRAN complier (PGF90). TITAN uses some FORTRAN 2003 features, which are mainly related to OOP and derived type enhancements. And it is originally compiled by IFC 8.1 and PGF90 6.1 in both WINDOWS and LINUX/UNIX with the same source files. As in April 2007, IFC v9.2 and PGF90 v7.0 are available in both operating systems.

The performance comparison between FORTRAN 77/90/95 and C/C++ has been discussed for years. C++ and its compilers evolve significantly over the years with a much larger user base. More and more scientific programmers consider C++ as one of their language choices. In the nuclear engineering field, however, FORTRAN still remains the first choice for two reasons. First, data abstraction penalty associated with language features such as OOP could undermine the performance of a scientific computing code. These new features are not always desirable or necessary in scientific computing as in computer applications because of the associated overheads. Codes can be ugly in human eyes, but very desirable in machines' viewpoint. Second, FORTRAN is traditionally widely used in our community with a large code base. It is not practical to rewrite the legacy codes in C/C++ or even with a newer FORTRAN standard.

It is difficult to provide a clear direct answer to the question which language is better for scientific computing, since the results can be affected by the individual coding practice and the compiler choice. C++ has a much richer feature set than FORTRAN. However, in scientific computing, one major concern of language choice is the array handling. Here we only provide an individual investigation on this aspect by comparing the C++ 'vector' class template with its FORTRAN counterpart.

We wrote two small Monte Carlo codes with the particle splitting/rouletting technique in FORTRAN and C++. The two codes follow the same logic with the same data structure. A particle object is defined with particle position and direction by a class in the C++ code, and a user-defined type structure in the FORTRAN code. An array of particle objects, called particle bank, is created by vector class template in the C++ code, and by defining an allocatable array in the FORTRAN code. We compiled the two codes with Intel Fortran compiler and Inter C++ compiler. The running times for both codes are compared in Table C-1.

Table C-1. Run time comparison of the sample FORTRAN and C++ codes.

Number of Particles	Run time of the FORTRAN Code	Run time of the C++ code
10 Million	7 sec	7 sec
100 Million	67 sec	64 sec

According to Table C-1, there is no significant performance difference between the two codes. However, it is worth noting that the size of the particle bank is required to be pre-defined in the FORTRAN code to avoid memory overflow. While the C++ vector class template provides a build-in mechanism to adjust the memory buffer after the last element of the vector. User can 'push' any number of particles into the bank without worrying memory overflow. It is safe to say that this mechanism in C++ vector template is very efficient, since even with this overhead, the C++ code still maintains the same level performance as the FORTRAN code, at least for the relatively small size array in our code. In handling very large size array, FORTRAN could have some advantages over C++, since it provides some build-in vector operation on arrays.

The key to a scientific computing code is always the algorithms and the physics underneath it. However, the paradigm of the code does make a difference on performance. If some desirable and crucial features are not available in FORTRAN, we should not hesitate to choose C++ or other languages.

```
/* C++ source code  for comparing performance with FORTRAN*/
/* shielding with variance reduction 1-D slab   */
/* Geometry splitting and roulette     */

/* Oct. 2005  Author :yice at ufl edu  */

#include <iostream>
#include <string>
#include <cmath>
#include <vector>

using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::vector;

/* for RN generator */
long int rn = 119;  /* seed */
/* GGL RN generator */
const __int64 a = 16807;    /* a=7^5 */
const __int64 c = 0;        /* c =0  */
const __int64 M=2147483647; /* M=2^31-1 */

class cParticle
{
public:
/* initial values */
cParticle(): x(0), w(1.0), reg(1), mu(1.0) { }

float x;      /* position */
float w;      /* weight */
int  reg;     /* region num */
float mu;     /* direction cosine */
};

/* track one particle inside  */
int TrackOne();
/* rn generator */
float MyRng();

const double sigma_t_d=10;
const double sigma_st=0.2;

/* num_cell : num of regions with diff. importance */
int num_cell = 6;
```

```
/* bon: region boundaries */
vector<float> bon;
/* region importance */
vector<float> imp;

/* w_counter: weight counter ; w_square: square sum (for R) ;
w_one: sum of the weights of each starting particle and its children
w_xxx[0] : absorbed
w_xxx[1] : back-scattered
w_xxx[2] : transmitted
w_xxx[3] : killed by rouletting */
vector<float> w_counter;
vector<float> w_square;
vector<float> w_one;

/* particle bank */
vector<cParticle> bank;
/* current partile being followed */
cParticle one;

const float Pi=2*asin(1.0);

/* ********************************************** */
int main()
{
int i,j,k;

int tot_part=10000000;
int tot_tracked=0;
float size_cell=sigma_t_d/num_cell;

/* Initialize varibles */
/* erase counter */
for( i = 0; i <4; ++i)
{
w_counter.push_back(0.0) ;          /* w_counter=0   */
w_square.push_back(0.0) ;
w_one.push_back(0.0);
}

/* imp and bod */
imp.push_back(0);  /* left outside imp=0 */
imp.push_back(1);  /* region 1 imp=1     */
bon.push_back(0);

for(  i = 1; i <num_cell+1; ++i)
```

```
{
imp.push_back(imp[i]*2) ;          /* imp=1,2,4,8,16 ..*/
bon.push_back(bon[i-1]+size_cell);  /* bon=0,2,4,6,8,10 */
// imp.push_back(1.0) ;
}
imp.push_back(0.0);   /* right outside imp=0 */

for (i = 0; i < tot_part ; ++i)
{
/* initial particle */
one.x=0; one.w=1.0; one.reg=1; one.mu=1.0;

/* push it into bank */
bank.push_back(one);

while( !bank.empty() )
{
 one=bank.back();  /* get the last particle in bank */
 bank.pop_back();  /* pop the last one out of bank */
 ++tot_tracked;    /* count tot particle tracked */
// j=bank.size();
 k=TrackOne();
 w_one[k]=w_one[k]+one.w;
 w_counter[k]= w_counter[k] + one.w;
// cout << " k=" << k << "  tot_tracked="  << tot_tracked <<endl;
// cout << " w=" <<  w_counter[k] <<endl;
}
for(j=0; j<4; ++j) {
w_square[j]=w_square[j] + w_one[j]*w_one[j] ;
w_one[j]=0.0;
}
}

cout << " tracked=" << tot_tracked << endl;
cout << " transmitted prob.= " << w_counter[2]/tot_part  << endl;
cout << " relative.   err. = " << sqrt( w_square[2]/pow(w_counter[2],2)-1.0/tot_part ) << endl;

return 0;
}
/* ********************************************************** */
/*  track one particle inside a 1-D multi-region shield      */
int TrackOne()
{
float eta, r, mu0, phi,ir;
int k;
while (one.reg >0 && one.reg < num_cell+1 )
```

```
{
eta=MyRng();
r=-log(eta);
one.x=one.x+r*one.mu;
while ( one.x >= bon[one.reg-1] && one.x <= bon[one.reg])
{
eta=MyRng();
if (eta <= sigma_st)
{                   /* scattered */
mu0 = 2*MyRng() - 1;
phi = 2*Pi*MyRng();
one.mu = one.mu*mu0 + sqrt(1-pow(one.mu,2))*sqrt(1-pow(mu0,2))*cos(phi);
r=-log(MyRng() );
one.x=one.x + r * one.mu ;
}
else   /* absorbed */
{
return 0;   /*absorbed */
}    /* end if eta */

} /* end while loop one.x */

/* cross the right region boundary */
if (one.x > bon[one.reg] )
{
/* to move foward one region */
 one.x=bon[one.reg++];
 ir=imp[one.reg]/imp[one.reg-1];
}
/* cross the left region boundary */
if (one.x < bon[one.reg-1] )
{
/* to move backward one region */
 one.x=bon[--one.reg];
 ir=imp[one.reg]/imp[one.reg+1];
}

/* splitting and rouletting */
k=int(ir);
if ( ir > 1)   /* splitting */
{
one.w=one.w/ir;
for (int j=1; j<k; ++j)
{
 bank.push_back(one);
}
```

```
if ( MyRng() < ir-k ) bank.push_back(one);

} /*end if ir greater than 1 */

if ( ir < 1 && ir > 0)   /* rouletting */
{

if (MyRng() < ir) {
one.w=one.w/ir;
}
else {
return 3;      /* killed by rouletting */
}
} /* end if ir less than 1 */
} /* end while loop one.reg */

if (one.reg <1 )
{
return 1; /* back scattered */
}
else
{
return 2; /* transmitted */
}  /* end if one.reg */
}

/* RN generator */
float MyRng()
{
rn=(a*rn + c)%M;
return 1.0*rn/M;
}
```

```
!/* FORTRAN 90 source code  for comparing performance with C++*/
!/* shielding with variance reduction 1-D slab   */
!/* Geometry splitting and roulette     */

module mRNG

integer :: x=119
integer*8 :: a=16807
integer*8 :: M=2_8**31-1

end module

module paraset1

type tParticle
real x
real w
integer reg
real mu
end type tParticle

integer :: banksize=100
type(tParticle), dimension(:), allocatable :: bank
type(tParticle) one
integer :: top=0

end module

module paraset2

real :: sigma_t_d=10
real :: sigma_st=0.2

! num_cell : num of regions with diff. importance
integer :: num_cell = 6

! bon: region boundaries
real , dimension(:), allocatable :: bon
! region importance
real, dimension(:) , allocatable :: imp

! w_counter: weight counter ; w_square: square sum (for R) ;
!w_one: sum of the weights of each starting particle and its children
!w_xxx[0] : absorbed
!w_xxx[1] : back-scattered
!w_xxx[2] : transmitted
```

```
!w_xxx[3] : killed by rouletting */
real :: w_counter(0:3)=0
real :: w_square(0:3)=0
real :: w_one(0:3)=0

real pi

end module

program shield
use paraset1
use paraset2
use DFPORT

integer i,k
real eta

integer tot_part,tot_tracked
real  size_cell

real s1,s2

s1=secnds(0.0)

pi=2*asin(1.0)
tot_part=1000000
tot_tracked=0
size_cell=sigma_t_d/num_cell

! /* Initialize varibles */
! /* erase counter */
w_counter=0
w_square=0
w_one=0

!/* imp and bod */
allocate ( imp(0:num_cell+1), bon(0:num_cell) )
imp(0)=0    !left outside
imp(1)=1    !/* region 1 imp=1     */
bon(0)=0

do i = 1, num_cell
imp(i+1)=imp(i)*2        ! /* imp=1,2,4,8,16 ..*/
bon(i)=bon(i-1)+size_cell !  /* bon=0,2,4,6,8,10 */
!imp(i+1)=1
enddo
```

```
imp(num_cell+1)=0         ! /* right outside imp=0 */

allocate ( bank(banksize) )
top=0

loop_part : do i = 1, tot_part
!/* initial particle */
one%x=0
one%w=1.0
one%reg=1
one%mu=1.0

! /* push it into bank */
top=top+1
bank(top)=one

do while( top .ne. 0 )

 one=bank(top)              ! /* get the last particle in bank */
 top=top-1                  ! /* pop the last one out of bank */
 tot_tracked=tot_tracked+1   ! /* count tot particle tracked */

 call TrackOne(k);
 w_one(k)=w_one(k)+one%w
 w_counter(k)= w_counter(k) + one%w

enddo

do j=0 , 3
w_square(j)=w_square(j) + w_one(j)**2
w_one(j)=0.0;
enddo

enddo loop_part

write(*,"('tracked=', I0)")  tot_tracked
write(*,"('transmitted prob. =', ES12.5)")  w_counter(2)/tot_part
write(*,"('relative err. = ', ES12.5)" )  &
  sqrt( w_square(2)/(w_counter(2)**2-1.0/tot_part ) )

write(*,'("run time=", f10.3, "sec" )  ') secnds(s1)

end program

subroutine TrackOne(flag)
use paraset1
```

```
use paraset2
integer flag

real eta, r, mu0, phi,ir,temp;
integer k

while_reg : do while (one%reg .gt. 0 .and. one%reg .lt. num_cell+1 )

call MyRng(eta)
r=-log(eta)
one%x=one%x + r*one%mu

while_xr : do while ( one%x .ge. bon(one%reg-1) .and. one%x .le. bon(one%reg) )

call MyRng(eta)

if (eta .le. sigma_st) then  ! /* scattered */

call MyRng(eta)
mu0 = 2*eta - 1
call MyRng(eta)
phi = 2*Pi*eta
one%mu = one%mu*mu0 + sqrt(1-one%mu**2)*sqrt(1-mu0**2)*cos(phi);
call MyRng(eta)
r=-log(eta)
one%x=one%x + r * one%mu

else                    ! /* absorbed */

flag=0
return

endif

enddo  while_xr

! /* cross the right region boundary */
if (one%x .gt. bon(one%reg) ) then

! /* to move foward one region */
one%x=bon(one.reg)
one%reg=one%reg+1
ir=imp(one%reg)/imp(one%reg-1)
endif

! /* cross the left region boundary */
```

```
if (one.x < bon(one%reg-1) ) then

!/* to move backward one region */
 one%reg=one%reg-1
 one%x=bon(one%reg)
 ir=imp(one%reg)/imp(one%reg+1)
endif

! /* splitting and rouletting */
k=int(ir)

if ( ir .gt. 1) then  !/* splitting */
one%w = one%w/ir

do j=1, k-1
  top=top+1
  bank(top)=one
enddo

call MyRng(eta)

if ( eta .lt. ir-k ) then
  top=top+1
  bank(top)=one
endif

endif

if ( ir .lt. 1 .and. ir .gt. 0)  then !/* rouletting */
one%w = one%w/ir
call MyRng(eta)

if ( eta .gt. ir) then
 flag=3
 return
endif

endif

enddo while_reg

if (one%reg .lt. 1 ) then
 flag=1
 return   ! /* back scattered */
else
 flag=2
```

```
 return ! /* transmitted */
end if


end subroutine

subroutine MyRng(rn)
use mRNG
real rn

!x=int( mod(a*x,M), 4 )
x=mod(a*x,M)
rn=1.0*x/M
return

end subroutine
```

APPENDIX D
TITAN I/O FILE FORMAT

TITAN Input Files

The TITAN code is developed based on the code base of PENMSH Express,[29] which is a mesh generator I wrote for generating PENTRAN input deck. PENMSH Express, or PENMSH XP, follows a similar input syntax with PENMSH.[28] Therefore, TITAN inherits most of the PENMSH input file format. Table D-1 lists the input files of the TITAN code.

Table D-1. TITAN input file list.

File #	File Name	Description	Memo
1	penmsh.inp	Meshing parameters	Required
2	prbname#.inp	Meshing per z level	Required
3	prbname.src	Fixed source grid	Optional
4	prbname.spc	Source spectrum	Optional
5	prbname.chi	Fission spectrum	Optional
6	prbname.mba	Material balance	Optional
7	bonphora.inp	General input parameters	Required
8	prbname.xs	Cross section data	Required

Input files #1 to #6 are general PENMSH input files, which define model geometries and source specifications. We use '*prbname*' to denote different problem names. General meshing parameters are specified in the '*penmsh.inp*', including number of *z* levels, *z*-level boundaries, etc. Geometries on each *z* level are specified in a separate file (Input file 2). For example, *prbname1.inp, prbname2.inp,....* These input files can describe various geometries with the 'overlay' feature. Figure D-1 shows the geometries generated by a sample *z*-level input file. The fixed source grid can be defined in the '*prbname.src*' file. *prbname.spc* and '*prbname.chi*' specify the source and fission spectrum, respectively. And *'prbname.mba'* is used to check the model material balance. More details on Input files #1 to #6 can be found in the manuals of PENMSH and PENMSH XP. And we will further discuss input file #7 in the next section.

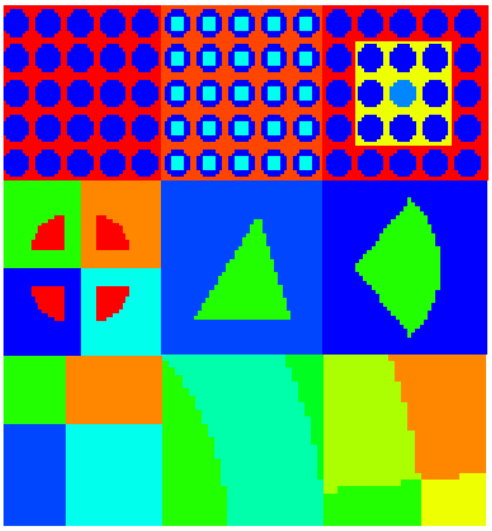


Figure D-1. A 3 by 3 coarse mesh model on one z level.

Bonphora.inp Input File

Input file 7 (*bonphora.inp*) is special file used by TITAN only, which specifies parameters for transport calculations, such as the quadrature set, differencing scheme, solver, etc. The file supports as many as 4 sections. The following is a sample *bonphora.inp* file.

```
/ bonphora.inp: TITAN input file to define transport parameters
#0  Section 0: Global varibles
2   0        /# of quadrature,  global DS id
#1 Section 1: Quadrature sets
/quad 1  Pn-Tn
/ Quadrature id , order, num of split directions
1 20 2
/spilited directions
46 47   /direction index
11 11   /splitted order
1  1     /splitted id : 1- pn-tn splitting
2  2     /# of directions on the top level
/quad 2 level symmetric
/ Quadrature id , order,num of split directions
0 20 1
/spilited directions ids
37     /direction index
8      /splitted order
0      /splitted id : 0- rectangular splitting
0      / not used
#2 Section 2: Coarse mesh specifications
/Solver_id
0 1 0
/qudra_id
1 2 1
/Diff scheme
1 1 2
```

Figure D-2. A sample *bonphora.inp* input file.

Section 0 is dedicated to specify two parameters: total number of quadrature sets used in the model, and the global differencing scheme id number, which define the differencing scheme for all coarse meshes if the number is a positive integer (*id=1*, diamond with zero fix-up; *id=2*, Directional Theta-Weighted). If zero is given as the global differencing scheme id, an additional card is required to specify an individual differencing scheme for each coarse mesh.

Section 1 is used to define all the quadrature sets used in the model. In this sample input file, two quadrature set are specified. The first one is a P_N-T_N quadrature (quadrature *id=1*) with an order of 20. The P_N-T_N splitting technique is applied on two directions in the quadrature set (direction index number: 46 and 47). The second one is a level-symmetric quadrature set with rectangular splitting on Direction 37.

Section 2 specifies the parameters for each coarse mesh. In this sample file, the S_N solver will be used for coarse meshes #1 and #3 (solver id=0). Coarse mesh #2 uses the characteristics solver. Quadrature set #1 specified in Section 1 is applied in coarse meshes #1 and #3. Quadrature set #2 is used in coarse mesh #2.

Another section can be used to specify the iteration number limitations and tolerances, especially for eigenvalue problems. The following is the input file for the C5G7 MOX benchmark problem.

```
/ bonphora.inp: TITAN input file to define transport
parameters
#0   Section 0: Global varibles
1  1        /# of quadrature,  global DS id
#1
0 6  0
#3 Iteration parameters
/tolout ,tolin
1.0e-5 1.0e-3
/outer,inner
-50 10
/rkdef
```

Figure D-3. C5G7 MOX benchmark problem *bonphora.inp* input file.

In this model, an S_6 level-symmetric quadrature is used with the diamond differencing scheme. The S_N solver is applied on all the coarse meshes (S_N solver is the default solver). Section 3 specifies some iteration parameters. The outer iteration tolerance is 1.0E-05 (variable *tolout*). And the inner iteration tolerance is 1.0E-03(variable *tolin*). Note if *tolin* is less than zero, the adaptive inner loop tolerance control will be engaged. The iteration number limitations are defined in the next card. The outer and inner iteration limits are 50 and 10 respectively. Negative numbers means the limitations are adaptive. The last card defines the initial guess of eigenvalue. Aitken extrapolation[37] is used on *k-effective* if users specify a negative initial guess.

TITAN can automatically convert a digital phantom into a transport calculation model. We use this feature for the SPECT benchmark problem. The input file format is slightly different for a medical phantom model. Details can be found in the PENMSH XP manual.

TITAN Output Files and TECPLOT Visualization

Table D-2 list the major output files of the TITAN code. The first file is an optional output, which contains a generated PENTRAN inputd deck. The second output file is a report of material balance check. The third file, *bonphora.log*, is the input processing log. And the solver log is stored in file *prbname_solver.log*, which records all the iteration output.

Table D-2. TITAN output file list.

File #	File Name	Description
1	Prbname_out.f90	
2	prbname_out.mba	Material balance tables
3	Bonphora.log	Processing log file
4	Prbname_solver.log	Solver log file
5	prbname_mix.plt.	TECPLOT binary file, contains all the calculation data
6	prbname.mcr	TECPLOT macro file

The last two files are used for visualization of the calculation results with the TECPLOT software. A TECPLOT I/O library is developed and included in the TITAN code. The library, composed of about 15 subroutines and modules, can generate TECPLOT binary data files as

many as necessary simultaneously. Some other TITAN output files, including the quadrature data file and the optional boundary angular flux files when a fictitious quadrature set is used, are also generated by this library. The last file in Table D-2, *prbname.mcr*, is a macro file, which can be loaded by TECPLOT, to help organize the data in *prbname_mix.plt*.

TECPLOT also provides an IO library (without source codes) for users to generate their own binary data files. However, for practical reasons, here we wrote our own version of TECPLOT IO library, which is optimized for our purpose. TECPLOT is an excellent visualization tool. However, it is a commercial software package. We consider migrating to the widely used visualization toolkit (VTK) platform, which is an open source library for scientific visualization. A number of front end software packages (e.g. PARAVIEW) are freely available to visualize the VTK format data file.

TITAN Command Line Option

The common command line option is '*-i*' option, which specifies the directories where the input files are located. The default input directory is the current one.

[home/user/]# bonphora –i test

The above command line reads input decks from the */home/user/test* directory. Other command options can be found in the PENMSH XP manual. Users can add their own modules and subroutines to extract the interested data from the calculation results. All the post-processing subroutines are called from a container subroutine named *Nirvana*. The user- defined post-processing routines can be triggered with a command line option with slight modification of the code. For example, the option *'-mox'* will trigger the C5G7 MOX post-processing subroutines. These subroutines are used to calculate the fuel pin powers based on the converged scalar fluxes.

LIST OF REFERENCES

1. B. V. ALEXEEV, *Generalized Boltzmann Physical Kinetics,* Elsevier Science Publishing (2004).

2. E. E. LEWIS and W. F. MILLER, *Computational Method of Neutron Transport*, John Wiley & Sons, New York (1984).

3. G. I. BELL, and S. GLASSTONE, *Nuclear Reactor Theory*, Robert E. Krieger Publishing, Malabar, FL (1985).

4. J. J. DUDERSTADT and L. J. HAMILTON, *Nuclear Reactor Analysis*, 1st ed., John Wiley & Sons, New York (1976).

5. B. G. CARLSON and K.D. LATHROP, "Discrete Ordinates Angular Quadrature of the Neutron Transport Equation," LA-3186, Los Alamos National Laboratory (1965).

6. J. R. ASKEW, "A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries," AEEW-M1108, United Kingdom Atomic Energy Authority (UKAEA), Winfrith (1972).

7. M. D. BROUGH and C.T. CHUDLEY, "Characteristic Ray Solution of the Transport Equation," *Advances in Nuclear Science and Technology Yearbook* (1980).

8. S. G. HONG and N. Z. CHO, "CRX: A Code for Rectangular and Hexagonal Lattices Based on the Method of Characteristics," *Ann. Nucl. Energy*, **25**, 547 (1998).

9. M. HURISN and T. JEVREMOVIC, "AGENT Code - Neutron Transport Benchmark Example and Extension to 3D Lattice Geometry," *Nuclear Technology and Radiation Protection*, **XX**, 10 (2005).

10. R. ROY, "Large-Scale 3D Characteristics Solver: Can the Dream Live On?" *Proc. Int. Conf. on Mathematics and Computation (M&C 2005)*, Avignon, France, American Nuclear Society (2005).

11. N. Z. CHO, G. S. LEE, and C. J. PARK, "Fusion of Method of Characteristics and Nodal Method for 3-D Whole Core Transport Calculation," *Trans. Am. Nucl. Soc.*, **86**, 322 (2002).

12. K. D. LATHROP, "Remedies for Ray Effects," *Nucl. Sci. Eng.*, **45**, 255 (1971).

13. G. E. SJODEN and A. HAGHIGHAT, "PENTRAN - Parallel Enviroment Neutral Particle TRANsport 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).

14. K. D. LATHROP, "Spatial Differencing of the Transport Equation: Positivity vs. Accuracy," *J. Comput. Phys.*, **4,** 475 (1969).

15. 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).

16. 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).

17. A. M. KIRK, "On the Propagation of Rays in Discrete Ordinates," *Nucl. Sci. Eng.*, **132**, 155 (1999).

18. W. RHOADES and W. ENGLE, "A New Weighted Difference Formulation for Discrete Ordinates Calculations," *Trans. Am. Nucl. Soc.*, **27**, 776 (1977).

19. B. PETROVIC and A. HAGHIGHAT, "New Directional Theta-Weighted S_N Differencing Scheme," *Trans. Am. Nucl. Soc.*, **73**, 195 (1995).

20. 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).

21. 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).

22. W. T. VETTERLING and B. P. FLANNERY, *Numerical Recipes in C++: the Art of Scientific Computing,* Cambridge University Press (2002).

23. B. G. CARLSON, "Transport Theory: Discrete Ordinates Quadrature over the Unit Sphere," LA-4554, Los Alamos National Laboratory (1970).

24. G. LONGONI, "Advanced Quadrature Sets, Acceleration and Preconditioning techniques for the Discrete Ordinates Method in Parallel Computing Environments," PhD Thesis, University of Florida (2004).

25. G. LONGONI and A. HAGHIGHAT, "Development of New Quadrature Sets with the Ordinate Splitting Technique," *Proc. Int. Conf. on Mathematical Methods and Supercomputing for Nuclear Applications (M&C 2001),* Salt Lake City, UT, American Nuclear Society (2001).

26. 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).

27. A. M. WEINBERG and E. P. WIGNER, *Physical Theory of Neutron Chain Reactors,* University of Chicago Press (1958).

28. A. HAGHIGHAT, “A Manual of PENMSH Version 5 –A Cartesian-Based 3-D Mesh Generator,” University of Florida (2004).

29. C. YI, “PENMSH XP manual: A Mesh Generator to Build PENTRAN Input Deck with Compatibility to PENMSH,” University of Florida (2007).

30. J. E. WHITE et al., “Bugle 96: Coupled 47 Neutron, 20 Gamma-ray Group Cross-Section Library Derived from ENDF/B-VI for the LWR Shielding and Pressure Vessel Dosimetry Applications” Oak Ridge National Laboratory (1996).

31. X-5 Monte Carlo Team, “MCNP-A General Monte Carlo Code for Neutron and Photon Transport, Version 5,” Los Alamos National Laboratory (2003).

32. J. C. WAGNER et al., “MCNP: Multigroup/Adjoint Capabilities,” Los Alamos National Laboratory (1994).

33. K. KOBAYASHI, N. SUGIMURA, and Y. NAGAYA, “3-D Radiation Transport Benchmarks for Simple Geometries with Void Regions,” OECD/NEA (2000).

34. A. HAGHIGHAT, G. E. SJODEN, and V. KUCUKBOYACI, “Effectiveness of PENTRAN's Unique Numerics for Simulation of the Kobayashi Benchmarks,” *Prog. Nucl. Energy*, **39**, 191 (2001).

35. E. E. LEWIS et al., “Benchmark Specification for Deterministic 2-D/3-D MOX Fuel Assembly Transport Calculations without Spatial Homogenization (C5G7 MOX),” OECD/NEA (2001).

36. E. E. LEWIS et al., “Proposal for Extended C5G7 MOX Benchmark,” OECD/NEA (2002).

37. K. ATKINSON, *An Introduction to Numerical Analysis,* 2nd ed., John Wiley & Sons, New York (1989).

38. W. P. SEGARS, “Development and application of the new dynamic NURBS-based cardiac-torso (NCAT) phantom,” PhD Thesis, University of North Carolina (2001).

39. L. J. LORENCE, J. E. MOREL, and G. D. VALDEZ, “User's Guide to CEPXS/ONELD: A One-Dimensional Coupled Electron-Photon Discrete Ordinates Code Package,” Sandia National Laboratory (1989).

40. M. LJUNGBERG, S. STRAND, and M. A. KING, “The SIMIND Monte Carlo program: Monte Carlo Calculation in Nuclear Medicine,” *Applications in Diagnostic Imaging*, **11**, 145 (1998).

41. A. YAMAMOTO, “Generalized Coarse-Mesh Rebalance Method for Acceleration of Neutron Transport Calculations,” *Nucl. Sci. Eng.*, **151**, 274 (2005).

42. J. S. WARSA, T. A. WAREING, and J. E. MOREL, "Krylov Iterative Methods and the Degraded Effectiveness of Diffusion Synthetic Acceleration for Multidimensional S_N Calculations in Problems with Material Discontinuities," *Nucl. Sci. Eng.*, **147,** 218 (2004).

43. V. KUCUKBOYACI and A. HAGHIGHAT, "Angular Multigrid Acceleration for Parallel S_N Method with Application to Shielding Problems," *Proc. Int. Conf. on Advances in Reactor Physics and Mathematics and Computation into the Next Millennium (PHYSOR 2000),* Pittsburgh, PA, American Nuclear Society (2000).

44. P. NOWAK, E. LARSEN, and W. MARTIN, "Multigrid Methods for S_N Problems," *Trans. Am. Nucl. Soc.*, **55**, 355 (1987).

45. Y. SAAD, *Numerical Methods for Large Eigenvalue Problems,* John Wiley & Sons, New York (1992).

46. Y. SAAD, *Iterative Methods for Sparse Linear Systems,* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA (2003)

47. A. HAGHIGHAT, M. HUNTER, and R. MATTIS, "Iterative Schemes for Parallel S_N Algorithms in a Shared Memory Computing Environment," *Nucl. Sci. Eng.*, **121**, 103 (1995).

48. A. HAGHIGHAT, G. E. SJODEN, and M. HUNTER, "Parallel Algorithms for the Linear Boltzmann Equation Complete Phase Space Decomposition," *Society for Industrial and Applied Mathematics (SIAM) Annual Meeting*, Kansas City, MO (1996).

49. G. ARFKEN, *Mathematical Methods for Physicists,* Academic Press, New York (1970).

50. L. I. SCHIFF, *Quantum Mechanics,* McGraw-Hill, New York (1968).

51. E. W. HOBSON, *The Theory of Spherical and Ellipsoidal Harmonics,* Cambridge University Press (1931).

BIOGRAPHICAL SKETCH

I was born in 1973 in Anshan, China. I went to Tsinghua University in 1992 and got my bachelor's degree in nuclear engineering in 1997. I continued on to the graduate school at Tsinghua, and graduated with a master's degree in nuclear engineering in 2000. The same year, I went to Penn State University to pursue a doctoral degree. In 2001, I followed Dr. Haghighat to the University of Florida.

The goal of my study was to develop a hybrid algorithm to solve the LBE efficiently in low-scattering media and to enhance the efficiencies of the PENTRAN code in medical applications. I started to write a 3-D S_N kernel in April 2005 from the PENMSH XP code base, which is a mesh generator I wrote for preparing PENTRAN input deck. The 3-D S_N code is originally designed as a test platform for the hybrid algorithm. By the summer of 2005, I completed the initial versions of both the S_N and characteristics solvers. In the summer, I dedicated most of the time to the University of Florida Training Reactor (UFTR) fuel conversion project. After that summer, I continued to work on the code and implemented a number of techniques, including P_N-T_N quadrature set, P_N-T_N ordinate splitting, and projection techniques. By April 2006, the framework of the code is completed. In the second half of 2006, I worked on the integration of characteristics solver into PENTRAN. In the first quarter of 2007, the fictitious quadrature technique is developed for the heart phantom benchmark. And some studies on the limitations of the hybrid algorithms are performed.