DEVELOPMENT AND VALIDATION OF METHOD OF CHARACTERISTICS BASED CODE FOR SOLVING NEUTRON TRANSPORT EQUATION FOR A TWO-DIMENSIONAL PIN CELL USING DELAUNAY TRIANGULATION

Submitted in partial fulfilment of the requirements
for the degree of

MASTER OF TECHNOLOGY

in

NUCLEAR SCIENCE AND ENGINEERING

By

Lakshay Jain
(2K13/NSE/03)

Under the guidance of

Dr. R. Karthikeyan
Scientific Officer - F,
Reactor Physics Design Division,
Bhabha Atomic Research Centre,
Mumbai, India

Dr. Nitin Kumar Puri
Assistant Professor,
Department of Applied Physics,
Delhi Technological University,
Delhi, India

Department of Applied Physics
Delhi Technological University
(Formerly Delhi College of Engineering)
Delhi

June 2015

दूरभाषी :
TELEPHONE : 2559 3726
Email: pdkrish@barc.gov.in
तार : बार्क-मुम्बई, चेम्बूर
TELEGRAMS : BARC-MUMBAI, CHEMBUR
टेलेक्स :
TELEX : 011-61017
फेक्स संख्या:
FAX NUMBER : 91-22-2550 5151 / 91-22-2551 9613

सत्यमेव जयते

हाल - 1,
ट्रॉम्बे,
मुंबई - 400 085,
HALL No. 1,
TROMBAY,
MUMBAI - 400 085

भारत सरकार
GOVERNMENT OF INDIA
भाभा परमाणु अनुसंधान केन्द्र
BHABHA ATOMIC RESEARCH CENTRE
रिएक्टर भौतिकी अभिकल्पन प्रभाग
REACTOR PHYSICS DESIGN DIVISION

Ref: RPDD/PDK/ 15059

June 25, 2015.

TO WHOMSOEVER IT MAY CONCERN

This is to certify that this major project work, entitled Development and Validation of Method of Characteristics Based Code for Solving Neutron Transport Equation for a Two-Dimensional Pin Cell Using Delaunay Triangulation, is a bonafide and authentic work carried out by Shri Lakshay Jain (2K13/NSE/03) in partial fulfillment for award of degree of Master of Technology (M. Tech) in Nuclear Science and Engineering by Delhi Technological University (Formerly Delhi College of Engineering), Delhi during the year 2014-2015. It is certified that all corrections/suggestions indicated for Internal Assessment have been approved as it satisfies the academic requirement in respect of major project prescribed for Master of Technology (M. Tech) degree.

[illegible]

(R. Karthikeyan)
Scientific Officer/F
Reactor Physics Design Division
BARC

[illegible]

(P.D. Krishnani)
Head, Reactor Physics Design Division
BARC

CERTIFICATE

This is to certify that this major project work, entitled **Development and Validation of Method of Characteristics Based Code for Solving Neutron Transport Equation for a Two-Dimensional Pin Cell Using Delaunay Triangulation**, is a bonafide and authentic work carried out by **Mr. Lakshay Jain (2K13/NSE/03)** in partial fulfillment for award of degree of **Master of Technology (M. Tech)** in **Nuclear Science and Engineering** by **Delhi Technological University** (Formerly Delhi College of Engineering), Delhi during the year 2014-2015. It is certified that all corrections/suggestions indicated for Internal Assessment have been approved as it satisfies the academic requirements in respect of major project prescribed for Master of Technology (M. Tech) degree.

(Signature of Internal Assessment Guide)
Dr. Nitin K. Puri
Assistant Professor
Department of Applied Physics
Delhi Technological University

(Signature of HOD)
Prof. Suresh C. Sharma
Head of Department
Department of Applied Physics
Delhi Technological University

DECLARATION

I, hereby, declare that the work being presented in this major project report, entitled **Development and Validation of Method of Characteristics Based Code for Solving Neutron Transport Equation for a Two-Dimensional Pin Cell Using Delaunay Triangulation**, is an authentic record of my own work carried out under the guidance of **Dr. R. Karthikeyan**, Scientific Officer - F, Reactor Physics Design Division, Bhabha Atomic Research Centre, Mumbai and **Dr. Nitin Kumar Puri**, Assistant Professor, Department of Applied Physics, Delhi Technological University (Formerly Delhi College of Engineering), Delhi. The work contained in this major project report has not been submitted in part or full, to any other university or institution for award of any degree or diploma.

This dissertation work is submitted to **Delhi Technological University** (Formerly Delhi College of Engineering) in partial fulfillment for the **Master of Technology (M. Tech)** in **Nuclear Science and Engineering** during the academic year 2014-2015.

DATE : June 21, 2015

____

(Signature and Name of Student)

Lakshay Jain

(2K13/NSE/03)

ACKNOWLEDGEMENTS

First and foremost, I wish to sincerely thank Dr. Ashutosh Bhardwaj for recommending my name for the dissertation work at Bhabha Atomic Research Centre (BARC). I would also like to extend my deep sense of appreciation to Professor S. C. Sharma and Dr. Nitin K. Puri for granting me permission to go to BARC, Mumbai for pursuing this work, and to Dr. P. D. Krishnani for allowing me to undertake this research, here, at BARC.

I wish to sincerely thank Dr. R. Karthikeyan, for accepting me as a student, persevering with my work and providing me all the support that a student needs to perform good research. I thank him for his constant encouragement and valuable guidance. Thanks are also due to Dr. Usha Pal, Mrs. Argala Srivastava and Mr. K. K. Yadav for their constant support and encouragement, for patiently listening to the never-ending review presentations, and for providing me with their invaluable suggestions and feedback. My special thanks to Mr. Anmol Singh for all the passionate discussions we had on the subject and introducing me to this work. I would like to thank all of them for making me feel at ease in the new environment at BARC from day one.

Last, and most importantly, this research would not have been possible but for the sacrifice of my family. The constant emotional support of all my relatives, colleagues and friends during my stay at Mumbai has been indispensable to this research.

ABSTRACT

The primary objective of this work is development and validation of a code for lattice level calculations using the method of characteristics (MOC) in 2-dimensions. The code solves the characteristic neutron transport equation for a 2-dimensional pin cell to compute the neutron flux values in different spatial regions of the problem domain, and uses the distribution so obtained to determine the effective multiplication factor for the region of interest. The problem is subdivided into smaller, triangular, unstructured meshes using Delaunay triangulation and mesh refinement techniques.

Qualified reactor physics codes are essential for the study of all existing and envisaged designs of nuclear reactors. Such codes estimate neutron fluxes in different regions of the problem domain as a function of space, angle, energy and time dependence. This provides a holistic description of all processes occurring in the reactor core and subsequent prediction of thermal–mechanical response and degradation of various components of the core. Thus, such codes are indispensable for thorough safety analysis and verification of economic feasibility of reactor design and operation.

The solution of the transport equation or linear Boltzmann equation for most practical problems must be obtained using numerical methods. Most computational schemes are based on two fundamentally different approaches, namely the deterministic and the stochastic or Monte Carlo. Major existing deterministic techniques are the spherical harmonics or P_n method, the discrete ordinates or S_N method, the collision probability (CP) or P_{ij} method and the method of characteristics (MOC). The method of characteristics (MOC) has been chosen over other available methods due to the many advantages associated with it. The spherical harmonics or P_n method leads to complicatedly coupled linear system of equations for 2- and 3-dimensional problems. The discrete ordinates or S_N method requires powerful pre-conditioners and acceleration strategies to ensure convergence. The collision probability (CP) or P_{ij} method, although capable of handling unstructured geometries unlike the P_n and S_N methods, requires isotropic sources in LAB and is only practically feasible for few-region problems. Monte Carlo simulations are accurate but are computationally expensive to set-up and run. MOC offers solution of neutron transport equation for unstructured geometries containing isotropic/anisotropic sources in LAB with reasonable accuracy at feasible computational costs.

The MOC-based lattice level code developed has been benchmarked for many two energy-group problems and the results have been compared with reference solutions obtained from DRAGON V4. The benchmark problems have 3 to 4 regions with varying material compositions. Problems with different fuel materials like uranium metal / uranium – plutonium mixed oxide MOX with varying geometrical size has been analyzed using this code.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

INTRODUCTION

The fundamental problem in design and analysis of nuclear reactors is the detailed prediction of neutron and photon distribution, as a function of space, angle, energy and time dependence, in all components of the reactor. Neutrons are responsible for propagating the chain reaction and releasing energy through fission, and also induce a thermal–mechanical response in the reactor core due to various nuclear and non-nuclear interactions. Consequently, degradation of the structural components, fuel rods, and control systems takes place through fuel depletion, fission product build up and radiation damage which, further affects the distribution of the neutron field through feedback mechanisms. A holistic description of all nuclear and non-nuclear processes is thus indispensable for safe and economically feasible operation and development of nuclear power plants and drives much of the need for development of an apposite mathematical and computational framework for adequate description of neutron distribution.

The process of transport of neutrons through matter is extremely well characterized by the *transport equation*, a linear version of the particle transport equation originally developed by Boltzmann for the kinetic theory of gases. It is an integrodifferential equation in seven independent variables, whose exact solution can be determined only for simple problems. The solution of the transport equation or linear Boltzmann equation is not smooth and thus, all problems of neutron transport of practical interest require a numerical solution. These numerical methods utilize a few approximation techniques, such as finite differences for differential operator, quadrature formulas for integral operators, or expansion methods.

Codes for reactor analysis are broadly classified as lattice level codes, and core level codes. Lattice level codes principally involve the generation of characteristics of representative cell in the core, which could be a simple pin-cell, or a super-cell or a complete assembly. Once the cell calculations are complete, the cell is effectively replaced by a homogeneous material whose cross sections and diffusion coefficients are constant on the complete cell volume. Hence, the lattice level code plays a very important role in reactor analysis. The core level code is used to model the coupled neutronics and thermal-hydraulics behavior of the entire reactor core during steady state and transient operation. It uses the homogenized cross sections developed by lattice code and analyses the overall behavior of the core, estimating the flux and power distribution as a function of burnup.

NEUTRON TRANSPORT EQUATION

The 3-dimensional neutron transport equation describes the neutron transport based on 7 independent variables, which are the three spatial components of position vector $\vec{r}$, two angles of directional unit vector $\hat{\Omega}$, kinetic energy E, and time t. The time dependent linear Boltzmann equation is given as,

$$\begin{aligned}\frac{1}{v}\frac{\partial\Phi(\vec{r},\hat{\Omega},E,t)}{\partial t} + \hat{\Omega}.\nabla\Phi(\vec{r},\hat{\Omega},E,t) + \Sigma_t(\vec{r},E)\Phi(\vec{r},\hat{\Omega},E,t) \\ = \int_0^\infty dE' \int_{4\pi} d\widehat{\Omega}'\Sigma_s(\vec{r},\widehat{\Omega}' \to \hat{\Omega}, E' \to E)\Phi(\vec{r},\widehat{\Omega}',E',t) \\ + \frac{\chi_p(\vec{r},E)}{4\pi}\int_0^\infty dE' \int_{4\pi} d\widehat{\Omega}' \,[1-\beta(E')]\nu\Sigma_f(\vec{r},E')\Phi(\vec{r},\widehat{\Omega}',E',t) \\ + \frac{1}{4\pi}\sum_{j=1}^{6}\chi_j(\vec{r},E)\lambda_j C_j(\vec{r},E) + Q(\vec{r},\hat{\Omega},E,t)\end{aligned} \quad (1)$$

where,

Φ	=	Angular neutron flux
C_j	=	Precursor density of a specific group
Q	=	Independent neutron source
Σ_t	=	Total macroscopic cross section
Σ_s	=	Macroscopic differential scattering cross section
$\nu\Sigma_f$	=	Macroscopic production cross section
β	=	Delayed neutron fraction
λ_j	=	Radioactive decay constant of a specific precursor group

χ_p, χ_j =	Prompt neutron fission spectrum and delayed neutron fission spectrum of j^{th} precursor respectively

The neutron transport equation is a simple particle balance equation which conserves the number of particles flowing through incremental volume of 6-dimensional phase space, $dVd\hat{\Omega}dE$ about $(\vec{r},\hat{\Omega},E)$ at any time t. The first term on the LHS represents the rate of change of number of neutrons, the second term represents the rate of neutron loss due to leakage, while the last term on LHS represents the rate of neutrons consumed in any interaction, mainly scattering, fission or absorption. Similarly, the terms on the RHS describe the rate at which neutrons are gained due to in-scattering source, prompt fission source, delayed fission source, and independent source, respectively. Thus, according to equation (1), rate of change of neutron number is equal to the difference of rate of gain of neutrons and the rate of loss of neutrons. Neutrons are lost due to leakage and collisions while are gained due to the processes of in-scattering, prompt fission, delayed decay of daughter nuclei and any independent sources.

In steady state, however, the rate of gain of neutrons is equal to the rate of loss of neutrons and thus, the rate of change of neutrons becomes zero. Neglecting the delayed fission neutrons, the steady state form of neutron transport equation is obtained as follows:

$$\begin{aligned}\hat{\Omega}.\nabla\Phi(\vec{r},\hat{\Omega},E,t) &+ \Sigma_t(\vec{r},E)\Phi(\vec{r},\hat{\Omega},E,t)\\ &= \int_0^\infty dE' \int_{4\pi} d\widehat{\Omega}'\,\Sigma_s(\vec{r},\hat{\Omega}'\to\hat{\Omega},E'\to E)\Phi(\vec{r},\widehat{\Omega}',E',t)\\ &+ \frac{\chi_p(\vec{r},E)}{4\pi}\int_0^\infty dE' \int_{4\pi} d\widehat{\Omega}'\nu\Sigma_f(\vec{r},E')\Phi(\vec{r},\widehat{\Omega}',E',t) + Q(\vec{r},\hat{\Omega},E,t)\end{aligned} \tag{2}$$

Various numerical methods have been developed since late 1950s and early 1960s to appropriately model and solve the neutron transport equation. These are as follows:

1. Spherical Harmonics or P_n Method

Spherical harmonics or *P_n method* is the oldest method, and evolved into reactor physics by Gelbard (1968) [1] and Lewis and Miller (1984) [2] from a similar approach used in astrophysics in the early twentieth century. It is based on the discretization of differential form of the neutron transport equation by expanding the angular flux and neutron sources in real spherical harmonics up to n terms. However, application of P_n method to 2- and 3-dimensional problem leads to a system of linear equations with complex couplings between discretized spatial and angular unknowns. An efficient, closely related approximation, based on the solution of the *simplified P_n* or *SP_n equation* as proposed by Gelbard (1960) [3], produces a simplified linear system which can be effectively solved for 2- and 3-dimensional cases, has been applied for full-core calculations. It is based on the expansion of angular flux in an incomplete basis of orthogonal functions.

2. Discrete Ordinates or S_N Method

   The *Discrete ordinates* or *S_N method* is also based on discretization of differential form of linear Boltzmann equation by discretizing the angular flux to a suitable directional quadrature and further evaluating neutron flux along each specific angular base point. It was adapted into neutron transport theory by Carlson and Bell (1958) [4] from a technique developed by Chandrasekhar (1960) [5] in radiation transfer theory for stellar atmospheres. A fixed-point iterative scheme is used to solve the difference relations obtained for every direction. However, powerful pre-conditioners and acceleration strategies must be employed to ensure convergence in most practical applications.

3. Collision Probabilities or P_{ij} Method

   Collision probability (CP) or *P_{ij} method*, given by Carlvik (1965) [6], is based on the spatial discretization of multi-group form of integral transport equation assuming isotropic sources. The probability of neutrons born in region i, undergoing a collision in region j is determined resulting to a square CP matrix of order $N \times N$ in each energy group for a problem with N regions. This technique is well suited for treating general unstructured meshes, unlike the P_n, SP_n and S_N methods, but is practically feasible only for few region problems since the complexity and computational expenses increase as $O(N^2)$. Sanchez and McCormick (1982) [7] proposed the *interface current (IC) method*, in which the problem is subdivided into uncoupled cells, for each of which CP matrices are determined. The detailed flux is then reconstructed using the interface currents associated with each cell.

4. Method of Characteristics (MoC)

   The method of characteristics (MoC) is based on the iterative calculation of neutron flux by solving the multi-group characteristic form of transport equation along straight neutron streaming paths or trajectories. This approach was proposed by Askew (1972) [8]. It can be effectively applied to unstructured domains, like the collision probability (CP) method, but does not require determining computationally expensive square CP matrices. It can also treat anisotropic sources, and is preferred when the number of regions is either large (more than a few hundred) or the sources are not isotropic in the LAB. The CACTUS module of WIMS-E by Halsall (1980) [9], MOCC module of DRAGON V3 by Roy (1998) [10], MCCG module of DRAGON V4 by le Tellier and Hébert (2006) [11] and OpenMOC [12] are some of the implementations of this approach.

5. Monte Carlo Method

   Unlike all the previous techniques which form a part of the class of deterministic solvers of neutron transport, the *Monte Carlo method* is a stochastic solver in which the process of neutron transport is modeled by applying known distribution functions to simulate the random histories of a large number of fictitious Monte Carlo particles, and averaging the results over all histories. It, thus, simulates the actual physical process and no approximate mathematical model, like the neutron transport equation, is required. If the geometry of the system and cross section data are available, the results of Monte Carlo simulations suffer only from statistical noise. However, the statistical error, which decreases as square root of number of Monte Carlo particles, is orders of magnitude larger than the actual physical process due to the limitation on the number of particles that can be simulated. The method is widely used due to its ability to treat arbitrary geometries, relatively easy implementation, and high accuracy. But, the simulations using this method are very slow and costly to set up and to run. Some implementations of the approach to reactor physics problems are by Carter and Cashwell (1975) [13], Kalos and Whitlock (1986) [14], Lux and Koblinger (1991) [15], Spanier and Gelbard (2008) [16] and the X-5 Monte Carlo team (2003) [17].

Hence, in this work, the method of characteristics has been preferred for solving the neutron transport equation for lattice level problems over the other available techniques due to the many advantages associated with it.

PROBLEM UNDER CONSIDERATION

The objective of this work is to develop a module for solving the neutron transport equation for a single square pin-cell using the MoC to obtain its effective multiplication factor. Figure 1 represents a typical region of interest.

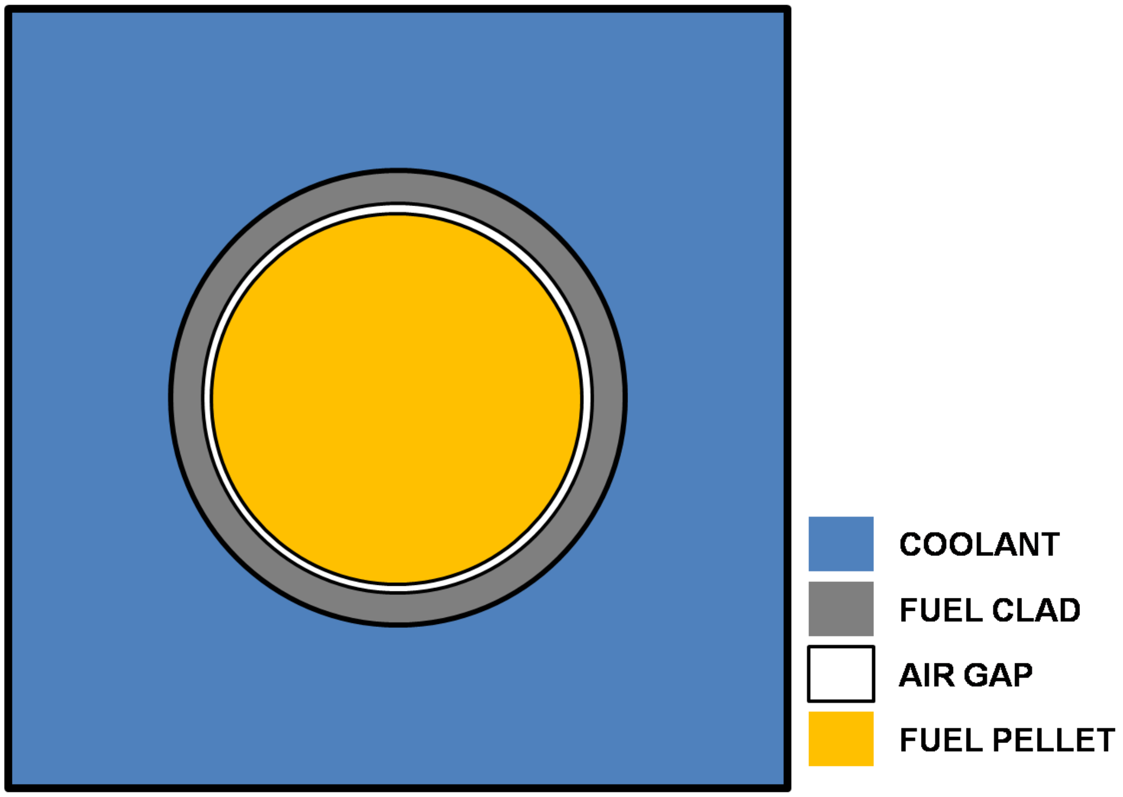


Figure 1 - Typical region of interest

ORGANIZATION OF THE THESIS

Chapter 1, entitled *Characteristic Form of Neutron Transport Equation*, provides the theoretical framework used for developing lattice level transport equation solver using the method of characteristics. It gives a derivation of the characteristic form of neutron transport equation from the original linear Boltzmann equation. The chapter then describes the analytical solution of the characteristic neutron transport equation to determine the desired quantities of interest, angular and scalar neutron flux values. It concludes with broad outline of the steps generally followed for method of characteristics based lattice level calculations.

Chapter 2, is entitled *Delaunay Triangulation*, and focuses on the meshing technique used for dividing the problem geometry into smaller triangular regions. It gives a detailed description of the algorithms developed for initial meshing, and further mesh refinement to obtain good quality meshes which conform to optimal shape and size parameters.

Chapter 3 is entitled *Ray Tracing* and explains the next stage of the work. It provides an exhaustive explanation of the technique used for selecting the directional quadrature for suitably representing 3-dimensional neutron motion and insertion and tracking of characteristic paths signifying neutron trajectories through the region of interest.

Chapter 4, entitled *Flux and Multiplication Factor Calculations*, explains the final stage of the lattice code developed. It describes inner – outer iteration scheme using the analytical solution of the characteristic neutron transport equation as described in chapter 1, to determine required flux values in each mesh, which are thereafter utilized for obtaining the effective multiplication factor for the region of interest.

Chapters 5 is entitled *Benchmarks Analyzed*. It provides details of the each of the several benchmarking problems solved and analyzed by the developed code to test and validate its operation for future use. The solution of each benchmarking problem is also discussed in this chapter.

The chapters are followed by conclusion and future scope of the work undertaken.

CHAPTER 1

CHARACTERISTIC FORM OF NEUTRON TRANSPORT EQUATION

1.1 METHOD OF CHARACTERISTICS

Method of characteristics (MoC) is a mathematical technique of numerical treatment of partial differential equation based on an analytical approach. Application of MoC on any partial differential equation simplifies it to an ordinary differential equation, which is generally much easier to handle. Simplification of the original partial differential equation is done by analytically solving it along certain *characteristic curves* or *characteristics*. *Characteristic curves* or *characteristics* are $(n-1)$-dimensional surfaces (curves) in n-dimensional space which exhibit certain special property, and this property is used advantageously for simplification via analytical means. In case of the neutron motion, these characteristic curves are the streaming paths along which the neutrons traverse through any region of interest.

1.2 DERIVATION OF CHARACTERISTIC NEUTRON TRANSPORT EQUATION

The integrodifferential form of Boltzmann transport equation describes the motion of neutrons as observed from a fixed reference point. This leads to the partial derivative in the streaming term. However, if the streaming operator, $\hat{\Omega}.\nabla\Phi$, is integrated along a straight line in direction $\hat{\Omega}$ corresponding to the trajectory of neutron motion, the characteristic form of neutron transport equation is obtained. If the neutron is assumed to be at distance s from a reference point $\vec{r}$ on its characteristic path $\vec{s}$, its actual position is given as $\vec{r} + s\hat{\Omega}$.

But,

$$\frac{d}{ds} = \frac{dx}{ds}\frac{\partial}{\partial x} + \frac{dy}{ds}\frac{\partial}{\partial y} + \frac{dz}{ds}\frac{\partial}{\partial z} \tag{1.1}$$

$$\hat{\Omega} ds = d\vec{r} = \hat{\imath} dx + \hat{\jmath} dy + \hat{k} dz$$

(1.2)

Taking the dot product of equation (1.2) with $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$,

$$\hat{\Omega} ds . \hat{\imath} = dx$$

(1.3a)

$$\hat{\Omega} ds . \hat{\jmath} = dy$$

(1.3b)

$$\hat{\Omega} ds . \hat{k} = dz$$

(1.3c)

Substituting equation (1.3) in (1.1)

$$\frac{d}{ds} = \left(\hat{\Omega}.\hat{\imath}\right)\frac{\partial}{\partial x} + \left(\hat{\Omega}.\hat{\jmath}\right)\frac{\partial}{\partial y} + \left(\hat{\Omega}.\hat{k}\right)\frac{\partial}{\partial z} = \hat{\Omega}.\nabla$$

(1.4)

Using (1.4) in equation (2), the backward characteristic form of neutron transport equation is obtained as

$$\frac{d\Phi(\vec{r} + s\hat{\Omega}, E, \hat{\Omega})}{ds} + \Sigma_t(\vec{r} + s\hat{\Omega}, E)\Phi(\vec{r} + s\hat{\Omega}, E, \hat{\Omega}) = Q(\vec{r} + s\hat{\Omega}, E, \hat{\Omega})$$

(1.5)

where, all source terms have been absorbed in $Q(\vec{r} + s\hat{\Omega}, E, \hat{\Omega})$, and are not given explicitly.

The first term on the L.H.S. of the neutron transport equation signifies the rate at which neutrons are lost out of the region of interest due to leakage; mathematically the rate of change of neutron flux along the characteristic path or ray or track segment while the second term signifies the loss due to scattering, absorption and those causing fission. The term on the R.H.S. denotes the rate of addition or generation of neutrons due to the presence of any sources (scattering, fission, actual sources). The forward characteristic form of neutron transport equation can be derived similarly.

1.2.1 Angular Flux

Dropping all indices, the characteristic neutron transport equation simplifies as follows,

$$\frac{d\Phi}{ds} + \Sigma_t \Phi = Q \tag{1.6}$$

Assuming that the source of neutrons is constant within the region of interest, the solution of the above differential equation is analytically obtained as

$$\Phi(s) = \Phi(0)e^{-\Sigma_t s} + \frac{Q}{\Sigma_t}(1 - e^{-\Sigma_t s}) \tag{1.7}$$

where, $s = 0$ is the location of neutron at initial time, $t = 0$.

If the neutron crosses through regions of different material composition, and having varying neutron source, as represented in Figure 1.1, the angular flux can be obtained by dividing the neutron trajectory into smaller regions with piecewise uniform properties, then

$$\Phi_n^{out} = \Phi_n^{in} e^{-\Sigma_{t,n} s_n} + \frac{Q_n}{\Sigma_{t,n}}(1 - e^{-\Sigma_{t,n} s_n}) \tag{1.8}$$

where,

Φ_{out}	=	Outgoing angular flux for a region
Φ_{in}	=	Incoming angular flux for a region
n	=	Material or region index

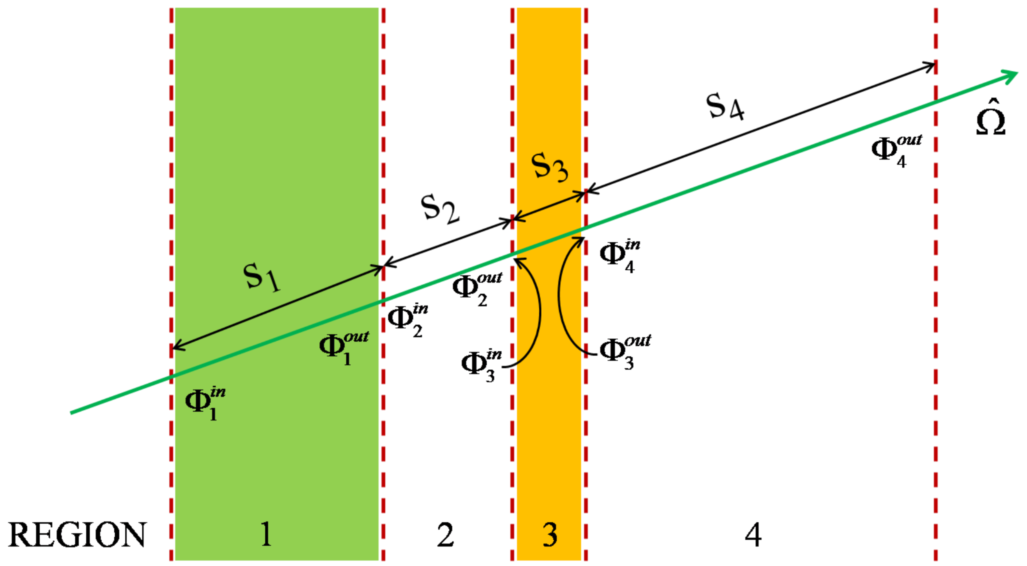


Figure 1.1 - Trajectory of neutron motion through regions with different material composition. Each individual region has piecewise uniform properties.

In multi group form,

$$\Phi_{g,n}^{out} = \Phi_{g,n}^{in} e^{-\Sigma_{t,n,g} s_n} + \frac{Q_{g,n}}{\Sigma_{t,n,g}} (1 - e^{-\Sigma_{t,n,g} s_n}) \tag{1.9}$$

1.2.2 Average Angular Flux

The average angular flux along the neutron streaming path along track segment extending from 0 to s is, thus, obtained as the ratio of integration of the angular flux values at every point along the track segment to the length of the track segment and is given as

$$\bar{\Phi} = \frac{\int_0^s \Phi(s')ds'}{\int_0^s ds'} \tag{1.10}$$

Using equation (1.7),

$$\overline{\Phi} = \frac{Q}{\Sigma_t} + \frac{\Phi(0) - \Phi(s)}{\Sigma_t s}$$

(1.11)

where,

$\overline{\Phi}$	=	Average angular flux along the track segment
Δ	=	$\Phi(0) - \Phi(s)$

Thus, average angular flux for any region n, is given as,

$$\overline{\Phi}_n = \frac{Q_n}{\Sigma_{t,n}} + \frac{\Phi_n^{in} - \Phi_n^{out}}{\Sigma_{t,n} s_n} = \frac{Q_n}{\Sigma_{t,n}} + \frac{\Delta_n}{\Sigma_{t,n} s_n}$$

(1.12)

where,

Δ	=	$\Phi^{in} - \Phi^{out}$

In multi group form,

$$\overline{\Phi}_{g,n} = \frac{Q_{g,n}}{\Sigma_{t,n,g}} + \frac{\Phi_{g,n}^{in} - \Phi_{g,n}^{out}}{\Sigma_{t,n,g} s_n} = \frac{Q_{g,n}}{\Sigma_{t,n,g}} + \frac{\Delta_{g,n}}{\Sigma_{t,n,g} s_n}$$

(1.13)

1.2.3 Scalar Flux

After average angular flux is determined in each direction, the scalar neutron flux in the region of interest is then calculated as the sum of average angular neutron flux in every direction weighted by the respective directional weights and is given as

$$\phi = \int_{\hat{\Omega}} \overline{\Phi}(\hat{\Omega})\, d\hat{\Omega}$$

where,

$d\hat{\Omega}$	=	Weight associated with a particular direction $\hat{\Omega}$

In discrete form,

$$\phi = \sum_{dir} \bar{\Phi}_{dir}\omega_{dir} = \sum_{m} \left(\frac{Q_{dir}}{\Sigma_t} + \frac{\Delta_{dir}}{\Sigma_t s_{dir}} \right) . \omega_{dir}$$

(1.14)

where,

ω_{dir}	=	Discrete directional weight equivalent of differential weight $d\hat{\Omega}$; $\sum_{dir} \omega_{dir} = 4\pi$
dir	=	Subscript representing direction

Hence, scalar flux of different material regions is determined using,

$$\phi_n = \sum_{dir} \bar{\Phi}_{dir,n}\omega_{dir} = \sum_{dir} \left(\frac{Q_{n,dir}}{\Sigma_{t,n}} + \frac{\Delta_{n,dir}}{\Sigma_{t,n} s_{n,dir}} \right) . \omega_{dir}$$

(1.15)

In multi group form, equation (1.14) becomes

$$\phi_{g,n} = \sum_{dir} \bar{\Phi}_{g,dir,n}\omega_{dir} = \sum_{dir} \left(\frac{Q_{g,n,dir}}{\Sigma_{t,n,g}} + \frac{\Delta_{g,n,dir}}{\Sigma_{t,n,g} s_{n,dir}} \right) . \omega_{dir}$$

(1.16)

Under the assumption of neutron motion with isotropic scattering, the modified expressions for angular flux, average angular flux and scalar flux are obtained by replacing total macroscopic cross section by transport corrected cross section as:

$$\Phi_{g,n}^{out} = \Phi_{g,n}^{in} e^{-\Sigma_{tr,n,g} s_n} + \frac{Q_{g,n,dir}}{\Sigma_{tr,n,g}} (1 - e^{-\Sigma_{tr,n,g} s_n})$$

(1.17)

$$\bar{\Phi}_{g,n} = \frac{Q_{g,n,dir}}{\Sigma_{tr,n,g}} + \frac{\Phi_{g,n}^{in} - \Phi_{g,n}^{out}}{\Sigma_{tr,n,g} s_n} = \frac{Q_{g,n,dir}}{\Sigma_{tr,n,g}} + \frac{\Delta_{g,n}}{\Sigma_{tr,n,g} s_n} \tag{1.18}$$

$$\phi_{g,n} = \sum_{dir} \bar{\Phi}_{g,dir,n} \omega_{dir} = \sum_{dir} \left(\frac{Q_{g,n,dir}}{\Sigma_{tr,n,g}} + \frac{\Delta_{g,n,dir}}{\Sigma_{tr,n,g} s_{n,dir}} \right) . \omega_{dir} \tag{1.19}$$

1.3 GEOMETRY ROUTINE

Prior to solving the characteristics form of neutron transport equation by means of computational schemes, some pre-processing steps are essential. These are as follows:

1. Problem region is divided into a suitable number of cells which are generally, the basic building blocks or lattices which can be repeated to regenerate the entire problem area.
2. Each cell is further divided into material regions and material IDs assigned to each such region of the problem.
3. Each material region is subdivided into a suitable number of meshes such that the flat source and flat flux approximations are applicable.
4. Streaming paths for the neutrons i.e. paths along which neutrons travel are laid down.
   a. First the streaming paths are laid down in azimuthal direction (plane of the problem)
   b. After calculating path length, τ''s, each path is raised in various polar directions by taking ratio of each of the path lengths with the sine of the various polar angles to obtain true path length in each direction of motion, $\hat{\Omega}_m$. Number of directions is obtained from the total number of azimuthal angles, I, and total number of polar angles, J.
   True path length, τ, is, thus, given by

$$\tau = \frac{\tau'}{\sin \theta_j} \tag{1.20}$$

5. Symmetry of the problem geometry can be used advantageously to simplify and reduce the number of directions for which flux calculation must be performed after making necessary and appropriate adjustments.

Figure 1.2 illustrates the broad stages using which the above mentioned geometry routine has been implemented in the code.

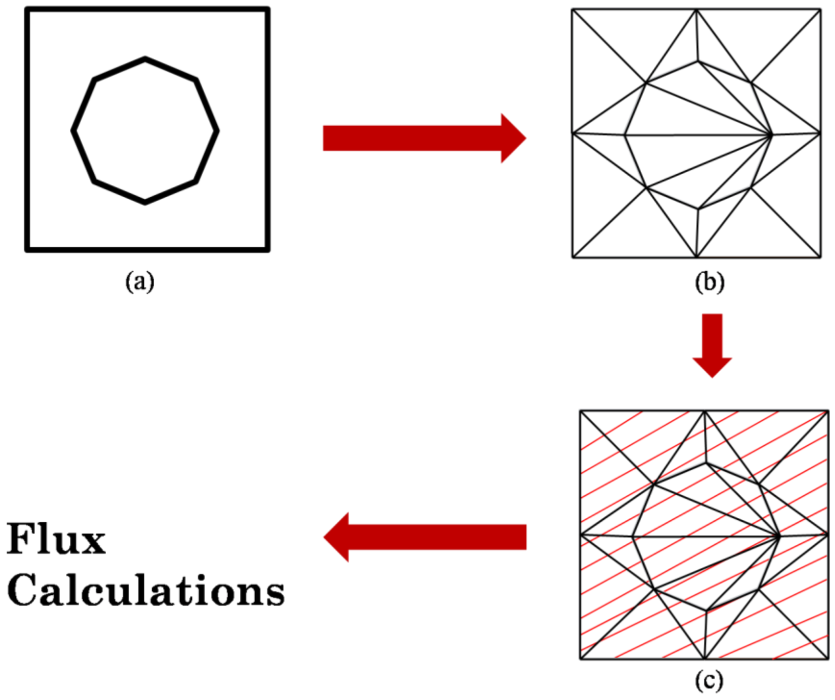


Figure 1.2 - Basic methodology (a) Reconstruction of the cell geometry from its input specifications (b) Subdivision of the cell into smaller meshes (c) Ray tracing throughout the cell followed by the flux calculations

CHAPTER 2

DELAUNAY TRIANGULATION

2.1 MESHING

Most physical phenomena in science and engineering are modeled by linear/non-linear, ordinary/partial differential equations. Solution of such equations can be numerically approximated by replacing the continuous system with a discrete system of a finite number of coupled linear/non-linear equations. This process of discretization involves partitioning of the problem domain into small regions of simple shapes, known as *meshes*, which are usually triangles or quadrilaterals (in two-dimensions). The solution of the discrete linear/non-linear equations accumulated over the entire problem domain provides the required approximated numerical solution.

It is desired that the process of mesh generation possesses certain properties. Firstly, it should generate meshes which can faithfully reproduce the problem domain being modeled, however complex its shape maybe. Secondly, it should provide as much flexibility as possible on the size of individual meshes and be able to offer rapid size-gradation from large to small meshes over relatively short distances. This gives control over mesh density depending upon behavior of the physical phenomena and hence, an important tradeoff between computation time and accuracy. The third, and probably the most difficult to achieve, is to generate meshes that are relatively “round” in shape, i.e. the ratio of its area to perimeter (in 2-dimensions) or volume to surface area (in 3-dimensions) should be as large as possible. The need for such a requirement stems out from the fact that elements with extremely large or small angles can degrade the quality of the numerical solution. While large angles can cause large *discretization error* and large errors in derivatives of the solution, as shown in Figure 2.1, small angles may yield ill-conditioned coupled systems of algebraic equations and introduce round-off errors. As shown by Babuška and Aziz [18], convergence might be prevented if mesh angles approach 180° even with decreasing mesh size.

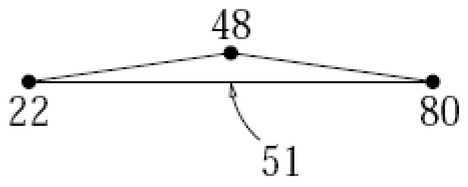


Figure 2.1 - Nodal values depicted may represent accurate estimate of correct solution. Nevertheless, as the large angle approaches 180°, the vertical directional derivative, estimated via linear interpolation, shows arbitrarily large error.

Meshes or grids can be broadly classified into two main classes, i.e. structured mesh and unstructured mesh. They are defined as follows:

1. Structured Mesh is defined as the one in which the indices of neighboring nodes can be determined using a simple formula. The internal mesh nodes exhibit a uniform topological structure and are connected to neighbors independent of their position.
2. Unstructured Mesh is defined as the one in which neighboring elements can't be determined using a formula, which necessitates storage of neighbor list of each node. Such meshes lack a regular topological structure and the pattern of connections varies from point to point.

Each of the above types has its own set of *pros* and *cons*. Although, unstructured meshes do suffer from a few drawbacks, they are perfectly suited to the nature of the physical system, the square pin-cell of a reactor core. The advantages and disadvantages associated with unstructured meshes are listed below.

Advantages of unstructured meshing:

1. It can effectively reproduce irregularly shaped domains due to which at times it is the preferred or even indispensable method. This aspect becomes even more pronounced when working with 3-dimensional geometries.
2. It offers rapid gradation of mesh size giving a great degree of control on mesh number and density. Thus, an optimization can be achieved between speed and accuracy.
3. It also offers far better multi-scale resolution and flexible tailoring conforming to complexity of the problem under consideration.

These very desirable advantages of unstructured meshing come at the cost of following disadvantages:

1. Determining neighboring nodes is a complex task since no direct formula can be used and might require extensive algorithms and complex data structures itself.
2. It is extremely expensive in terms of storage space and memory traffic demands due to the need to store neighboring nodes list of individual nodes.
3. It is extremely difficult to parallelize computation because of the irregular structure of such meshes. Sophisticated partitioning algorithms and parallel unstructured solvers may be required for this purpose.

Also, since it is desired that the characteristic form of neutron transport equation be solved in 2 dimensions, 2-dimensional unstructured meshing becomes an obvious choice.

Triangles are the simplest 2-dimensional objects and numerous *triangulation* algorithms exist in engineering and computational geometry literature. *Delaunay triangulation* introduced by Russian mathematician Boris Delaunay in 1934 [19], has been an extremely popular technique since the very early stages of development of mesh generation techniques. This geometric structure, which is a geometric dual of *Dirichlet Tessellations* or *Voronoi Diagrams* [20], has been extensively researched and widely used in two dimensions due to its ability to generate good quality meshes. Extensions to higher dimensions have enjoyed no less popularity.

In two dimensions, *triangulation* of a set V of vertices results in a set T of triangles whose:

1. vertices collectively are V,
2. interiors don’t intersect each other, and
3. union is the convex hull of V,

if every triangle intersects V only at the triangle’s vertices. Figure 2.2 represents triangulation of a set of vertices.

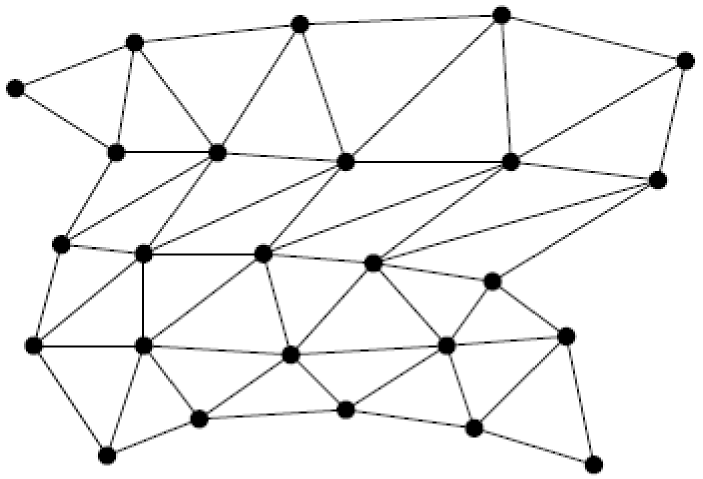


Figure 2.2 - Triangulation of a set of vertices

Any circle in a plane is said to be *empty* if it encloses no vertices of *V*. An edge is said to be Delaunay if it has at least one empty circum-circle. Figure 2.3 shows a Delaunay edge, one of whose circum-circles does not contain any vertex in its interior.

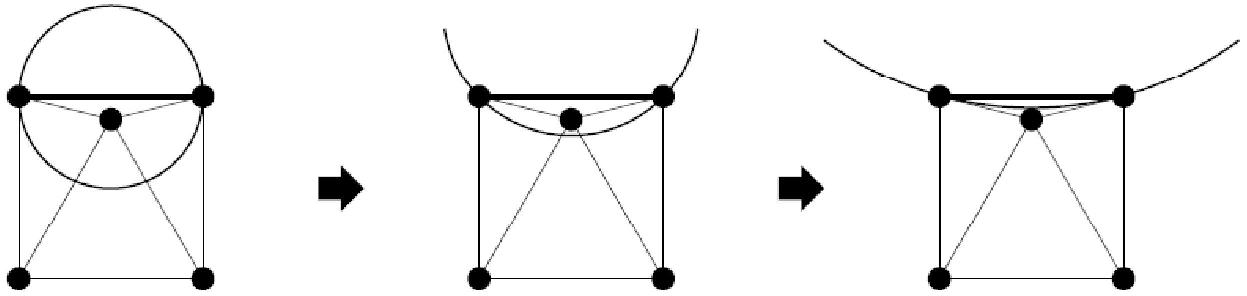


Figure 2.3 - Delaunay Edge

Similarly, a triangle is said to be Delaunay if and only if its circum-circle is empty. Figure 2.4(a) shows two Delaunay triangles formed from a set of 4 vertices while Figure 2.4(b) shows two non-Delaunay triangles from the same set of four vertices. It should also be noted that the common edge *e* of the two triangles formed, satisfies empty circum-circle property when the triangles are Delaunay but not otherwise and flipping *e* converts non-Delaunay triangles on the right to Delaunay triangles on the left.

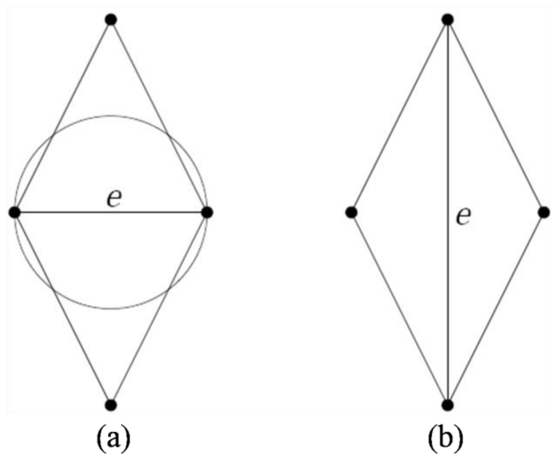


Figure 2.4 - (a)Delaunay and (b)non-Delaunay triangle

If every triangle of a triangulation of a set of vertices *V* is Delaunay, then it is said to be *Delaunay Triangulation D* of *V*, as given in Figure 2.5. Circum-circle of every triangle of the triangulation shown is empty.

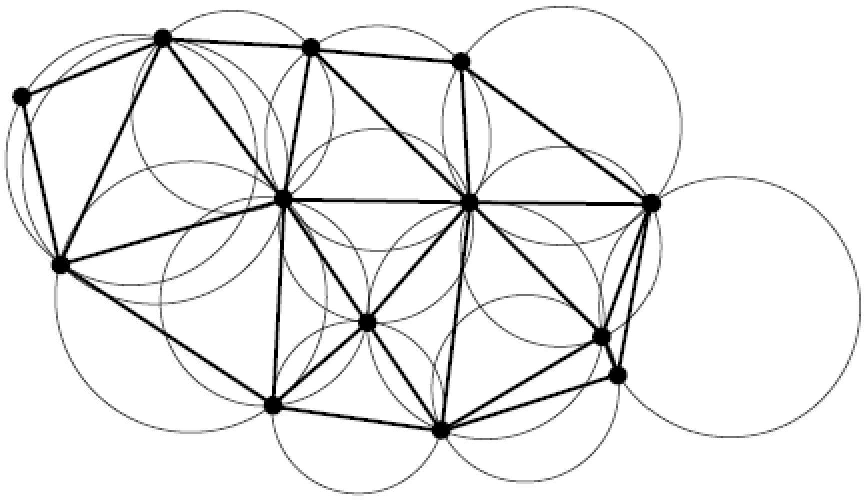


Figure 2.5 - Delaunay Triangulation of a set of vertices

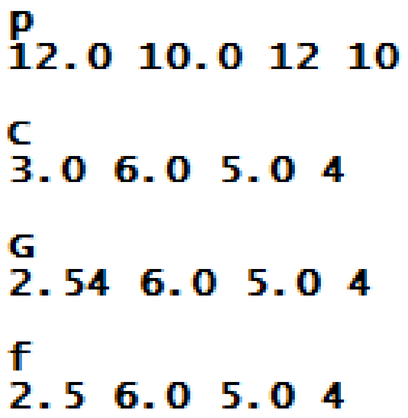
Use of Delaunay triangulation as a mesh construction technique and a guide to refinement by new vertex insertion has been intensively studied in the engineering community since the 1980s and, with the improvement in power and capability of computers, began attracting interest from the computational geometry community since the 1990s. Continued popularity of the technique stems out from the several favorable characteristics associated with it. These are as follows:

1. The most important feature of Delaunay triangulation is its ability to maximize minimum angle among all possible triangulations of a set of vertices due to which meshing is of better quality.
2. It is extremely suited to adaptive solution strategies and can be flexibly tailored according to the requirement of the problem geometry and the nature of the physical process being modeled.
3. The insertion of a new vertex, being a local operation, in one part of the mesh does not unnecessarily disturb a distant part of the mesh, and hence is inexpensive except in unusual cases.
4. Another great advantage which is much less obvious is that connections are made between nearest neighbors and new vertices are inserted “as far away from other vertices as possible”.
5. Due to all the above features, Delaunay triangulation has been extensively studied resulting in the availability of numerous good algorithms, which simplifies the process significantly.

2.2 INITIAL MESH NODES GENERATION

First and the foremost step for meshing of the problem geometry is generation of initial mesh points/nodes from the input specifications which are later *triangulated*, or connected together to form triangular shaped meshes using *Delaunay Triangulation* technique.

The specifications of the lattice are provided by the user in the form of an input file, *pin_cell_geom_params.in*. Figure 2.6 gives a snapshot of the input file. The dimensions and the number of segments of each material region is input by the user through this file, which is read by the subroutine, *pin_cell_mesh_nodes_generator*.

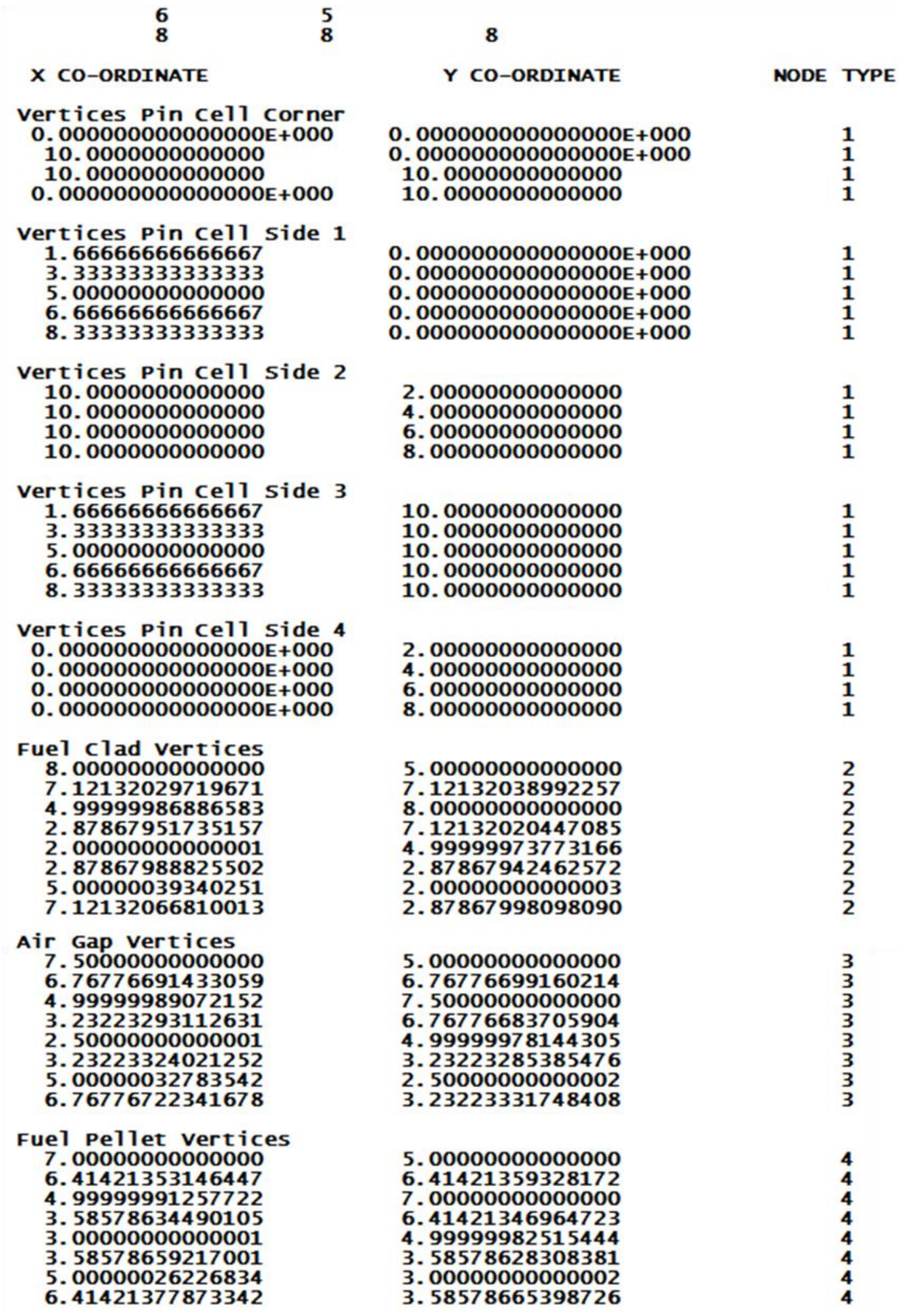


```
p
12.0 10.0 12 10

c
3.0 6.0 5.0 4

G
2.54 6.0 5.0 4

f
2.5 6.0 5.0 4
```

Figure 2.6 - Snapshot of input file for geometry specifications

The subroutine processes this information to generate the coordinates of mesh nodes, the underlying philosophy being to divide the boundaries of each region in segments of equal size. Each point is also assigned a *type*, which basically specifies the boundary on which the point is located. This information is useful to determine the material type of each mesh required during calculations. The points, thus, generated are stored in an output file, *pin_cell_mesh_nodes.out*, which is used further to generate meshes. Figure 2.7 gives a snapshot of the output file.

The complete process of generating initial mesh nodes is depicted by the block diagram as given in Figure 2.8.



```
            6              5
            8              8              8

 X CO-ORDINATE                    Y CO-ORDINATE            NODE TYPE

Vertices Pin Cell Corner
 0.000000000000000E+000       0.000000000000000E+000           1
  10.0000000000000            0.000000000000000E+000           1
  10.0000000000000             10.0000000000000                1
 0.000000000000000E+000        10.0000000000000                1

Vertices Pin Cell Side 1
  1.66666666666667            0.000000000000000E+000           1
  3.33333333333333            0.000000000000000E+000           1
  5.00000000000000            0.000000000000000E+000           1
  6.66666666666667            0.000000000000000E+000           1
  8.33333333333333            0.000000000000000E+000           1

Vertices Pin Cell Side 2
  10.0000000000000             2.00000000000000                1
  10.0000000000000             4.00000000000000                1
  10.0000000000000             6.00000000000000                1
  10.0000000000000             8.00000000000000                1

Vertices Pin Cell Side 3
  1.66666666666667             10.0000000000000                1
  3.33333333333333             10.0000000000000                1
  5.00000000000000             10.0000000000000                1
  6.66666666666667             10.0000000000000                1
  8.33333333333333             10.0000000000000                1

Vertices Pin Cell Side 4
 0.000000000000000E+000        2.00000000000000                1
 0.000000000000000E+000        4.00000000000000                1
 0.000000000000000E+000        6.00000000000000                1
 0.000000000000000E+000        8.00000000000000                1

Fuel Clad Vertices
  8.00000000000000             5.00000000000000                2
  7.12132029719671             7.12132038992257                2
  4.99999986886583             8.00000000000000                2
  2.87867951735157             7.12132020447085                2
  2.00000000000001             4.99999973773166                2
  2.87867988825502             2.87867942462572                2
  5.00000039340251             2.00000000000003                2
  7.12132066810013             2.87867998098090                2
Air Gap Vertices
  7.50000000000000             5.00000000000000                3
  6.76776691433059             6.76776699160214                3
  4.99999989072152             7.50000000000000                3
  3.23223293112631             6.76776683705904                3
  2.50000000000001             4.99999978144305                3
  3.23223324021252             3.23223285385476                3
  5.00000032783542             2.50000000000002                3
  6.76776722341678             3.23223331748408                3

Fuel Pellet Vertices
  7.00000000000000             5.00000000000000                4
  6.41421353146447             6.41421359328172                4
  4.99999991257722             7.00000000000000                4
  3.58578634490105             6.41421346964723                4
  3.00000000000001             4.99999982515444                4
  3.58578659217001             3.58578628308381                4
  5.00000026226834             3.00000000000002                4
  6.41421377873342             3.58578665398726                4
```

Figure 2.7 - Snapshot of output file containing initial mesh vertices

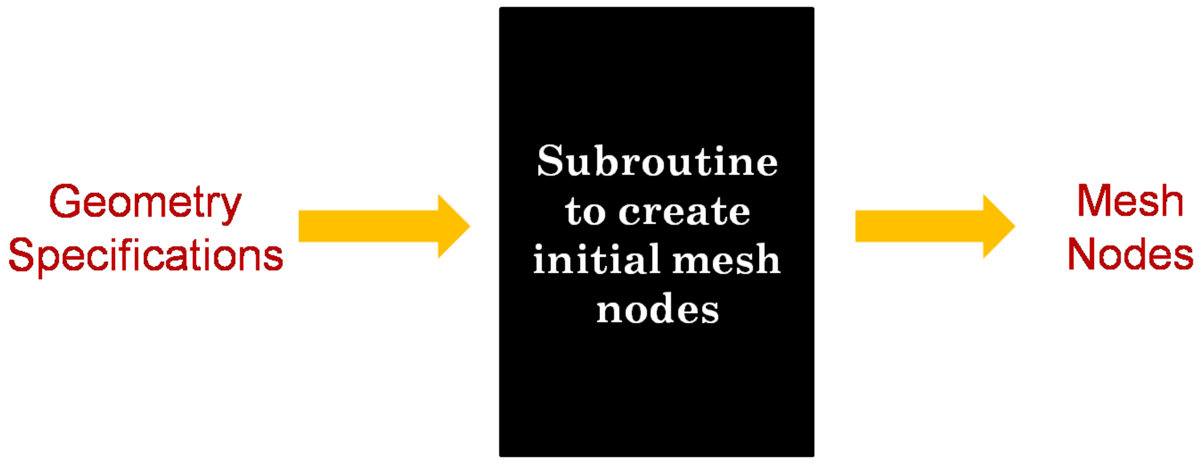


Figure 2.8 - Block diagram of process for generating initial mesh nodes

2.3 MESH GENERATION

Delaunay triangulations can be constructed using a variety of algorithms, easiest being incremental insertion algorithms, divide-and-conquer and sweepline techniques being the other two. Incremental insertion algorithms are based on the local nature of vertex insertion. The principle of operation is simply to maintain Delaunay triangulation of the grid in which mesh nodes are being inserted one at a time. Earliest such algorithm, applicable to 2-dimensions, was introduced by Lawson [21]. However, a more convenient procedure, which can be generalized to higher dimensions, is the Bowyer [22] and/or Watson [23] algorithm.

The Bowyer-Watson algorithm is a “reconnection” method based on the empty circum-circle property of Delaunay triangulation. All triangles of the existing graph, whose empty circum-circle property is violated and hence, no longer remain Delaunay, because of the insertion of new vertex, are eliminated. All other triangles remain Delaunay and are not disturbed. Bowyer [22] and Watson [23] have shown that:

1. All these affected triangles are contiguous and located in the local neighborhood of the new vertex, and form a connected cavity surrounding it.
2. Connecting the new vertex to each edge of the enclosing cavity always gives Delaunay triangles and the triangulation, thus obtained is again Delaunay.

A sequential repetition of this process until all mesh nodes have been inserted leads to a Delaunay triangulation. The initial triangulation so obtained, however, may contain poor quality triangles that are either not size and shape optimal or unsuitable for numerical solution and associated assumptions involved. More about this will be discussed in later chapters. In order to overcome this problem, refinement of the initial triangulation is required. Thus, the process of mesh generation is divided into: 1) initial triangulation of the system, 2) mesh refinement to improve quality of triangulation

2.4 INITIAL TRIANGULATION ALGORITHM

The basic Bowyer-Watson algorithm lacks the ability to deal with the *degeneracy case*. Hence, a modified form of the algorithm, which can handle the *degenerate vertices*, has been employed for triangulation of the initial mesh nodes. This algorithm is implemented through a subroutine which gives an initial Delaunay triangulation. The process is depicted through the block diagram given in Figure 2.9.

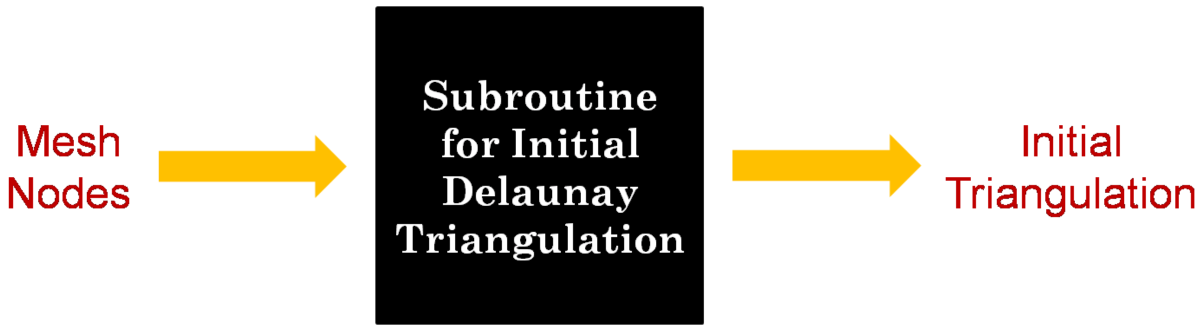


Figure 2.9 - Block diagram of process of obtaining initial triangulation

In an n-dimensional space, infinitely many “circum-spheres” pass through a set of *n* points and there always exists a unique “circum-sphere” which passes through *n+1* points. If more than *n+1* points lie on the same “circum-sphere”, they are said to be *degenerate* and a *degeneracy* is said to have occurred. In 2-dimensional space, a line segment has infinitely many circum-circles and every triangle has a unique circum-circle, as shown in Figure 2.10. But if 4 or more points share a common circum-circle, then these points are said to be *degenerate*. Figure 2.11 represents a set of *degenerate points*.

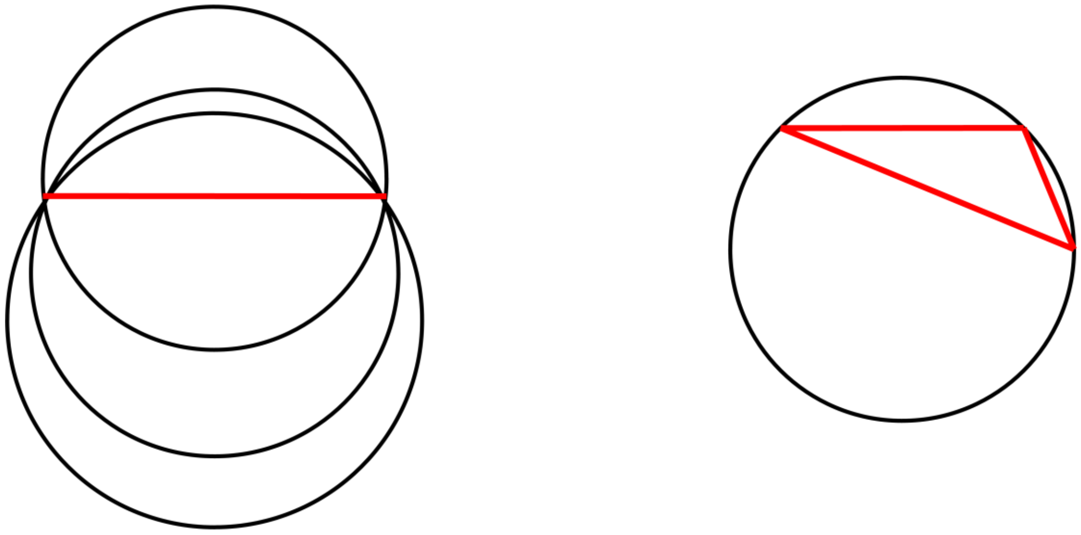


Figure 2.10 - Set of possible circum-circles for 2 points (line segment) and 3 points (triangle) lying on a 2-dimensional plane

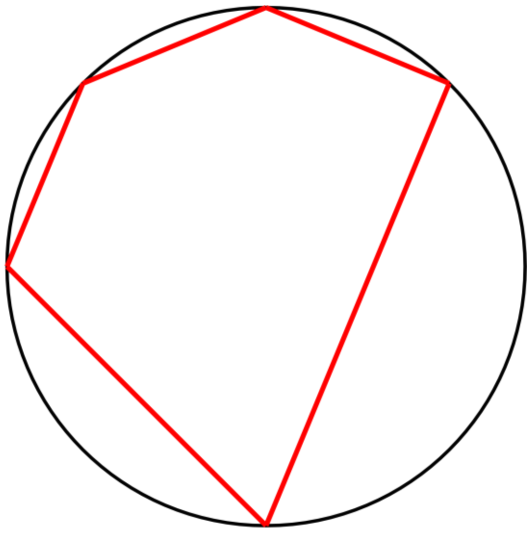


Figure 2.11 - A set of degenerate points in a plane

Since occurrence of the degeneracy case is certain due to the geometry of the pin-cell, modifications have been made to the basic Bowyer-Watson algorithm. The initial triangulation algorithm is as follows:

STEP 1 – Obtain first triangle from any three corner nodes of the pin cell
STEP 2 – For each vertex in the mesh node list,
STEP 3 – Check whether Degeneracy case or Bad Triangle case
STEP 4 – Address particular case appropriately
STEP 5 – Repeat steps 3 and 4 till all initial vertices have been inserted

Degeneracy Case

STEP 1 – If all triangles in the existing triangulation share a common circum-circle,
STEP 2 – Then, if distance of new vertex from the “common” circum-center is equal to the circum-radius
STEP 3 – Then, prioritize existing vertices in ascending order of distance from new vertex
STEP 4 – Connect new vertex to the 2 highest priority vertices to create a new triangle
STEP 5 – Add new triangle to existing triangulation

Bad Triangle Case

STEP 1 – For each triangle in existing triangulation,
STEP 2 – If new vertex violates empty circum-circle property of the triangle,
STEP 3 – Then, add *bad triangle* to bad triangle list
STEP 4 – Repeat steps 2 and 3 till all bad triangles have been identified
STEP 5 – Identify polygonal hole (open/closed) formed by these bad triangles; all non-shared edges among bad triangles form the boundary of this polygonal cavity
STEP 6 – Delete all bad triangles from existing triangulation list
STEP 7 – Connect each boundary edge of the polygonal hole to new vertex, forming a triangle with each edge
STEP 8 – Add each new triangle, thus, created to the triangulation.

Figure 2.12 illustrates the classification of a triangles as good or bad based on its empty circum-circle property.

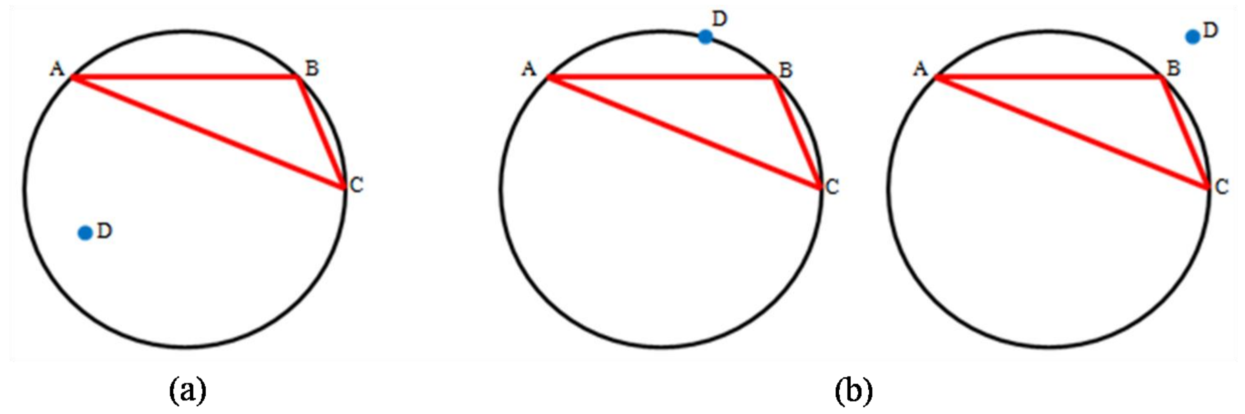


Figure 2.12 - (a) Bad Triangle ABC since vertex D lies in the interior of its circum-circle (b) Good Triangle ABC since vertex D lies either on or outside its circum-circle

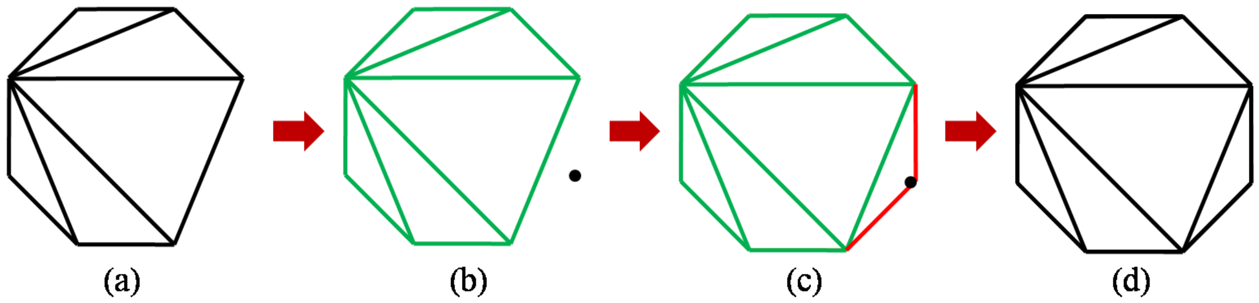


Figure 2.13 - Illustration of degeneracy case (a) Existing triangulation (b) New vertex (in black) being inserted is degenerate with all existing vertices and all triangles (in green) are affected (c) New vertex connected to 2 highest priority nodes (d) New updated triangulation

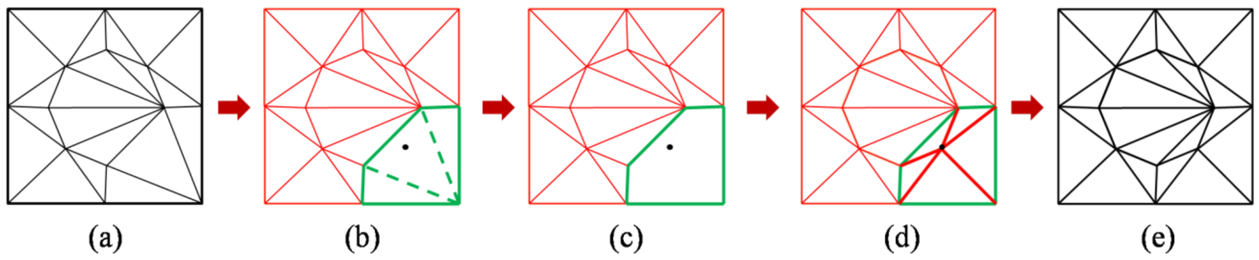


Figure 2.14 - Illustration of bad triangle case (a) Existing triangulation (b) All triangles that become “bad” (in green) due to insertion of the new vertex (in black) (c) Polygonal cavity formed by all contiguous bad triangles (d) Reconnection of new vertex with edges of polygonal hole (e) New updated triangulation

After all mesh nodes have been added, initial Delaunay triangulation is obtained. Figure 2.13 and Figure 2.14 give illustration of algorithms for handling degeneracy case and bad triangle case, respectively.

The initial mesh nodes generated from the specifications of the problem domain are read from the file, *pin_cell_mesh_nodes.out*, by the subroutine *initial_triangulation* to create initial Delaunay triangles. Figure 2.15 provides an implementation of the initial triangulation algorithm i.e. the modified Bowyer-Watson algorithm.

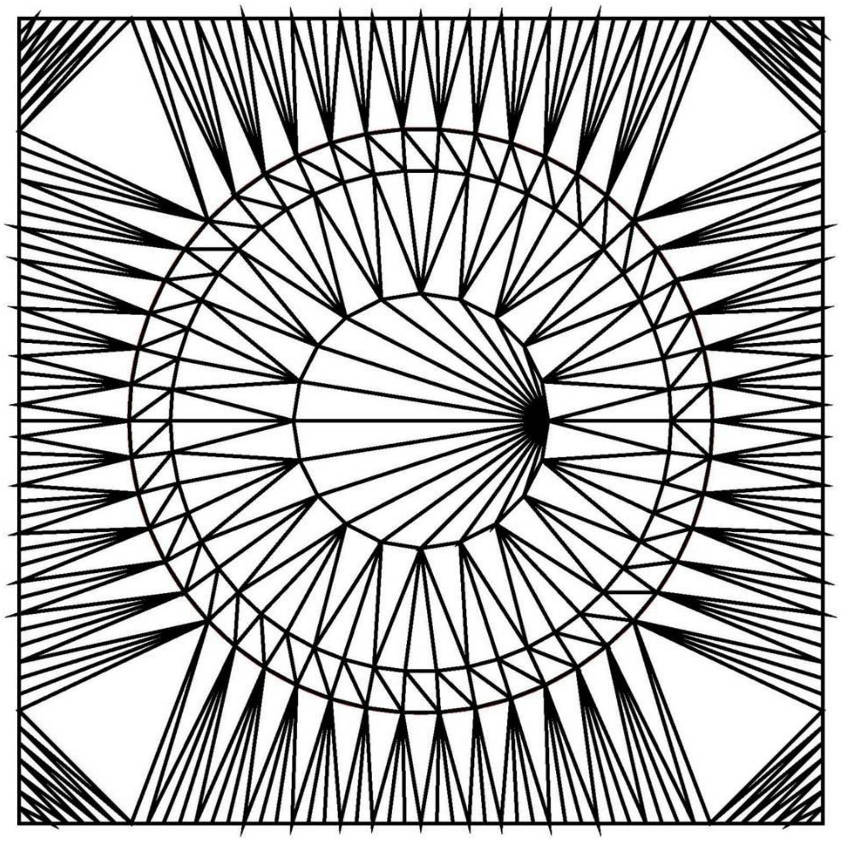


Figure 2.15 - Illustration of modified Bowyer-Watson algorithm to obtain initial triangulation

2.5 MESH REFINEMENT ALGORITHM

Mesh refinement is the process of identifying all poor quality meshes in the initial triangulation and replace them with triangles of better quality. Categorization of meshes as poor or not is based on their size and minimum angle. A triangle is considered to be of poor quality if, either it is too large, or it is a *skinny triangle*.

A triangle is *skinny* if it contains extremely small or large angles i.e. it is too thin. Miller *et al* [24] have shown that the most natural and elegant measure for analyzing Delaunay mesh quality is the *aspect ratio*, β, of the simplex defined as the circum-radius *r* to shortest edge *d* ratio of the simplex.

$$\beta = \frac{r}{d}$$

It is desirable to have meshes with this ratio as small as possible and Delaunay refinement algorithms naturally optimize this metric. In 2-dimensions, the minimum angle α of the triangle increases with decreasing value of aspect ratio as:

$$\beta = \frac{1}{2\sin\alpha}$$

Thus, any triangle having aspect ratio greater than the maximum allowed aspect ratio, β_{max}, is classified as *skinny triangle*, as shown in Figure 2.16. A value of $\beta_{max} = \sqrt{2}$ has been used which ensures $\alpha_{min} = 20.7°$.

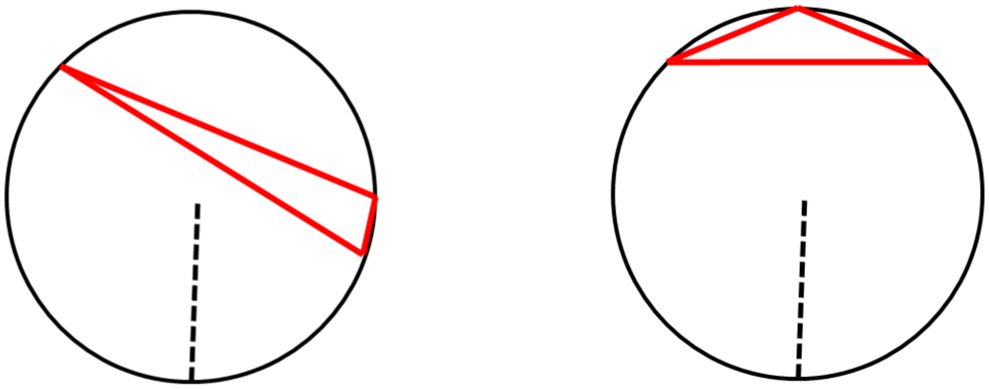


Figure 2.16 - Skinny triangles having large aspect ratio

A triangle is poor if:

1. Its aspect ratio is greater than the maximum allowed aspect ratio i.e. it is a skinny triangle.
2. Size of any of the sides exceeds the maximum mesh size allowed.

The choice of the two quality parameters is a tradeoff between speed, accuracy and memory costs. No characteristic rays may pass through a skinny triangle, as shown in Figure 2.17(a), or a small displacement in the streaming path might bring a large change in track segment length, as shown in Figure 2.17(b), which can cause large error, especially in case of strongly absorbing media. Thus, the maximum aspect ratio must depend upon the minimum angle requirement and desired mesh density and guarantee of termination of the refinement process.

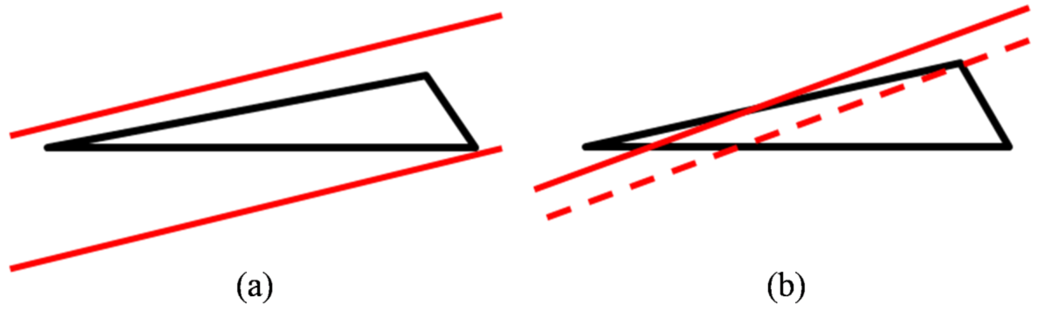


Figure 2.17 - (a) No characteristic rays passing through a skinny triangle (b) Large change in length of track segment formed in a skinny triangle due to small displacement in position of streaming path

Similarly, validity of the flat flux and flat source approximations may be compromised if the mesh size is too large which can in turn lead to an error. Maximum mesh size for each material type should be such that the approximations remain valid along with having minimum number of meshes possible. Thus, maximum mesh size should be a function of the sum of all macroscopic cross-sections of all reactions in which neutrons are either consumed or generated by nuclei of the mesh material, primarily, capture and fission. This can be determined from the nuclear data associated with material of the mesh.

Despite the different nature of the two metrics on which mesh quality is decided, the manner in which the quality is improved is the same for both cases. To obtain meshes of better quality, the poor quality triangles are *split*. Under this process, a new vertex is inserted at the circum-center of the poor quality triangle. The insertion of the new vertex requires re-triangulation to remove all triangles, whose Delaunay property is violated as a result. The modified Bowyer-Watson algorithm, developed for initial triangulation, is again utilized for re-triangulation. Mesh refinement is undertaken in the subroutine *improve_triangulation* which takes the initial triangulation as input and gives the final, good quality Delaunay triangulation as output. The mesh refinement algorithm is as given below:

STEP 1 – Read nuclear data from file.
STEP 2 – Calculate maximum mesh size for each region/material.
STEP 3 – For each triangle in triangulation list,
STEP 4 – If triangle is skinny; aspect ratio is greater than maximum aspect ratio,
STEP 5 – Then, *split triangle*
STEP 6 – If triangle size exceeds maximum mesh size,
STEP 7 – Then, *split triangle*
STEP 8 – Repeat steps 4 to 7 till all triangles satisfy desired aspect ratio and size criteria

Maximum Mesh Size

STEP 1 – Determine absorption cross-section for each material in each group

$$\Sigma_{a,g}^{mat} = \Sigma_{t,g}^{mat} + \sum_{g'=1}^{G} \Sigma_{s,g\to g}^{mat}$$

(Assuming Total cross-section = Absorption cross-section + Scattering cross-section)

STEP 2 – Determine maximum cross-section for each material among all groups
STEP 3 – Determine maximum mesh size for each material as reciprocal of the maximum cross-section, as calculated in step 2

Split Triangle

STEP 1 – Insert new node at circum-center of the triangle
STEP 2 – Determine node type of new node

STEP 3 – Update mesh nodes list with new vertex and number of mesh nodes

STEP 4 – Re-triangulate using modified Bowyer – Watson algorithm

At the end of mesh refinement, final Delaunay triangulation is obtained after all poor quality triangles have been eliminated. Figure 2.18 gives an illustration of the split triangle algorithm, respectively. Figure 2.19 provides an illustration of the *improve_triangulation* i.e. mesh refinement algorithm.

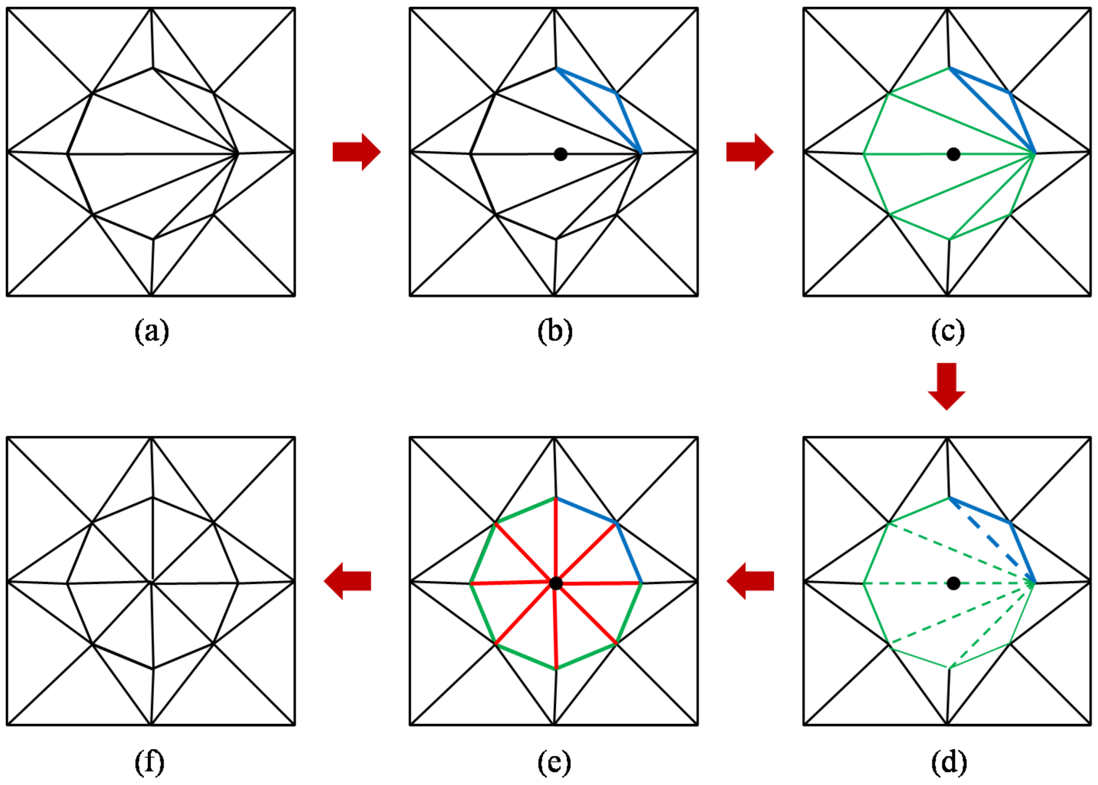


Figure 2.18 - Illustration of split triangle algorithm (a) Existing triangulation (b) Poor quality triangle (in blue) identified to be split (c) New vertex (in black) to be inserted at circum-centre of poor quality triangle (in blue). All triangles that become "bad" (in green) due to insertion of new vertex (d) Polygonal cavity formed by all contiguous bad triangles (e) Reconnection of new vertex with edges of polygonal hole (f) New refined triangulation

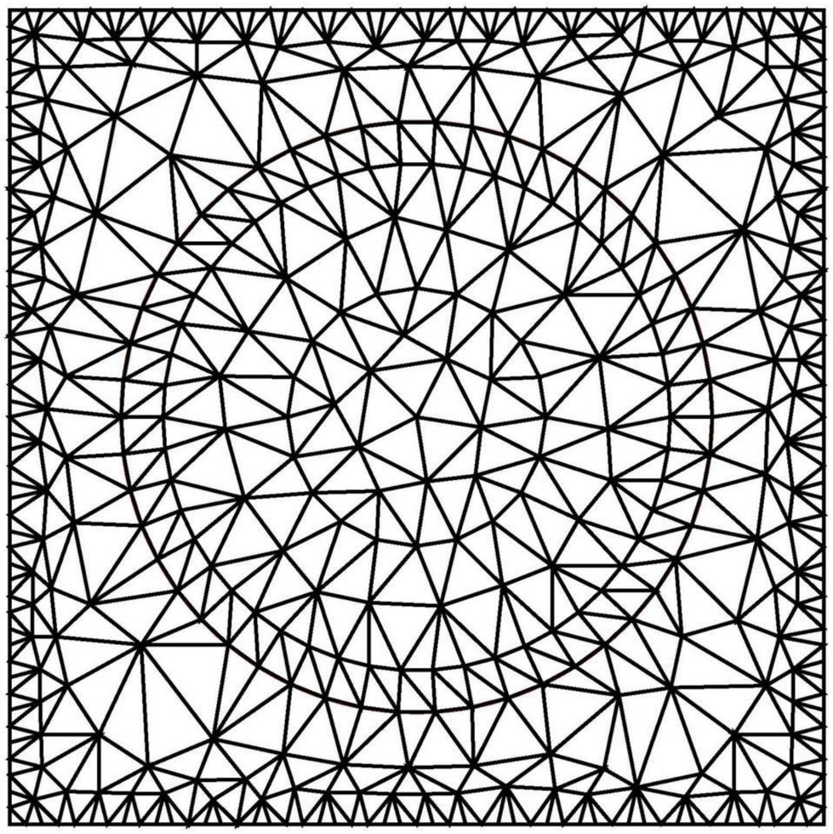


Figure 2.19 - Improvement of mesh quality through implementation of mesh refinement algorithm

CHAPTER 3

RAY TRACING

MOC technique of solving neutron transport equation for lattice level calculation involves solving for neutron flux along each characteristic neutron streaming path. The complete process of inserting a desired number of parallel rays, in every desired direction so as to appropriately model the neutron motion in 3-D space, and determining and storing the intersection made by each characteristic path with different meshes is referred to as *ray tracing*. Thus, after the completion of ray tracing, complete information about the number, length and order of *track segments* and the mesh in which each track segment lies in the *x-y* plane of the problem is available. Information about all the characteristic rays passing through each mesh is also obtained which is required during the flux and multiplication factor calculations.

Insertion of rays, determining the initial triangle in which the ray enters, and also the subsequent triangles in which the ray proceeds requires the knowledge of the boundary triangles as well as neighbors of each mesh. Also, since the region of interest is only bounded in 2-dimensions, and extends infinitely along the z-axis, rays must be traced only in the azimuthal plane of the problem and can be subsequently raised out of the plane of the problem into different polar directions.

3.1 DIRECTION QUADRATURE

The neutrons always travel in three-dimensional space irrespective of the dimensionality of the problem being considered. The dimensionality of the problem is determined by the number of dimensions in which material boundaries exist; i.e. in case the material region is bounded only along one axis, then the problem is one dimensional, for existence of boundaries along two coordinate axes, the problem becomes two dimensional and so on. However, the neutrons always have motion in all the three dimensions regardless of the existence of material boundaries.

Thus, in order to model neutron transport, characteristic streaming paths must be considered in all possible directions in 3D space (i.e. in all 4π directions) and calculation of angular flux is done along each streaming path. However, since only a finite number of directions can be considered during calculations to approximate the complete problem domain, these should be chosen in such a manner so as to appropriately model neutron motion. This is done by choosing a set of desired number of azimuthal angles and similarly, a set of polar angles, which together determine each unique direction of motion, as represented in Figure 3.1.

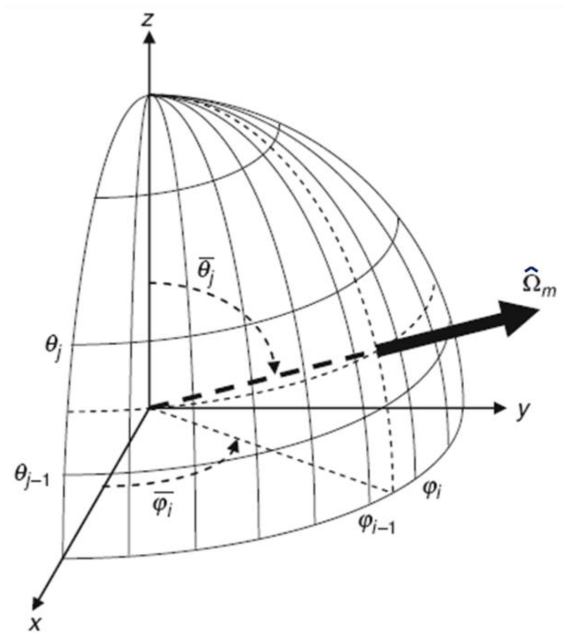


Figure 3.1 - Typical quadrature ball

3.1.1 Azimuthal Angles

Generally, the azimuthal angles in the method of characteristics are evenly spaced in the *xy*-plane of the problem. The boundaries of motion, thus, occur at intervals given by

$$\Delta\varphi = \frac{2\pi}{I} \tag{3.1}$$

where,

I	=	Total number of azimuthal directions of motion

The azimuthal direction of motion lies along the center of two neighboring boundaries. Thus, azimuthal direction is obtained as

$$\bar{\varphi}_i = \frac{1}{2}(\varphi_{i-1} + \varphi_i) \tag{3.2}$$

All neutrons that travel at angles between the two neighboring boundaries are represented by the associated azimuthal direction of motion. This is accounted for by the weight associated with each direction of motion which is given as,

$$\omega_i = \frac{2\pi}{I} \tag{3.3}$$

Figure 3.2 shows the representative azimuthal directions for $I = 4$ case.

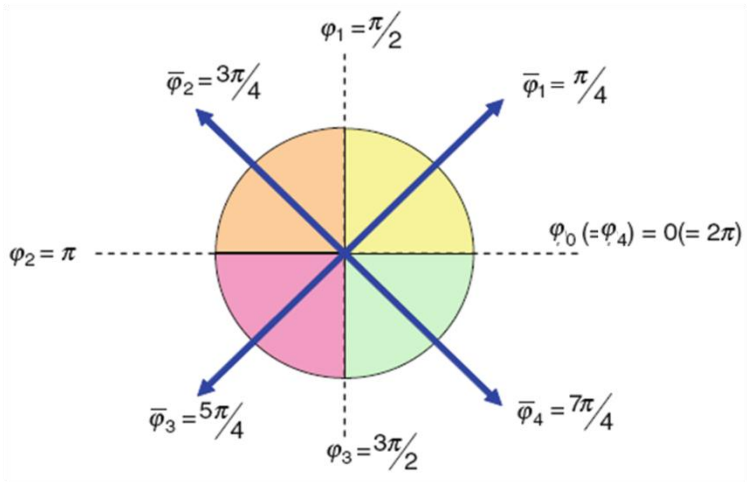

$\varphi_1 = \pi/2$
$\bar{\varphi}_2 = 3\pi/4$
$\bar{\varphi}_1 = \pi/4$
$\varphi_0 \, (=\varphi_4) = 0(= 2\pi)$
$\varphi_2 = \pi$
$\bar{\varphi}_3 = 5\pi/4$
$\bar{\varphi}_4 = 7\pi/4$
$\varphi_3 = 3\pi/2$

Figure 3.2 - Azimuthal directions for $I = 4$

3.1.2 Polar Angles

Polar angles of motion can be chosen in a variety of ways ranging from simple but not-so accurate to complex but more accurate methods. Some of the methods available are

3.1.2.1 Equal Weights Quadrature Set

In this method, the polar directions are chosen such that the weights associated are equal in all directions. More emphasis is laid on neutrons streaming at angles close to the plane of the problem. The weights can be calculated from the differential area of a unit sphere subtended by all boundaries for a given direction. The differential area on a unit sphere with an arbitrary direction $\hat{\Omega}_m$ is

$$dA = \hat{\Omega}_m^2 sin\theta_m d\theta d\varphi \tag{3.4}$$

where

θ = Polar angle associated with $\hat{\Omega}_m$

φ = Azimuthal angle associated with $\hat{\Omega}_m$

$d\theta, d\varphi$ are the differential spreads in polar and azimuthal directions respectively

For a unit sphere, $|\hat{\Omega}| = 1$, and thus

$$dA = sin\theta_m . d\theta . d\varphi \tag{3.5}$$

The total area subtended on the unit sphere associated with direction $\hat{\Omega}_m$ then is

$$A = \int_{\theta_{j-1}}^{\theta_j} \int_{\varphi_{i-1}}^{\varphi_i} dA = \int_{\theta_{j-1}}^{\theta_j} \int_{\varphi_{i-1}}^{\varphi_i} sin\bar{\theta} . d\theta . d\varphi = (\varphi_i - \varphi_{i-1})(\cos\theta_{j-1} - \cos\theta_j) \tag{3.6}$$

Since spacing between azimuthal angles, $\Delta\varphi = \varphi_i - \varphi_{i-1}$, is equal and equal areas on unit sphere require

$$A = \frac{4\pi}{I.J} \tag{3.7}$$

where, *I* and *J* are the number of azimuthal and polar angles respectively.

Thus,

$$\left(\cos\theta_{j-1} - \cos\theta_j\right) = \frac{4\pi}{I.J.\Delta\varphi} = \frac{4\pi}{I.J.2\pi/I}$$

$$\theta_j = \cos^{-1}\left(\cos\theta_{j-1} - \frac{2}{J}\right) \; and \; \theta_0 = 0 \tag{3.8}$$

Since the polar direction of motion should pass through the centroid of the surface area subtended by the polar boundaries, they are given as

$$\bar{\theta}_j = \cos^{-1}\left\{\frac{1}{2}\left(\cos\theta_j + \cos\theta_{j-1}\right)\right\} \tag{3.9}$$

The corresponding associated weights are the differential areas on the surface of the unit sphere created by the polar boundaries

$$\omega_j = \cos\theta_j - \cos\theta_{j-1} \tag{3.10}$$

3.1.2.2 Equal Angles Quadrature Set

In the equal angles quadrature, the angles are distributed equally in the polar directions. Equal emphasis is laid on neutrons irrespective of the directions that they stream in. The polar boundaries of motion, thus, occur at intervals given as

$$\Delta\theta = \frac{\pi}{J}$$

(3.11)

where, the polar boundaries extend from $\theta_0 = 0$ to $\theta_J = \pi$; J being the total number of polar directions of motion

The polar direction of motion lies along the center of two neighboring boundaries. Thus, polar direction is obtained as

$$\bar{\theta}_j = \cos^{-1}\left\{\frac{1}{2}\left(\cos\theta_j + \cos\theta_{j-1}\right)\right\}$$

(3.12)

Also, the associated weights can be easily calculated as

$$\omega_j = \cos\theta_j - \cos\theta_{j-1}$$

(3.13)

Both of the quadrature sets are perfectly acceptable although the number of polar angles might differ to obtain the same amount of accuracy. Equal angles quadrature set is preferable for three-dimensional transport equation due to its symmetric treatment in all directions where as equal weights quadrature set is more preferable for one and two-dimensional equation as the angles are weighted towards the plane of the problem.

3.1.2.3 Legendre Quadrature

Legendre and Gaussian quadratures provide an even better distribution of angles for one- and two-dimensional problems. Thus, the polar directions are taken as an input from the user along with the number of polar directions required. A 16-point double angular quadrature set (DP$_7$), symmetric about the azimuthal plane of the problem, has been used for better modeling neutron motion out of the azimuthal plane in various polar directions [25]. Table 3.1 gives the DP_7 quadrature.

Table 3.1 - 16-point double angular quadrature set (DP_7) [25]

COSINE OF POLAR ANGLE $(\cos\theta_j)$	DIRECTIONAL WEIGHT (w_j)
.98014493	0.05061427
.89833324	0.11119052
.76276620	0.15685332
.59171732	0.18134189
.40828268	0.18134189
.23723380	0.15685332
.10166676	0.11119052
.01985507	0.05061427

3.2 SINGULARITIES

Any ray may enter or exit a triangle either across one of its edges or through one of its vertices. Also, the ray may or may not have the same slope as one of the triangle edges. Thus, the possible ways of passage of a ray through a mesh have been given in Table 3.2 and illustrated through Figure 3.3.

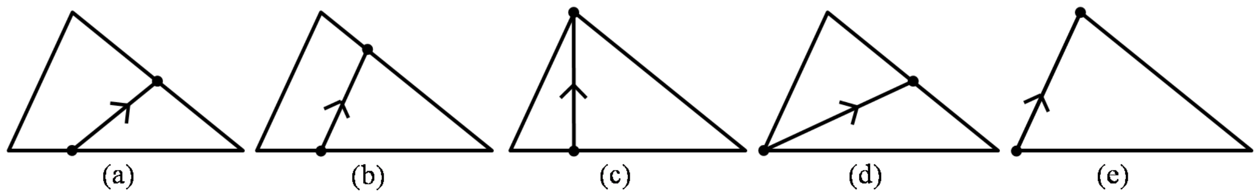


Figure 3.3 - Possible ways of passage of a characteristic ray through a mesh (a) Case I, (b) Case II, (c) Case III, (d) Case IV and (e) Case V

Table 3.2 - Possible ways of passage of a characteristic ray through a mesh

CASE	RAY ENTRY	RAY EXIT	PARALLEL
I	Across Edge	Across Edge	No
II	Across Edge	Across Edge	Yes
III	Across Edge	Through Vertex	No
IV	Through Vertex	Across Edge	No
V	Through Vertex	Through Vertex	Yes

However, whenever a ray passes through a vertex i.e. case III and case IV, an ambiguity arises which is referred to as a *vertex-based singularity* and special consideration is required as multiple meshes may share a vertex. In the special case when a ray lies along an edge, case V, an *edge-based singularity* occurs and it is even more difficult to handle. In both cases the ray can be associated with multiple meshes and thus, multiple material properties.

Vertex-based singularity is treated by associating the ray with the mesh to which it proceeds. The triangle to which the ray proceeds can be identified by determining whether the angle, θ_0, between the two triangle edges which form the vertex is equal to the sum of angles, θ_1 and θ_2, made by the ray with these edges, where φ, φ_1 and φ_2 are the angles of the ray and the 2 edges w.r.t. X-axis, as shown in Figure 3.4. Values of θ_0, θ_1 and θ_2 can be calculated as,

$$\theta_0 = |\varphi_1 - \varphi_2|$$
$$\theta_1 = |\varphi - \varphi_1|$$
$$\theta_2 = |\varphi - \varphi_2|$$

Ray proceeds to the mesh which satisfies the condition, $\theta_0 = \theta_1 + \theta_2$.

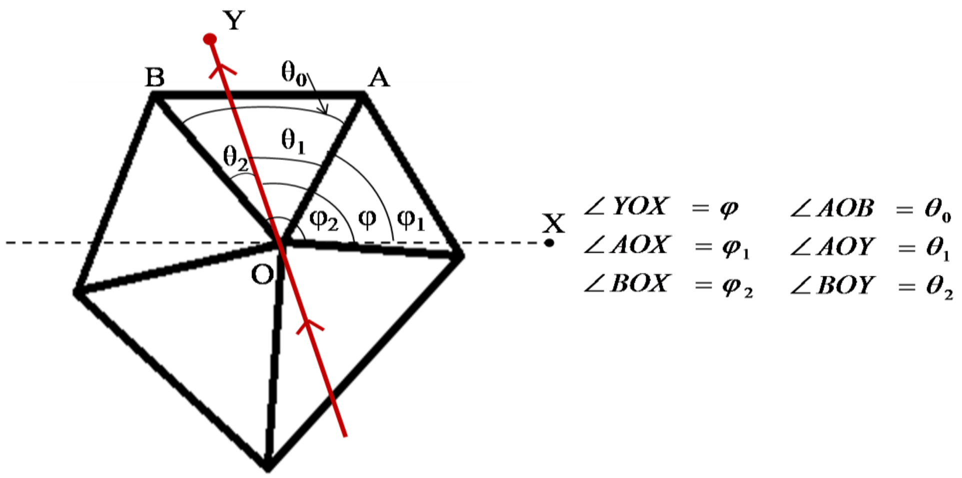


Figure 3.4 – Identification of mesh to which a ray proceeds after passing through the common vertex

As far as edge-based singularities are concerned, if the edge along which the ray passes is shared between triangles having the same material composition, then the ray is associated to either of the two. However, when the material composition of the two triangles differs, the ray is associated to the mesh belonging to the inner regions of the problem geometry.

3.3 NEIGHBORHOOD AND BOUNDARY TRIANGLES

Neighborhood of each mesh is identified based on the *vertex criterion*, according to which, any two meshes which share at least one vertex classify as neighbors. Vertex criterion has been employed because a ray may exit a mesh and enter into the neighboring mesh through common vertex, and not necessarily pass from one mesh to another across its edge. The algorithm for determining mesh neighborhood is as follows:

STEP 1 – For each triangle
STEP 2 – For each vertex of the triangle
STEP 3 – If vertex is also a vertex of the triangle whose neighborhood is being determined
STEP 4 – Then, add triangle to neighbor list
STEP 5 – Repeat steps 2 to 4 till all neighbors have been identified

Figure 3.5 shows the vertex neighbors of red colored triangle using this algorithm.

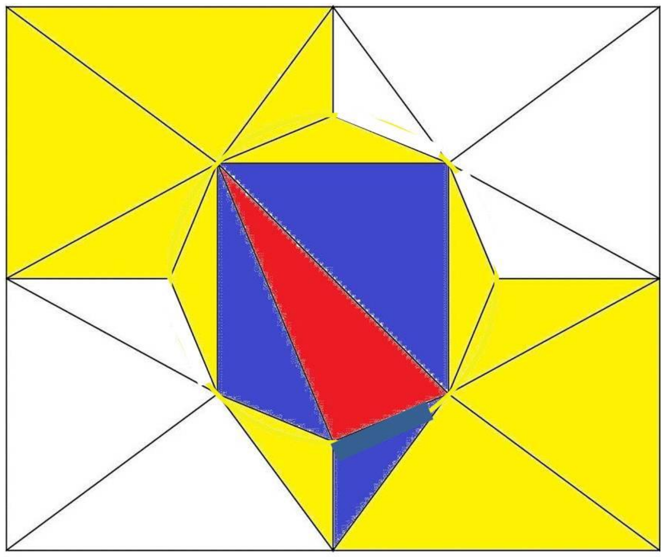


Figure 3.5 - Neighboring triangles of triangle in red. All triangles in yellow share only 1 vertex with the red triangle; all blue triangles share 2 vertices with the red triangle

All triangles lying on the boundary of the pin cell are the boundary triangles. As in case of mesh neighbors, any triangle having at least one vertex on the boundary of region of interest is classified as a boundary triangle. Vertex criterion has been employed because a ray may enter the problem domain through either a vertex located on the boundary or through a triangle edge lying on the boundary. The algorithm for determining mesh neighborhood is as follows:

```
STEP 1 – For each triangle
STEP 2 – For each vertex of the triangle
STEP 3 – If vertex is a corner vertex of the region of interest
STEP 4 – Then, add triangle to both boundary lists on which the corner is located and EXIT
STEP 5 – If vertex is located on one of the boundaries of the region of interest but is not a
         corner vertex
STEP 6 – Then, add triangle to respective boundary list on which it is located and EXIT
STEP 7 – Repeat steps 2 to 6 till all boundary triangles have been identified
```

Figure 3.6 shows a sample illustration of this algorithm.

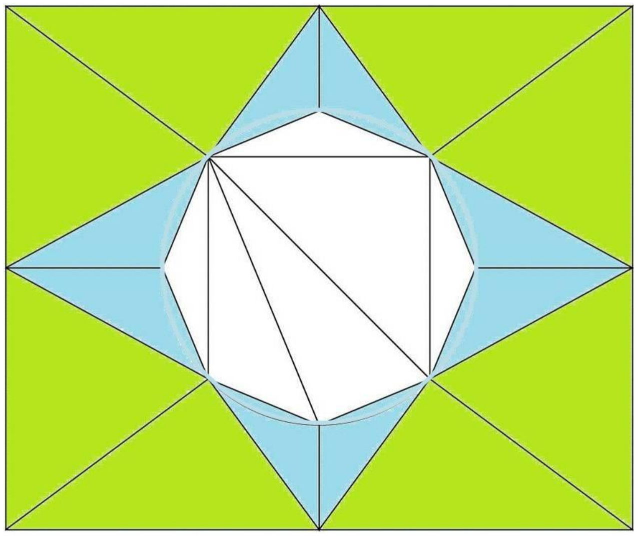


Figure 3.6 - Boundary triangles. All blue triangles have 1 vertex located on the boundary; all green triangles have more than 1 vertices located on the boundary

3.4 INTER – RAY SPACING AND ALIGNMENT

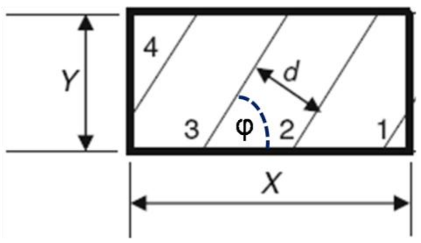


Figure 3.7 – Inter–ray spacing

Since multiple parallel characteristic rays must be inserted in every azimuthal direction and spread uniformly throughout the region of interest, the inter-ray spacing is a function of azimuthal direction and the desired number of parallel rays 'N' per direction. The inter-ray spacing, d, in every direction, φ, as shown in Figure 3.7, is obtained as:

$$d = \frac{X.\sin\varphi + Y.\cos\varphi}{N}$$

where,

X	=	Horizontal size of the system
Y	=	Vertical size of the system

However, the end of each incident characteristic ray must be aligned precisely to the beginning of its reflective counterpart to ensure perfect reflection under reflective boundary conditions. To achieve this, track separation and azimuthal angle must be altered appropriately based on the overall dimensions of the problem. This process of alignment illustrated in Figure 3.8, is achieved using the following set of equations:

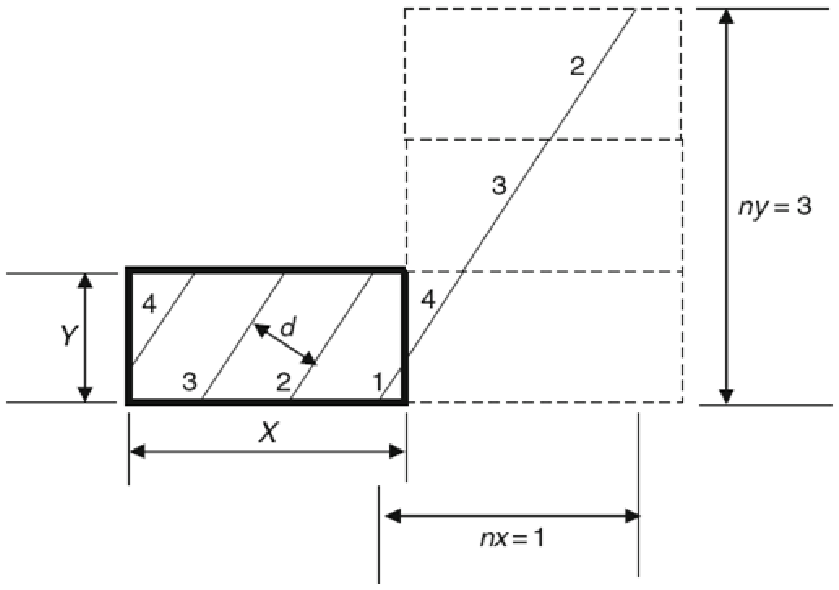


Figure 3.8 - Alignment of rays with their reflective counterpart

Total number of horizontal system lengths scanned, n_x, and vertical system lengths scanned, n_y, is given, respectively, as:

$$n_x = \frac{Y.\cos\varphi}{d}$$

$$n_y = \frac{X.\sin\varphi}{d}$$

For the streaming paths to align to their reflective counterparts at the boundaries, n_x and n_y must be integers. This can be done by rounding off n_x and n_y, as obtained above, to the nearest integers. Thus,

$$n'_x = round - off\ (n_x)$$

$$n'_y = round - off\ (n_y)$$

This causes a slight adjustment in the angle of streaming and the updated azimuthal angle, *φ'*, is calculated as:

$$\varphi' = \tan^{-1}\left(\frac{n'_y \, . Y}{n'_x \, . X}\right) \tag{3.14}$$

The new adjusted inter-ray spacing is thus, obtained as:

$$d' = \sqrt{\left(\frac{Y}{n'_x}\right)^2 + \left(\frac{X}{n'_y}\right)^2} \tag{3.15}$$

The entire process of ray alignment is repeated for each azimuthal angle. Using the corresponding updated track separations and azimuthal angles, parallel, equidistant tracks, which are perfectly aligned to their reflective counterparts at the boundaries, are laid across *x*-*y* plane of the problem.

3.5 RAY DISTRIBUTION

The process of ray distribution involves determining the initial point, c_0, for each parallel ray in a particular direction. This is the first step to be undertaken for laying down streaming paths and tracing each of these through the geometry. If the starting point of any of the parallel rays is known, then the rest can be obtained easily by shifting it appropriately along the concerned boundary.

A simple choice for the initial point of first ray in a particular direction is a point located at a distance equal to half inter-ray spacing along the corresponding boundary away from the corner of the region. However, such a choice may cause the following problems:

1. Rays near the corners at certain angles may create tracks of very small lengths
2. Number of rays actually inserted, for some angles, may be less than the desired number as a few rays do not intersect the problem geometry at all.

3. Rays at certain angles may not be uniformly spread across the region of interest as a large portion may remain vacant.

In order to ensure that the rays are uniformly spread through the azimuthal plane of the problem, a more prudent choice is to determine the distance of starting point of first ray from the corner as a function of azimuthal angle of the ray rather than fixing it as half the separation along the concerned axis. In accordance with the given argument, ray distribution is started along the vertical boundaries and the distance of initial point of the first ray from the corner is determined as the *spacing fraction* times the inter-ray separation along vertical boundary, where, *spacing fraction* is defined as:

$$Spacing\ fraction = \begin{cases} \dfrac{\left|{}^{\pi}/_{2} - \varphi\right|}{{}^{\pi}/_{2}}, & \forall \quad \varphi \in (0, \pi) \\ \dfrac{\left|{}^{3\pi}/_{2} - \varphi\right|}{{}^{\pi}/_{2}}, & \forall \quad \varphi \in (\pi, 2\pi) \end{cases} \tag{3.16}$$

Once the initial point of the first ray is known, the initial point of all the remaining rays can be determined using size of projection of inter-ray spacing along suitable boundary. Initial point is found along the vertical boundary till the point remains within the extent of the boundary. When the initial point surpasses the boundary extremities, the intersection of this ray with apposite horizontal boundary becomes initial point of this ray. The process is now repeated, but, along the horizontal boundary using inter-ray separation along horizontal boundary, till desired number of rays has been distributed. The algorithm of ray distribution is as follows:

STEP 1 – Determine inter-ray spacing along vertical and horizontal boundaries

STEP 2 – Determine spacing fraction

STEP 3 – Determine c_0 for first ray on vertical boundary

STEP 4 – Determine c_0 for next ray on vertical boundary

STEP 5 – If c_0 lies on boundary

STEP 6 – Then, repeat steps 4 and 5

STEP 7 – Else, find intersection of this ray on horizontal boundary

STEP 8 – Initial point of ray, c_0, is equal to intersection point

STEP 9 – Determine c_0 for next ray on horizontal boundary

STEP 10 – Repeat step 9 till desired number of rays has been distributed

3.6 INITIAL MESH IDENTIFICATION

After the ray distribution is completed in each azimuthal direction, i.e. initial points of all rays have been determined, the next step is to use this information coupled with the list of boundary triangles to identify the initial mesh in which the ray enters the region of interest. The initial point, c_0, could be either a vertex or any other point located on the boundary. In the former case, technique for identifying mesh to which a ray proceeds when it passes through a vertex, as discussed in Section 3.2, is employed for every triangle on suitable boundary till the initial mesh is successfully identified. To identify the initial triangle in the latter case, a check is made on each triangle on the concerned boundary about whether or not c_0 lies on its edges. The algorithm is as follows:

STEP 1 – Identify boundary on which initial point, c_0, is located

STEP 2 – For each triangle on the appropriate boundary (identified in step 1)

STEP 3 – Check whether c_0 is a vertex of the triangle or not

STEP 4 – Address each case appropriately

STEP 5 – Repeat steps 3 and 4 till initial mesh determined

Initial point, c_0, is a triangle vertex

STEP 1 – Determine the 2 triangle edges containing c_0

STEP 2 – Determine angle of inclination of the 2 edges w.r.t. +ve X-axis, φ_1 and φ_2

STEP 3 – Determine angle between the 2 edges containing c_0, θ_0

STEP 4 – Determine angle between ray and the 2 edges, θ_1 and θ_2

STEP 5 – If $\theta_0 = \theta_1 + \theta_2$, then current triangle is the required initial mesh

Initial point, c_0, is a not a triangle vertex

```
STEP 1 – Determine edges of triangle
STEP 2 – For each edge of triangle
STEP 3 – Check if edge lies on boundary
STEP 4 – Repeat step 3 till boundary edge identified
STEP 5 – If boundary edge identified
STEP 6 – Then, if initial point, $c_0$, lies on the boundary edge
STEP 7 – Then, current triangle is the required initial mesh
```

3.7 TRACE RAY

Once the initial point, c_0, and initial mesh of a ray have been identified, the process of identifying successive track segments formed by a ray is undertaken. This involves determining the points of intersection made by the ray with successive meshes through which it passes. The knowledge of in-point of a track segment and the mesh in which it lies is used to determine the next intersection point, i.e. out-point of the segment. After determining out-point of the segment, the next triangle along the path of the ray is ascertained which can be done by determining the neighboring triangle containing edge on which out-point is located. However, to determine the mesh to which the ray proceeds when out-point is a triangle vertex, the method for handling vertex-based singularities as described in Section 3.2 has been adopted. Out-point for current track segment and its next triangle so determined become the in-point and mesh, respectively, for next track segment. These steps are repeated till the ray egresses the region of interest. The algorithm for trace ray process is given below

```
STEP 1 – Assign initial point, $c_0$, and initial mesh as in-point and triangle for track segment 1
STEP 2 – Determine out-point for track segment
STEP 3 – Check whether out-point is a triangle vertex or not
STEP 4 – Determine next triangle to which ray proceeds, appropriately
STEP 5 – Add segment to track segment list of ray
STEP 6 – Update in-point for next track segment with out-point of current segment
STEP 7 – Update triangle for next track segment with next triangle of current segment
STEP 8 – Repeat steps 2 to 7 till the ray exits the region of interest at the opposite boundary
```

Determining out-point

```
STEP 1 – Formulate equation describing ray from its azimuthal angle and in-point
STEP 2 – Determine edges of triangle in which track segment lies
STEP 3 – For each edge
STEP 4 – Formulate equation describing edge
STEP 5 – If ray is not parallel to edge
STEP 6 – Then, determine intersection point of the two lines
STEP 7 – If intersection point lies between the edge vertices i.e. within the range of edge
STEP 8 – Then, out-point = intersection point
STEP 9 – If ray is parallel to edge
STEP 10 – If in-point is also an end point of the edge
STEP 11 – Then, out-point = other end point of edge
```

Out-point is a triangle vertex

```
STEP 1 – For each neighboring triangle of mesh in which track segment lies
STEP 2 – Determine the 2 triangle edges containing out-point
STEP 3 – If 2 such edges do not exist, move to next neighbor
STEP 4 – Determine angle of inclination of the 2 edges w.r.t. +ve X-axis, $\varphi_1$ and $\varphi_2$
STEP 5 – Determine angle between the 2 edges containing out-point, $\theta_0$
STEP 6 – Determine angle between ray and the 2 edges, $\theta_1$ and $\theta_2$
STEP 7 – If $\theta_0 = \theta_1 + \theta_2$, then current neighboring triangle is the required mesh to which
         ray proceeds next
```

Out-point is not a triangle vertex

STEP 1 – For each neighboring triangle of mesh in which track segment lies
STEP 2 – Determine edges of triangle
STEP 3 – For each edge of triangle
STEP 4 – If out-point lies between the edge vertices i.e. lies on the edge
STEP 5 – Then, then current neighboring triangle is the required mesh to which ray proceeds next

3.8 REFLECTED RAYS

After establishing the path made by every ray, i.e. finding and storing the list of successive track segments made by each ray during its passage through the problem under consideration, the process of ray tracing is essentially complete. However, another step, although not necessary, is undertaken. The reflective counterpart of each ray is identified and stored. This information is useful during the flux and multiplication factor calculations since the ray to which the final angular neutron flux along a given ray is transferred under reflective boundary conditions is known *apriori*. To determine the reflected ray for each incident ray, a simple yet effective scheme is used in which the azimuthal angle of the possible reflected ray is first calculated followed by verifying if the initial point, c_0, of the reflected ray is same as the point at which the incident ray exits the geometry, as shown in Figure 3.9.

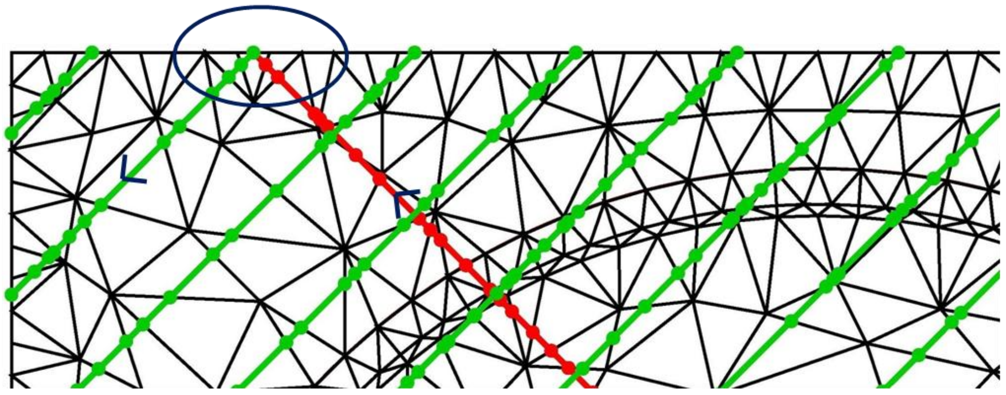


Figure 3.9 - Identification of reflective counterpart of an incident ray

For any incident ray with azimuthal angle φ_{inc}, crossing a square pin cell, the region of interest, the azimuthal angle of its reflective counterpart φ_{ref}, can either be given as $|\pi - \varphi_{inc}|$ or $2\pi - \varphi_{inc}$. This is because for reflection off left or right boundaries, $\varphi_{ref} = |\pi - \varphi_{inc}|$ and for ray reflection off top or bottom boundaries, $\varphi_{ref} = 2\pi - \varphi_{inc}$. Table 3.3 lists all the possible cases for ray reflection in a square pin cell and the possible combinations of φ_{inc} and φ_{ref} in the form of a reflection matrix.

Table 3.3 – Reflection matrix for φ_{inc} and φ_{ref}

AZIMUTHAL ANGLE OF INCIDENT RAY (φ_{inc})		REFLECTING BOUNDARY			
QUADRANT	RANGE	BOTTOM	RIGHT	TOP	LEFT
I	$0\ to\ {}^{\pi}/_{2}$	Not applicable	I $(\pi - \varphi_{inc})$	II $(2\pi - \varphi_{inc})$	Not applicable
II	${}^{\pi}/_{2}\ to\ \pi$	Not applicable	Not applicable	III $(2\pi - \varphi_{inc})$	I $(\pi - \varphi_{inc})$
III	$\pi\ to\ {}^{3\pi}/_{2}$	II $(2\pi - \varphi_{inc})$	Not applicable	Not applicable	IV $(\varphi_{inc} - \pi)$
IV	${}^{3\pi}/_{2}\ to\ 2\pi$	I $(2\pi - \varphi_{inc})$	III $(\varphi_{inc} - \pi)$	Not applicable	Not applicable

Hence, φ_{inc} belonging to a particular quadrant, the corresponding φ_{ref} can only belong to one of the two neighboring quadrants and never lie in the diagonally opposite quadrant of the azimuthal plane. Thus, any ray on reflection at the boundaries of a square geometry may only have a reflective counterpart belonging to either of its neighboring quadrants. This is represented in Table 3.4.

Table 3.4 - Relationship between quadrants of φ_{inc} and φ_{ref}

QUADRANT OF AZIMUTHAL ANGLE OF INCIDENT RAY (φ_{inc})	QUADRANT OF AZIMUTHAL ANGLE OF INCIDENT RAY (φ_{ref})			
	I	II	III	IV
I	No	Yes	No	Yes
II	Yes	No	Yes	No
III	No	Yes	No	Yes
IV	Yes	No	Yes	No

The algorithm for determining the reflective counterpart of each ray, designed on the basis of this reflection matrix, is as follows:

STEP 1 – Determine possible azimuthal angles for reflected ray
STEP 2 – For each characteristic ray
STEP 3 – If out-point of last track segment of incident ray is same as initial point c_0, of the ray
STEP 4 – If azimuthal angle of ray is equal to the possible azimuthal angles of reflective counterpart, as determined in step 1
STEP 5 – Then, current ray is the required reflective counterpart and EXIT

3.9 ALGORITHM

The final algorithm for ray tracing involves laying down the desired number of rays in each azimuthal direction. Once rays have been spread uniformly throughout the problem domain, an ordered list of intersection points and thus, track segments made by each of these is stored which is further used during flux and multiplication factor calculations. In addition to the basic procedure, the reflective counterpart of each ray is also identified and stored.

The steps of ray tracing algorithm are given below:

STEP 1 – Determine neighbors of each mesh
STEP 2 – Identify all triangles lying on different boundaries of the problem geometry
STEP 3 – Determine azimuthal directions in which *xy*-plane is divided
STEP 4 – Calculate inter-ray spacing for rays of each direction and adjust it to ensure perfect alignment of all rays to reflective counterparts
STEP 5 – Generate and distribute the desired number of rays in each azimuthal direction
STEP 6 – For each characteristic ray
STEP 7 – Identify initial mesh
STEP 8 – Determine trace list i.e. trace ray

STEP 9 – Determine reflected counterpart

Figure 3.10 represents an implementation of the ray tracing algorithm for a Delaunay triangulated region of interest.

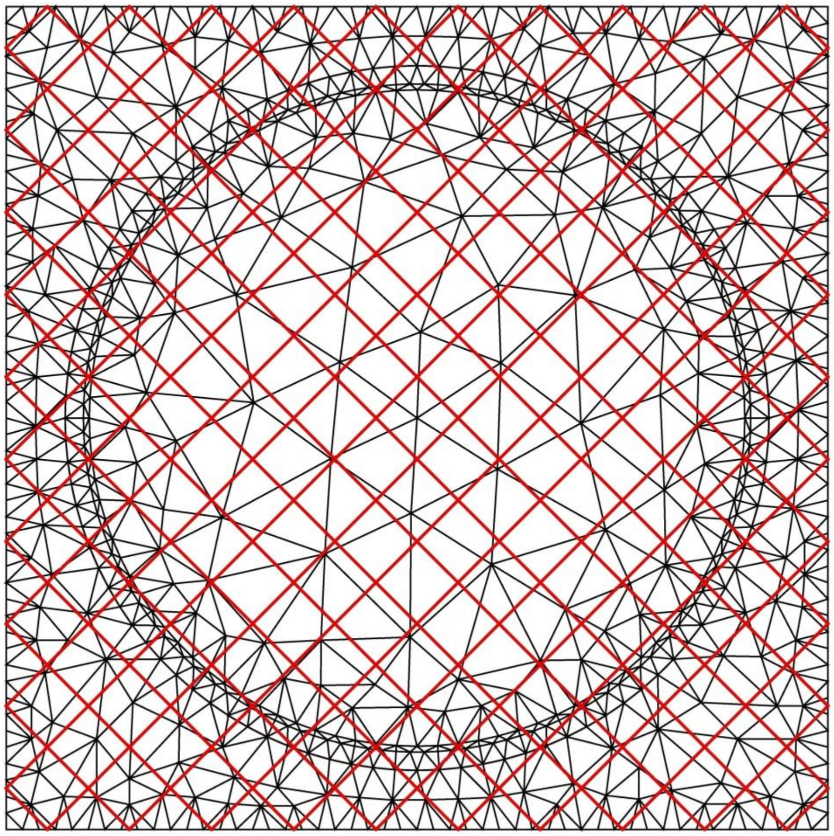


Figure 3.10 - Ray tracing in 4 azimuthal directions with 20 parallel rays per direction

CHAPTER 4

FLUX AND MULTIPLICATION FACTOR CALCULATIONS

4.1 SOLUTION TO THE CHARACTERISTIC EQUATION

Essentially, the problem is to obtain solution of the following equation for each mesh,

$$\Phi_{out} = \Phi_{in} e^{-\Sigma_{tr}\tau} + \frac{q}{4\pi\Sigma_{tr}}\left(1 - e^{-\Sigma_{tr}\tau}\right) \tag{4.1}$$

where,

Φ_{out}	=	Outgoing angular flux for the mesh
Φ_{in}	=	Incoming angular flux for the mesh
τ	=	Length of track segment formed by the characteristic ray in the mesh
Σ_{tr}	=	Transport corrected macroscopic cross section

The above equation is solved under the following assumptions:

1. All scattering can be modeled as isotropic and is transport corrected.
2. All sources are inherently isotropic.
3. Flat flux approximation, i.e., angular and, hence, scalar flux is constant within a mesh.
4. Flat source approximation, i.e., scalar neutron source is constant within a mesh.

The first term on the RHS represents the number of neutrons that stream across the mesh without undergoing a collision; and the second term on the RHS is the number of neutrons that are picked up along the track from scatterings and sources as track passes through the mesh. The sum of these two terms gives the number of neutrons that exit the mesh at the end of the track represented by the LHS.

In order to obtain the angular flux value exiting the mesh along the track, Φ_{out}, only angular flux entering the mesh at the beginning of the track, Φ_{in}, and total source term within the mesh, q, is further required since transport corrected cross-sections are known beforehand while the track length, τ, can be readily calculated by raising its projection in the azimuthal plane, τ', which has already been calculated in the ray tracing routine, in the desired polar direction using,

$$\tau = \frac{\tau'}{\sin\theta_j} \tag{4.2}$$

where,

θ_j = j^{th} polar direction

4.2 FLUX AND MUTIPLICATION FACTOR INITIALIZATION

Initial flux values are required due to the need of Φ_{in} and q values in order to solve the characteristics equation. An initial guess of scalar flux in every mesh, starting angular flux along each ray and multiplication factor is a pre-requisite to beginning the iterative solution process. So, a seed value of $\phi = 1$ is set for every energy group, in every mesh and thus, $\Phi_{in} = {}^{1}/_{4\pi}$ in all groups, in all directions along the problem geometry. Also, the initial value of multiplication factor is set to $k = 1$

4.3 SOURCE TERM CALCULATION

The total angular source term comprises of three separate sources:

1. Scattering source

$$Q_s(\vec{r}, E, \hat{\Omega}) = \int_0^\infty dE' \int_0^{4\pi} d\widehat{\Omega}'\, \Sigma_s(\vec{r}, E' \rightarrow E, \widehat{\Omega}' \rightarrow \hat{\Omega}) \Phi(\vec{r}, E', \widehat{\Omega}') \tag{4.3}$$

2. Fission source

$$Q_f\left(\vec{r}, E, \hat{\Omega}\right) = \chi(\vec{r}, E) \int_0^\infty \nu\Sigma_f\left(\vec{r}, E', \hat{\Omega}\right)\phi(\vec{r}, E')\, dE'$$

(4.4)

3. Fixed external source

$$Q_{ext}\left(\vec{r}, E, \hat{\Omega}\right) = S\left(\vec{r}, E, \hat{\Omega}\right)$$

(4.5)

These must be calculated prior to solving the characteristic equation to plug in the values of total scalar source term, q.

4.3.1 Scattering Source

All scattering is assumed to be isotropic and effects of anisotropic scattering are accounted for by transport-correcting the self scattering and total cross-sections. Thus,

$$\Sigma_s\left(\vec{r}, E' \to E, \widehat{\Omega'} \to \hat{\Omega}\right) = \frac{1}{4\pi}\Sigma_s(\vec{r}, E' \to E)$$

(4.6)

and scattering source term is reduced to

$$Q_s\left(\vec{r}, E, \hat{\Omega}\right) = \frac{1}{4\pi}\int_0^\infty \Sigma_s(\vec{r}, E' \to E)\phi(\vec{r}, E')\, dE'$$

(4.7)

Since the entire energy range is discretized into energy groups,

$$Q_{s,g} = \frac{1}{4\pi}\sum_{g'} \Sigma_{s,g' \to g} . \phi_{g'}$$

(4.8)

where,

$\Sigma_{s,g' \to g} . \phi_{g'}$	=	Total number of neutrons in energy group g', which get scattered into energy group g, and only 1/4π of these are travelling in direction Ω.

4.3.2 Fission Source

Neutron generation due to fission is assumed to be an isotropic process, and thus, the fission source term becomes

$$Q_f\left(\vec{r}, E, \hat{\Omega}\right) = \chi(\vec{r}, E) \int_0^\infty \frac{\nu\Sigma_f(\vec{r}, E')}{4\pi} \phi(\vec{r}, E')\, dE'$$

(4.9)

In discretized multi-group form,

$$Q_{f,g} = \frac{\chi_g}{4\pi} \sum_{g'} \nu\Sigma_{f,g'} . \phi_{g'}$$

(4.10)

4.3.3 External Source

Isotropic distribution of neutrons due to an external source with energy E, located at position $\vec{r}$, with total strength $s(\vec{r}, E)$, is obtained as

$$Q_{ext}\left(\vec{r}, E, \hat{\Omega}\right) = \frac{q(\vec{r}, E)}{4\pi}$$

(4.11)

In multi-group form,

$$Q_{ext,g} = \frac{s_g}{4\pi}$$

(4.12)

Source term calculation is repeated over each mesh, n, which gives respective source terms as follows:

Scattering Source,

$$Q_{s,n,g} = \frac{1}{4\pi} \sum_{g'} \Sigma_{s,n,g' \to g} . \phi_{g',n}$$

(4.13)

Fission Source,

$$Q_{f,n,g} = \frac{\chi_{n,g}}{4\pi} \sum_{g'} \nu\Sigma_{f,n,g'} . \phi_{g',n} = \sum_{g'} \frac{\Sigma_{p,n,g'\to g}}{4\pi} . \phi_{g',n}$$

(4.14)

where,

$$\Sigma_{p,n,g'\to g} = \chi_{n,g} . \nu\Sigma_{f,n,g'}$$

(4.15)

External Source,

$$Q_{ext,n,g} = \frac{s_{n,g}}{4\pi}$$

(4.16)

However, external source is usually not considered.

4.3.4 Total Source for Multiplying System

In a multiplying system, external sources may or may not be present. The total source term, in presence of external source, is given as

$$q_{n,g} = q_{s,n,g} + q_{f,n,g} + q_{ext,n,g} = \sum_{g'} \left(\Sigma_{s,n,g'\to g} + \frac{\Sigma_{p,n,g'\to g}}{k_\infty} \right) . \phi_{g',n} + s_{n,g}$$

(4.17)

where,

k_∞	=	Infinite multiplication factor

In absence of an external source, the total scalar source can be expressed as

$$q_{n,g} = q_{s,n,g} + q_{f,n,g} = \sum_{g'} \left(\Sigma_{s,n,g'\to g} + \frac{\Sigma_{p,n,g'\to g}}{k_\infty} \right) . \phi_{g',n}$$

(4.18)

Infinite multiplication factor is the ratio of the total number of neutrons born through fission to the total number of neutrons absorbed in the system.

$$k_\infty = \frac{\sum_n \sum_g \nu\Sigma_{f,n,g}\phi_{g,n}V_n}{\sum_n \sum_g \Sigma_{a,n,g}\phi_{g,n}V_n}$$

(4.19)

where,

V_n = Volume of nth mesh

Thus, the characteristic transport equation, equation (4.1), may now be solved as all required quantities have been calculated.

4.4 BOUNDARY CONDITIONS

Like any other differential equation, boundary conditions must be applied on the characteristic transport equation to obtain the exact solution. In the method of characteristics, boundary conditions are applied to the angular flux only. There are three types of boundary conditions available for any system in a reactor core. The choice of which boundary condition is applied to the system is made on the basis of the physical environment in its neighborhood. The boundary conditions can be broadly classified as: periodic, reflective and vacuum.

4.4.1 Periodic Boundary Condition

Periodic boundary condition is applied when the physical system is infinite and exhibits a periodic structure due to which, the angular flux itself is spatially periodic. As a result, angular flux for the complete system can be determined by solving the transport equation for an equivalent problem on a single lattice cell which is periodically repeated.

Periodic boundary condition, is expressed mathematically as,

$$\Phi^g_{in,n'}(\bar{\varphi}_i, \bar{\theta}_j) = \Phi^g_{out,n}(\bar{\varphi}_i, \bar{\theta}_j)$$

(4.20)

Thus, periodic boundary condition require the value of angular flux in group g; in the direction $\hat{\Omega} \equiv \left(\bar{\varphi}_i, \bar{\theta}_j\right)$, leaving a system across some boundary from mesh n to be equal to the value of angular flux in group g; in the same direction $\hat{\Omega} \equiv \left(\bar{\varphi}_i, \bar{\theta}_j\right)$, entering at the opposite system boundary in mesh n', as illustrated in Figure 4.1.

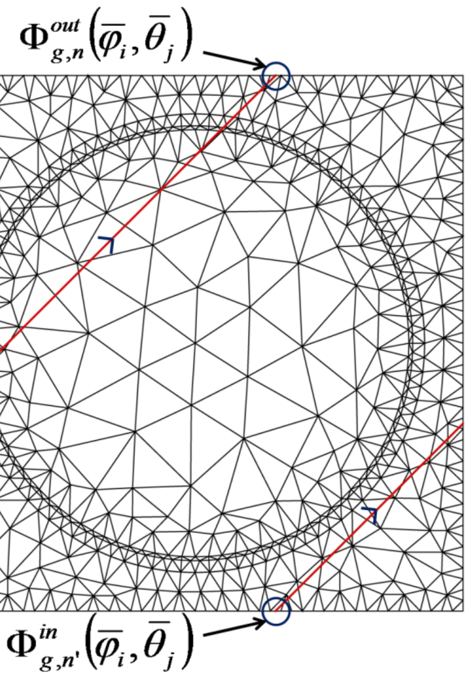

$\Phi^{out}_{g,n}\left(\bar{\varphi}_i, \bar{\theta}_j\right)$
$\Phi^{in}_{g,n'}\left(\bar{\varphi}_i, \bar{\theta}_j\right)$

Figure 4.1 - Periodic boundary condition

4.4.2 Reflective Boundary Condition

In case, the physical problem possesses planar symmetry, solution can be obtained for only a sub region of the system using reflective boundary condition to determine the flux profile of the entire system. For each plane of symmetry that exists, the size of the sub region is half the size of its parent, which greatly improves the efficiency of simulation.

Mathematically, reflective boundary condition is expressed as,

$$\Phi^{g}_{in,n}\left(\bar{\varphi}'_i, \bar{\theta}'_j\right) = \alpha \, . \, \Phi^{g}_{out,n}\left(\bar{\varphi}_i, \bar{\theta}_j\right) \tag{4.21}$$

Thus, reflective boundary condition requires a fraction α, of the value of angular flux in group g; hitting the boundary of the system along the characteristic path travelling in the direction $\hat{\Omega} \equiv (\bar{\varphi}_i, \bar{\theta}_j)$ to be reflected along its companion characteristic along the direction $\hat{\Omega}' \equiv (\bar{\varphi}'_i, \bar{\theta}'_j).(\bar{\varphi}'_i, \bar{\theta}'_j)$. For the problem of interest, $\hat{\Omega}'$ is obtained from $\hat{\Omega}$ by substituting $\bar{\varphi}'_i = |\pi - \bar{\varphi}_i|$ or $\bar{\varphi}'_i = 2\pi - \bar{\varphi}_i$, as shown in Figure 4.2. Also, reflection being a planar phenomenon, $\bar{\theta}'_j = \bar{\theta}_j$ and thus, no change in polar direction takes place between the incident and the reflected rays. The primes on the azimuthal and polar angles represent the reflective counterparts to the incident angles. Perfectly reflecting boundaries occur in the special case with $\alpha = 1$ while $\alpha = 0$ represents the perfect vacuum condition where no reflection takes place.

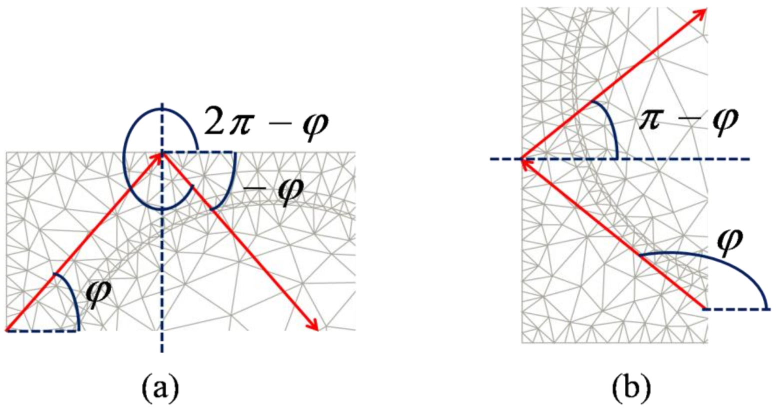


Figure 4.2 - Reflective boundary condition. Change in direction when reflection occurs off (a) top (or bottom) boundaries (b) left (or right) boundaries

4.5 TRACK ANGULAR FLUX CALCULATION

Calculation of angular flux along a track segment involves determining the value of both, the outgoing Φ_{out} and average Φ_{avg} angular flux along the track segment. Figure 4.3 shows the significance of both the quantities. Outgoing angular flux can be easily determined using equation (1.9) as discussed in Section 1.2.1 once the incoming angular flux along a track segment is known, while Φ_{avg} can be calculated using Φ_{out} and Φ_{in}. The expressions used are,

$$\Phi^{out}_{g,dir,r,n} = \Phi^{in}_{g,dir,r,n} . e^{-\Sigma_{tr,n,g}\tau_{dir,r,n}} + \frac{Q_{g,n}}{\Sigma_{tr,n,g}}(1 - e^{-\Sigma_{tr,n,g}\tau_{dir,r,n}}) \tag{4.22}$$

$$\Phi^{avg}_{g,dir,r,n} = \frac{Q_{g,n}}{\Sigma_{tr,n,g}} + \frac{\Phi^{in}_{g,dir,r,n} - \Phi^{out}_{g,dir,r,n}}{\Sigma_{tr,n,g} \cdot \tau_{dir,r,n}} \tag{4.23}$$

where, *n*, *g*, *dir* and *r* are the indices representing mesh, energy group, direction and parallel characteristic ray in a given direction, respectively.

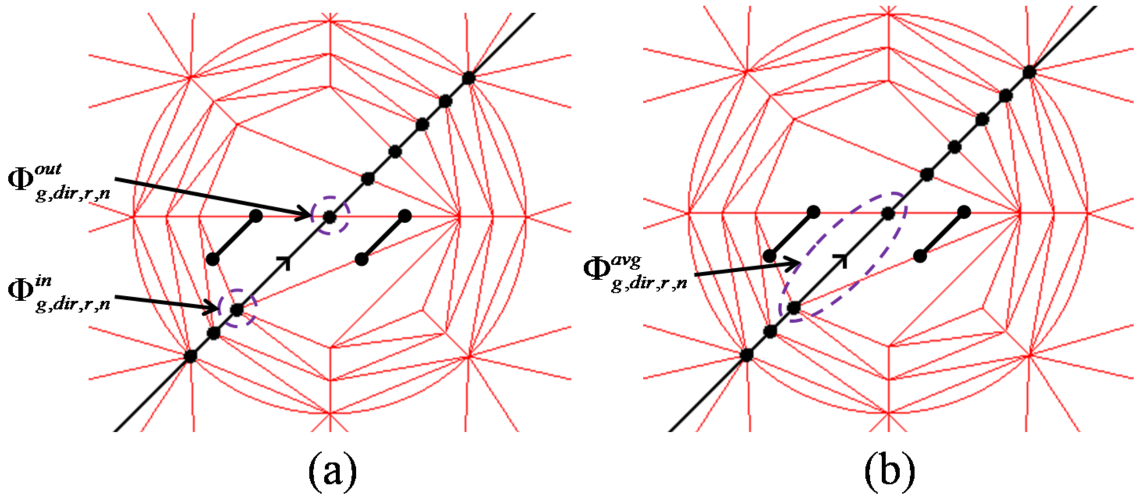
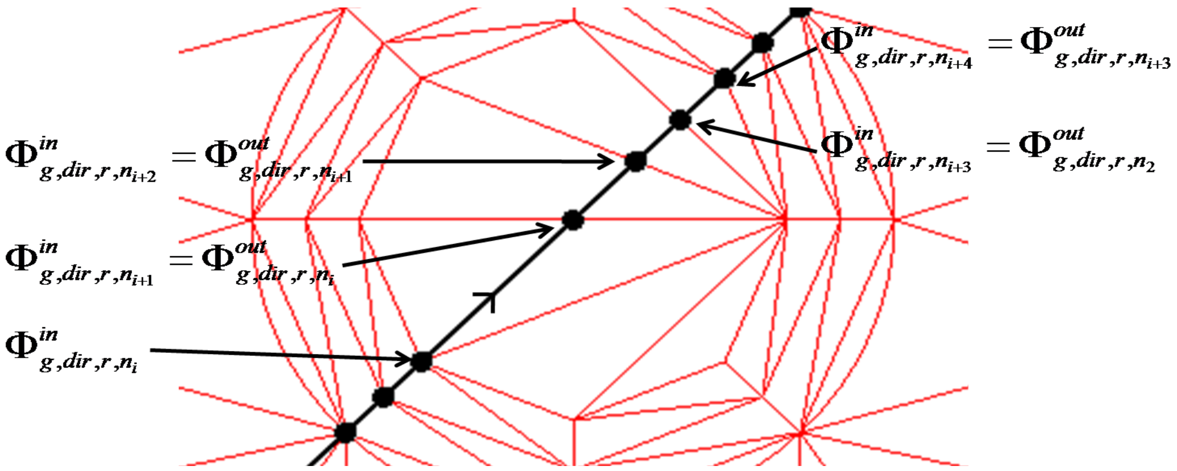


Figure 4.3 - Angular flux along a track segment (Φ_{out}) (a) Outgoing angular flux (b) Average angular flux (Φ_{avg})

This process is repeated for every track segment along a given ray beginning with the incoming angular flux at the point where the ray enters the region of interest to get the desired angular flux values for track segment 1. The outgoing angular flux for track segment 1 then becomes the incoming angular flux for track segment 2 along the characteristic and thus, angular flux is calculated for every track segment until the ray exits the problem domain at the opposite boundary, as depicted in Figure 4.4.

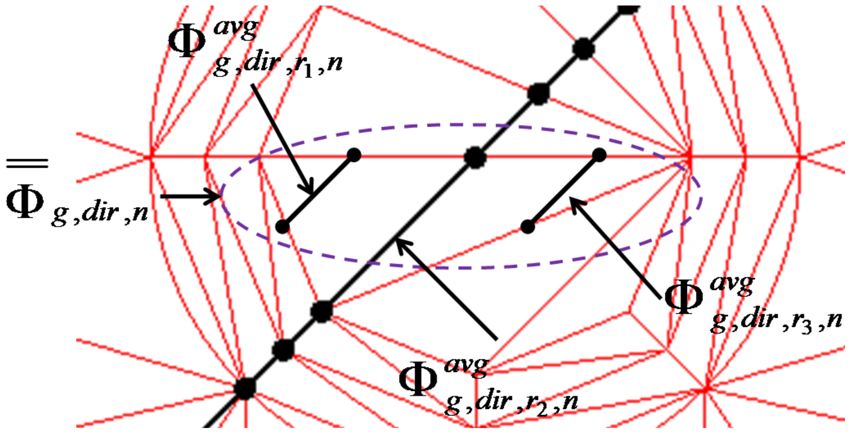


$\Phi^{in}_{g,dir,r,n_{i+4}} = \Phi^{out}_{g,dir,r,n_{i+3}}$
$\Phi^{in}_{g,dir,r,n_{i+2}} = \Phi^{out}_{g,dir,r,n_{i+1}}$
$\Phi^{in}_{g,dir,r,n_{i+3}} = \Phi^{out}_{g,dir,r,n_{2}}$
$\Phi^{in}_{g,dir,r,n_{i+1}} = \Phi^{out}_{g,dir,r,n_{i}}$
$\Phi^{in}_{g,dir,r,n_{i}}$

Figure 4.4 - Angular flux along a characteristic ray

4.6 AVERAGE MESH ANGULAR FLUX

Once the angular flux calculation is completed along all parallel neutron streaming paths in one direction, the average angular flux associated with each mesh, $\overline{\overline{\Phi}}_{g,dir,n}$, in the direction under consideration is determined. The average mesh angular flux is nothing but the weighted average of $\Phi^{avg}_{g,dir,r,n}$, the average angular flux along a track segment, along all rays passing through mesh n in the direction of interest, as shown in Figure 4.5. The area of region represented by a ray forms the corresponding weight.



$\Phi^{avg}_{g,dir,r_1,n}$
$\overline{\overline{\Phi}}_{g,dir,n}$
$\Phi^{avg}_{g,dir,r_3,n}$
$\Phi^{avg}_{g,dir,r_2,n}$

Figure 4.5 - Average angular flux ($\overline{\overline{\Phi}}_{g,dir,n}$) associated with a mesh through which multiple parallel characteristics in a given direction pass

The average mesh angular flux is, thus, given as:

$$\bar{\bar{\Phi}}_{g,dir,n} = \frac{\sum_{r(dir)} \Phi^{avg}_{g,dir,r,n} . \tau_{dir,r,n} . w_{r(dir)}}{\sum_{r(dir)} \tau_{dir,r,n} . w_{r(dir)}} \tag{4.24}$$

where,

$w_{r(dir)}$	=	Width associated with a characteristic *r* in direction *dir*, i.e. inter-ray spacing in direction *dir*
$\tau_{dir,r,n} . w_{r(dir)}$	=	Area of region represented by track segment made by each ray; corresponding weight for respective average angular flux values

4.7 MESH SCALAR FLUX

Mesh scalar flux, $\phi_{g,n}$, for a mesh, is the summation of average mesh angular flux, $\bar{\bar{\Phi}}_{g,dir,n}$, in all the directions, weighted by appropriate directional weight, w_{dir}, for a given mesh. The average mesh angular flux, as obtained previously, is determined in all directions for every mesh and weighted by the associated directional weight. The summation of this quantity in all directions is hence, used to compute the scalar flux. Mathematically,

$$\phi_{g,n} = \sum_{dir} \bar{\bar{\Phi}}_{g,dir,n} . w_{dir} \tag{4.25}$$

where,

w_{dir}	=	Directional weight of direction *dir*

Also,

$$\sum_{dir} w_{dir} = 4\pi \quad and \quad w_{dir} = w_{polar} . w_{azimuthal}$$

where,

w_{polar}	=	Weight of polar component of direction *dir*, or polar weight
$w_{azimuthal}$	=	Weight of azimuthal component of direction *dir*, or azimuthal weight

4.8 MULTIPLICATION FACTOR

After mesh scalar flux, $\phi_{g,n}$, for all meshes in every energy group has been resolved, multiplication factor of the system, k_{eff}, can be determined as per its definition, as given in Section 4.3.4 using equation (4.19). However, another expression can be used to estimate k_{eff} using the value of multiplication factor as calculated in the previous outer iteration. This is as follows:

$$k_{eff}^{new} = k_{eff}^{old} . \frac{\sum_{n,g} \nu\Sigma_{f,n,g} . \phi_{g,n}^{new} . A_n}{\sum_{n,g} \nu\Sigma_{f,n,g} . \phi_{g,n}^{old} . A_n} \tag{4.26}$$

where,

k_{eff}^{old}	=	Multiplication factor of previous outer iteration
$\phi_{g,n}^{new}$	=	Scalar flux at the end of inner convergence loop of current outer iteration
$\phi_{g,n}^{old}$	=	Scalar flux at the end of inner convergence loop of previous outer iteration

4.9 FLUX AND MULTIPLICATION FACTOR ITERATION SCHEME

Flux and multiplication factor calculations are implemented in the form of an inner – outer iteration scheme. The outer iteration involves calculation of the source term for each mesh from the old scalar flux values in each mesh in each group while in the inner loop, the scalar flux value of every mesh is updated using the source term calculated in the current outer iteration. The inner loop is repeated till the scalar flux in every mesh is converged while the outer iteration ends only on meeting desired convergence criteria for multiplication factor. The outer iteration consists of an inner loop for each energy group while to determine updated scalar flux in the inner iteration, outgoing and average angular flux is calculated along every track segment formed by every characteristic ray in every direction. Figure 4.6 gives the flowchart of the inner – outer iteration scheme used for calculating scalar flux and multiplication factor, implemented using the following algorithm.

```
  STEP 1 – Outer convergence loop
    STEP 2 – For each energy group (g = 1, G)
      STEP 3 – Calculate fission source, $Q_{g,n}^{f}$, in each mesh n = 1, N
      STEP 4 – Inner convergence loop
        STEP 5 – Calculate scattering source, $Q_{g,n}^{f}$, in each mesh n = 1, N
        STEP 6 – For each direction (dir)
          STEP 7 – For each parallel ray (r = 1, R)
            STEP 8 – For each track segment on ray (k = 1, K(r))
              STEP 9 – Calculate outgoing angular flux, $\Phi_{g,dir,r,n}^{out}$
              STEP 10 – Calculate average angular flux, $\Phi_{g,dir,r,n}^{avg}$
            End of track segment loop
            STEP 11 – Transfer outgoing angular flux of last track segment to
                      reflective counterpart
          End of parallel rays loop
          STEP 12 – For each mesh (n = 1, N)
            STEP 13 – Calculate average mesh angular flux, $\bar{\bar{\Phi}}_{g,dir,n}$
          End of mesh loop
        End of direction loop
        STEP 14 – For each mesh (n = 1, N)
          STEP 15 – Calculate scalar flux, $\phi_{g,n}$
        End of mesh loop
        STEP 16 – Check scalar flux convergence
      End of inner convergence loop
    End of energy groups loop
    STEP 17 – Calculate multiplication factor
    STEP 18 – Check multiplication factor convergence
End of outer convergence loop
```

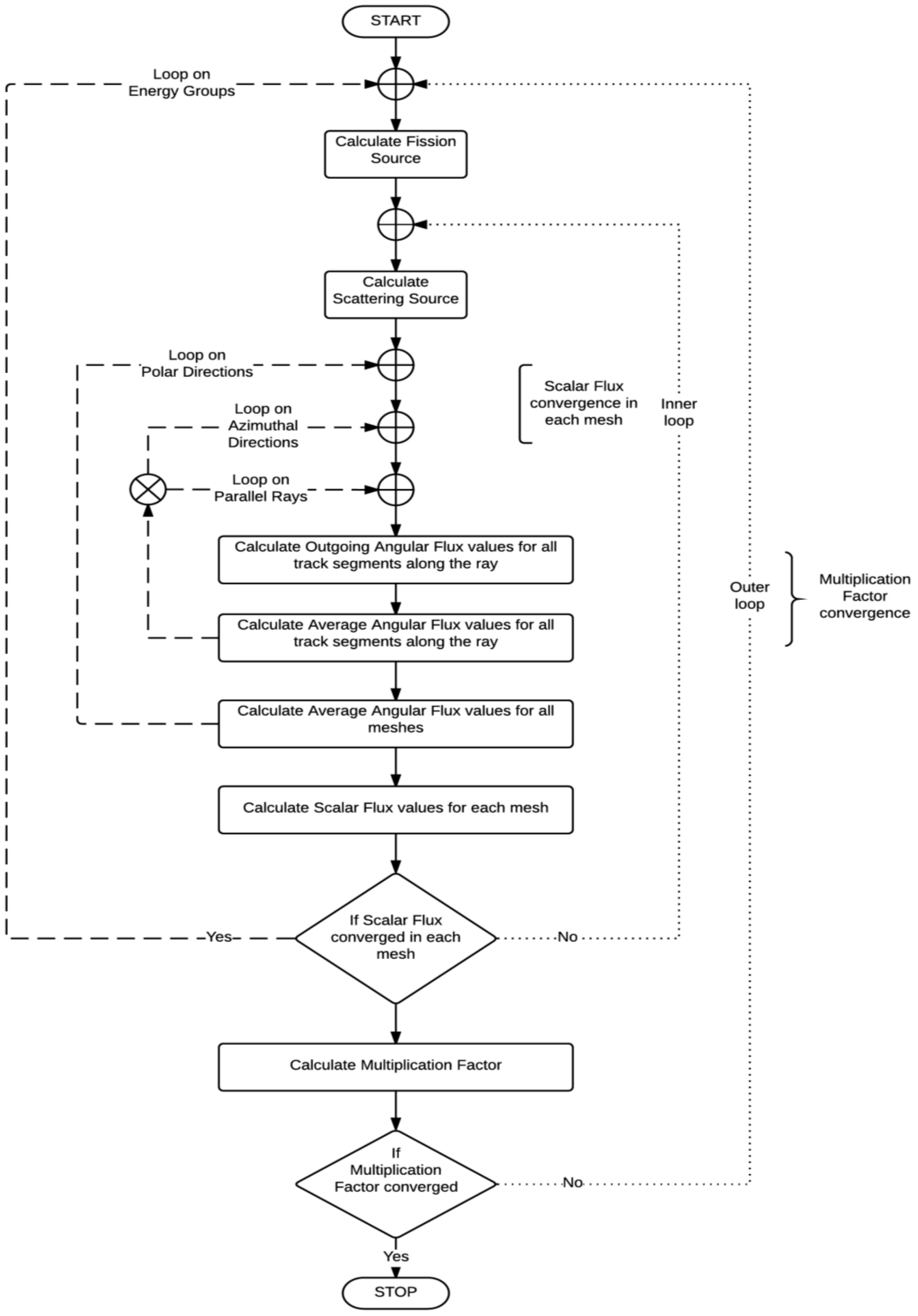

START
Loop on Energy Groups
Calculate Fission Source
Calculate Scattering Source
Loop on Polar Directions
Loop on Azimuthal Directions
Loop on Parallel Rays
Scalar Flux convergence in each mesh
Inner loop
Calculate Outgoing Angular Flux values for all track segments along the ray
Calculate Average Angular Flux values for all track segments along the ray
Outer loop
Multiplication Factor convergence
Calculate Average Angular Flux values for all meshes
Calculate Scalar Flux values for each mesh
If Scalar Flux converged in each mesh
Yes
No
Calculate Multiplication Factor
If Multiplication Factor converged
No
Yes
STOP

Figure 4.6 - Flowchart of inner–outer iteration scheme for scalar flux and multiplication factor calculations

4.10 CONVERGENCE

Since the seed values of inward angular flux along a boundary surface, scalar flux in each mesh and multiplication factor provided to initiate the solution process are merely a guess (which may or may not be correct), the characteristic equation must be iteratively solved until the true solution has been obtained.

A measure of closeness of the solution to the true solution is the convergence of the iterative solution. The difference among the solution obtained at the end of the present iteration to that obtained at the end of the previous iteration can be used to represent the degree of closeness to the true solution. Thus, if this difference, for all inward directions, along the boundaries of the system lies within a certain criterion, ε, the solution is said to have converged. The *convergence criterion* is given as:

$$\left|\frac{New\ Value - Old\ Value}{New\ Value}\right| < \varepsilon$$

4.10.1 Convergence of Scalar Flux

At the end of every inner iteration, convergence of scalar flux is tested for each energy group, g. Scalar flux for each energy group, g, at the beginning of the iteration, $\phi_{g,n}^{old}$, is used to calculate the source term, $Q_{n,g}$. The new scalar flux, $\phi_{g,n}^{new}$, is then obtained as discussed above in Section 4.7 using equation (4.25). The convergence criterion for scalar flux in each mesh is, thus, given as:

$$\left|\frac{\phi_{g,n}^{new} - \phi_{g,n}^{old}}{\phi_{g,n}^{new}}\right| < \varepsilon_{\phi}$$

4.10.2 Convergence of Multiplication Factor

The multiplication factor, k_∞ , for the old flux distribution is calculated at the beginning of every outer iteration. k_∞ converges along with the fission source term which itself takes place as a result of the convergence of scalar flux. The convergence criterion for multiplication is, thus, given as:

$$\left|\frac{k_{eff}\,(new) - \; k_{eff}(old)}{k_{eff}\,(new)}\right| < \; \varepsilon_k$$

Multiplication factor, however, essentially converges if the flux distributions converge.

CHAPTER 5

BENCHMARKS ANALYZED

The quality assurance of any code developed for analysis of reactor systems is mainly achieved by validating it against experimental data, or if such data is not available, against some standard numerical benchmarks. The choice of benchmarks depends on the intended application/s of the code, and must be such that a broad spectrum of problems is considered.

As a part of the validation process, the MOC-based lattice level code developed has been benchmarked for many two energy-group problems and the results have been compared with those obtained from DRAGON V4. The benchmark problems consist of a square pin cell having 3 to 4 regions having varying material compositions and annular fuel geometry, as shown in Figure 1. They include uranium metal, uranium – plutonium MOX, as fuel material; H_2O, D_2O, as moderator and coolant material; different clad material like aluminium, zircalloy; and have varying geometrical size. A total of five square pin cell problems have been analyzed using this code. Sections 5.1 to 5.5 present benchmark problems 1 to 5 respectively.

5.1 BENCHMARK PROBLEM 1

Benchmark problem 1 is a two – energy group problem consisting of three regions, as shown in Figure 5.1. The identification of materials of which the various regions of the problem are composed is not known. However, the dimensional specifications and nuclear data specifications are provided in Table 5.1 and Table 5.2, respectively.

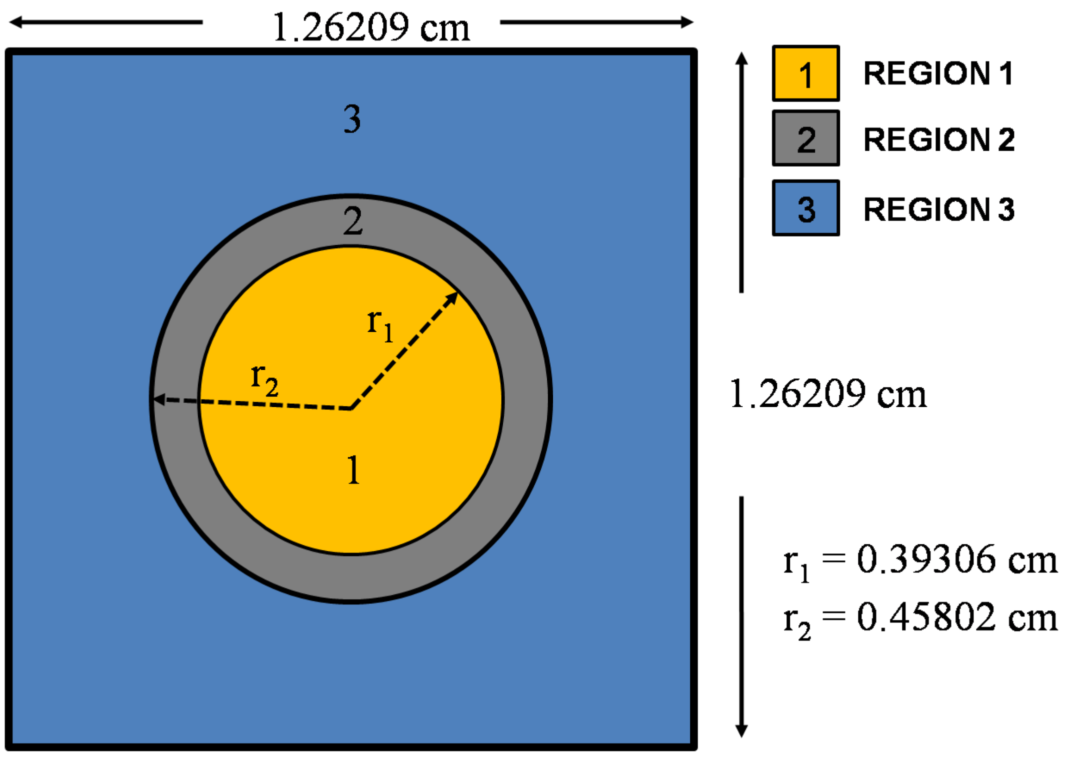


Figure 5.1 - Benchmark Problem 1

Table 5.1 – Dimensional specifications of benchmark problem 1

PARAMETER	DIMENSION (in cm)
Fuel Pellet Radius	0.39306
Cladding Outer Radius	0.45802
Pin Pitch	1.26209

Table 5.2 - Nuclear data specifications of benchmark problem 1

REGION	MATERIAL	ENERGY GROUP (g)	χ_g	$\Sigma_{t,g}$ (in cm^{-1})	$\nu\Sigma_{f,g}$ (in cm^{-1})	$\Sigma_{s,g\rightarrow 1}$ (in cm^{-1})	$\Sigma_{s,g\rightarrow 2}$ (in cm^{-1})
1	1	1	1.0	0.22222	0.0	0.19222	0.02
		2	0.0	0.83333	0.135	0.0	0.75333
2	2	1	0.0	0.16667	0.0	0.12667	0.04
		2	0.0	1.11111	0.0	0.00015	1.10111
3	2	1	0.0	0.16667	0.0	0.12667	0.04
		2	0.0	1.11111	0.0	0.00015	1.10111

5.1.1 Result

The number of triangles formed in every region of the problem as obtained after the initial triangulation and mesh refinement along with the user-specified number of segments on each material boundary are listed in Table 5.3. Table 5.4 provides a comparison of the multiplication factor calculated using this code with respect to the results obtained from DRAGON V4. The number of azimuthal directions and number of rays per direction used have also been mentioned.

Table 5.3 - Number of Delaunay triangles formed for benchmark problem 1

REGION	NO. OF TRIANGLES IN INITIAL TRIANGULATION	NO. OF TRIANGLES IN REFINED TRIANGULATION
1	88	140
2	100	100
3	250	528

BOUNDARY	NO. OF SEGMENTS
Region 1	50
Region 2	50
Region 3 horizontal	50
Region 3 vertical	50

Table 5.4 - Comparison of multiplication factor for benchmark problem 1

Reference k_∞ (k_1)	1.199
Calculated k_∞ (k_2)	1.1983342
Relative Error (in %) $\frac{\lvert k_1 - k_2 \rvert}{k_1} X\ 100$	5.554×10^{-2}
Number of Azimuthal Directions	12
Number of Rays per Direction	100

5.2 BENCHMARK PROBLEM 2

Benchmark problem 2 is also a two – energy group problem consisting of three regions, as shown in Figure 5.2. The fuel material is not known but the clad region consists of *Pt + gap* and the coolant or moderator material is heavy water (D_2O). The dimensional specifications and nuclear data specifications associated with different material regions of the problem are listed in Table 5.5 and Table 5.6, respectively.

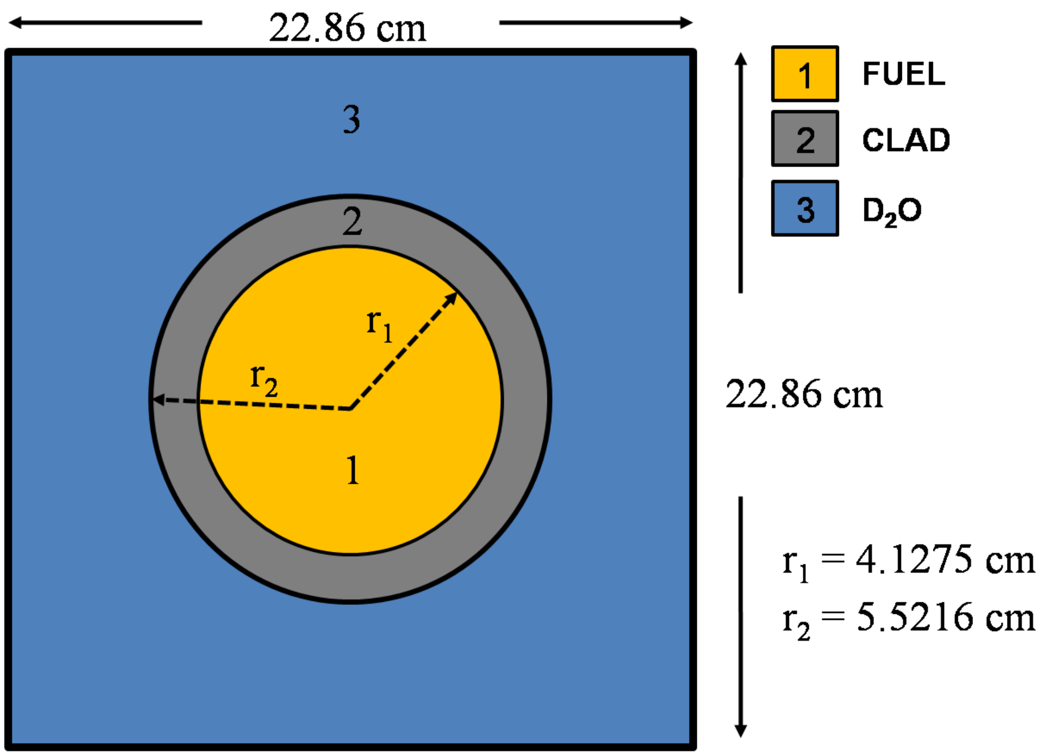


Figure 5.2 - Benchmark Problem 2

Table 5.5 – Dimensional specifications of benchmark problem 2

PARAMETER	DIMENSION (in cm)
Fuel Pellet Radius	4.1275
Cladding Outer Radius	5.5216
Pin Pitch	22.86

Table 5.6 - Nuclear data specifications of benchmark problem 2

REGION	MATERIAL	ENERGY GROUP (g)	χ_g	$\Sigma_{t,g}$ (in cm^{-1})	$\nu\Sigma_{f,g}$ (in cm^{-1})	$\Sigma_{s,g\rightarrow 1}$ (in cm^{-1})	$\Sigma_{s,g\rightarrow 2}$ (in cm^{-1})
1	1	1	1.0	2.8327e-1	5.5988e-3	2.6864e-1	2.3582e-3
		2	0.0	4.4071e-1	9.3182e-2	1.0652e-4	3.6181e-1
2	2	1	0.0	1.0259e-1	0.0	1.0159e-1	1.5817e-4
		2	0.0	1.1418e-1	0.0	3.8957e-5	1.1177e-1
3	3	1	0.0	2.5423e-1	0.0	2.4304e-1	1.1171e-2
		2	0.0	3.8498e-1	0.0	5.4575e-5	3.8451e-1

5.2.1 Result

The number of triangles formed in every region of the problem as obtained after the initial triangulation and mesh refinement along with the user-specified number of segments on each material boundary are listed in Table 5.7. Table 5.8 provides a comparison of the multiplication factor calculated using this code with respect to the results obtained from DRAGON V4. The number of azimuthal directions and number of rays per direction used have also been mentioned.

Table 5.7 - Number of Delaunay triangles formed for benchmark problem 2

REGION	NO. OF TRIANGLES IN INITIAL TRIANGULATION	NO. OF TRIANGLES IN REFINED TRIANGULATION
1	178	378
2	200	336
3	300	802

BOUNDARY	NO. OF SEGMENTS
Fuel Pellet	100
Clad	100
Pin cell horizontal	50
Pin cell vertical	50

Table 5.8 - Comparison of multiplication factor for benchmark problem 2

Reference k_∞ (k_1)	0.95
Calculated k_∞ (k_2)	0.95056271
Relative Error (in %) $\frac{\lvert k_1 - k_2 \rvert}{k_1} X\ 100$	5.9233 x 10^{-2}
Number of Azimuthal Directions	4
Number of Rays per Direction	150

5.3 BENCHMARK PROBLEM 3

Benchmark problem 3 is also a two – energy group problem consisting of three regions, as shown in Figure 5.3. Uranium oxide (UO_2), zirconium (Zr) and light water (H_2O), respectively constitute the fuel (region 1), clad (region 2) and coolant (region 3). The material specifications, dimensional specifications and nuclear data specifications associated with different material regions of the problem are given in Table 5.9, Table 5.10, and Table 5.11, respectively.

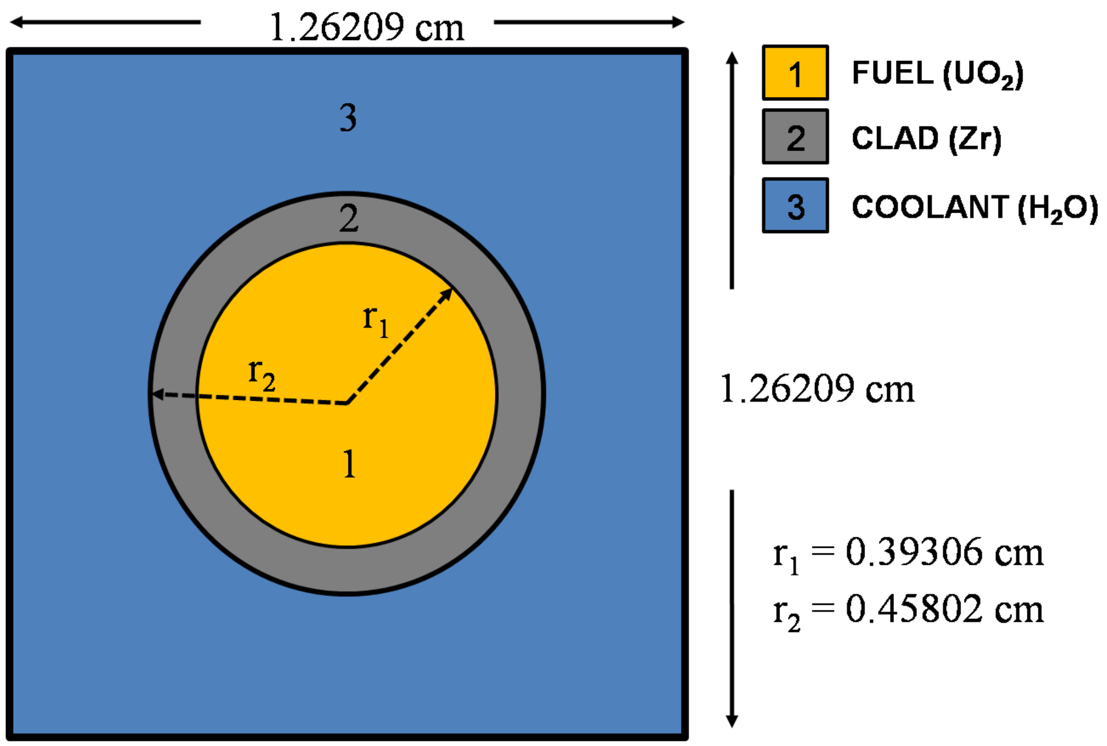


Figure 5.3 - Benchmark Problem 3

Table 5.9 – Material specifications of benchmark problem 3

REGION	MATERIAL COMPOSITION
Fuel	U^{235}, U^{238}, O^{16} (UO_2)
Clad	Zr^{91}
Coolant	H^1, O^{16}, B^{10} (H_2O + Boron)

Table 5.10 - Dimensional specifications of benchmark problem 3

PARAMETER	DIMENSION (in cm)
Fuel Pellet Radius	0.39306
Cladding Outer Radius	0.45802
Pin Pitch	1.26209

Table 5.11 - Nuclear data specifications of benchmark problem 3

REGION	MATERIAL	ENERGY GROUP (g)	χ_g	$\Sigma_{t,g}$ (in cm^{-1})	$\nu\Sigma_{f,g}$ (in cm^{-1})	$\Sigma_{s,g\rightarrow 1}$ (in cm^{-1})	$\Sigma_{s,g\rightarrow 2}$ (in cm^{-1})
1	1	1	1.0	3.7489e-1	1.0928e-2	3.4989e-1	9.1375e-4
		2	0.0	4.8509e-1	1.1999e-1	6.3921e-4	3.9240e-1
2	2	1	0.0	2.4462e-1	0.0	2.4302e-1	3.0198e-4
		2	0.0	2.5044e-1	0.0	3.8882e-4	2.4614e-1
3	3	1	0.0	6.0615e-1	0.0	5.7716e-1	2.8167e-2
		2	0.0	1.7624e0	0.0	1.1758e-3	1.73117e0

5.3.1 Result

The number of triangles formed in every region of the problem as obtained after the initial triangulation and mesh refinement along with the user-specified number of segments on each material boundary are listed in Table 5.12. Table 5.13 provides a comparison of the multiplication factor calculated using this code with respect to the results obtained from DRAGON V4. The number of azimuthal directions and number of rays per direction used have also been mentioned.

Table 5.12 - Number of Delaunay triangles formed for benchmark problem 3

REGION	NO. OF TRIANGLES IN INITIAL TRIANGULATION	NO. OF TRIANGLES IN REFINED TRIANGULATION
1	88	140
2	100	100
3	250	528

BOUNDARY	NO. OF SEGMENTS
Fuel Pellet	50
Clad	50
Pin cell horizontal	50
Pin cell vertical	50

Table 5.13 - Comparison of multiplication factor for benchmark problem 3

Reference k_∞ (k_1)	0.6624341
Calculated k_∞ (k_2)	0.66279911
Relative Error (in %) $\frac{\lvert k_1 - k_2 \rvert}{k_1} X\ 100$	5.5101 x 10^{-2}
Number of Azimuthal Directions	8
Number of Rays per Direction	80

5.4 BENCHMARK PROBLEM 4

Benchmark problem 4 is also a two – energy group problem, but, unlike the previous three cases, is made up of four regions, as shown in Figure 5.4. Uranium (U-metal), aluminium (Zr) and light water (H_2O), respectively constitute the fuel (region 1), clad (region 3) and coolant (region 4). The material specifications, dimensional specifications and nuclear data specifications associated with different material regions of the problem are given in Table 5.14, Table 5.15 and Table 5.16, respectively.

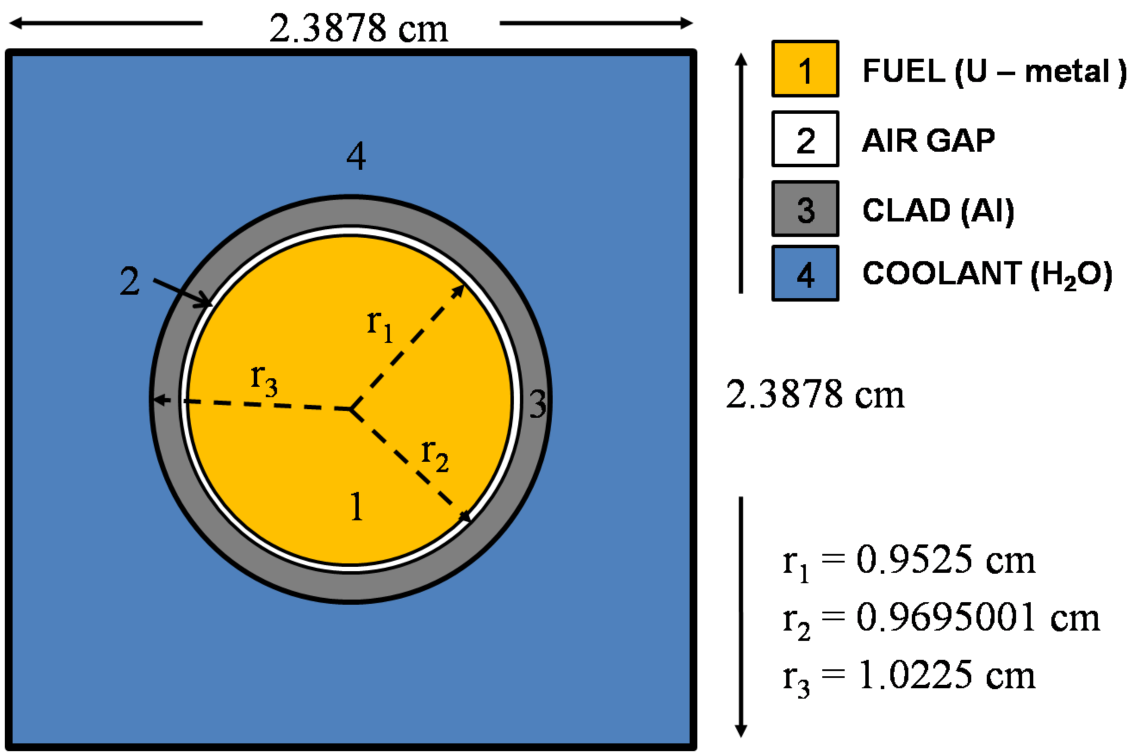


Figure 5.4 - Benchmark Problem 4

Table 5.14 – Material specifications of benchmark problem 4

REGION	MATERIAL COMPOSITION
Fuel	U^{235}, U^{238} (U-metal)
Clad	Al^{27}
Coolant	H^{1}, O^{16} (H_2O)

Table 5.15 - Dimensional specifications of benchmark problem 4

PARAMETER	DIMENSION (in cm)
Fuel Pellet Radius	0.9525
Cladding Inner Radius	0.9695001
Cladding Outer Radius	1.0225
Pin Pitch	2.3878

Table 5.16 - Nuclear data specifications of benchmark problem 4

REGION	MATERIAL	ENERGY GROUP (g)	χ_g	$\Sigma_{t,g}$ (in cm^{-1})	$\nu\Sigma_{f,g}$ (in cm^{-1})	$\Sigma_{s,g\to 1}$ (in cm^{-1})	$\Sigma_{s,g\to 2}$ (in cm^{-1})
1	1	1	1.0	4.3868e-1	2.0767e-2	4.0581e-1	2.2427e-4
		2	0.0	6.7533e-1	3.4309e-1	9.2262e-4	4.3455e-1
2	2	1	0.0	1.0e-8	0.0	1.0e-10	1.0e-10
		2	0.0	1.0e-8	0.0	1.0e-10	1.0e-10
3	3	1	0.0	1.5476e-1	0.0	1.5422e-1	1.8625e-4
		2	0.0	8.8524e-2	0.0	1.3047e-4	7.9783e-2
4	4	1	0.0	9.4568e-1	0.0	9.0755e-1	3.7733e-2
		2	0.0	2.72043e0	0.0	8.8535e-4	2.70547e0

5.4.1 Result

The number of triangles formed in every region of the problem as obtained after the initial triangulation and mesh refinement along with the user-specified number of segments on each material boundary are listed in Table 5.17. Table 5.18 provides a comparison of the multiplication factor calculated using this code with respect to the results obtained from DRAGON V4. The number of azimuthal directions and number of rays per direction used have also been mentioned.

Table 5.17 - Number of Delaunay triangles formed for benchmark problem 4

REGION	NO. OF TRIANGLES IN INITIAL TRIANGULATION	NO. OF TRIANGLES IN REFINED TRIANGULATION
1	98	250
2	250	250
3	250	250
4	300	576

BOUNDARY	NO. OF SEGMENTS
Fuel Pellet	100
Air gap	150
Clad	100
Pin cell horizontal	50
Pin cell vertical	50

Table 5.18 - Comparison of multiplication factor for benchmark problem 4

Reference k_∞ (k_1)	0.9745523
Calculated k_∞ (k_2)	0.97552953
Relative Error (in %) $\frac{\lvert k_1 - k_2 \rvert}{k_1} X\ 100$	1.0027×10^{-1}
Number of Azimuthal Directions	4
Number of Rays per Direction	160

5.5 BENCHMARK PROBLEM 5

Benchmark problem 5 is also a two – energy group problem consisting of four regions, as shown in Figure 5.5. Fuel (region 1) is composed of uranium - plutonium MOX (U-Pu-MOX), fuel cladding (region 3) is composed of zircalloy (Zr, Sn, Fe, Cr, O), and coolant (region 4) is composed of light water (H_2O). Table 5.19, Table 5.20 and Table 5.21 provide the material specifications, dimensional specifications and nuclear data specifications associated with different material regions of the problem, respectively.

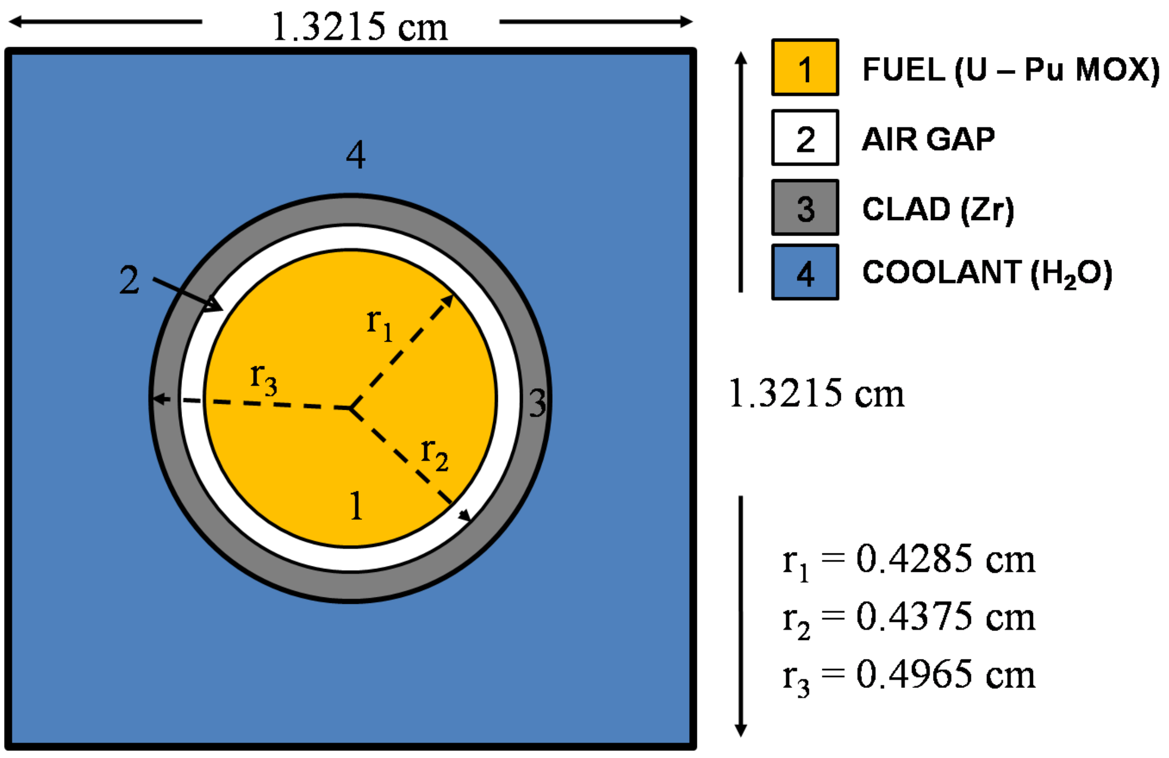


Figure 5.5 - Benchmark Problem 5

Table 5.19 – Material specifications of benchmark problem 5

REGION	MATERIAL COMPOSITION
Fuel	U^{235}, U^{238}, Pu^{239}, Pu^{240}, Pu^{241}, Pu^{242}, O^{16} (U – Pu - MOX)
Clad	Zr^{91}, Sn^{118}, Fe^{56}, Cr^{52}, O^{16}
Coolant	H^1, O^{16} (H_2O)

Table 5.20 - Dimensional specifications of benchmark problem 5

PARAMETER	DIMENSION (in cm)
Fuel Pellet Radius	0.4285
Cladding Inner Radius	0.4375
Cladding Outer Radius	0.4965
Pin Pitch	1.3215

Table 5.21 - Nuclear data specifications of benchmark problem 5

REGION	MATERIAL	ENERGY GROUP (g)	χ_g	$\Sigma_{t,g}$ (in cm^{-1})	$\nu\Sigma_{f,g}$ (in cm^{-1})	$\Sigma_{s,g\to1}$ (in cm^{-1})	$\Sigma_{s,g\to2}$ (in cm^{-1})
1	1	1	1.0	4.0283e-1	4.1718e-2	3.5711e-1	6.0950e-4
		2	0.0	1.84233e0	2.61407e0	1.8484e-3	4.0275e-1
2	2	1	0.0	1.0e-6	0.0	1.0e-8	1.0e-8
		2	0.0	1.0e-6	0.0	1.0e-8	1.0e-8
3	3	1	0.0	2.7391e-1	0.0	2.7215e-1	2.1081e-4
		2	0.0	2.8486e-1	0.0	9.7459e-4	2.7902e-1
4	4	1	0.0	8.8557e-1	0.0	8.5059e-1	3.4576e-2
		2	0.0	2.53925e0	0.0	1.7204e-3	2.52504e0

5.5.1 Result

The number of triangles formed in every region of the problem as obtained after the initial triangulation and mesh refinement along with the user-specified number of segments on each material boundary are listed in Table 5.22. Table 5.23 provides a comparison of the multiplication factor calculated using this code with respect to the results obtained from DRAGON V4. The number of azimuthal directions and number of rays per direction used have also been mentioned.

Table 5.22 - Number of Delaunay triangles formed for benchmark problem 5

REGION	NO. OF TRIANGLES IN INITIAL TRIANGULATION	NO. OF TRIANGLES IN REFINED TRIANGULATION
1	98	296
2	225	225
3	225	275
4	300	672

BOUNDARY	NO. OF SEGMENTS
Fuel Pellet	100
Air gap	125
Clad	100
Pin cell horizontal	50
Pin cell vertical	50

Table 5.23 - Comparison of multiplication factor for benchmark problem 5

Reference k_∞ (k_1)	1.382806
Calculated k_∞ (k_2)	1.38289184
Relative Error (in %) $\frac{\lvert k_1 - k_2 \rvert}{k_1} X\ 100$	6.2077 x 10^{-3}
Number of Azimuthal Directions	8
Number of Rays per Direction	101

5.6 CPU EXECUTION TIME ANALYSIS

The results obtained from analysis CPU execution time analysis of each benchmark problem (with least absolute relative error) are listed in Table 5.24. The table also provides a comparison of these results with those obtained from DRAGON V4.

Table 5.24 - Module wise CPU Execution Time

PROBLEM	CPU EXECUTION TIME (in s)		
	DELAUNAY TRIANGULATION	RAY TRACING	FLUX AND k_∞ CALCULATIONS
Benchmark 1	0.516	1.077	300.234
Benchmark 2	0.749	0.749	178.355
Benchmark 3	0.483	0.593	127.950
Benchmark 4	0.828	0.671	104.344
Benchmark 5	0.827	0.952	218.978

CONCLUSION

This work is primarily aimed at developing and validating a MOC-based code for lattice level calculations for a 2-dimensional pin cell. The code solves the characteristic neutron transport equation for a 2-dimensional pin cell to compute the neutron flux values in different spatial regions of the problem domain, and uses the distribution so obtained to determine the effective multiplication factor for the region of interest. Delaunay triangulation and mesh refinement techniques have been used to subdivide the problem into smaller, triangular, unstructured meshes. Once the meshes are available, a suitable quadrature set of azimuthal and polar directions is selected and a user defined number of characteristic neutron streaming paths or rays are inserted in each direction, and the track segments made in various meshes are identified and stored using the ray tracing routine. This information is used for the calculation of neutron flux and multiplication factor values. The developed code has been tested and validated against a variety of pin cells which cover a broad spectrum of fuel, clad, coolant/moderator and dimensional specifications.

The code requires following three input files from the user:

1. *pin_cell_geom_params.in* provides the dimensional specifications of the region of interest including the number of segments required to be made on each region boundary. The number of azimuthal directions and number of rays per direction are also given in this file.
2. *polar_angle.in* provides the polar quadrature to be used. Double angular 16-point quadrature set (DP_7), symmetric about the azimuthal plane of the problem, has been used for accurately modeling neutron motion out of the azimuthal plane in various polar directions.
3. *nuclear_data.in* provides the nuclear data i.e. macroscopic group-wise transport, production, and scattering cross – sections and the neutron fission spectrum associated with the different materials constituting the problem domain.

Five infinite square pin cell benchmarks were chosen for the analysis. All the problems analyzed are two energy-group having three or four material regions. The infinite multiplication factor

(k_∞) calculated using the developed code was compared with benchmark solutions computed using DRAGON V4.

In general, the MOC-based code was found to perform satisfactorily irrespective of the material number and type, and dimensional specifications. The k_∞ values predicted by this code were in good agreement with those obtained from DRAGON V4. The relative error was found to be small, < $\pm$1%. A maximum absolute relative error of 1.0027 x 10^{-1} % was obtained in benchmark problem 4.

The region of interest is faithfully reproduced by Delaunay triangulation and mesh refinement. The triangular meshes formed have optimal size and aspect ratio specifications. The theoretical bound on minimum angle in a mesh is $20.7°$. The maximum mesh size for each mesh is automatically computed as a function of its material composition. However, no minimum size bound is available for the meshes, and it is only governed by the number of segments specified by the user. The meshes also show fast size gradation over short distances, as expected.

The CPU execution time for different modules of the code was also obtained. Delaunay triangulation and mesh refinement, and ray tracing modules of the code require negligible time as compared to the inner – outer iteration scheme used in the flux and multiplication factor calculation module.

SCOPE FOR FUTURE STUDIES

Currently, the MOC-based code solves the neutron transport equation for square pin cell to obtain the infinite multiplication factor (k_∞) using few group homogenized cross section. It will be interesting to extend the code to obtain fluxes in individual regions and reaction rates thereof. Also the extension of this code may include solution for the neutron transport equation in hexagonal pin cell. At a later stage the code can be extended to solve for the current configuration of lattices (17x17 as in western PWR or VVER).

It is also essential to include periodic and vacuum boundary conditions for analysis of problems having asymmetric material composition. It would be interesting to study the behavior of accuracy of the result as a function of number of azimuthal directions by inclusion of anisotropic scattering effects over and above the currently considered isotropic scattering.

It will also be interesting to prescribe an optimum track density to the user via minimization of the error between actual volume of a region/mesh and approximate volume obtained from the characteristic rays. This would help improve the robustness of the code and also reduce the dependence of the results on user-defined inputs.

REFERENCES

1. Gelbard E. M. (1968). Computing Methods in Reactor Physics (Chapter 4). In: Greenspan H., Kelber K., Okrent D. (eds). Gordon and Breach, New York.
2. Lewis E. E., Miller W. F. Jr (1984). Computational Methods of Neutron Transport. John Wiley, New York. (Republished by American Nuclear Society, Inc., 1993).
3. Gelbard E. M. (1960). Application of Spherical Harmonics Methods to Reactor Problems. WAPDBT-20, Bettis Atomic Power Laboratory.
4. Carlson B. G., Bell G. I. (1958). Solution of the Transport Equation by the S_N Method. *Proceedings of the United Nations International Conference on Peaceful Uses of Atomic Energy, Vol. 2*, Geneva, 2386p.
5. Chandrasekhar S. (1960). Radiative Transfer. Dover, New York.
6. Carlvik I. (1965). A Method for Calculating Collision Probabilities in General Cylindrical Geometry and Applications to Flux Distributions and Dancoff Factors, *Proceedings of the United Nations International Conference on Peaceful Uses of Atomic Energy*, *Vol. 2*, Geneva, 225p.
7. Sanchez R., McCormick N. J. (1982). A Review of Neutron Transport Approximations. *Nuclear Science and Engineering*, *80*, 481.
8. Askew J. R. (1972). A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries. Report AEEW-M 1108, United Kingdom Atomic Energy Establishment, Winfrith.
9. Halsall M. J. (1980). CACTUS, A Characteristics Solution to the Neutron Transport Equation in Complicated Geometries. Report AEEW-R 1291, United Kingdom Atomic Energy Establishment, Winfrith.
10. Roy R. (1998). The Cyclic Characteristics Method. *International Conference on Physics of Nuclear Science and Technology*, Long Island, New York, October 5-8.
11. Le Tellier R., Hebert A. (2006). On the Integration Scheme along a Trajectory for the Characteristics Method. *Annals of Nuclear Energy*, *33*, 1260.
12. Boyd W., Shaner S., Li L., Forget B. Smith K. (2014). The OpenMOC Method of Characteristics Neutral Particle Transport Code. *Annals of Nuclear Energy*, *68*, 43–52.

13. Carter L. L., Cashwell E. D. (1975). Particle Transport Simulation with the Monte Carlo Method. TID-26607, National Technical Information Service, U.S. Department of Commerce.

14. Kalos M. H., Whitlock P. A. (1986). Monte Carlo Methods. Wiley, New York.

15. Lux I., Koblinger L. (1991). Monte Carlo Particle Transport Methods: Neutron and Photon Calculations. CRC Press, Boca Raton.

16. Spanier J., Gelbard E. (2008). Monte Carlo Principles and Neutron Transport Problems. Dover, New York. Originally published by Addison-Wesley, Reading (1969).

17. X-5 Monte Carlo Team (2003). MCNP – A General Monte Carlo N-particle Transport Code, Version 5. Los Alamos National Laboratory, LA-UR-03-1987 (Revised October 2005).

18. Babuška I., Aziz A. K. (1976). On the Angle Condition in the Finite Element Method. *SIAM Journal on Numerical Analysis*, *13*(2), 214–226.

19. Delaunay B. (1934). Sur la Sphère vide. *A la mémoire de Georges Voronoi*, *Bulletin de l'Académie des Sciences de l'URSS*, *Classe des Sciences Mathématiques et Naturelles, 6*. 793–800.

20. Rebay S. (1993). Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer – Watson Algorithm. *Journal of Computational Physics*, *106*, 125–138.

21. Lawson C. L. (1977). Software for C^1 Surface Interpolation. *Mathematical Software III*. 161–194. Academic Press, New York.

22. Bowyer A. (1981). Computing Dirichlet Tessellations. *Computer Journal*, *24*(2), 162–166.

23. Watson D. F. (1981). Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes. *Computer Journal*, *24*(2), 167–172.

24. Miller G. L., Talmor D., Teng S., Walkington N. (1995). A Delaunay Based Numerical Method for Three Dimensions: Generation, Formulation, and Partition. *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, Las Vegas, Nevada, pages 683–692.

25. Lee C. E. (1962). The Discrete S_N Approximation to Transport Theory. Los Alamos National Laboratory. LA-2595.

APPENDIX I

DATA STRUCTURES

A modular programming approach has been used for implementation of the algorithms for the three modules of the code, namely, Delaunay triangulation and mesh refinement, ray tracing and flux and multiplication factor calculations, as discussed in chapters 2, 3 and 4 respectively. The code has been developed in FORTRAN 90, which lacks advantageous concepts of object oriented programming (OOP). However, since OOP considerably simplifies code development and extension, its basic concept of laying emphasis on data rather than procedure has been improvised in the code to ensure robustness, versatility and ease of modification. Thus, all data associated with one physical entity is put into one container, and referred using a single entity name. This further reduces the number of variables required by a considerable amount, and hence simplifies book keeping of variable names within the code. The various data structures and their significance used in the code are listed below in Table I.1 and are described in the following sections.

Table I.1 - List of data structures used

NAME	TYPE	SIGNIFICANCE
tri_id_list	LIFO Linked List	Linked list of triangle IDs based on LIFO principle
ray_id_list	LIFO Linked List	Linked list of ray IDs of different rays based on LIFO principle
point	Derived Data Type	Represents a real world point in 2-D space
edge	Derived Data Type	Represents a real world line segment formed from 2 end-points
triangle	Derived Data Type	Represents a triangle formed from 2 vertices or points

NAME	TYPE	SIGNIFICANCE
edge_list	LIFO Linked List	Linked list of edges based on LIFO principle
triang_list	FIFO Linked List	Linked list of triangles based on FIFO principle
track_segment	Derived Data Type	Represents a track segment formed due to passage of the ray through a mesh
trace_list	FIFO Linked List	Linked list of track segments based on FIFO principle
ray	Derived Data Type	Represents a characteristic neutron streaming path
psi	Array	Represents angular flux values along a ray at its points of intersection with different mesh boundaries (Φ)

I.1 TRIANGLE ID LIST

tri_id_list is a linked list which works on LIFO principle. It consists of a list of IDs of triangle/mesh which satisfy some specific criteria. Its component variables are given in Table I.2.

Table I.2 - Elements of *tri_id_list*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
tri_id	Integer	Triangle ID
next	tri_id_list pointer	Address of the next element in the list

I.2 RAY ID LIST

Similarly *ray_id_list* is also a linked list which works on LIFO principle. But it represents a list of ray Ids. Its component variables are given in Table I.3.

Table I.3 - Elements of *ray_id_list*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
tri_id	Integer	Ray ID
next	ray_id_list pointer	Address of the next element in the list

I.3 POINT

point is a derived data type which represents a real world point in 2-dimensional space. All initial mesh nodes, intersection points and new vertices inserted for mesh refinement are represented using this data type. Its component variables are given in Table I.4.

Table I.4 - Elements of *point*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
X	Real	X – coordinate of point
Y	Real	Y – coordinate of point
p_type	Integer	Region in which the point is located

I.4 EDGE

edge is a derived data type representing a line segment formed from two points which are its end points. All triangle edges may be represented using this data type. Its component variables are given in Table I.5.

Table I.5 - Elements of *edge*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
end_point	Integer array	Vertex IDs of the 2 end-points which form the edge

I.5 TRIANGLE

triangle is another derived data type created to represent a triangle. All the meshes, being triangular shaped, are represented using this data type. Its component variables are given in Table I.6.

Table I.6 - Elements of *triangle*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
vert	Integer array	Vertex IDs of the 3 vertices of the triangle
c_center	Point	Center of circum-circle of the triangle
c_rad	Real	Radius of the circum-circle of the triangle
mat_type	Integer	ID of material constituting the mesh
nbor_head	tri_id_list	Header element of list of mesh IDs of every triangle neighbor
ray_head	ray_id_list	Header element of list of ray IDs of all rays passing through the triangle

I.6 EDGE LIST

edge_list represents a LIFO – based linked list of edges. The list is mainly used while finding the boundary edges of the polygonal cavity formed by contiguous bad triangles due to the process of new vertex insertion in an existing Delaunay triangulation. It enables us for determination and modification of the list of edges of the polygonal hole. Its component variables are given in Table I.7.

Table I.7 - Elements of *edge_list*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
e	edge	Represents an edge, which forms one element of the list
next	edge_list pointer	Address of the next element in the list

I.7 TRIANGLE LIST

triang_list represents a FIFO – based linked list of triangles. The list is used for the Delaunay triangulation of the region of interest. The existing triangulation formed from all mesh nodes/vertices that have been inserted is stored using this data structure. Thus, easy deletion of bad triangles and insertion of new triangles is possible. Its component variables are given in Table I.8.

Table I.8 - Elements of *triang_list*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
t	Triangle	Represents a triangle, which forms one element of the list
next	triang_list pointer	Address of the next element in the list

I.8 TRACK SEGMENT

track_segment is a data structure designed to store all information related a track segment. Its component variables are given in Table I.9.

Table I.9 - Elements of *track_segment*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
in_pt	Point	In-point of the ray in some mesh
out_point	Point	Out-point of the ray in the same mesh
tri_id	Integer	Mesh/triangle ID of the mesh through which the ray passes

I.9 TRACE LIST

trace_list is also a FIFO – based linked list of the track segments formed due to the passage of a ray through the region of interest. The ray intersects triangles successively till it finally exits the problem geometry. Its component variables are given in Table I.10.

Table I.10 - Elements of *trace_list*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
segment	track_segment	Represents one track segment along a ray, which forms one of the several elements of the list
Next	trace_list pointer	Address of the next element in the list

I.10 RAY

ray represents a derived data structure used to assemble all data related with a characteristic streaming path. Its component variables are given in Table I.11.

Table I.11 - Elements of *ray*

VARIABLE NAME	DATA TYPE	SIGNIFICANCE
fi	Real	Angle of inclination of the ray w.r.t +ve X – axis
c0	Point	Initial point of a ray on the problem geometry
tri_id_0	Integer	ID of initial mesh in which the ray enters
n_track_segments	Integer	Number of track segments formed by the ray
reflected_ray_id	Integer	ID of reflective counterpart of the ray
rt_head	triang_list pointer	Header element of the trace list of the ray
rt_tail	triang_list pointer	Tail (last) element of the trace list of the ray

I.11 PSI

psi is a dynamic array which contains the angular flux values along a ray at various points of intersection of ray with the triangular meshes, size being the number of track segments, n_track, or $(n\_track\ +1)$.