CAD Based Monte Carlo Method: Algorithms for Geometric Evaluation in Support of Monte Carlo Radiation Transport Calculation

Mengkuo Wang

August 2006

UWFDM-1353

Ph.D. thesis.

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

CAD Based Monte Carlo Method: Algorithms for Geometric Evaluation in Support of Monte Carlo Radiation Transport Calculation

Mengkuo Wang

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

August 2006

UWFDM-1353

Ph.D. thesis.

CAD based Monte Carlo Method

Algorithms for geometric evaluation in support of Monte Carlo radiation transport calculation

By
Mengkuo Wang

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy
(Nuclear Engineering and Engineering Physics)

at the
UNIVERSITY OF WISCONSIN – MADISON
2006

Abstract

In particle transport computations, the Monte Carlo simulation method is a widely used algorithm. There are several Monte Carlo codes available that perform particle transport simulations. However the geometry packages and geometric modeling capability of Monte Carlo codes are limited as they can not handle complicated geometries made up of complex surfaces. Previous research exists that take advantage of the modeling capabilities of CAD software. The two major approaches are the Converter approach and the CAD engine based approach. By carefully analyzing the strategies and algorithms of these two approaches, the CAD engine based approach has been identified as the more promising approach. Though currently the performance of this approach is not satisfactory, there is room for improvement. The development and implementation of an improved CAD based approach is the focus of this thesis.

Algorithms to accelerate the CAD engine based approach are studied. The major acceleration algorithm is the Oriented Bounding Box algorithm, which is used in computer graphics. The difference in application between computer graphics and particle transport has been considered and the algorithm has been modified for particle transport.

The major work of this thesis has been the development of the MCNPX/CGM code and the testing, benchmarking and implementation of the acceleration algorithms. MCNPX is a Monte Carlo code and CGM is a CAD geometry engine. A facet representation of the geometry provided the least slowdown of the Monte Carlo code. The CAD model generates the facet representation. The Oriented Bounding Box algorithm was the fastest acceleration

technique adopted for this work. The slowdown of the MCNPX/CGM to MCNPX was reduced to a factor of 3 when the facet model is used.

MCNPX/CGM has been successfully validated against test problems in medical physics and a fusion energy device. MCNPX/CGM gives exactly the same results as the standard MCNPX when an MCNPX geometry model is available. For the case of the complicated fusion device – the stellerator, the MCNPX/CGM's results closely match a one-dimension model calculation performed by ARIES team.

Acknowledgements

I want to extend my sincere gratitude and appreciation to the many people who helped make this dissertation possible and in many other ways helped my Ph.D endeavor. First, I am deeply indebted to my advisors, Professor Douglass Henderson and Professor Tim Tautges. That this dissertation project occurred and could be completed is due in large part to the enlightenment and support from Prof. Henderson and Prof. Tautges. Their guidance, help, and training in many aspects during my Ph.D study will have a long-term influence on my future career.

I also want to thank Prof. Paul Wilson, for his insightful advices on this research and his kindness answering many questions throughout my Ph.D study here in Madison.

Special thanks are also due to Prof. Laila El-Guebaly, Prof. Mohamed E. Sawan for supporting me and providing valuable feedback at the early stages of this dissertation research.

Special thank you is owed to my friends in our university, including Jiankui Yuan, Po Hu, especially Qiguang Zhu. Their friendship and kindness help has made my research and life in Madison a wonderful experience.

Words cannot express my deepest gratitude to my family. Without my father's vision and my mother's unconditional support, I would not have been going on this far away from home.

Finally, I want to thank my wife, Feng Qi for her constant support and optimism. In many ways, the graduate student life has been a mixture of stress, pain, and success. I thank Feng for helping me through all the difficult times, for sharing my happiness, and for always being there.

List of figures:

List of tables:

Chapter 1

Introduction

Particle transport is used in many disciplines including Nuclear Engineering and Medical Physics. Researchers first became interested in this topic when they studied radiation shielding. Later the knowledge of particle transport was applied to nuclear reactor design and dose calculation in the field of Medical Physics. The typical problem is that in a given system, the radiation source characteristics are specified (location, energy and angular dependence) and one is interested in the particle behavior in the surrounding domain in order to compute a reaction rate or determine a response. For example in a Medical Physics application, a cobalt - 60 source is used to treat cancerous tissue. The goal is to give a prescribed dose to the diseased tissue and spare the surrounding normal tissue. In order to provide the correct dose we need to perform a photon transport simulation. In a shielding problem, for example, in a nuclear reactor, the reactor and human working area are separated and the human working area should be a safe environment. Neutron and photon transport simulations are required to quantify the radiation environment in the working area. If the dose is too high we need to redesign the shield to reduce the dose in that area. Although experiments are one way to obtain the dose or particle flux, a computational transport simulation provides these quantities in a less expensive, easier and faster way.

In applications of particle transport, the correct modeling of the geometry is an important factor. In computational transport simulations, we hope the geometry model used is the same as in the real application. However, many current computational transport simulation codes only provide a limited geometry modeling capability and hence only an approximation of the realistic geometry configuration is available. Inevitably an inaccurate geometry model will affect the accuracy of the simulation result. There is some previous research on this topic but the computational performance is not satisfactory. In this work some techniques that speed up the computation are presented in Monte Carlo radiation transport code application.

1.1 Deterministic and Statistical methods

Two primary methods are used to compute radiation transport solutions. One is the deterministic method and the other is the statistical method.

Deterministic methods usually deal with the solution of a differential equation with appropriate boundary conditions and for time-dependent problems, an initial value. The domain of interest is subdivided into a computational grid. In particle transport, continuous time, space and energy radiation transport equations are discretized and the solution is on grids over the space, angle, energy, and possibly time domains. That is to say: we need to solve an algebraic system based on discrete differential equations. There is always an error associated with the solution. The grid size is carefully chosen as a compromise between the large relative error (mesh too coarse) and long computational time (mesh too fine). The grid should also be chosen

carefully to best represent the boundary conditions.

A statistical method models the problem on an individual particle basis and simulates the behavior of a particle as it traverses the medium. Interactions of particles with the medium are determined by probabilities of interaction. Therefore, theoretically, a differential equation for the system is not required. Only probability density functions (pdf's) are needed to describe the system, which is related to the physical problem being modeled. According to the law of large numbers, the solution is the average of many simulated particles. The more particles we simulate, the more accurate the solution. In addition the statistical algorithm complexity is largely geometry independent. This means the algorithm complexity does not change when applied to various geometric configurations, which makes the statistical method especially suitable for three-dimensional problems.

1.2 Monte Carlo Method

The Monte Carlo method is a statistical method, which has been in existence for many years [1]. The computational speed to arrive at a highly accurate solution using this method can be very slow since the accuracy is directly related to the square root of the number of particles simulated. Because a large number of particles must be simulated, it is nearly impossible for a human being to solve a problem by this method. Only after computer science and special computational techniques were invented, could this method be widely used in many fields, and mostly in complex applications. To understand this method, we need to know the content of

the method.

The major components of a Monte Carlo method are the pdf's, random number generator, sampling, tallying (scoring), and error estimation. The details of each component can be found in reference [1].

The Monte Carlo method repetitively simulates the physical process of a particle traveling through material. Each simulation is called a "history". In each simulation, the source particle has a PDF function to describe energy, position and the direction distribution. Each physics interaction has a PDF function to describe energy and the direction distribution of outgoing particles. A Random number generator is used in sampling these PDF functions. The desired result is the average of the number of histories.

The random number generator is an important component of a Monte Carlo method. In Monte Carlo codes, all the random numbers are just pseudo-random numbers. A good random number algorithm produces a uniform, unbiased random number with a long period. Hammersley and Handscomb give a detailed discussion on this topic [1]. The random number and its generator are the basis of all sampling in a Monte Carlo code. All samplings from distributions are begun with a random number generator.

The pdf function is another important component. It is the statistical description of the

physical system and describes the statistical behavior of a large sample of particles. A large database is needed to store a particle's statistical information, such as the probability of various interactions, the probability of a particle's energy after an interaction, and the probability of a particle's direction after an interaction. Only if the pdf is a precise representation of the particle's physical behavior, can the result of a Monte Carlo simulation be accurate.

The Monte Carlo method constructs a tally by averaging contributions in specific places over all simulation histories. Therefore a statistical error is generated as the variance of histories. To improve the tally accuracy, the statistical error must achieve a given small error. According to the central limit theory, the statistical error is proportional to the reciprocal of the square root of the history number:

$$R \propto 1/\sqrt{N} . \qquad (1)$$

In equation (1), R is the statistical error and N is the history number. Hence, in order to obtain a highly accurate solution a large number of histories are required. This translates into a high computational time expense. For most cases we need to wait hours or even days to obtain an accurate result. Therefore, the Monte Carlo method is considered a slowly converging method.

The Monte Carlo method requires a continuous representation of the domain and a discretized grid is not needed. It can be applied to an arbitrary 3-D geometric configuration and many realistic systems can be readily simulated. These realistic systems can be quite complicated

geometries. Modeling of increased geometrical complexity will prolong the computational time of current Monte Carlo codes and will make the problem setup more difficult and time consuming.

1.3 MCNP Introduction

MCNP is a widely used Monte Carlo code and it will be used as an example to introduce a current Monte Carlo code used for particle transport simulations.

MCNP is "a general-purpose Monte Carlo N-Particle code that can be used for neutron, photon, electron, or coupled neutron/photon/electron transport, including the capability to calculate eigenvalues for a critical system" [2]. For neutrons, the code uses the evaluated pointwise cross-section data. For photons, it takes account of incoherent and coherent scattering, photoelectric absorption, the possibility of fluorescent emission after photoelectric absorption, pair production, annihilation radiation after pair production, and bremsstrahlung. For electrons, it uses a continuous slowing down model which includes positrons, k x-rays, and bremsstrahlung. Besides those physics applications, MCNP has some important features that make it useful in a number of situations: a powerful general source; surface and body sources; geometry and output tally plotters; many kinds of variance reduction techniques; a flexible tally structure; and an abundance of cross-section data.

MCNP was developed by the Transport Methods Croup (Group XTM) at Los Alamos National Laboratory. This group also improves and maintains MCNP by releasing a new version every two to three years. The current highest version of MCNP is MCNP5 [3].

MCNPX [4] is a major extension of the MCNP code. It can track several particles (neutrons, photons, electrons and general charged particles) through a very large energy range (TeV to μeV). Geometry, basic tally and graphical capabilities of MCNPX are the same as the standard MCNP. MCNPX has a feature that is not available in the MCNP5 code. It is the pin-hole projection tally and it will be discussed and used in Chapter 5. My developmental research utilizes the framework of MCNPX.

MCNPX is composed of about 360 subroutines. The code can be divided into 3 parts – code initialization, run time and output, as shown in Fig 1.1.

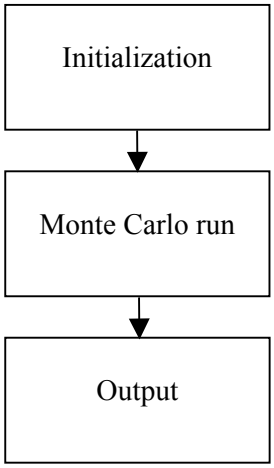


Figure 1.1. MCNP flow diagram

The initialization part includes problem initiation and cross section processing. The code will read the input file, setup the problem, process the source, process tallies, process the material specifications, and calculate the cell volumes and surface areas. The geometric information, such as surface, cell, and point, are also initialized here. One thing to note is that initialization is just a one-time job and requires little run time when compared to the actual Monte Carlo run time.

Execution performs the particle transport executing the number of histories requested by the user. Each history starts with a source particle by sampling its position, direction and energy of a source particle. Then MCNP tracks this particle as it transports through the domain, until it is absorbed or leaves the system.

The execution stage is similar in all Monte Carlo transport codes and involves geometry evaluation. The common situation is that one particle can be in only one cell at any given time. For the case of neutrons and photons, particles travel in a straight line and will cross (hit) the boundary of the current cell unless a collision with the material has occurred. However, for charged particles like the electron, they will experience the coulomb force and change direction all the time. Therefore, a charged particle track is a curve. In a Monte Carlo code, a curve is usually subdivided into many small line segments. In every segment, the description above still holds.

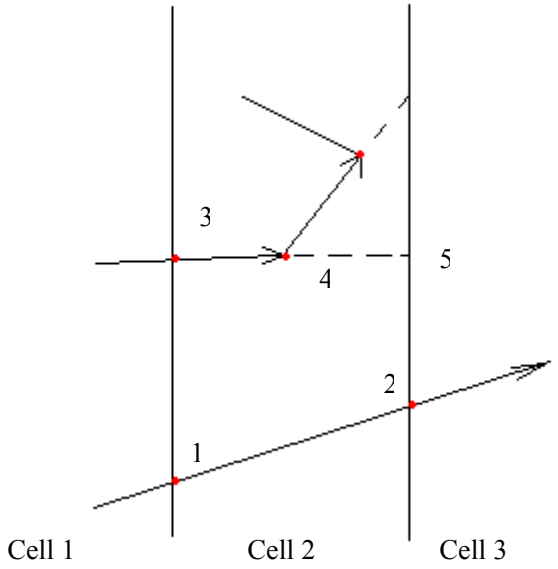


Figure 1.2. Monte Carlo particle transport diagram

Figure 1.2 shows a Monte Carlo particle transport diagram. MCNP first determines the distance to the boundary along the direction of travel, and the boundary surface number. In Fig. 1.2, for point 1, MCNP will find point 2, and for point 3 MCNP will find point 5. It then obtains a cell number, which is on the other side of the surface. Then MCNP will sample the distance to the next collision. If this distance is greater than the distance to the boundary, MCNP will move the particle just across the boundary and start the process again. That is the case of point 1 and point 2. The particle is in a new cell. That is because MCNP defines a cell as a region with uniform material properties and once a particle enters a new cell, there will be a new material and re-sampling of the distance to a collision is needed. Therefore we can only move the particle a distance L across the surface. It makes the distance to boundary necessary for particle transport. If the distance to a collision is less than the distance to the

boundary (that is the case of points 3, and 4) MCNP will move the particle that distance along the current direction and the particle will have a collision there. After a collision, the particle will have a new energy and new direction, MCNP will continue these processes until the particle leaves the system or is removed due to variance reduction.

During execution, MCNP will repeat the history many times until the run time or run history reaches the limit set by a user. To obtain a better accuracy, the user must set a high run time or history limit. The "find the distance to the boundary" function should also be efficient because it is used in a very high frequency.

In the output part, MCNP gathers the information saved during computation time and periodically writes an output file which contains summary tables and tallies; for example, the following information is written: the surface flux, surface current, and energy deposition within cells.

Although MCNP is already widely used in the areas of Nuclear Engineering and Medical Physics, the geometry domain of MCNP is limited. These limitations have already restricted the application of MCNP on some problems with very complicated geometry. In section 1.3.1, the geometry limitations are discussed. In 1.3.2, the execution time limitations are discussed.

1.3.1 Modeling

MCNP can treat “an arbitrary three-dimensional configuration of materials in geometric cells bounded by first- and second-degree surfaces and fourth-degree elliptical tori”. The geometric module within MCNP performs this task. The process of constructing the geometry configuration is called geometric modeling.

MCNP [2] uses an input file to input geometric information, which describes geometric surfaces and cells as Boolean combinations of surfaces. MCNP defines the geometry by defining surfaces first. Then it uses the surfaces to construct cells. The cells can be used in Boolean expressions to define other complicated cells. Surfaces are defined by providing the coefficients to the analytic surface equations.

A total of 29 mnemonics can help the user define various types of surfaces. The user will assign a surface number for every defined surface. Table 1.1 presents the surface types that can be represented by MCNP.

Table 1.1 The surfaces that can be represented by MCNP [2]

Mnemonic	Type	Description	Equation	Card Entries
P	Plane	General	$Ax + By + Cz - D = 0$	$A B C D$
PX	↓	Normal to X-axis	$x - D = 0$	D
PY	↓	Normal to Y-axis	$y - D = 0$	D
PZ	↓	Normal to Z-axis	$z - D = 0$	D
SO	Sphere	Centered at Origin	$x^2 + y^2 + z^2 - R^2 = 0$	R
S	↓	General	$(x-\bar{x})^2 + (y-\bar{y})^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{x}\ \bar{y}\ \bar{z}\ R$
SX	↓	Centered on X-axis	$(x-\bar{x})^2 + y^2 + z^2 - R^2 = 0$	$\bar{x}\ R$
SY	↓	Centered on Y-axis	$x^2 + (y-\bar{y})^2 + z^2 - R^2 = 0$	$\bar{y}\ R$
SZ	↓	Centered on Z-axis	$x^2 + y^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{z}\ R$
C/X	Cylinder	Parallel to X-axis	$(y-\bar{y})^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{y}\ \bar{z}\ R$
C/Y	↓	Parallel to Y-axis	$(x-\bar{x})^2 + (z-\bar{z})^2 - R^2 = 0$	$\bar{x}\ \bar{z}\ R$
C/Z	↓	Parallel to Z-axis	$(x-\bar{x})^2 + (y-\bar{y})^2 - R^2 = 0$	$\bar{x}\ \bar{y}\ R$
CX	↓	On X-axis	$y^2 + z^2 - R^2 = 0$	R
CY	↓	On Y-axis	$x^2 + z^2 - R^2 = 0$	R
CZ	↓	On Z-axis	$x^2 + y^2 - R^2 = 0$	R
K/X	Cone	Parallel to X-axis	$\sqrt{(y-\bar{y})^2 + (z-\bar{z})^2} - t(x-\bar{x}) = 0$	$\bar{x}\ \bar{y}\ \bar{z}\ t^2\ \pm 1$
K/Y	↓	Parallel to Y-axis	$\sqrt{(x-\bar{x})^2 + (z-\bar{z})^2} - t(y-\bar{y}) = 0$	$\bar{x}\ \bar{y}\ \bar{z}\ t^2\ \pm 1$
K/Z	↓	Parallel to Z-axis	$\sqrt{(x-\bar{x})^2 + (y-\bar{y})^2} - t(z-\bar{z}) = 0$	$\bar{x}\ \bar{y}\ \bar{z}\ t^2\ \pm 1$
KX	↓	On X-axis	$\sqrt{y^2 + z^2} - t(x-\bar{x}) = 0$	$\bar{x}\ t^2\ \pm 1$
KY	↓	On Y-axis	$\sqrt{x^2 + z^2} - t(y-\bar{y}) = 0$	$\bar{y}\ t^2\ \pm 1$
KZ	↓	On Z-axis	$\sqrt{x^2 + y^2} - t(z-\bar{z}) = 0$	$\bar{z}\ t^2\ \pm 1$ ±1 used only for 1 sheet cone
SQ	Ellipsoid Hyperboloid Paraboloid	Axes parallel to X-, Y-, or Z-axis	$A(x-\bar{x})^2 + B(y-\bar{y})^2 + C(z-\bar{z})^2 + 2D(x-\bar{x}) + 2E(y-\bar{y}) + 2F(z-\bar{z}) + G = 0$	$A\ B\ C\ D\ E$ $F\ G\ \bar{x}\ \bar{y}\ \bar{z}$
GQ	Cylinder Cone Ellipsoid Hyperboloid Paraboloid	Axes not parallel to X-, Y-, or Z-axis	$Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fzx + Gx + Hy + Jz + K = 0$	$A\ B\ C\ D\ E$ $F\ G\ H\ J\ K$
TX	Elliptical or circular torus.	$(x-\bar{x})^2/B^2 + (\sqrt{(y-\bar{y})^2 + (z-\bar{z})^2} - A)^2/C^2 - 1 = 0$		$\bar{x}\ \bar{y}\ \bar{z}\ A\ B\ C$
TY	Axis is Parallel to	$(y-\bar{y})^2/B^2 + (\sqrt{(x-\bar{x})^2 + (z-\bar{z})^2} - A)^2/C^2 - 1 = 0$		$\bar{x}\ \bar{y}\ \bar{z}\ A\ B\ C$
TZ	X-,Y-, or Z-axis	$(z-\bar{z})^2/B^2 + (\sqrt{(x-\bar{x})^2 + (y-\bar{y})^2} - A)^2/C^2 - 1 = 0$		$\bar{x}\ \bar{y}\ \bar{z}\ A\ B\ C$
X Y Z P		Surfaces defined by points		

A cell is defined by surfaces. One important concept is the "sense" of a point to a surface. Suppose the surface function is f(x,y,z)=0, then a given point is (x0,y0,z0). If f(x0,y0,z0) > 0, the given point is on the positive side of the surface; it is denoted as "positive sense". If

f(x0,y0,z0) < 0, the given point is at the negative side of the surface, it is denoted as “negative sense”. Therefore every surface divides the space into two subspaces, one positive, and another negative. The cell can be defined by Boolean of these subspaces. And the user will assign a cell number to every defined cell. Boolean includes intersection (AND), union (OR) and complement (NOT). A blank space is the symbol of intersection. A “;” is the symbol of union. A “#” is the symbol of complement. For example, “1 0 1 -2 -3” defines a cell. The last three numbers mean the cell is the positive side of surface 1 intersecting with the negative side of surface 2 and intersecting with the negative side of surface 3. A 0 means there is no material in this cell. The first 1 is the cell number assigned by the user. The cell can also be defined by combining (AND, OR, NOT operations) on existing cells. For example, “4 0 #3; #2” means cell number 4 is composed by the space either not belonging to cell 3 or not belonging to cell 2.

MCNPX use the same geometry-modeling package as MCNP.

Table 1.2 is an example of a MCNP input file which constructs three cylinders; the geometry produced by this input is shown in Fig. 1.3

Table 1.2 Sample MCNP input for geometry

Testprob – n p
1 1 -0.675 -1 2 -3
2 0 -4 5 -6
3 0 -7 8 -9 #1 #2
4 0 7:-8:9

c surface card
1 cz 20
2 pz 10
3 pz 50
4 cz 5
5 pz 60
6 pz 70
7 cz 30
8 pz -5
9 pz 75

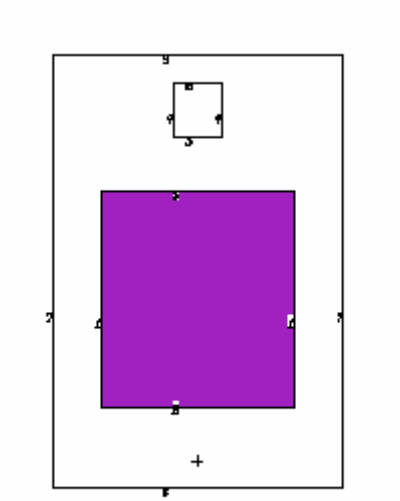


Figure 1.3. 3 cylinders generate by MCNP

However, there are some limitations of the geometry input portion of MCNPX. The input method is tedious. A user needs to draw many sketches to define the surface and cell. Thereafter, the MCNPX input file is typically written. MCNPX provides a feature to plot the geometry. But it can only be plotted after the input file is ready. For a complicated geometric configuration, it is hard to let the user check the correctness of all surface and cell number assignments. Using this input method, it is hard to model complex geometries. Once a complex geometric configuration is set up in MCNP, it is hard to modify [5].

There is no geometry analysis at the initialization stage. MCNP only stores the basic geometry information provided in the input file; such as surface number, cell number, and type of surface and sense. Therefore it actually requires each user to know every detail of geometry. Although there is an intensive internal check on geometry, only the basic and low-level errors such as no surface to construct a cell are reported. Higher-level errors such as cell overlap and gap can be detected only with lots of user effort. Because MCNP requires every point in the space to belong to one and only one cell or be the boundary of one or more cells, if there is a gap between the cells, the particle will be lost in this gap. If two cells overlap, the particle in the overlap region will belong to two cells, which is an unacceptable situation. MCNPX does not provide a built in feature to detect these errors. A user needs to run a zero material interaction case to examine those errors.

There is no compatibility of the MCNP geometry format with other graphics or geometry generating systems. There are many graphics or geometry software packages that can generate and modify the geometric configuration and store it on the computer in some format. But MCNP's geometry is incompatible with other kinds of geometry file formats. Even if the user wanted to run a problem on a geometric configuration, which has already been established by other software, the user will still need to generate MCNP's geometric configuration. (There have been efforts for translating CAD models into MCNP input, these efforts are described in Chapter 2.)

There are no advanced geometric features in MCNP such as discrete or facet based surface and B-spline surface. Facet based surface is widely used in representing some realistic or non-analytic surfaces. Many patches compose a facet-based surface. The patch normally is a polygon and is the basic unit of the total surface. All patches are connected to each other to represent the original object. Because the complicated object is subdivided into patches and every patch is relatively small and simple when compared to the original object, the surface is accurate enough and the analytic properties are easy to calculate.

We are interested in a facet based surface representation because (1) we can use it to represent a real geometry for which a solid geometry model does not exist, such as a scanned organ in Medical Physics; (2) we can use it to substitute a complicated solid model and then the geometry evaluation is much easier because we use plane (facet) to substitute the high order curve surface; and (3) we can use it as a flexible approximation of solid model because we can change the facet size to obtain a coarse or fine approximation.

The B-spline is also an important feature in geometric modeling. It can subdivide a complicated curve into pieces but still keeps some degree of continuity. Therefore B-splines are used extensively in CAD modeling for real parts such as blending objects. MCNP cannot generate a B-spline. If the user wants to use it, the user will have to input every part of a B-spline as a normal first-order or second-order curve or surface.

1.3.2 Performance

During execution, as mentioned before, MCNP performs the following sequence: “find the distance to the boundary, obtain the hit surface number and get the cell number on the other side of the hit surface”. It is a purely geometry problem. Therefore if necessary, the MCNP runtime geometry can be substituted by another geometric package.

In MCNP, to implement the “find the distance to boundary”, the “minimized positive distance principle” is used. It calculates the distance to each bounding surface along the particle direction. The surface with the minimum positive distance is the surface that is hit. This distance will be compared with the distance to a collision later.

To obtain a good performance, the “find the distance to the boundary” algorithm is simple. For example, in each cell, the boundary surface type can be directly obtained. Therefore MCNP can calculate the distance to the each infinite surface. If the cell is a convex object (or in the terminology of MCNP, an object is totally on the one side of each boundary surface), the minimum distance is the distance to the boundary. If the cell is concave (or in the terminology of MCNP, an object is not totally on the one side of each boundary surface), MCNP only needs to test the hit distance and compared to the ranked order from the minimum distance, then second minimum, third minimum until the last one. If a hit point is located on the surface portion, the part of the bounding surface of object and not the extension of

bounding surface, this distance is the distance to the boundary.

Therefore, MCNP does use simple techniques to speedup the computation. But there is still room left for optimization or acceleration. Because MCNP does not implement any advance acceleration techniques, it just simply compares the distances to all the bounding surfaces, which means we always need to calculate distances to every surface. It is not an efficient method. An example is that of charge particle transport; a particle can cross the bounding surface only when it is near the bounding surface (how near the distance depends on particle energy, material and so on). If the particle is not "near" to the surface, we won't need to calculate the distance to the surface. It can save a lot of computation time. Unfortunately, this feature is not provided in MCNP. In addition, the "find the distance to surface" is called "ray-object intersection" in the area of computer graphics. A lot of research has been done on this topic and a lot of good algorithms are available. But MCNP does not use the latest or best algorithms from among them. This will be discussed later.

In theory, a facet based surface can be input into MCNP by inputting each facet patch. But in fact, it is difficult for the user to input every patch, and even if every patch is already input, it is very inefficient to calculate it in MCNP because MCNP will calculate every patch for minimized distance to find a single hit. It will be extremely slow to use and is unacceptable.

1.4 Complex device

Usually a real object is not simple geometry. Fig 1.4 depicts a clothespin with a match clinched between its jaws. The spring is moved out of normal position to illustrate the complicated geometries, which make up this real object. This model cannot be analyzed with the MCNP geometric package because of the complicated helical surface of the spring.

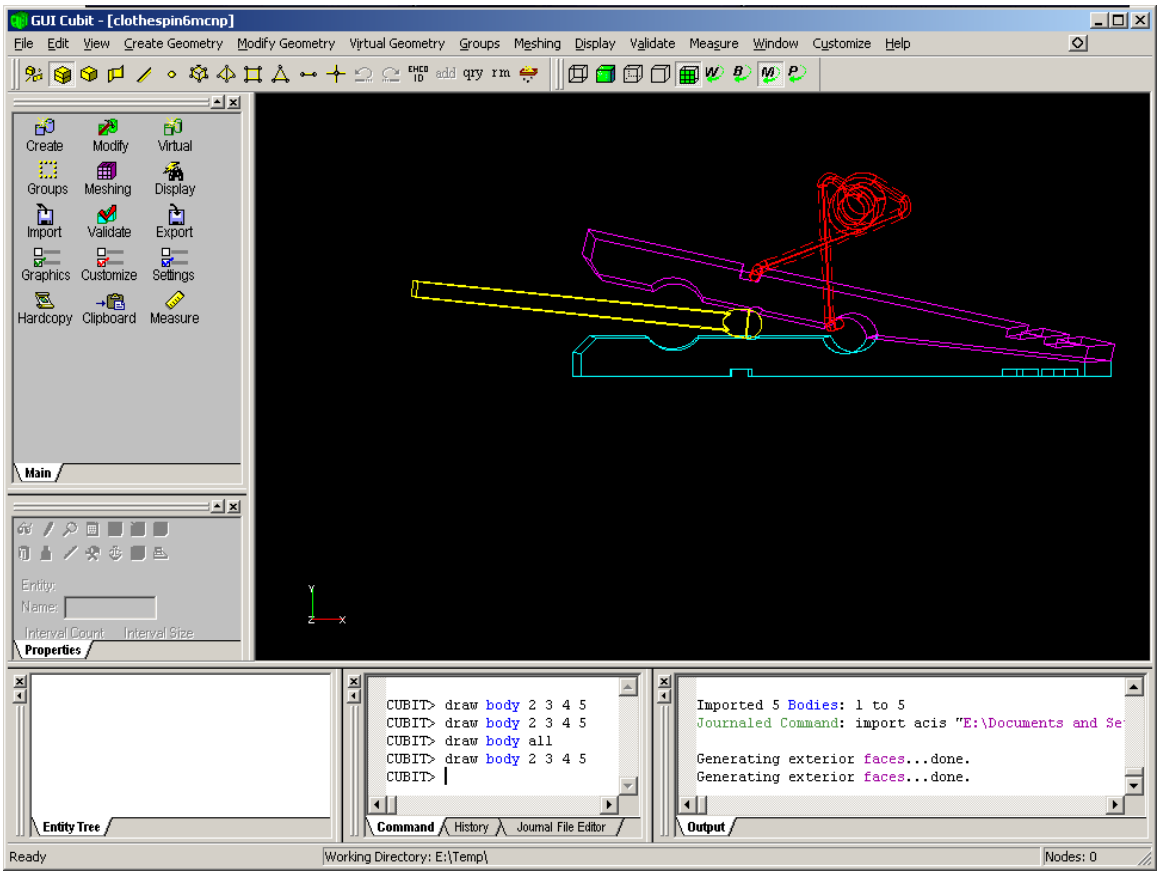


Figure 1.4. Complex device a clothespin

More complex nuclear engineering or medical physics devices and applications are being designed. For example, a Stellerator device, depicted in Fig 1.5, is a complex fusion machine currently being investigated as a proposed design for a fusion power plant.

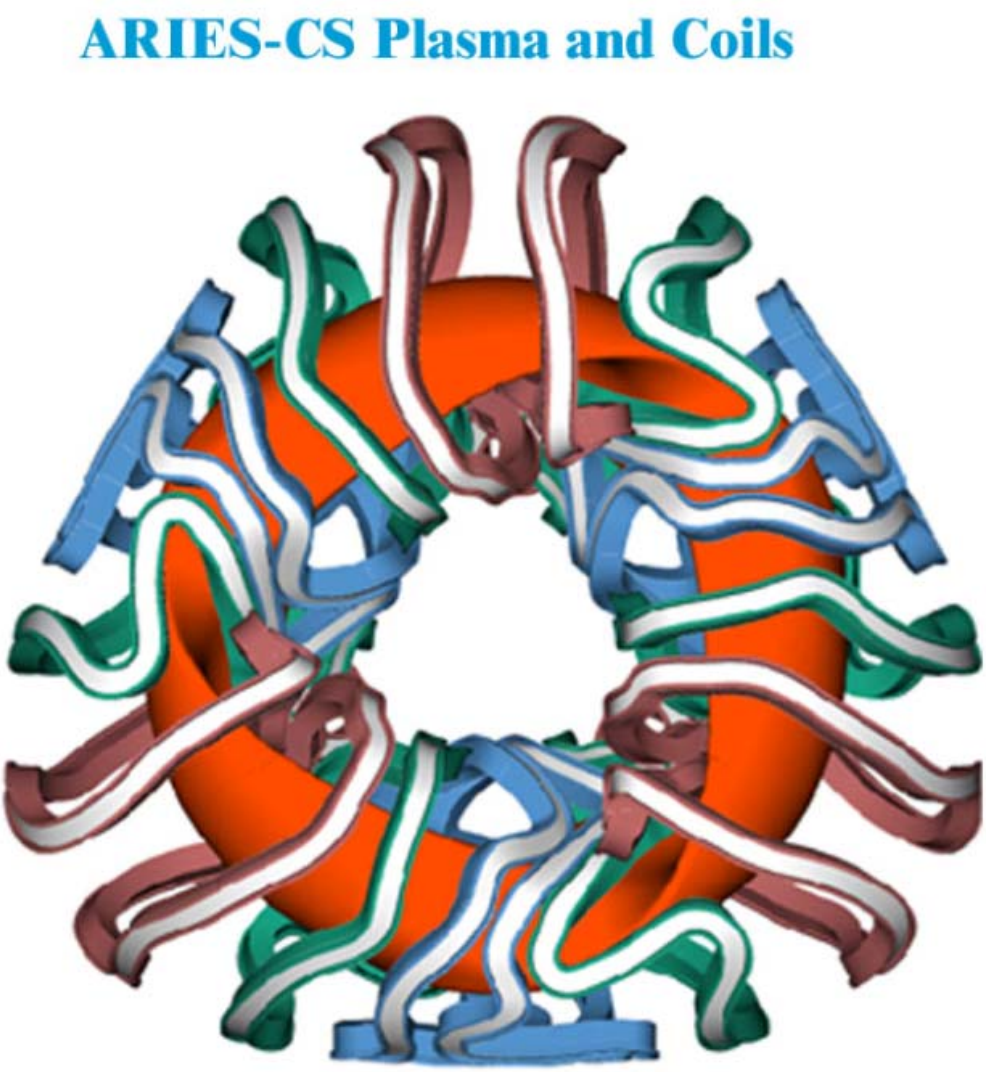


Figure 1.5. A Stellerator magnetic fusion device

In order to design a Stellerator fusion power plant, multiple neutronics related parameters and structures must be analyzed such as the neutron wall loading, average and peak dpa, the tritium breeding ratio, neutron induced radioactivity, magnet damage and energy/heat extraction blanket. However, because its curved surface is very complicated and is best modeled using a B-spline patch surface, the Stellerator cannot be directly modeled by current Monte Carlo transport codes.

There are also other complex devices, for example ITER, which has many (~1000 or more) cells. Usually these complex devices are designed with CAD software and can be precisely represented only in CAD software. These devices would all benefit from a Monte Carlo code for which the geometry could be read directly from CAD representations.

1.5 Motivation

The Monte Carlo method can be applied to many physical problems and there is no restriction on the geometric configuration. Modern computational geometry tools with all there extensions can deal with complicated geometric configurations and provide the analytic properties of them. If we merge the Monte Carlo method and modern computational geometry generation tools, we can apply the Monte Carlo method to physics problems with complicated geometries

However, usually an approximate model is used to substitute for the real geometry because it is not easy to model complicated geometries in current Monte Carlo codes. In addition the computational performance is low when Monte Carlo codes are running on a complicated geometry.

We can also take advantage of work performed in the “computer graphics” area involving “ray-object intersection” algorithms. High performance “ray-object intersection” algorithms

can speed up the execution time of the Monte Carlo codes. Thus, when we obtain the benefit of working on a physics problem with complicated geometry, execution time will not need to be sacrificed.

Because of the difficulty of modifying existing Monte Carlo codes, individuals have focused on writing GUI (Graphical User Interface) driven geometric converters. Converters read and analyze the geometry, rewrite the geometric configuration in Monte Carlo code's geometry primitive types and save it in the Monte Carlo code's input file format. This work helps the user when using existing Monte Carlo codes but it does not add or extend the geometric functions in Monte Carlo codes. Therefore it still has the geometric restrictions that are part of the current Monte Carlo code, such as missing facet base surface representation capabilities, missing high order curve surface and missing ray-object intersection acceleration techniques.

My research project is to:

1. Merge the Monte Carlo method and CAD based geometry engine by performing the Monte Carlo simulation directly on the CAD geometry model.
2. Take advantage of ray-object intersection algorithms in computer graphics to speed up the ray intersection inside the CAD geometry engine.
3. Make use of a generic geometry interface so that different types of geometric models (solid models, facet-based, etc.) can be evaluated through the same interface in Monte Carlo codes.

This research will be performed within a general purpose Monte Carlo code, MCNPX, which can be applied to complicated geometric configurations and has a reasonable execution time.

Chapter 2

Literature review

In this chapter, we will review the current research on how to improve upon the geometric limitations of current Monte Carlo codes. First, taking advantage of CAD software is a common solution. However, there are two approaches to take advantage of CAD software. One is a converter approach and the other is CAD based Monte Carlo transport approach. Each has its own limitations. The limitations can be solved by techniques in the computer graphics area. The approachs and limitations will be reviewed in this chapter.

2.1 Monte Carlo codes

Because the convergence speed of the Monte Carlo method does not depend on the dimensions of problem, the Monte Carlo codes are fast becoming the preferred particle transport simulation tool. Besides MCNP mentioned in Chapter 1, there are many other Monte Carlo codes. EGS (Electron Gamma Shower) is a general purpose Monte Carlo code, which is famous for its coupled transport of electrons and photons. It can deal with “an arbitrary geometry and the energy of particles can be from a few keV up to several TeV” [6]. But to use it, the user needs to write a “user code” to setup the problem and input the geometric

information. The main part of the "user code" is the function "how_far". This function takes as input the current particle position and direction, and returns the distance to the boundary of a current cell along the particle direction. The advantage is that EGS4 has almost no restriction on geometry. But the disadvantage is that it requires the user not only to have the ability to write a code, but also a working knowledge of the algorithms dealing with the geometry calculation. It is also an error-prone method and is hard to debug. There is a user code called EGSNRC BEAM that allows simulation of most of the geometry associated with radiation therapy. But it will involve limitations on geometries.

FLUKA [7] is another Monte Carlo code. It transports hadrons, muons, neutrons, electrons, photons and neutrinos. The FLUKA geometry input is achieved by the "combinatorial geometry package" [7]. It provides 20 kinds of basic geometry types, and the user can combine these to generate a geometry model (Table 2.1). The advantage is the user can input geometric information much easier than for EGS. But the disadvantage is the geometric is limited to the types that the "combinatorial geometry package" provides and can construct.

Table 2.1 The geometry types provided by FLUKA [7]

Code	Meaning
RPP	Rectangular Parallelepiped
BOX	General Rectangular Parallelepiped
SPH	Sphere
RCC	Right Circular Cylinder
REC	Right Elliptical Cylinder
TRC	Truncated Right Angle Cone
ELL	Ellipsoid of Revolution
WED or RAW	Right Angle Wedge
ARB	Arbitrary Convex Polyhedron of 4, 5, or 6 sides
XYP, XZP, YZP	Infinite half-space delimited by a coordinate plane
PLA	Generic infinite half-space
XCC, YCC, ZCC	Infinite Circular Cylinder parallel to a coordinate axis
XEC, YEC, ZEC	Infinite Elliptical Cylinder parallel to an axis

In general, the geometric modeling capability of existing Monte Carlo codes, such as EGS [6], MCNP [2] and FLUKA [7], is not satisfactory. Actually the existing Monte Carlo codes focus more on the physics of the problem than on geometric modeling. The focus is on more kinds of particles, a wider energy range, more interactions, and better physics models.

2.2 CAD

CAD means using computer technology to design parts and rendering them on the computer. For example, many mechanical parts are designed in a three-dimensional computer model. The basic capabilities are representations and operations on three-dimensional geometries. The features include creating, storing and modifying a geometric configuration. Current CAD software also has some powerful features such as overlap/gap detection, mesh generation, and multi-mode three-dimensional display. Some current CAD software include ProEngineer[8], SolidWorks[9] and Unigraphics[10].

The current CAD software usually has two primary components - geometric engine and user interface structure. The geometric engine will implement the basic geometric functions such as create functions, which can create a simple geometric unit like a sphere, cylinder, plane and high order surface/object; modify functions, which can move, expand/reduce or Boolean the geometry; evaluate functions, which can obtain the topology information, test a point whether inside a body and ray-object intersection; and I/O functions which can import the geometry from the other systems and export the geometry to other systems. Some geometric engines, such as ACIS [11] and Parasolids [12], are widely used in CAD software. The user interface will be built on the top of the geometry engine. It will provide powerful interactive abilities/features and provide a user-friendly interface.

An important contribution of current CAD software was to allow parametric modeling – construct models, which can be changed by changing values of parameters. This feature

changes the design process and most complex designs are developed by this feature.

Usually CAD software uses a Boundary Representation, or BREP, to represent the basic geometry. BREP models use a boundary to construct the geometry object. BREP will involve the notion of vertex, edge, surface and cell. BREP is different with feature-based modeling where, feature-based modeling refers to construct geometries as a combination of form features like holes and slots.

In summary, the current CAD software represents the achievement made on geometric modeling. The CAD software normally focuses on the ability/feature issue; its focus is on overall efficiency and performance than the optimization of a single function.

2.2.1 CAD/Geometry Engines

The Common Geometry Module (CGM) is a "code library, which provides geometric functionality used for mesh generation and other applications" [13]. CGM is built upon the ACIS geometric modeling engine, but it includes additional geometry capabilities beside those in ACIS. ACIS is a solid modeling engine used by hundreds of software developers in many industries worldwide, including CAD/CAM/CAE. Therefore, CGM is compatible with software that is compatible with ACIS. CGM can be used as a code library to provide geometric functions, and it also allows the geometric model to be used as the basis of another application, such as mesh generation.

The geometric functionality of CGM includes that commonly found in solid modeling engines of CAD software, like geometry creation, query and modification. CGM also includes capabilities not commonly found in solid modeling engines, like geometry decomposition tools and support for shared material interfaces. The geometry functionality not found in ACIS includes non-manifold geometry and virtual geometry. One example of "non-manifold geometry" is a single surface shared between volumes. Therefore, CGM is not simply a wrapper on ACIS; it is rather a set of tools providing added capabilities on top of ACIS, an interface to ACIS and other solid modeling engines and a mesh-based representation, providing facet-based surfaces.

2.2.2 CAD Tools

We will use CUBIT as an example of CAD tools. CUBIT is a two- and three-dimensional finite element mesh generation toolkit for solid models [14]. CUBIT provides a lot of geometry capability with a user-friendly graphics interface (see Fig. 2.1).

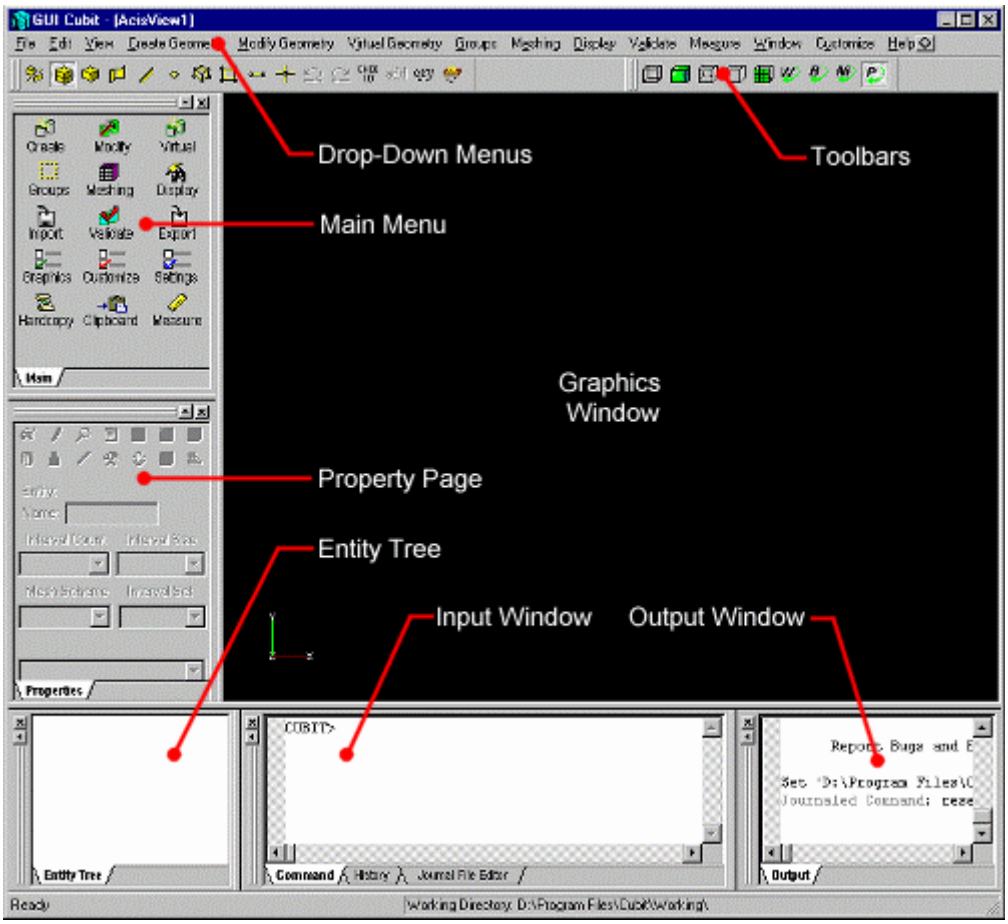


Figure 2.1. CUBIT user interface

CGM (Common Geometry Module) provides most of the geometry functionality required by CUBIT. Therefore CGM can be considered as the geometric engine of CUBIT.

2.2.3 The Imprint/merge process

CUBIT provides two important features: the imprint and merge process. By these two processes we can convert manifold geometry into non-manifold geometry. If we still use the example of non-manifold geometry as a single surface shared between volumes, the manifold geometry is two volumes having partial or total overlapping boundary surfaces. What imprint does is find the overlap portion for both volumes. It splits and generates surfaces for both

volumes, which are totally overlapped. What merge does is merge two totally overlapped surfaces into a single surface shared by volumes, which is a non-manifold geometry.

CUBIT provides graphical interaction with the model, which is important to verify results of the imprint/merge process. CUBIT can also save and restore a geometry model in an imprinted/merged state. CGM, as the geometric engine of CUBIT, can also restore the exact state saved by CUBIT. Therefore any application of CGM has this feature.

2.2.4 Faceting algorithm

Usually CAD software provides the functionality called faceting algorithm. The algorithm is to generate a set of triangles or facets, which approximates the surface. The most common application of this algorithm is graphics display. Because it is difficult to display all kinds of curve surfaces, it is easier to just display the triangles. When many facets are used, the faceted surface can be a good approximation of the real curved surface.

This algorithm is standard in the CAD field and provides a parameter to control how fine the faceting surface is. The idea of using a faceted surface to represent a real surface and then use unified display techniques on any surface is very important. The faceted representation can also be used as an approximation of a surface for other purposes including ray tracing. This is discussed in more detail in a later section.

2.3 Converter approach

To take advantage of CAD software in a Monte Carlo code, the most straightforward idea is to write a converter, which translates the CAD representation to geometry input accepted by the Monte Carlo code. This approach is depicted in Fig. 2.2:

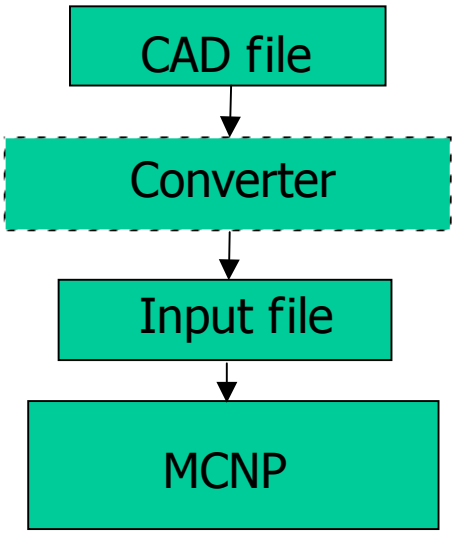


Figure 2.2. Converter approach diagram

The converter usually provides the feature that defines materials and material properties of cells.

2.3.1 GUI converter

Schwarz, Carter and Manke [15] developed a visual editor of MCNP (see Fig. 2.3). This editor has a GUI interface and the user can create and modify the geometry. After that, the user can save the geometry in MCNP input format. The latest update is a CAD to MCNP conversion tool, which currently works on circles and planes and arcs. It can read two-dimensional AutoCAD (dxf format) and three-dimensional ACIS geometry (SAT format)

then convert and save it in MCNP format.

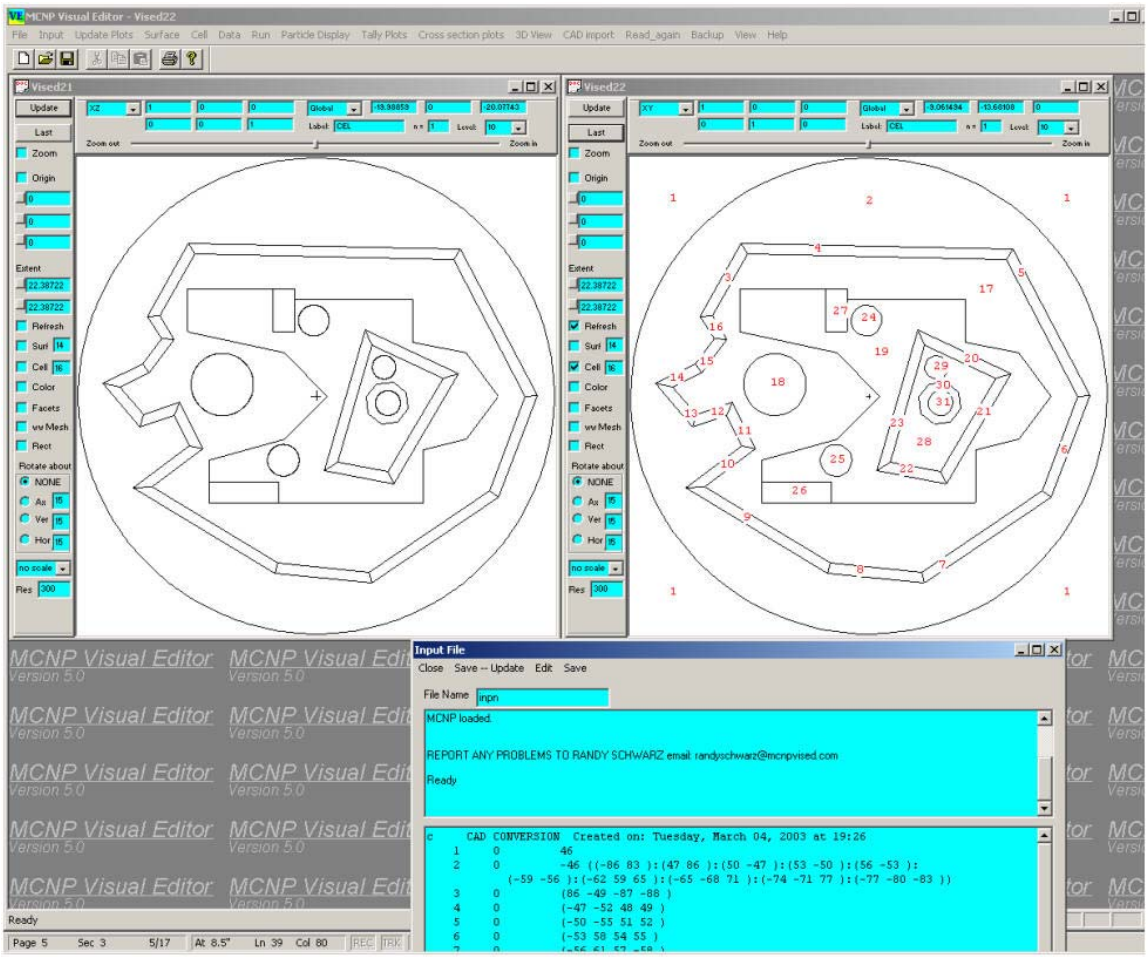


Figure 2.3. MCNP Visual Editor and Converter

2.3.2 MCAM 3.0

Wu [16] and his team developed MCNP Auto- Modeling Tools called MCAM. It is a CAD tool. The user can generate the geometry with MCAM and MCAM can exchange data with other CAD software like STEP, IGES or ACIS (.SAT) formatted files. Then MCAM can convert the CAD geometry model into a MCNP geometry model. Like CGM, MCAM is also

based on the ACIS geometric engine, however it only uses the ACIS engine to translate ACIS geometry into MCNP format.

2.3.3 Other converter

TOPACT[17] (Automated Translation from CAD to Combinatorial Geometry) is developed by Raytheon. TOPACT can automate the translation of CAD geometry to combinatorial geometry representations used by some Monte Carlo codes, such as MCNP. It is the most recent of converter-based approach efforts.

H Tsige-Tamirat also has a tool called McCAD[18] to convert CAD models into MCNP models. It was used with MCAM together in A Serikov's work [19] in ITER.

2.3.4 Limitation

The problem or limitations of an editor or a converter is they need to be updated to stay abreast with the rapid progress of CAD software and new geometry types and renderings. In addition, they did not remove any geometry limitations of the Monte Carlo code. Even if current CAD software supports many new geometry types, the converter converts to limited MCNP geometry types. Therefore this conversion is not a loss-less conversion, which means that the converter will perform an approximation to change advanced CAD geometry types to primitive Monte Carlo geometry types. There will be an algorithm to perform these approximations. Usually it will use many small analytic surface pieces to represent the CAD

surface. It is hard to decide which algorithm can achieve a better approximation. Even if a user puts that approximate geometry into a Monte Carlo code, the Monte Carlo code will test each small piece for collisions and the computational performance will be low. And this approximate model is different from the CAD model. The differences are called translation artifacts and are inevitable for this translation approach. The CAD – Monte Carlo geometry is different from CAD – CAD geometry translation. Because CAD supports advanced geometry types, we can represent complicated geometries on either CAD platform with minor translation artifacts. This translation is still difficult and is the subject of much work. Fortunately, this work is included in the CAD geometry package.

The converter approach is limited by the modeling limitation of the Monte Carlo code. Although converters reduce the user time for constructing a geometric model, they do not deal with the performance of the ray-object intersection algorithm. The computational time is still high if a converter uses many small analytic surfaces to substitute for a complex surface. And converters are also needed to overcome robustness problems and lack of automation problems.

2.4 CAD based Transport Approach

Another idea is to couple the Monte Carlo code directly to a CAD engine. The particle will still undergo the Monte Carlo simulation within the Monte Carlo code. For example, the source sampling, collision interaction and outcome particle, energy, direction sampling, tally

are still within Monte Carlo code portion. All other geometry-relevant calculations will be performed within the CAD geometry engine. For example, geometry configuration setup, ray surface intersection, and finding the next cell when a particle crosses the boundary are performed within CAD geometry engine.

2.4.1 ACIS based work

Franke, Kensek and Warren [5] used the ACIS solid model geometry engine as the geometry engine of a Monte Carlo code. Therefore they obtain the full geometry representation capability of the CAD software. They have implemented bounding box acceleration, a simple ray-object intersection acceleration technique. But they still suffered from the low performance of ray-object intersection function of the CAD software. The Monte Carlo code used in the work was ITS (integrated TIGER series) [20]. Without any acceleration techniques, their performance was 120 times slower than the unmodified ITS code. After implementing the bounding box, their performance improved to 30 times slower than the standard ITS code.

Franke, Kensek and Warren's work proved the applicability of CAD based transport approach. However, performance is a major problem. They have not published an updated paper on their performance improvements.

2.5 Computer Graphics

The term “computer graphics” describes any use of computers to create or manipulate images. The major techniques include modeling, rendering and animation. The modeling deals with the “mathematical specification of shape and appearance properties in a way that can be stored on the computer” [21]. Rendering deals with the creation of images from three-dimensional computer models. Animation deals with the creation of an illusion of motion through sequences of images.

“Ray tracing” is an important method to generate the image. The basic idea is to trace the light ray from the light source. The ray will experience reflection and refraction as it encounters material interfaces. If the ray intersects and hits the eye, this means that there is a visible point. All visible points will compose the image. “Ray-object intersection” is an important topic in “ray tracing” because only when a ray hits an object can reflection or refraction occurs. A ray is a vector in space. It includes an origin and a direction. An object is a three-dimensional object in the space. The “ray-object intersection” will answer the question “Will this ray hit the object?” and “If it is hit, how long will this vector travel?” These are basic questions and people use “ray-object intersection algorithm” to solve them. A lot of high performance algorithms exist and this area is still undergoing rapid development.

The “ray-object intersection” algorithms are also useful for the Monte Carlo method. As the

particle travels in a straight line, it can be considered a ray. Therefore the "ray-object intersection algorithm" can be applied to "find the distance to boundary" within a Monte Carlo code. Monte Carlo codes are very sensitive to the computational time of this function, since it is used extensively in computing particle histories.

2.5.1 Ray-Object intersection

The basic techniques of "ray-object intersection" are the same as the ones in computational geometry, which is to find the first object hit by a given ray with given position and direction. However some advanced ray-object intersection acceleration techniques have been invented in the area of computer graphics. That is because efficiency is the greatest challenge in ray tracing. However, there is a slight difference in ray tracing as applied to the Monte Carlo method. In the standard ray tracing usage, a ray would only interact with an object at a surface (reflection and refraction) and an image point needs to be tested to determine whether or not the point is in the shadow area of the other object. In a Monte Carlo analysis, the particle may interact with the medium, in which case its intersection with the next surface is not needed. Therefore some ray tracing techniques need to be modified before implementation within a Monte Carlo code. Arvo and Kirk classified ray-object intersection techniques as "faster ray-object intersections" and "fewer ray-object intersections" [22]. Faster ray-object intersections techniques try to optimize the actual computed intersections. Fewer ray-object intersections try to rule out intersections with actual objects. In the

following section, I will discuss the techniques including bounding volume, spatial subdivision and direction techniques.

2.5.1.1 Bounding Volume

The bounding volume is a fundamental and ubiquitous technique. "It is a volume which contains a given object and permits a simpler ray intersection check than the object." [22] This technique uses a very efficient intersection calculation to determine if there will be an intersection with objects when a ray comes near the bounding volume. When a ray intersects the bounding volume we need to check the object itself for intersection. If the ray comes near to an object and hits its bounding volume, it will increase the computation. But usually rays that come close and hit the bounding volume are only a small fraction of total rays and checking the intersection of object is computational expensive. We still obtain a significant net gain in efficiency. This approach will use many if-then loops which can not be pipelined efficiently on some old CPU hardware architectures. New CPU architectures usually come with "branch predictor" feature. With this feature, if-then loops will have much better pipeline efficiency. However, bounding volume does not decrease the number of "ray-object intersections", it is only a quicker way of determining if a ray hits an object.

Decreasing the number of "ray-object intersections" is the purpose of hierarchical bounding volumes. Rubin and Whitted [23] introduced this concept and the time complexity is logarithmic in the number of objects instead of linear. Hierarchical bounding volume

encloses a number of bounding volumes within a larger bounding volume. If a ray does not intersect the outer parent volume, we do not need to test the inside bounding volumes or objects.

2.5.1.1.1 Distance limit of bounding volume

An advantage of a bounding volume is if a point of intersection has been found with an object, all objects or bounding volumes which intersect the ray beyond this bound can be ignored. Therefore an intersection infers an upper bound of the distance. When a ray hits the bounding volume, if the distance is greater than the bound, we still do not need to test the content of bounding. This will reduce the computational time.

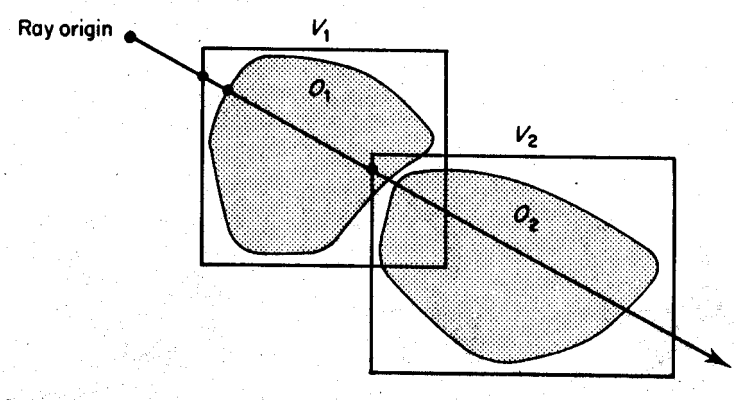


Figure 2.4. Bounding Volume[22]

Figure 2.4 depicts this situation. If the intersection with object O1 is found first, the contents of volume V2 need not be tested. In computer graphics, the upper bound can be obtained only by calculating the ray object intersection because the ray is an infinite ray. However in a Monte Carlo simulation, there is an upper bound, which is the collision distance. The

distance from the ray origin to the intersection point is to be compared with the distance to collision. If the distance to collision is larger, we need to move the particle to the boundary, which is the distance to intersection. However if the distance to intersection is larger, we only use distance to collision. Therefore the distance to collision is an automatic upper bound of the distance to intersection. In the best scenario, even though a ray may hit many bounding volumes, if the distance to collision is less than the distance to the nearest bounding volume, we do not need to test any content (object). This scenario applies to the case in which the mean-free-path is very small. To take advantage of the distance to collision is a new strategy in nuclear particle transport Monte Carlo simulations because a particle can go inside of an object. We call this algorithm “distance limit”.

2.5.1.1.2 Shape of bounding volume

If the bounding volume is a rectangular parallelepiped, it is called a bounding box. The bounding volume can also be a sphere or any other kind of geometric form. There is a trade-off between two competing factors: tightness of fit and cost of intersection. If a bounding volume tightly encloses the object, we will decrease the case that a ray hits the bounding volume but does hit the object. It will save computational time by reducing the number of ray-object intersection calculations. But a tight bounding volume will involve a complicated bounding volume and result in a higher cost of ray-bounding volume intersection calculations. Although some computational techniques exist like lookup table to speed up, it will involve approximations.

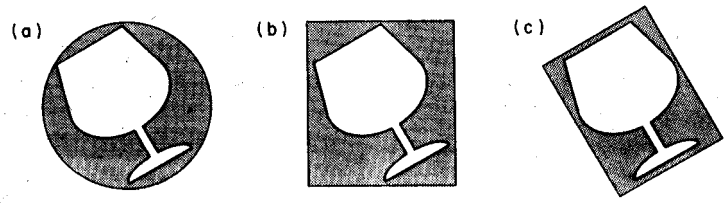


Figure 2.5. Different Bounding Volumes[22]

Figure 2.5 shows the different cost/fit ratio. Volume (b) gives the worst fit, however the bounding surface is an axis-aligned plane, which provides a fast ray-bounding intersection calculation. Volume (a) gives a better fit, but a spherical surface is a second order surface and the ray-bounding intersection is more costly than for an axis-aligned plane. We can use look-up table for square root function to speed up, but look-up table function is approximation function. Volume (c) gives the best fit, but the bounding surface is a transformed plane, which will involve trigonometric functions. According to Arvo and Kirk [22], if the object is complex, the additional cost of ray-bounding can be paid back by a significant reduction in number of ray-object intersections calculations. Case (c) brings forth the idea of OBB (Oriented Bounding Box), this will be discussed more in later section. Note these bounding surfaces work are ideal for individual particle track length simulation. Other routines and models will need to be employed for simulations such as the condensed model for electron transport.

Furthermore, we can use multiple bounding volumes for a single object. This can result in a better fit, but it also increases the cost of testing the bounding volume.

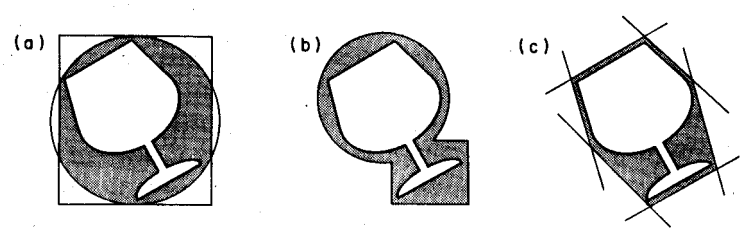


Figure 2.6. Multiple Bounding Volumes[22]

Figure 2.6 depicts this situation. For a ray, we need to test two bounding volumes ((a) and (b)) or 6 planes (c) for an object. The OBB tree algorithm addresses cases where there are a number of ray bounding surface tests as in case (c). This will also be discussed more in section 2.5.1.4.

If a particle is inside the object, the bounding box of the object will fail because the ray will always hit the object. However if we apply the bounding box to a surface, this algorithm will work again. This issue will be discussed in Chapter 4.

2.5.1.2 Spatial subdivision

Three-dimensional spatial subdivision is based on the observation that “the further an object is from the path of a ray, the less work we can afford to do in eliminating it from consideration” [22]. It is a kind of divide-and-conquer approach that divides the space surrounding the objects and finds the good candidates for intersection. Therefore this method can obtain “fewer ray-object intersections”.

We need to partition a volume bounding the environment into non-overlapping cuboids pieces.

Each piece will be labeled as totally or partially or not occupied by objects. Therefore a pre-processing step is required. The only objects that need to be tested are those that intersect the piece pierced by the ray. If we process the pieces in the order in which they are encountered along the ray, we need not test any remaining pieces when an intersection has been found.

2.5.1.2.1 Nonuniform subdivision

The spatial subdivision algorithms include nonuniform subdivision and uniform subdivision. The nonuniform subdivision means that the sizes of cuboid pieces are variable and cuboids pieces can be adaptive to the object. Usually the octree is used in this subdivision. Octree means subdividing the rectangular volumes into eight subordinate octants until the leaf cuboids meet some criterion.

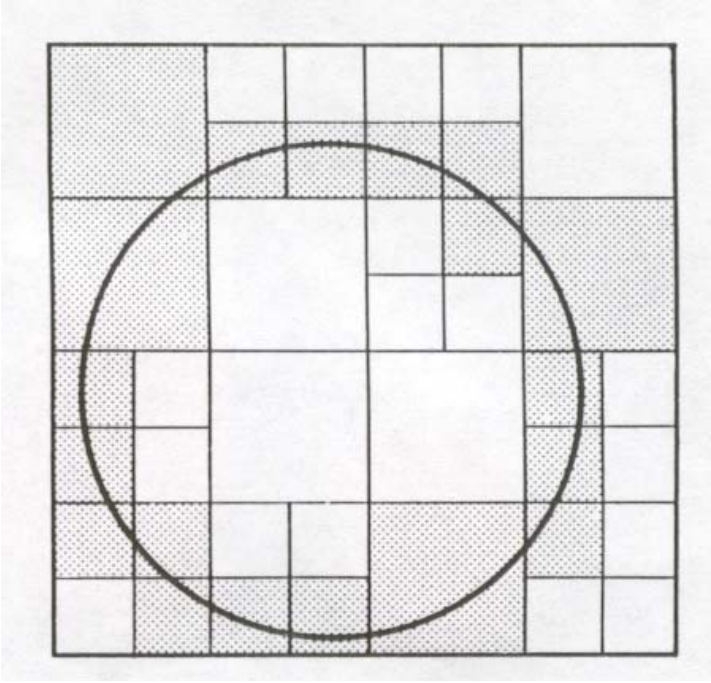


Figure 2.7. Nonuniform Spatial Subdivision[22]

Figure 2.7 shows a two-dimensional analogy of a sphere and the octree. The shaded areas are the candidate list of ray-sphere intersection.

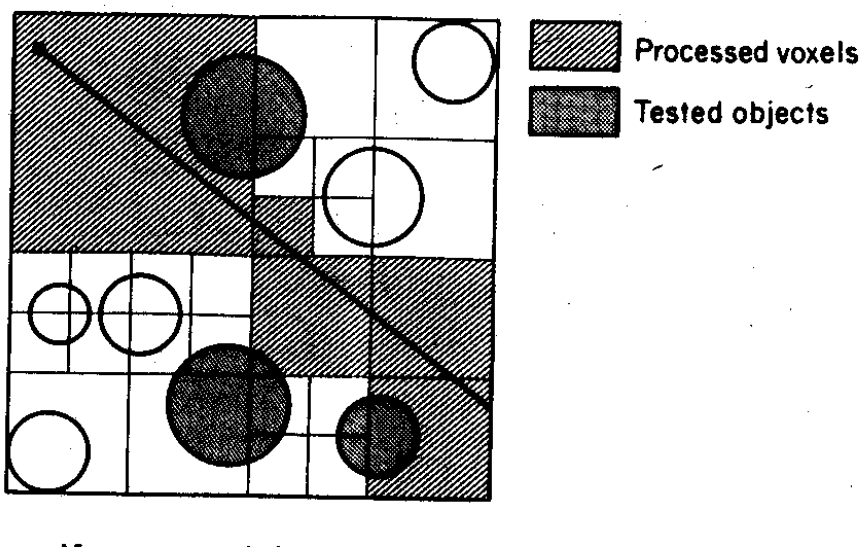


Figure 2.8. Using Nonuniform Subdivision to Accelerate Ray-Object Intersection Calculation[22]

Figure 2.8 shows an example of the nonuniform spatial subdivision implementation. First, we need to construct the octree. The criterion used were subdividing the cuboids that have two or more intersection candidates and subdividing no more than three levels deep. By this subdivision process cuboid pieces are obtained. Second, we process the ray and obtain the cuboids that the ray crosses. These cuboids are shown as shaded cuboids in Fig 2.8. Third, we obtain the objects that intersect the shaded cuboids. These objects are shown as shaded objects in Fig 2.8. These shaded objects are the objects that are tested for intersections. It is

clear to see that we only need to test 3 of a total 8 objects.

In the Monte Carlo simulation, after the distance to collision is obtained, we know the beginning point and the end point of a ray, therefore there are even less cuboids to be tested than for the infinite ray case.

2.5.1.2.2 Uniform Subdivision

Fujimoto [24] introduced a different spatial subdivision approach using cuboid pieces that are uniform in size. Each cuboid piece will be marked as "empty" or "full" or "part occupied" during pre-processing. The spatial subdivision is easier than nonuniform subdivision. But there will be more cuboid pieces in the space, which requires more memory, and the ray will cross more pieces.

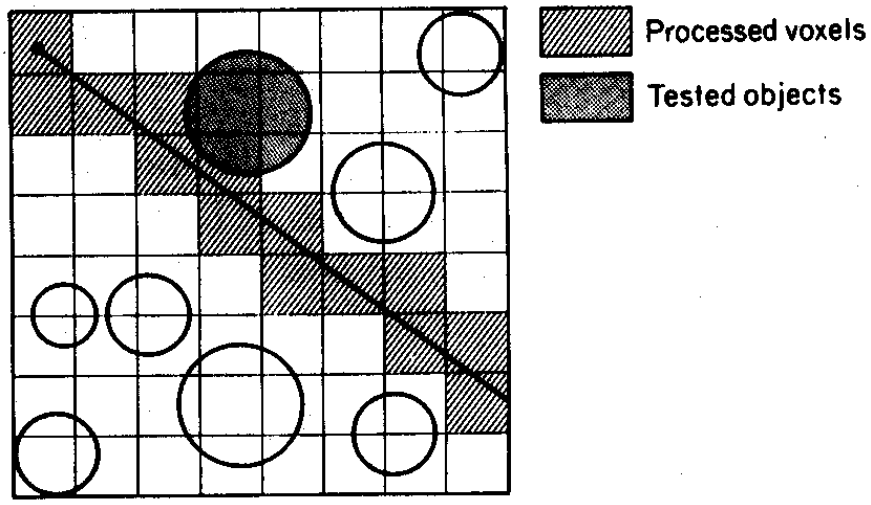


Figure 2.9. Uniform Spatial Subdivision[24]

Figure 2.9 shows the same environment and objects as Fig 2.8. By using the uniform spatial

subdivision process, the number of cuboid pieces increases from 25 to 64; the number of cuboids requiring testing increases from 5 to 14. However, only one object needs to be tested.

2.5.1.3 Direction Techniques

The directional technique is a recent category to emerge in ray-object intersection techniques than the first two algorithms. It exploits the direction information at a level above that of individual rays. The purpose is trying to eliminate the consideration of objects that are not in the direction of the rays.

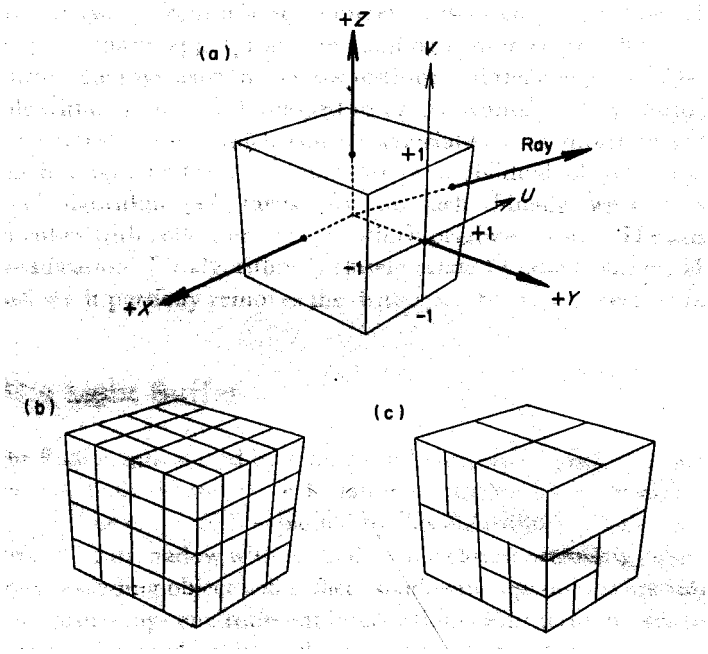


Figure 2.10. Direction Cube[25]

Figure 2.10 shows a direction cube which is used in the light buffer algorithm [25]. A direction cube is an axis-aligned cube centered at a ray origin with an edge length of 2. The ray will hit one of these six surfaces. In Fig 2.10 the ray hits the plane y=1. We call the +Y axes as the “dominant axes” of the ray. The hit point also generates two-dimensional

coordinates on the plane y=1. The dominant axes and two-dimensional coordinates can represent a ray's direction.

The direction cube allows easy subdivision of the solid angle. The surface of the direction cube can be uniformly subdivided ((b) in Fig 2.10) or nonuniformly subdivided ((c) in Fig 2.10). For a nonuniform subdivision, the quadtrees or BSP (Binary Space Partitioning) tree can be applied. Each rectangular on each plane represents a solid angle which is a "direction pyramid" in three-dimension.

2.5.1.3.1 Light buffer

Haines and Greenberg [25] introduce the light buffer algorithm for shadow calculation of a point light source. But it can be directly applied to a Monte Carlo code for the point source case. We only need to preprocess the direction cube, find the intersection candidate list of each direction pyramid. In the Monte Carlo computation, after we sample the direction of a source particle, we can find the corresponding direction pyramids. The intersection candidates are then known. The nonuniform direction cube subdivision applies to the non-isotropic point source or non-uniform distribution of objects that can be intersected; but we need to determine the subdivision, which best fits the source direction distribution or object distribution. The uniform direction cube subdivision can be applied to the isotropic point source and for the first collision, though the direction pyramids may not have the same solid angle.

The pre-processing step will require some computational overhead (time). But this is not a big problem. For the case of a non-isotropic source, some direction pyramids will have no particle going through them. We will waste memory storing it. The solution can be “lazy pre-processing”. This means we will process the direction pyramids when a real particle goes through it. This method will allow the use of the uniform direction cube subdivision technique for the non-isotropic source case.

The restriction of the light buffer algorithm is that it can only be applied to a point source and only for the first collision. That is because there are infinite points in a cell and it is impossible to construct a direction cube for infinite points. The ray coherence algorithm solves this problem and it can also be applied to cell source.

2.5.1.3.2 Ray Coherence

Ohta and Maekawa [26] introduced the “ray coherence theorem” in 1987.

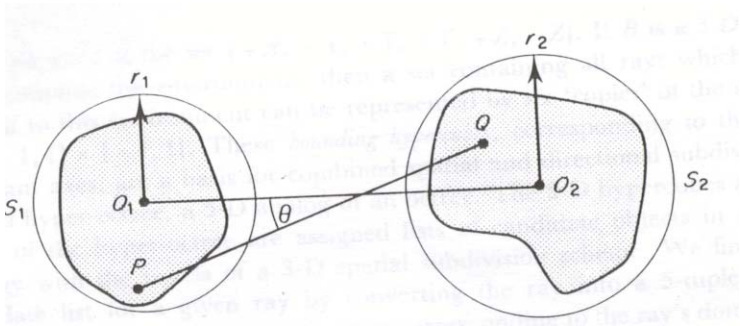


Figure 2.11. Ray Coherence[26]

Figure 2.11 shows the ray coherence algorithm [27]. There are two objects. S1 and S2 are

the spherical bounding volumes of these two objects. O1 and O2 are the center of the bounding spheres. r1 and r2 are the radii of the bounding spheres. Any ray that begins in a point in S1 and ends in a point in S2 will define an angle (θ) with the line through the sphere centers. The maximum of θ will satisfy:

$$\tan(\theta) < (r1+r2)/||O1-O2||. \qquad (1)$$

The meaning for a cell source is; if the sampled angle is greater than the θmaximum, we do not need to test the intersection. We note for real applications, there may be many objects, each object will construct a θmax with respect to the source cell. Therefore we will construct a spatial direction cube of the source cell, each direction pyramid will store the intersection candidate list. Here the candidate list means for a source particle with its direction inside the current direction pyramid, if the particle can hit an object when the origin of this particle is variable inside the source cell, the hit object will be put in the candidate list of the current direction pyramid.

We note that not only can a particle go from source cell to another cell, the particle can go between any two objects. Therefore we can construct a direction cube with any object. However, the storage requirements are large, therefore it is a typical space-time trade off. Fortunately, the Monte Carlo codes are usually execution time bound, not memory-bound.

2.5.1.3.3 Ray Classification

Another algorithm can be considered as a synthesis of spatial subdivision and direction techniques. It is called “ray classification” algorithm and was introduced by Arvo and Krik [22]. This algorithm is based on the observation that a three-dimensional ray has five degrees of freedom. Therefore a ray is a point in five-dimensional space. We can subdivide this space into many non-overlapping five dimensional cells. A cell encapsulates the space similarity and direction similarity of the ray in three-dimensional space. We can place the intersection candidate list into the five-dimensional unit cell by pre-processing.

A five-dimensional cell is hard to understand and hard to illustrate. Figure 2.12 depicts the two-dimensional and three-dimensional explanation of the ray classification algorithm.

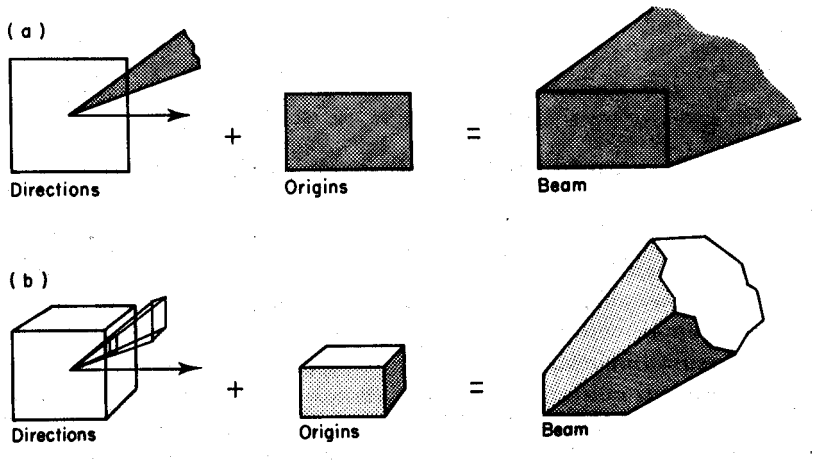


Figure 2.12. Ray classification[22]

Figure 2.12(a) depicts the two-dimensional case. The set of rays with the same origin but directions inside some range describes an angle. But when the origins of the rays are varied, the set become the shape of a beam. The three-dimensional case is shown in Fig 2.12(b). The set of rays with the same origin but directions inside some range describes a direction

pyramid. But when the origins of the rays are varied, the set becomes a three-dimensional beam. A ray inside this three-dimensional beam is a point in the five-dimensional cell. The candidate list is all objects, which intersect this beam.

The ray classification algorithm is similar to the ray coherence algorithm. They both change the direction pyramid into a three-dimensional beam. The difference is the starting region of the three-dimensional beam. For the ray coherence algorithm, the starting region is in the object; while for the ray classification algorithm, the starting region of the three-dimensional beam can be any cuboids in a three-dimensional space. Therefore the cuboids can be inside of an object, which gives the object a further subdivision.

2.5.1.4 OBB and OBB tree

The idea of oriented bounding box (OBB) was mentioned in Fig 2.5 (c). The algorithm is to find a space oriented bounding box that best fits the object. This is depicted in Fig 2.13. The OBB computational procedure is discussed in Chapter 4.

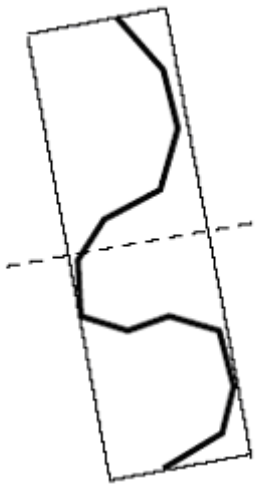


Figure 2.13. The OBB

The oriented bounding box (OBB) tree [28] is an extension of OBB. The algorithm is to apply the hierarchical bounding box process to OBB. The idea of the OBB tree is shown in Fig 2.14.

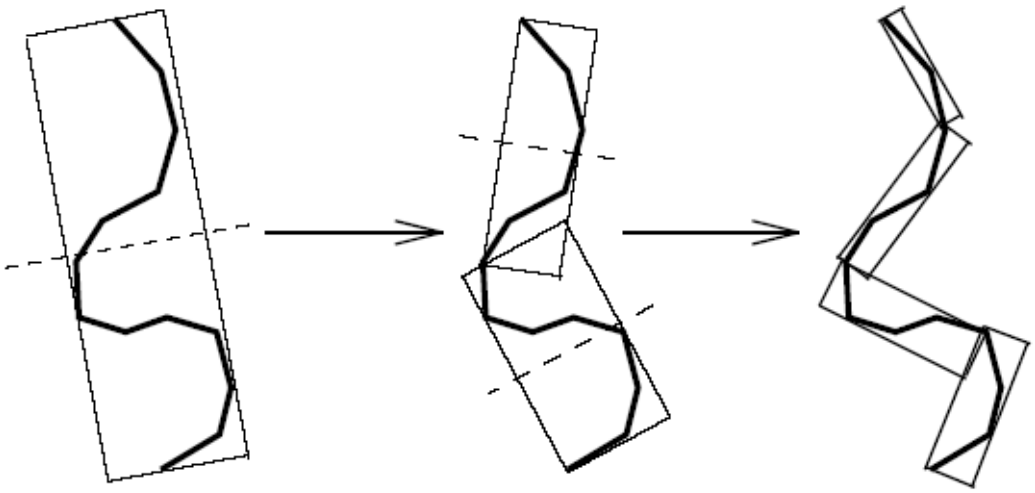


Figure 2.14. The OBB tree[28]

The OBB and OBB tree were developed in the computer graphics field for collision detection. Collision detection is a test performed to detect if there is contact between two spatial objects. The typical approach is: 1) any object in system is composed of a list of polygons, or in simple form, a list of triangles; 2) build an oriented bounding box for each object and 3) at runtime, traverse the tree of each object to test the overlap, if leaf node bounding box is still overlapped, directly test the object (triangle) itself.

In Cottschalk, Lin and Manocha's paper [28], they developed a software package to prove the OBB tree algorithm called RAPID (Rapid and Accurate Polygon Interference Detection). This

software is for collision detection purposes in computer graphics. It is to detect whether there is contact or overlap between two objects. Each object is inputted as a list of polygons. This software generates an OBB tree for each object and performs collision detection between the objects. Bounding box – bounding box test performs the collision detection. To begin it uses a root bounding box for both OBB trees of objects. When there is contact, it will go through the tree until it reaches a leaf node. In the case contact exists, it will return the contact pair of leaf nodes.

Cottschalk, Lin and Manocha's group is continuing research on this algorithm. Their major focus is on development of a fast tree building algorithm. The reason for this lies in the application of computer animation. In animation the shape of an object changes; for example, when a man is walking, the body shape changes. Therefore, from time to time, the OBB tree needs to be rebuilt.

This algorithm can be applied to surface objects found in Monte Carlo particle transport and therefore it can be used to accelerate the Monte Carlo simulation. The only problem is usually the object modeled is not a polygon object.

The OBB tree plays an important role in my research because it can be applied to any geometry and can achieve the tightest fit bounding box. Further review and discussion will be in Chapter 4.

2.5.1.5 Comparison and Combination

All these ray object intersection acceleration algorithms have storage and time trade-offs. They all store the information in a pre-processing step and this information helps the ray object intersection during execution time. The more information we store, the more help we obtain but the more storage space is required. Therefore the storage space will be a key aspect to consider during implementation.

Besides the storage space consideration, these three groups of ray object intersection acceleration algorithms focus on different aspects. The spatial subdivision algorithm focuses on space, especially the space pierced by the ray. If the space is sparse this algorithm would not outperform the bounding volume because it will deal with too many empty space cuboids. The bounding volume algorithm focuses on objects; if there are a large number of objects this algorithm must deal with too many bounding volumes that are not intersected. Therefore it would not outperform the spatial subdivision algorithm. The direction techniques focus on the direction of the ray. The idea of this algorithm is the ray with different direction will hit different objects. If there are a number of objects for which each inside object is encircled by an outer one and the ray is from the innermost object, this algorithm will totally fail.

Snyder and Barr [27] compared the performance of uniform three-dimensional spatial subdivision, octree-based nonuniform subdivision and bounding volume hierarchies. Each

algorithm has its favorite scenarios. For example, they observed that for large numbers of homogeneously distributed objects of similar scale, a regular grid outperforms octree methods because for a regular grid, voxel walking is more efficient. And due to large number of homogeneously distributed objects of similar scale, there are many voxel walking calculations.

Also, because these ray-object intersection algorithms focus on different aspects, they can be combined together to achieve a performance that is better than any single one. For example, the bounding volume method can be combined with direction techniques that put the direction cube in the volume. Then a ray coming out of the object can use the candidate list of direction cube. Actually, the ray classification algorithm can be considered as the combination of spatial subdivision and ray coherence. In combining optimizations, the storage space will be more important because combining optimizations will require an increase in memory greater than the usage of any single one. Kirk and Arvo [22] give a general mechanism for combining optimizations. They encapsulated acceleration techniques then present the same interface for pre-defined primitive objects. Now an acceleration technique becomes an aggregate object. Because they have a uniform interface for all aggregate objects, they can create meta hierarchies that include a cadre of algorithms like octree, uniform grid and bounding volume hierarchies. Their work focused on how to choose or compare from different combinational approaches. Their goal was to achieve the best performance in a complicated environment that contains millions of objects. From Kirk and Avro's work one notes that research on combinational approaches requires thorough research on all single

algorithms.

2.6 Summary of previous work

To summarize all the previous work, to take advantage of CAD software is the right approach because it can fully take advantage of the powerful features, tools and shape rendering in CAD software. A CAD substitution would get rid of geometry modeling limitation of current Monte Carlo codes. There are two approaches on how to utilize CAD; the converter approach and the CAD based transport approach. In the converter approach, the Monte Carlo simulations still suffers from the limitation on geometry modeling and the computation performance of the original code. In the CAD based transport approach, the geometric modeling of the simulation has been improved, however, computational performance becomes the main issue (problem) and the ray tracing function becomes the bottleneck.

In the field of computer graphics, there are many ray-object intersection acceleration techniques. However, ray tracing acceleration algorithms in computer graphics need to be adapted for use in particle transport Monte Carlo. The major differences between ray tracing and Monte Carlo transport are:

(1) A particle track can pierce into an object while a ray will not.

(2) When a particle is inside an object, it has the possibility of colliding with any boundary segment. We can obtain the distance to collision from physics sampling. This will be a benefit for most ray-object intersection algorithms.

(3) In computer graphics, there is an environment and objects. A ray only traverses the environment, which is outside of the object. However in a Monte Carlo particle transport, the rays only travel inside a cell, though it may be an empty cell.

(4) In ray tracing, people prefer a large number of objects (for example 1000 or more) but each object is simple (such as a triangle, sphere or box). But in Monte Carlo transport, there are also cases that there are not many objects (for example less than 100), however each object is composed of many surfaces (for example 100 surfaces). Therefore in a simulation, there is interest in not only, which object the ray intersects, but also which surface of the object is hit.

(5) In computer graphics, the distance to an object needs to be known. In Monte Carlo simulations, the distance to surface intersection is compared with the distance to collision. In the case where the distance to the collision is smaller, the exact distance to a surface intersection is not necessary.

Chapter 3

Overall Approach and implementation

In this chapter, our approach and implementation will be described in detail. It includes the motivation and the benefits of our approach, the implementation of the CGM geometry engine, changes to the MCNPX code and the difference of usage of the MCNPX/CGM and the standard MCNPX codes.

3.1 CAD geometry engine based implementation and its benefits

By providing both CGM and CUBIT, we have both the geometry constructing software (CUBIT) and its geometric engine (CGM). By using CUBIT we can easily create and edit a geometry configuration. Finally we can save it in a CAD geometric format. The next question is how to combine the CAD software/engine to obtain the most benefit from the CAD capabilities.

Our approach is called “engine based implementation”. We use CGM to substitute all the geometric functions in MCNPX. The particle physics interactions are still performed within MCNPX, but the particle transport (or “ray tracing” in computer graphics terminology) is performed within the CAD engine. Because the CAD geometry engine is part of

MCNPX/CGM, the MCNPX/CGM can directly read CAD geometry files and perform the Monte Carlo simulation within it. This approach not only allows the user to easily create geometry configurations, and to provide a compatible interface between MCNPX and the CAD software, but also allows complicated geometries to be modeled with MCNPX. The structure of the MCNPX/CGM code is shown in Fig 3.1.

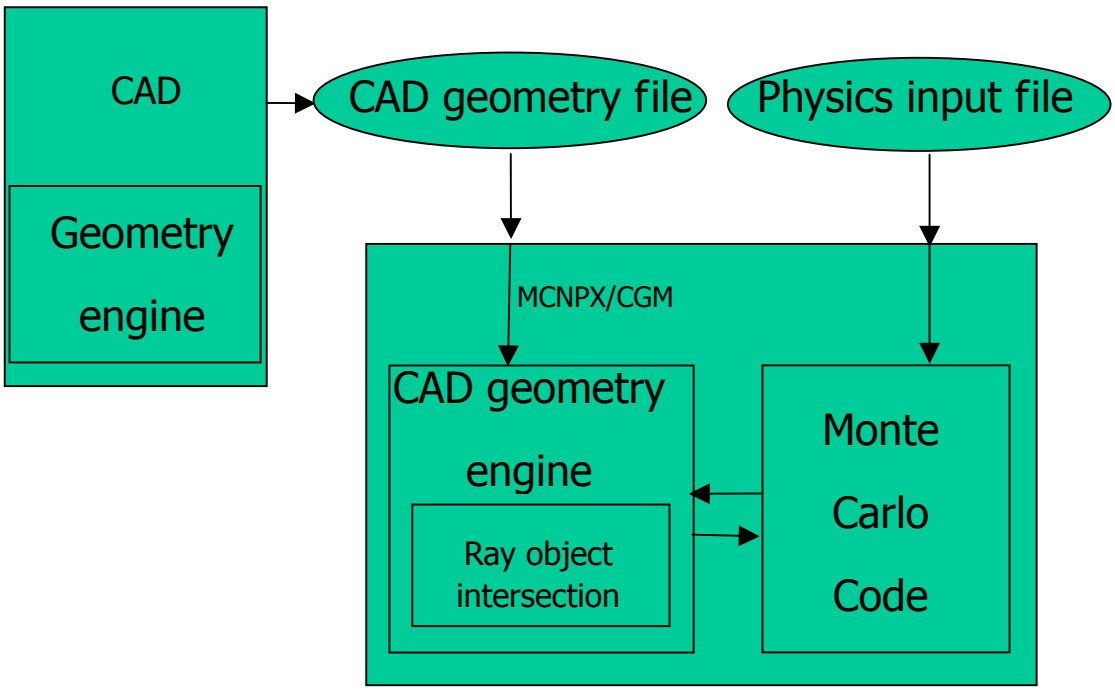


Figure 3.1. MCNPX/CGM flow chart

3.2 Implementation details

The implementation idea is to modify the MCNPX code to incorporate calls directly to CGM and to incorporate the ray-tracing acceleration techniques to speedup the code. The combined code is called MCNPX/CGM.

As stated in the introduction, the MCNPX execution can be subdivided into the initialization,

Monte Carlo simulation and tally stages. Our implementation involves these three stages.

3.2.1 Initialization and changes

There are two major changes in the initialization, one is to use CGM to read the geometric information of the problem and perform a geometry analysis. The other is to get rid of all the geometric information from the MCNPX input file.

The first one is achieved by implement a C++ main function for MCNPX. This is required when calling C++ functions. This function performs:

1. Initialization of the CGM geometry engine.
2. Reads in the geometry file. (ACIS format)
3. Geometry pre-processing.
4. Calls MCNPX to begin the Monte Carlo simulation.

The geometry pre-processing, task 3, includes obtaining the total cell numbers, total surface numbers, surface faceting and OBB tree building. The surface faceting and OBB tree building are used for ray-tracing acceleration techniques. This will be discussed in detail in Chapter 4.

The other modifications are within the MCNPX code. We need to input the geometry information into the code; like the total surface number and the total cell number into MCNPX

code MCNPX will have the cell and surface numbers, but the geometry information, like the surface type, will only be available in the CAD geometry engine portion.

After these modifications, the MCNPX part of the code will require a geometry input. In the MCNPX input file, the geometry information is left blank and the previous example input file (Table 1.2 and Fig 1.3) will become:

Table 3.1 MCNPX input file

testprob - n p
1 1 -0.675
2 0
3 0
4 0
c surface card
1
2
3
4
5
6
7
8
9

In this example the cell description section only contains material and density information. The surface Boolean information has been eliminated. In the surface description section, only the surface number is kept. No surface type and position information input is required.

3.2.2 Monte Carlo simulation and changes

In the modified Monte Carlo simulation code, all the geometric computations, such as "ray-object intersection" and "finding the cell when a particle crosses the boundary", are performed by CGM functions.

Originally "ray-object intersection" was performed by the MCNPX function "track". Now it is performed with the CGM function "cgmtrack". The ray-tracing acceleration techniques are also inside this function. This function will be described in detail in Chapter 4.

Another function being substituted for is the "finding the cell when a particle crosses the boundary" function. The MCNPX function "newcel" is substituted by the CGM function "cgmnewcel". This function will also be described in detail in Chapter 4.

3.2.3 Tally

Usually the MCNPX tally involves a specific surface or cell. In the unmodified MCNPX, the tally is assigned at some surface or cell number, for example, the current on surface 1 or, energy deposition in cell 2. Originally the surface and cell numbers were provided to MCNPX via the input file. Now all geometric information is provided to the CAD geometry engine. There are also surface and cell numbers for each surface and cell. These numbers can be passed to MCNPX during the Monte Carlo simulation. Therefore, the tally section of the modified MCNPX input file contains cell or surface numbers, which are obtained from the

CAD geometry engine or CAD software. In MCNPX, when a particle crosses the tally surface or enters the tally cell, a tally of this particle is automatically generated as in the unmodified MCNPX case. After the MCNPX simulation is completed, the modified MCNPX will generate the tally report just like the original MCNPX. The tally report format has the same format as before.

3.2.4 Code modifications

Table 3.2 presents the modifications to the MCNPX functions. Major changes mean that the whole function has been rewritten. Minor changes mean that only a few lines of the function have been commented out, changed or have been added. As can be seen, there are not many changes to original code.

Table 3.2 Modifications to MCNPX

Minor changes	Major changes
Hstory.F	Main.cpp
Imcn.F	Track.F
Mcnp.F	Newcel.F
Rdprob.F	Chkcel.F
Surface.F	
Transm.F	

Chapter 4

Run-time Accelerations

From the literature review we can see that most Monte Carlo codes do not take full advantage of modern modeling software for their computational geometry. Focusing primarily on the physics side, their current geometry engines just implement a small set of functions and do not provide complex surface modeling. In contrast CAD software implements a much greater set of functions that allow for complex surface modeling. They provide powerful, advanced geometric functions and abilities. A better approach would be to use CAD software as the geometric engine of the Monte Carlo code. It allows the new CAD based Monte Carlo code to obtain all the geometric merit from the CAD software. And some generic interfaces to CAD, like CGM, even provide some non-CAD geometry modeling capabilities like facet-based and virtual geometry. It will be shown that a CAD based Monte Carlo code can be used effectively and efficiently for complicated geometries.

However, in Chapter 2 Franke, Kensek and Warren's [5] work showed that performance will be the major problem in CAD based transport. My research will focus on this problem.

In my research, MCNPX represents the Monte Carlo code and CGM represents the CAD software. In Chapter 3 we described the implementation of MCNPX/CGM. We have already

accomplished a CGM based MCNPX. But without ray tracing acceleration techniques, MCNPX/CGM will have performance issues similar to Franke's efforts. This chapter will show the detail of several ray tracing acceleration techniques. It includes OBB tree, sorted distance and distance limit algorithm on CAD geometry models and OBB tree, sorted distance and distance limit algorithm on facet models and source particle region determination.

4.1 Ray Tracing accelerations

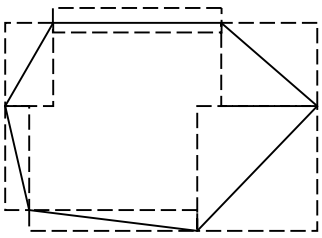
4.1.1 Bounding box for object or bounding box for surface

The bounding box is the most easily and wildly used ray-tracing acceleration technique. It uses a box to enclose a given object and a simpler check for ray intersection with the box than the object to eliminate unnecessary ray object intersection. The first decision made in this work was to use a bounding box on an object or on a surface.

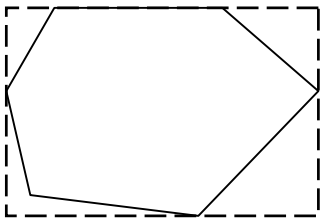
In the computer graphics field, bounding boxes on objects are typically used because in an environment, there are many objects, and a ray usually starts from the outside of the object. The use of a bounding box on an object can easily filter out objects whose boxes are not hit by the ray.

However, there are some problems with application of a bounding box. First, if a ray point is inside an object, the bounding box is useless. Because when the start point is inside an object,

the ray coming from this point will always hit the bounding box. Second, to obtain the best fit ratio, at times requires the use of a sphere, cylinder or other higher order shape as the bounding volume. In the worst case, we need to use multiple bounding volumes on one object. This not only requires a more complicated algorithm to generate bounding volumes, but also requires more ray-volume test time because a second order bounding volume surface is being used. This issue was also discussed in Section 2.5.1.1.



(a) 2D analog of Bounding on surface



(b) 2D analog of Bounding on

Figure 4.1. Bounding to surface and bounding to object

As mentioned before, the situation in Monte Carlo particle simulation is different from that in computer graphics. A particle is always inside some object and this object could be very complicated. Therefore we decided to apply the bounding box to each surface. This is depicted in Fig 4.1. Figure 4.1(a) shows a bounding box on each surface. When a particle is

inside the object, it still can take advantage of the bounding when the test is with a surface. Figure 4.1(b) shows the bounding on the object. Any particle inside the object is inside bounding box. There is no benefit in the application of the bounding test for the surface.

4.1.2 Axis-aligned bounding boxes (AABB)

The AABB is the simplest bounding box. The axis-aligned means the surfaces of the bounding box are always parallel to the x,y,z coordinate planes. Therefore, the AABB is very easy to calculate. And because the bounding box algorithm is easy to implement, most CAD geometry engines already include it in their ray-tracing algorithm.

However, the simple AABB has a limitation: because the bounding box planes are always parallel to the coordinate planes, the bounding box is sometimes not a tight fit to the object. Figure 4.2 depicts an example of this.

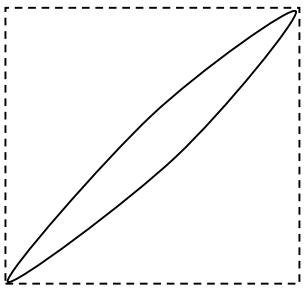


Figure 4.2. Worst case of the AABB

Because the AABB is axis-aligned, boxes can be much larger than the object, which as seen, depends on the object's orientation. The performance of the AABB will be

problem-dependent. In the computer graphics area, a non-tight fitting bounding box will have many cases where the ray hits the bounding box but does not hit object. This degrades the performance of the bounding box algorithm.

This limitation is even worse for a Monte Carlo simulation because a particle is always inside a cell. Figure 4.3 depicts the worst case of AABB on for a surface. Because the surface is a cylinder, the AABB of a cylinder enclose the object in it. The particle inside the cylinder surface is always inside the bounding box. The AABB of cylinder surface provides no benefit. In this case, AABB on surface will be same as AABB on object.

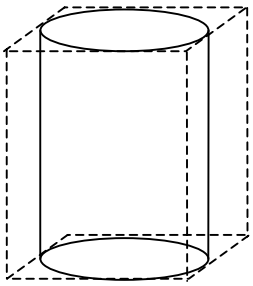


Figure 4.3. AABB of cylinder surface

4.1.3 Oriented bounding box (OBB) tree

Cottschalk, Lin and Manocha [28] introduced the OBB tree algorithm to solve the limitations of the AABB. It is widely used in the computer graphics area, especially for collision detection. There is some similarity between the collision detection application and the Monte Carlo particle simulation. The idea of this algorithm can be applied to particle Monte Carlo

transport, but the algorithm requires modification to be of better use.

4.1.3.1 Calculate OBB

The central idea of the OBB algorithm is: the bounding box will not need to be parallel to coordinates axis. The algorithm will find the x′, y′, z′ axes and construct the bounding box aligned to these new axes to make a tightly fitted bounding box to the object. The purpose of OBB is to obtain a bounding box as small as possible. However a coordinate transformation needs to be performed before a ray-bounding box intersection test is made.

The first step is to calculate the OBB for a triangular object, or in particle Monte Carlo transport, faceting object. In Cottschalk, Lin and Manocha's paper [28], present the algorithm to generate OBB as follows: suppose we have a list of triangles, numbered from 1 to n. The coordinates of the vertices of the i′th triangle are vectors p^i, q^i and r^i. If written in scalar form: $(p_1^i, p_2^i, p_3^i), (q_1^i, q_2^i, q_3^i)$ and (r_1^i, r_2^i, r_3^i). The 1, 2, 3 here represent the three coordinate directions. The mean, μ, is equal to:

$$\mu_1 = \frac{1}{3n}\sum_{i=1}^{n}(p_1^i + q_1^i + r_1^i) \tag{1}$$

$$\mu_2 = \frac{1}{3n}\sum_{i=1}^{n}(p_2^i + q_2^i + r_2^i) \tag{2}$$

$$\mu_3 = \frac{1}{3n}\sum_{i=1}^{n}(p_3^i + q_3^i + r_3^i) \tag{3}$$

Therefore the μ is the center of the object. Then the covariance matrix C is equal to:

$$C_{jk} = \frac{1}{3n}\sum_{i=1}^{n}(\overline{p_j^i}\,\overline{p_k^i} + \overline{q_j^i}\,\overline{q_k^i} + \overline{r_j^i}\,\overline{r_k^i}) \quad (4)$$

Where vector $\overline{p^i} = p^i - \mu$, $\overline{q^i} = q^i - \mu$, and $\overline{r^i} = q^i - \mu$. C_{jk} is a scalar. All the C_{jk} for j from 1 to 3 and k from 1 to 3 compose a 3 by 3 covariance matrix.

The eigenvectors of this matrix are mutually orthogonal because the covariance matrix is a symmetric matrix [29]. We normalize it and use it as a basis. That is also the orientation of the bounding box. The matrix of eigenvectors also forms a rotation matrix, which rotates the bounding box from local coordinates to world coordinates. Then the extreme vertices along each axis of the basis are found, which is the size of the bounding bo. It also defines the maximum and minimum coordinates in the bounding box's local coordinates.

4.1.3.2 Faceting and enlarge OBB

In a CAD geometry configuration, there are many kinds of objects or surfaces other than polygon or facet surfaces. The faceting algorithm mentioned in Chapter 2 will generate a faceting surface for any surface. To use faceting surface is because ray facet intersection test is more efficient than ray intersection with any other high order surface. Figure 4.4 shows an example of a faceting surface.

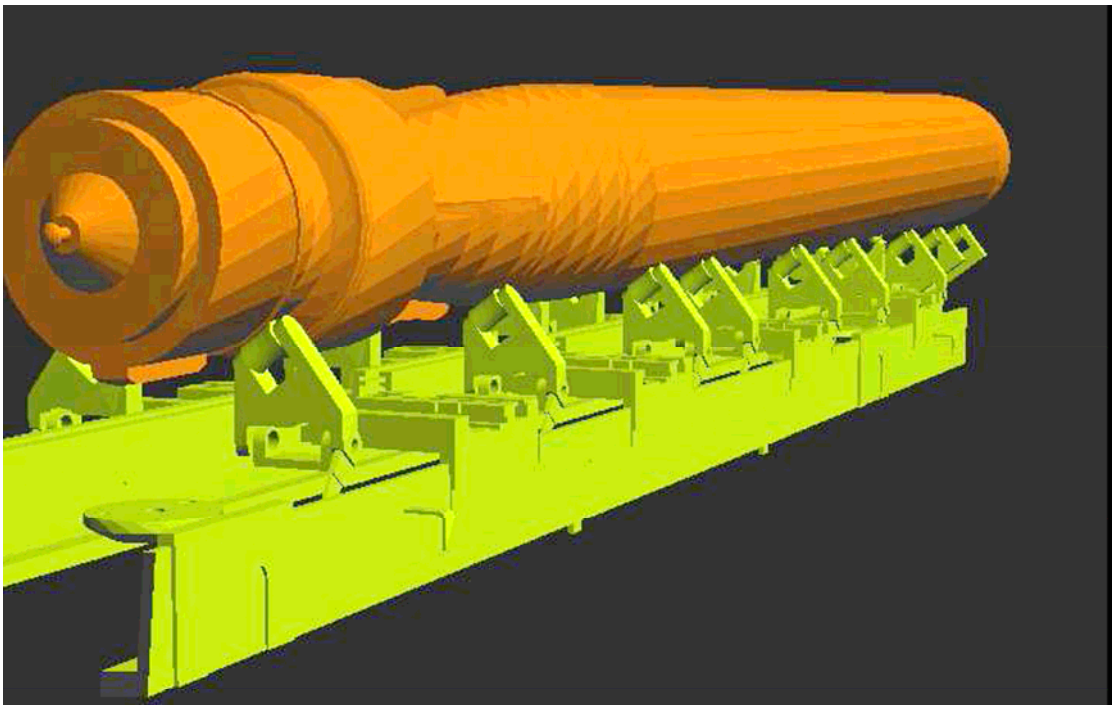


Figure 4.4. Example of facet object[28]

The faceting algorithm is controllable. The major input parameter is the distance tolerance (See Fig. 4.5).

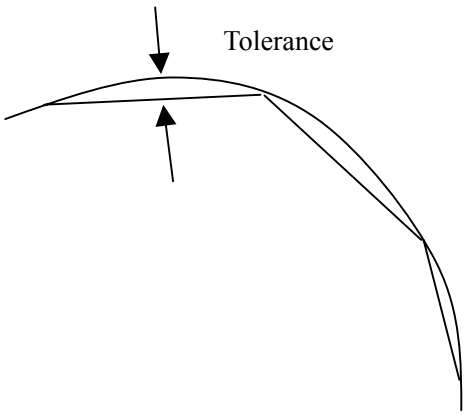


Figure 4.4. Defining the tolerance

By inputting a larger or smaller tolerance, we can control the coarseness or fineness of the

faceting surface. Tolerance also means the maximum difference between facet surface and real object surface.

However, the faceting surface is only an approximation of the real object surface. We can only generate bounding boxes on facet surfaces and the real object surface may not fall completely inside the bounding box. This means that the bounding box is not the correct bounding box. But by the conclusion drawn before, the maximum difference between the faceting surface and the real object surface is the tolerance. A simple modification can solve this problem: we only need to enlarge each bounding box by a distance equal to the tolerance in each coordinate. This will guarantee that the real object surface is inside the bounding box (See Fig 4.6).

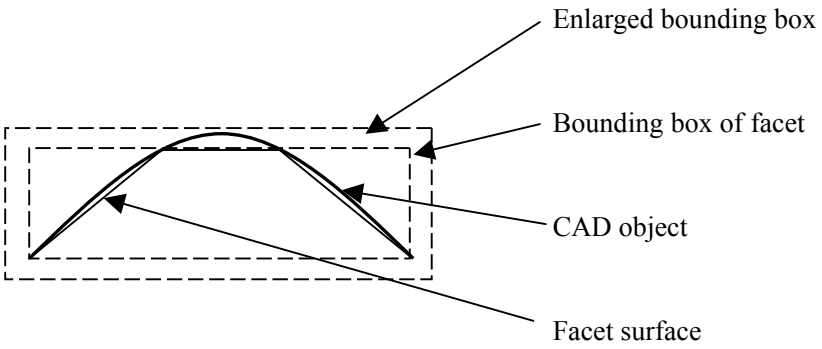


Figure 4.5. Enlarge bounding box to include CAD object

The faceting surface is an approximation of a real surface by many first order surfaces. Therefore there is a trade-off if we use a second order surface to approximate the real surface.

If we use a second order surface, we can have a better approximation for curved surface. However, a more complicated algorithm is needed to generate second order surface. The ray second order surface test is more expensive than the ray facet test. Since a tolerance is already being used because of the accuracy requirement to generate a faceting surface, we therefore choose the first order surface – facet to approximate real surface.

4.1.3.3 Ray-OBB intersection test

We will use the OBB tree to test the ray-surface intersection. This will require the use of the ray-bounding box test algorithm. However in RAPID mentioned in Chapter 2, the OBB tree is applied to the object-object collision detection, therefore, it is only a bounding box-bounding box test algorithm. We require a ray bounding test algorithm, which we will develop for the CAD/MCNPX code

The first question is how to test a ray to a spatial oriented box. We recall our ray, bounding box and whole system are in a "world coordinates" with x, y, z axis. Each oriented bounding box is stored with the origin, dimension and direction of x′, y′, z′ axis. We call the x′, y′, z′ axis as "local coordinates" of each oriented bounding box. It is harder to test the collision of a ray and a spatial oriented box in world coordinates because we need to test 6 spatial oriented surfaces. A better idea is to transform the ray into the oriented box's "local coordinates". Then we obtain the benefit that the bounding box is an axis aligned symmetrical box.

To transform an object from one coordinate system to another, we need to do a "rotation" and a

"shift" operation. The shift is the origin of bounding box and the rotation is a matrix composed by x′, y′, z′ vectors. An OBB can be considered as a rotation and a shift of an AABB whose origin is at the coordinate's origin. To test a ray and an oriented box, we need to inverse transform the ray to the oriented box's local coordinates. In this process, we need to use the computational expensive matrix inversion operation. However, the rotation matrix is an orthogonal matrix. The inverse matrix is just the transpose of the matrix, which is trivial in computation. That is:

The eigenvector matrix of the covariance matrix C_{jk} is A ……(5)

A is an orthogonal matrix. Therefore $A^{-1} = A^{T}$ [30] ……(6)

After the transformation, we face a simple computational geometry problem: we have a spatial ray with a spatial point as origin and a spatial direction, we also have a axial aligned box which is centered at the origin of the coordinate system. The question is whether or not the ray hits the box and if hits it, what is the distance to the box.

There are many ways to achieve this. The most straightforward way is to test the ray with 6 axis paralleled plane. I will use a two-dimensional analog to illustrate it.

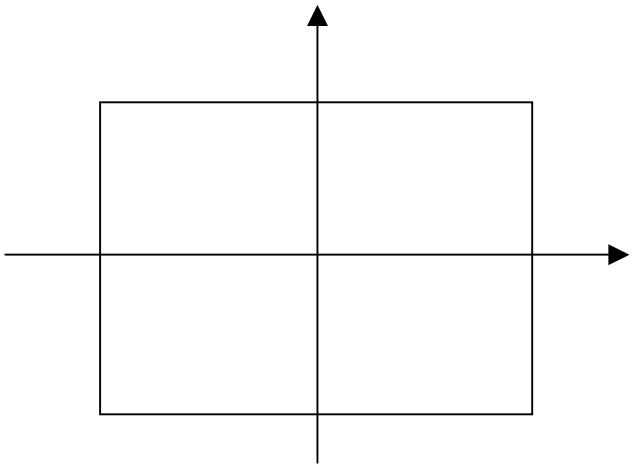


Figure 4.6. Box at the center

The general ray function is: $\begin{cases} x = x_0 + ud \\ y = y_0 + vd \end{cases}$. (7)

The point (x_0,y_0) is the origin of the ray. u and v are the direction vectors and d is the distance of the point (x, y) from the point (x_0,y_0).

For a box centered on the origin, the box area is described as:

$$|x| \le l/2, \quad |y| \le w/2 \qquad (8)$$

where l and w are the length and width of the box, respectively. Note that the box can be written in this form only if box is axis aligned and centered at the origin.

To test the ray box intersection, we need to test 4 edges of the box. For example the top edge:

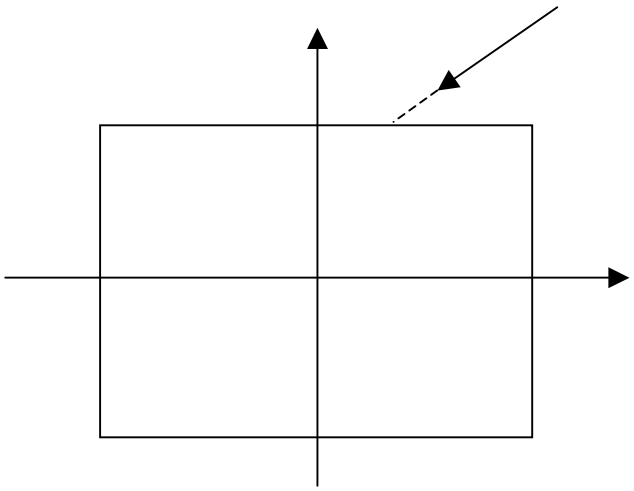


Figure 4.7. The ray test with the upper edge

Let $w/2 = y$ and put it into (7)

$$w/2 = y_0 + vd \quad \Rightarrow d = \frac{1}{v}(w/2 - y_0) \tag{9}$$

Substituting (9) into (7), the corresponding x coordinate is:

$$x = x_0 + ud = x_0 + \frac{u}{v}(w/2 - y_0) \tag{10}$$

We need to test if $|x| \le l/2$ (11)

Therefore put (10) into (11): $\left| x_0 + \frac{u}{v}(w/2 - y_0) \right| \le l/2$ (12)

If this condition (12) is satisfied, it means there is a hit. Then the distance *d* is a candidate. We test all 4 edges and find the minimal distance *d*, which is the distance to the bounding box.

In (12), $l/2$ and $w/2$ can be pre-computed and stored, therefore there is no division operation in the ray-bounding box test. But the vectors, *u,v* are changed for a ray test with a different box. That is because we transfer the ray from world coordinate into each bounding box's local coordinate. Fortunately we can change the condition (12) into:

$$\left|x_0 v + u(w/2 - y_0)\right| \le \frac{l}{2}|v| \qquad (13)$$

The difference between (13) and (12) is that there is no division operation in (13). Therefore (13) has a better, more efficient performance of the ray bounding box test than (12).

The three-dimensional version of equation (12) and (13) are respectively:

$$\left|x_0 + \frac{u}{w}(c/2 - z_0)\right| \le a/2 \quad \text{and} \quad \left|y_0 + \frac{v}{w}(c/2 - z_0)\right| \le b/2 \qquad (14)$$

$$\left|x_0 w + u(c/2 - z_0)\right| \le \frac{a}{2}|w| \quad \text{and} \quad \left|y_0 w + v(c/2 - z_0)\right| \le \frac{b}{2}|w| \qquad (15)$$

The length, width and height of bounding is *a,b,c* respectively.

When testing the ray with the bounding box of root and intermediate (non-leaf) node of the OBB tree, the algorithm will only return if the ray hits the bounding box, no distance information will be calculated. We will use condition (15) to test. When testing the ray with the leaf node-bounding box, the algorithm will return whether it is a hit and the distance if it does hit. Therefore we use (9) to calculate the distance d and use condition (14) to test.

The reason to have two version of the ray bounding box test algorithm is: for each hit in the leaf node, there are many ray intermediate nodes tests. There are also many cases where the ray bounding box algorithm tests many intermediate nodes but does not hit a leaf node. Therefore overall there are many more ray intermediate node tests than ray leaf node tests. Therefore, we need a higher performance ray bounding box test algorithm here than for the ray leaf node bounding box test.

There is another difference in our non-leaf ray bounding box test: once a ray hit with one of the bounding planes of the bounding box is found, we don't need to continue testing other planes because we only need to test if there is a hit. This modification will also improve the performance of ray intermediate bounding box test algorithm.

Condition (15) is only for one surface of the bounding box, the test conditions for all surfaces in the box are:

$$\left|x_0 w + u(c/2 - z_0)\right| \le \frac{a}{2}|w| \quad \text{and} \quad \left|y_0 w + v(c/2 - z_0)\right| \le \frac{b}{2}|w| \tag{16}$$

$$\text{or} \quad \left|x_0 w + u(-c/2 - z_0)\right| \le \frac{a}{2}|w| \quad \text{and} \quad \left|y_0 w + v(-c/2 - z_0)\right| \le \frac{b}{2}|w| \tag{17}$$

$$\text{or} \quad \left|y_0 u + v(a/2 - x_0)\right| \le \frac{b}{2}|u| \quad \text{and} \quad \left|z_0 u + w(a/2 - x_0)\right| \le \frac{c}{2}|u| \tag{18}$$

$$\text{or} \quad \left|y_0 u + v(-a/2 - x_0)\right| \le \frac{b}{2}|u| \quad \text{and} \quad \left|z_0 u + w(-a/2 - x_0)\right| \le \frac{c}{2}|u| \tag{19}$$

$$\text{or} \quad \left|x_0 v + u(b/2 - y_0)\right| \le \frac{a}{2}|v| \quad \text{and} \quad \left|z_0 v + w(b/2 - y_0)\right| \le \frac{c}{2}|v| \tag{20}$$

$$\text{or} \quad \left|x_0 v + u(-b/2 - y_0)\right| \le \frac{a}{2}|v| \quad \text{and} \quad \left|z_0 v + w(-b/2 - y_0)\right| \le \frac{c}{2}|v| \tag{21}$$

4.1.3.4 OBB tree building

If we combine the concept of an OBB and the hierarchical bounding box discussed in Chapter 2, we obtain the OBB tree. In computer graphics, an object is represented by a list of polygons. The root node-bounding box will be the bounding box for whole object. The leaf node-bounding box will be the bounding box for a single polygon. The ray bounding box test is started from root node. If a ray does not hit a bounding box in a level, all ray bounding box

tests below this level are not needed.

To build an OBB tree, Cottschalk, Lin and Manocha, recursively partition the bounded polygons and calculate the bounding box for each group [28].

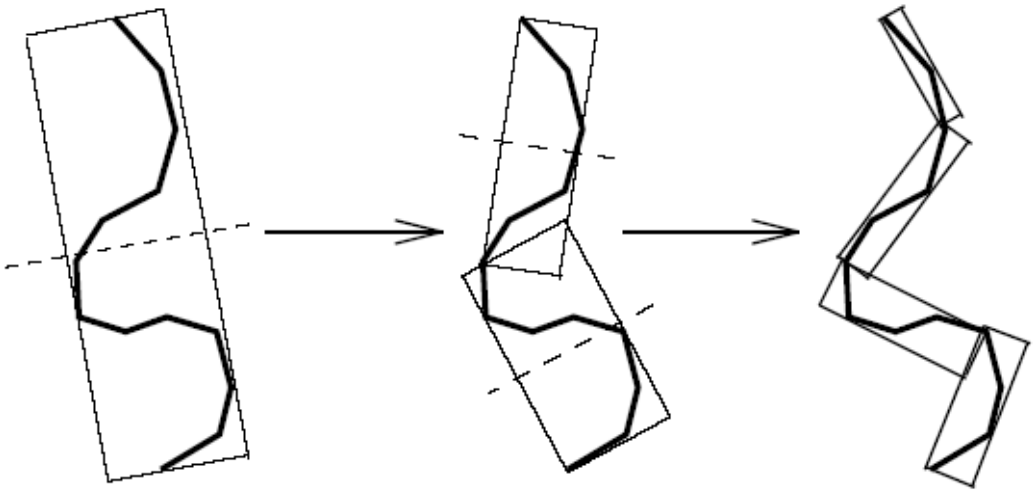


Figure 4.8. Building the OBB tree

They first calculate the bounding box for whole object, which is the root of the OBB tree. Then the longest axis of a bounding box is split partitioning the polygons according to the vertices points lying on the side of split plane. If the subdivision line crosses a polygon, we can use the mean point of that polygon to determine which side this polygon belongs to. If the longest axis is subdivided, we cannot obtain a two-polygon group (see Fig 4.10), we will try the second longest axis. The reason to begin with the longest axis is that it is most likely to successfully subdivide into a two-polygon group (See Fig 4.9). If it still cannot be subdivided, we will try the shortest axis. If all three axes cannot be subdivided, this means the current bounding box is the leaf node of the OBB tree.

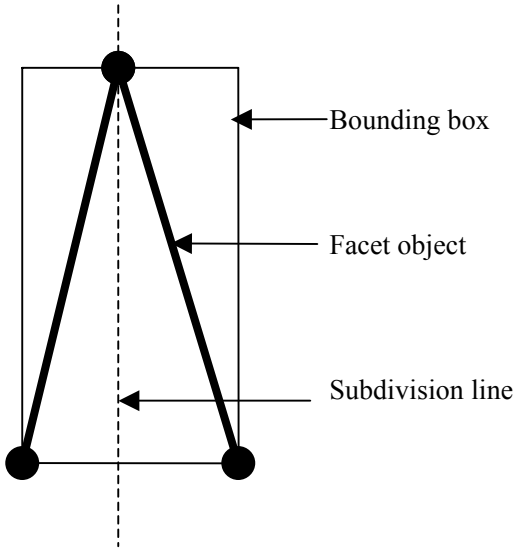


Figure 4.9. In this case we need to use the second longest axis to subdivide

Given a object with n triangles, the overall time to build the tree is O(nlog(n)). The recursion is similar to the quick sort algorithm in sorting theory. The depth of the tree is O(log(n)), which means O(log(n)) ray bounding box tests to find a hit.

Although building an OBB tree requires computational resources, the geometry configuration will not change because the configuration is time-independent. Therefore the OBB tree construction only happens once. Hence for particle Monte Carlo simulations, we do not worry about the time required to build a tree.

To test an OBB tree, we start from top level bounding box and do a ray bounding box high performance test. If there is a hit to the boxes, we will go through the tree. Because the tree is pre-computed and stored, we just need to retrieve two children node-bounding boxes and

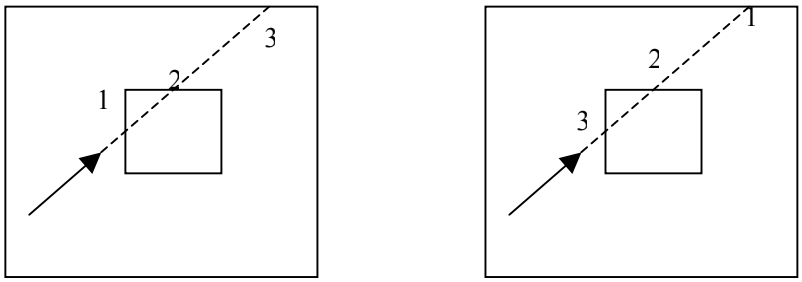
continue applying the high performance bounding box test algorithm. If not, it means there is no hit to the objects. As to the leaf node, we will just use the regular version ray bounding box test algorithm to test if there is a hit and return the hit distance if the ray does hit the bounding box.

4.1.3.5 RAPID

This RAPID software package mentioned in Chapter 2 already provides some functionality we need, such as the OBB tree building functions. It also provides some functions, which are very similar to what we want, such as the bounding box – bounding box test, which can be modified for a ray bounding box test.

4.1.4 Sorting distance

In the previous section, we show the case when a cell is concave. For this case there is the possibility that the ray will have more than one hit. We have a solution that takes advantage of the hit distance to the bounding box. In the best scenario, the first ray-trace calculation on the CAD surface is also the closest hit and all the other hits to bounding box have a larger hit distance than the hit distance to CAD surface. We can still achieve one CAD surface test per ray. However, in the worst scenario, the closest hit to the CAD surface is the last ray-trace performed. We still need to do CAD surface test for each ray-bounding box hit. In this case, we do not obtain any benefit from the distance to the bounding box returned from ray bounding box test function. Figure 4.11 depicts the best and the worst case scenarios.



Best case scenario Worse case scenario

Figure 4.10. Best and worst case scenario

How to make the best scenario always happen? Our approach is after finding a hit to the bounding box, we do not instantly perform a ray CAD surface test. We just store the hit surface and hit distance to bounding box, then continue testing the ray with the bounding box tree of other surfaces. After testing all the surfaces, we have a list of stored hit surface and corresponding hit distance to bounding box. We sort this list by the hit distance in ascending order. Then we can always test the CAD surface from minimum ray bounding box hit distance, and not test boxes whose hit distances are greater than the closest CAD surface hit. Because the bounding box is different from the object, the bounding box with the minimum hit distance is not necessarily the bounding the surface with the minimum hit distance. Therefore this algorithm does not always give ideal results, but when the faceting is very fine, the bounding box closely approximates the real surface. Hence, this algorithm also closely approximates the ideal case.

4.1.5 Distance limit

The accelerations above can be implemented strictly inside the geometry engine without regard for the application (Monte Carlo). However, there are also acceleration techniques that take advantage of both geometry and application.

In particle transport, particles will undergo interactions with materials. The Monte Carlo code will sample a distance to collision. Only if this distance is less than the distance to boundary, that is to say, the collision point is inside current cell, will a collision happen. We find the distance to boundary is just for comparison with distance to collision. Only if distance to collision is larger do we need a precise distance to the boundary. This is because this particle will cross the boundary. If the distance to collision is less, we do not need a precise distance to boundary. We can just use an approximate distance and if we can make sure the distance to collision is smaller, this accuracy level of distance to surface is enough for us.

The OBB trees give exactly this approximation to us. The leaf node bounding-box is an approximation of the CAD surface. And when a particle is outside of the bounding box, the distance to the bounding box is always smaller then distance to CAD surface. If the distance to bounding box is already larger than distance to a collision, the distance to surface is guaranteed to be larger than the distance to collision. In this case, we do not need to calculate distance to surface because particle will undergo collision instead. Then we can save computation time. We call this algorithm “distance limit”.

After implementing this step, we believe on average a particle will need < 1.0 ray CAD surface test per cell. The amount of benefit we can obtain depends on physics of the problem. In a cell with dimensions much larger than a particle's mean free path, this algorithm will have a big benefit because most of the time a particle will undergo an interaction.

This algorithm is possible only because the OBB tree provides a good approximation of the surface (actually the leaf node of OBB tree of a facet surface is a triangle, the bounding box of a triangle is degraded to a square. Even if we consider enlarging the bounding box, one dimension of bounding box will be very small.) and we can obtain the distance to a collision from the physics portion of the Monte Carlo code. This algorithm is unique in particle transport.

4.1.6 Implementation

We use the software package RAPID to perform the OBB tree building and the ray-bounding box test [28]. This approach can save a lot of time for software developing. But as discussed in subsection 4.1.3.2, some modifications are necessary. We keep the OBB tree building but in the ray-bounding box test algorithm, we only need to build an OBB tree for one object. We also implement a ray-object intersection test based on the OBB tree. It will read in the facet surface generated by CAD geometry engine, generate the OBB tree for each facet surface and store it. In the Monte Carlo particle simulation, a ray is inputted and determines if

the ray hits the OBB tree, if a leaf node bounding-box is hit, the distance to it is also returned. In the case that a facet model is used, a ray-triangle test is performed in the leaf node and the distance is returned if there is a hit.

To implement the sorting distance algorithm, we use an “insert sort” algorithm. We have a list to store the hit distance and hit surface sorted on distance. If there is a new distance to insert, we will find a place in this list and insert it.

To implement a distance limit algorithm, we change the order of calculation in MCNPX. It will now calculate the distance to a collision first and pass it to the ray surface test functions. We compare this distance with the sorted hit distance list. If the distance to the bounding box is larger than the distance to collision, we stop the calculation and return “the particle will not hit the surface”.

4.2 Other Geometric Accelerations

Besides the ray tracing, there are some other geometry evaluations in MCNPX. Some of them can be very costly in computational time. Our approach is to either use the functionality provided by the CAD geometry engine or to use a simple approach to improve performance.

4.2.1 Surface traversal

After a particle hits the boundary, the next question is what the cell number on the other side of the boundary is. This is a topology question. The feature “non-manifold geometry” we mentioned in Section 2.2 can be used to solve this problem. The “non-manifold geometry” means there will be a single surface shared for two bodies. The CGM has a function called “merge” which changes the manifold geometry to a non-manifold geometry.

The benefit of using a non-manifold geometry in MCNPX is applicable for any surface. There is either one cell or are two cells on each side of a surface. For an outmost surface, there will be only one cell on one side of it. For an internal surface, there will be two cells that share it. We can use the functionalities provided by CGM to retrieve the one or two cells bounded by this surface; combining this with the current cell number, we can know the other cell number or the particle leaves the system. This approach also improves performance by eliminating one ray-surface testing each time a particle crosses a surface. This optimization was not in Franke’s work.

4.2.2 Determine Region of Source Particle

In MCNPX, there is a function to determine the cell number of a source particle, given a source particle’s position. A user will not need to input the start cell number. However, in a complicated geometry configuration, this “determine source particle cell” function will be as expensive as or even more expensive than calling the ray tracing function. We can apply ray tracing acceleration techniques like spatial subdivision for source positions, but for most cases,

a user would know which cell the source particle comes from. Therefore we just let the user input the start cell information and bypass the function in MCNPX. The performance of this approach will be analyzed in Chapter 6.

4.3 Facet Model

In the previous approach, the surface faceting is only for the purpose of generating the OBB tree. Although for each level of an OBB tree there is a faceted surface,, we never do any ray faceting surface test. When we need the distance to surface, we always use the CAD surface to test.

However, the faceting algorithm can obtain an approximate surface with controllable accuracy. When we use a very small tolerance to generate the faceting surface, we can obtain a very accurate approximation of the surface with just the facets. Therefore we provide the option to just use the facet-based surface, completely bypassing the ray CAD surface test.

The direct benefit is that we do not need to perform an expensive ray CAD surface test. Hence, there is a huge improvement in computational performance. The major problem is it is just an approximate model. The computational results are questionable. However, in Chapter 6 we will see that because the facet model is a high accuracy approximation model, the computation result achieved are sometimes identical to CAD model results.

4.3.1 Implement and ray triangle test

Implementation is straightforward because a lot of work has already been done in previous steps. We need to modify the software package RAPID [28]. In the CGM ray tracing function, there is a flag set to indicate whether a CAD model or a facet model is used. For the case of a facet model, the ray bounding test will not add the tolerance to the bounding box dimension because the facet model is used directly. And for the leaf node test, we do not use the ray bounding box test with distance return function, we use the ray triangle test function with distance return. We still sort the hit distance in a list. In the forgoing case, if the ray hits a leaf node bounding box, a ray CAD surface test is performed. In the latter case, the minimized hit distance to triangle along with hit surface number is returned directly.

The ray triangle test algorithm basically is to find the barycentric coordinates of the hit point. By definition, the barycentric coordinates are (α, β, γ). When $0<\alpha<1$, $0<\beta<1$, $0<\gamma<1$ and $\alpha+\beta+\gamma=1$, the point is inside the triangle.

The ray triangle test algorithm we used is from Ref [21]. It is high performance algorithm where at most there is one division operation. The algorithm is:

the triangle's three vertices are (a_x, a_y, a_z), (b_x, b_y, b_z), (c_x, c_y, c_z).

The ray origin is (p_x, p_y, p_z) and the ray direction is (d_x, d_y, d_z).

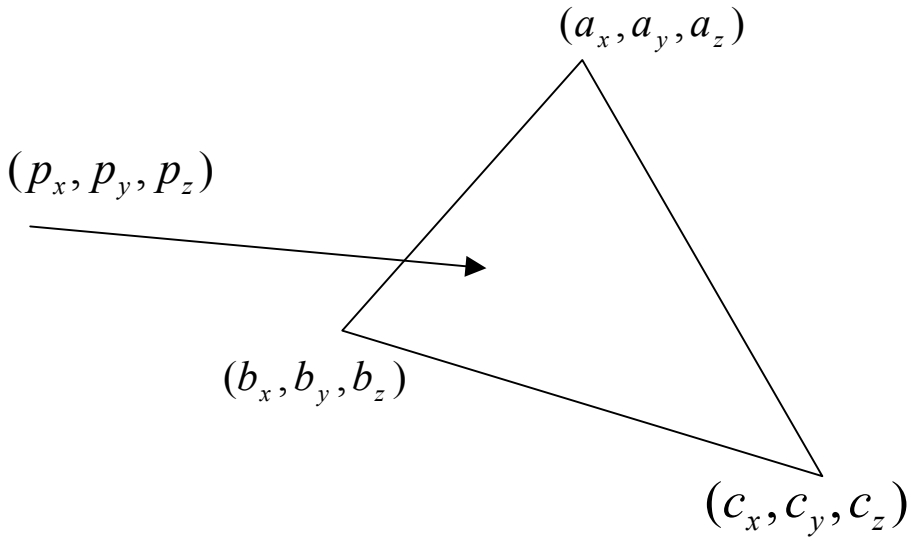


Figure 4.11. Ray triangle test

We should solve $\begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - p_x \\ a_y - p_y \\ a_z - p_z \end{bmatrix}$. [21] (22)

On the condition $t>0$, $\beta>0$, $\gamma>0$, $\beta+\gamma<1$, the triangle is hit.

We rewrite the equation as:

$$\begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} J \\ K \\ L \end{bmatrix}. \quad (23)$$

Then Cramer's rule gives us:

$$\beta = \frac{J(EI - HF) + K(GF - DI) + L(DH - EG)}{M}; \quad (24)$$

$$\gamma = \frac{I(AK - JB) + H(JC - AL) + G(BL - KC)}{M}; \quad (25)$$

and $$t = \frac{F(AK - JB) + E(JC - AL) + D(BL - KC)}{M} \quad (26)$$

where

$$M = A(EI - HF) + B(GF - AL) + D(BL - KC) . \quad (27)$$

The pseudo code is [21], the code use "if – then" structure will not hurt performance too much because current CPUs are usually designed with a "branch predict" feature:

```
double raytri(ray r, vector a, vector b, vector c)
compute t
if(t<0) then return -1
computeγ
if(γ<0) or (γ>1) then return -1
computeβ
if(β<0) or (β+γ>1) then return -1
return t
```

Figure 4.12. Pseudo code of ray triangle test

Chapter 5

Applications

In this chapter, we will run benchmark test problems to show the correctness of our MCNPX/CGM and its applicability to complicated geometries. The complicated geometry is a real fusion device – the ARIES-CS compact stellerator. The neutronics parameters simulated by MCNPX/CGM are used to guide the design and analysis of this device.

5.1 Simple Comparison and Benchmark

We will run two simple test problems. They are problems one may encounter in a nuclear engineering analysis or medical physics application. A comparison with the standard MCNPX is performed and it is shown that MCNPX/CGM generates identical results with the standard MCNPX version.

5.1.1 Test Problem 1: Three cylinders

The first test problem is the “three cylinders” case and is considered a simple nuclear analysis problem. We have a point neutron source which irradiates an object. A detector is located on the far side of the object and we want to measure the gamma photon spectrum, which is induced by the interactions of neutrons with the object. The problem diagram is shown below. Figure 5.1 depicts the standard MCNPX two-dimensional model of the problem. Figure 5.2

depicts the CGM geometry. The CGM rendering of the computational geometry is much better than the standard MCNPX plot.

With reference to the CGM model, the red cylinder is the outer boundary of the problem. The green cylinder is the object being irradiated and is made of carbon. The yellow cylinder is the detector. The space between boundary, object and detector is a vacuum. The 11 MeV neutron source is a point source and located below the green cylinder but inside the red cylinder.

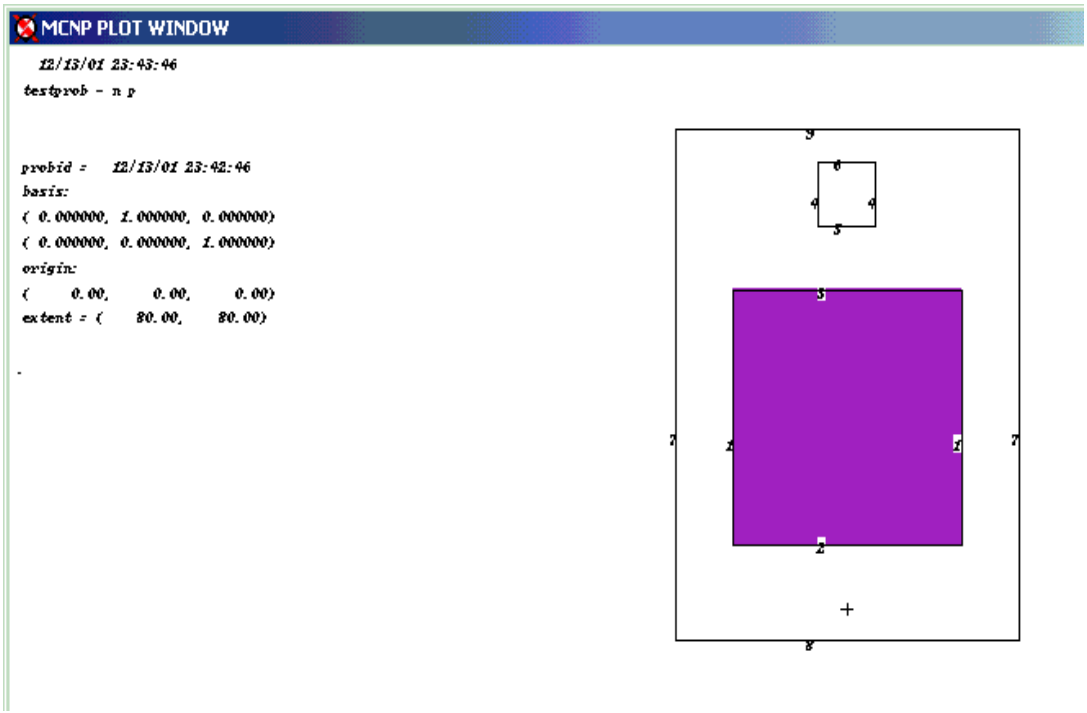


Figure 5.1. MCNPX plot of three cylinders problem

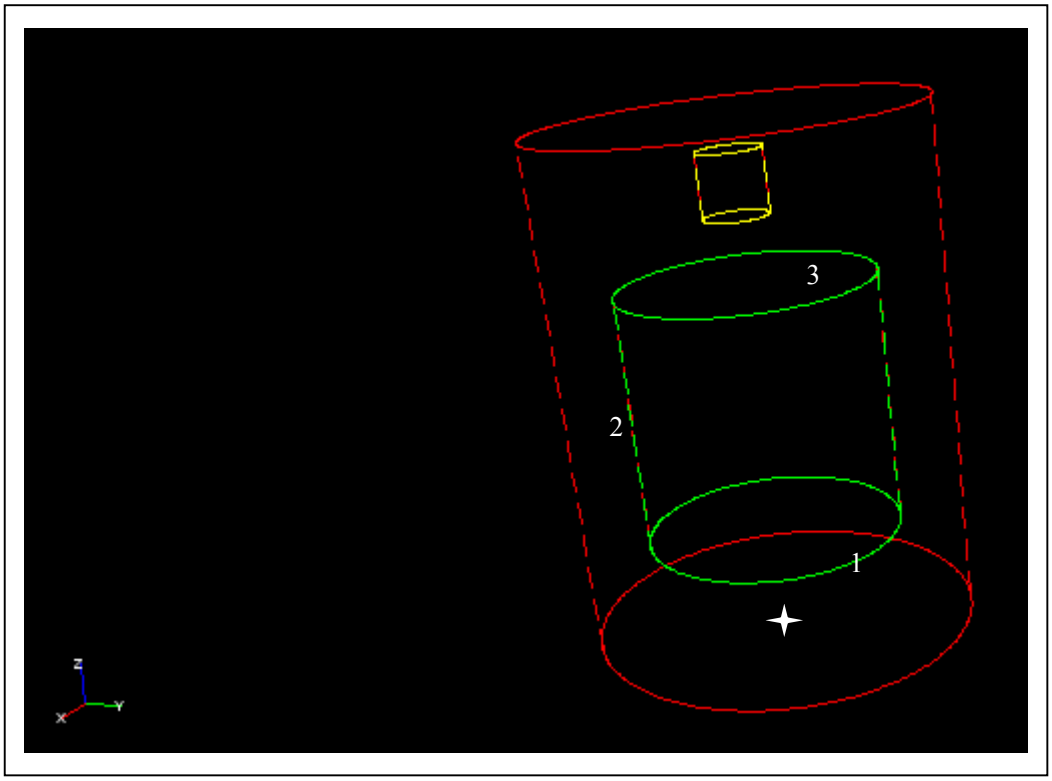


Figure 5.2. CGM/Cubit view of three cylinders problem

A tally was set on the three surfaces of the green cylinder. It is the F1 (surface current) tally for neutrons. Surface 1 is the circular plane surface near the source. Surface 2 is the cylindrical side surface. Surface 3 is the circular plane surface near to the detector. The energy distribution spectrum is shown below.

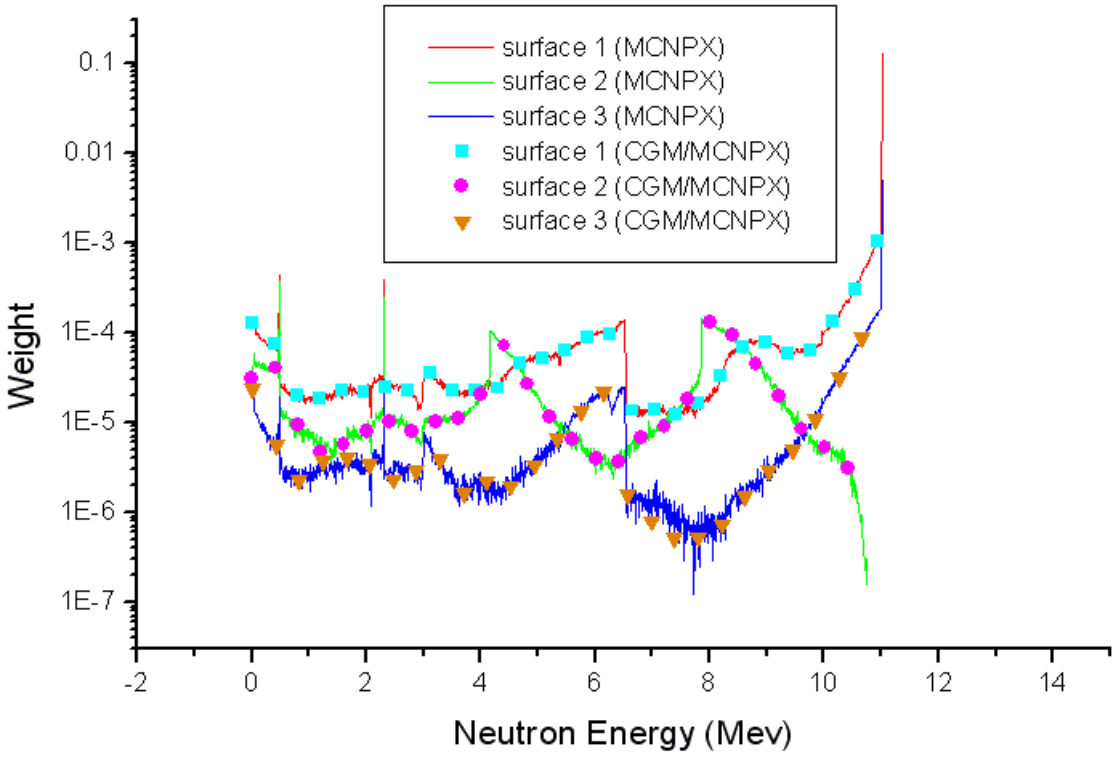


Figure 5.3. Tally spectrum of three cylinders problem

The spectrum is computed by both the standard MCNPX and MCNPX/CGM. Since the sampling and physics functions were not modified and if the particle tracking algorithms were modified and implemented correctly a particle in MCNPX/CGM will experience the exact same tracks and interactions as in the standard version. The MCNPX/CGM routines cgmtrack() and cgmnewcel() return the exact same values as MCNPX's routines track() and newcel() and the tallies are exactly the same. In Fig. 5.3 above, each spectrum curve is actually two spectra, one from each of the codes, which totally coincide.

5.1.2 Test Problem 2: Cobalt

This is an application in the area of medical physics for cancer therapy. The device is a ^{60}Co (Cobalt-60) gamma photon cancer therapy unit where ^{60}Co is the photon source. ^{60}Co is radioactive and emits two photons per disintegration with energies of 1.17 and 1.33 MeV, respectively. Lead (Pb) is used as the surrounding shield material. The device is constructed to obtain a controllable therapeutic beam on the patient. The diagram of the device is shown in Fig. 5.4 (MCNPX plot) and Fig. 5.5 (Cubit).

In the CGM diagram, the big red sphere is the Pb shielding material. Its outer surface is also the outer boundary of the problem. The larger green cylinder is the source container and the small green cylinder is the gamma source. For the test problem, the ^{60}Co gamma source was simplified to be a point source located on the center of the top surface of the small green cylinder. The four jaws are movable to control the dose and the projected area of the beam. If the jaws move into the beams path, part of the gamma photon beam will be scattered while the other part will be absorbed. The overall effect is that the dose to the patient decreases. The beam exits the device at the red solid circle on the top of the Pb shield.

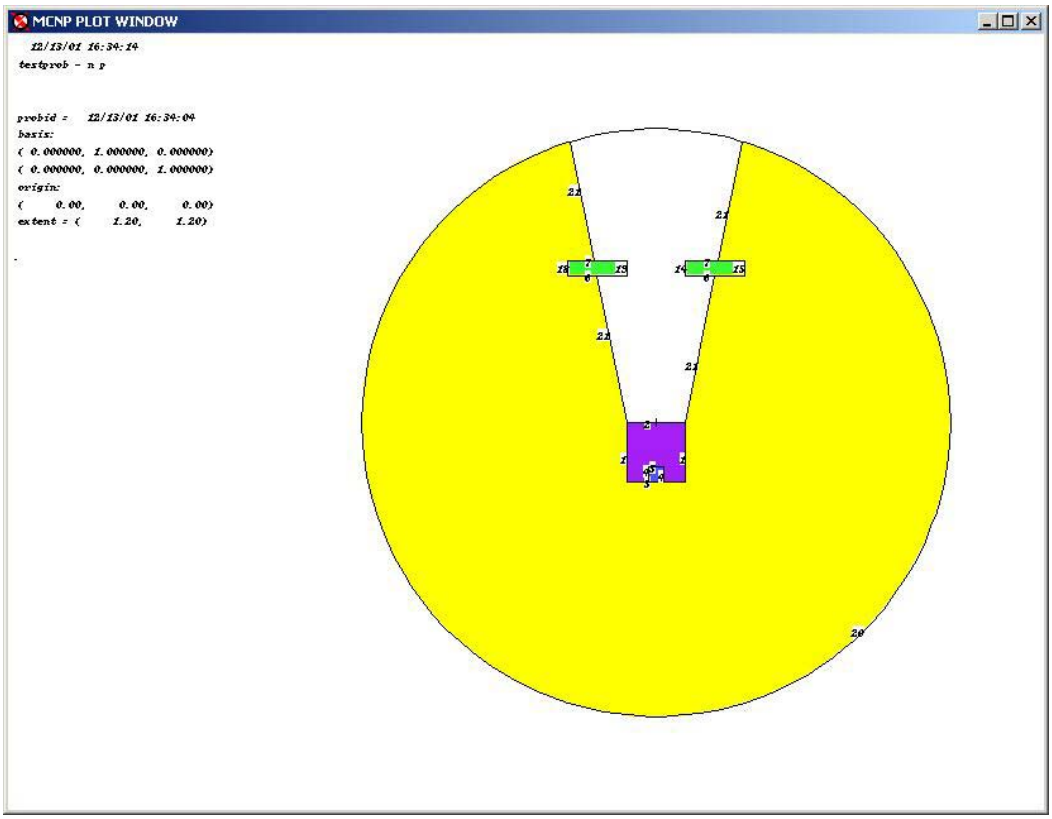


Figure 5.4. MCNPX plot of cobalt device

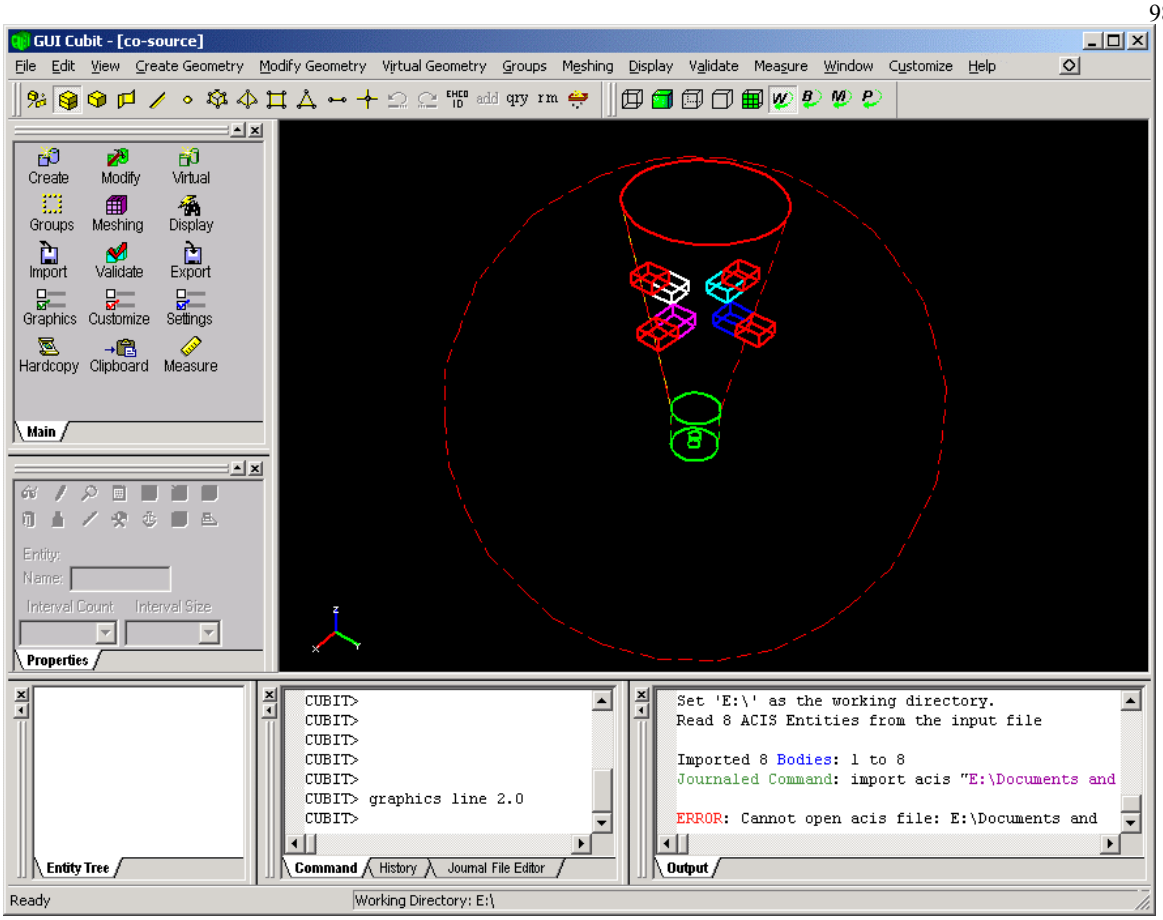


Figure 5.5. MCNPX/CGM view of cobalt device

The scoring tally for the problem was the F1 tally (surface current) for photons crossing at the red circular plane. MCNPX running mode is mode P. Two cases were run: the base case (jaws open) and the case when the jaws are moved inside to half close. When the jaws are moved inside, there will be increased photon scattering out of the primary beam, decreasing the dose to the patient.

The tally results are shown in Fig. 5.6.

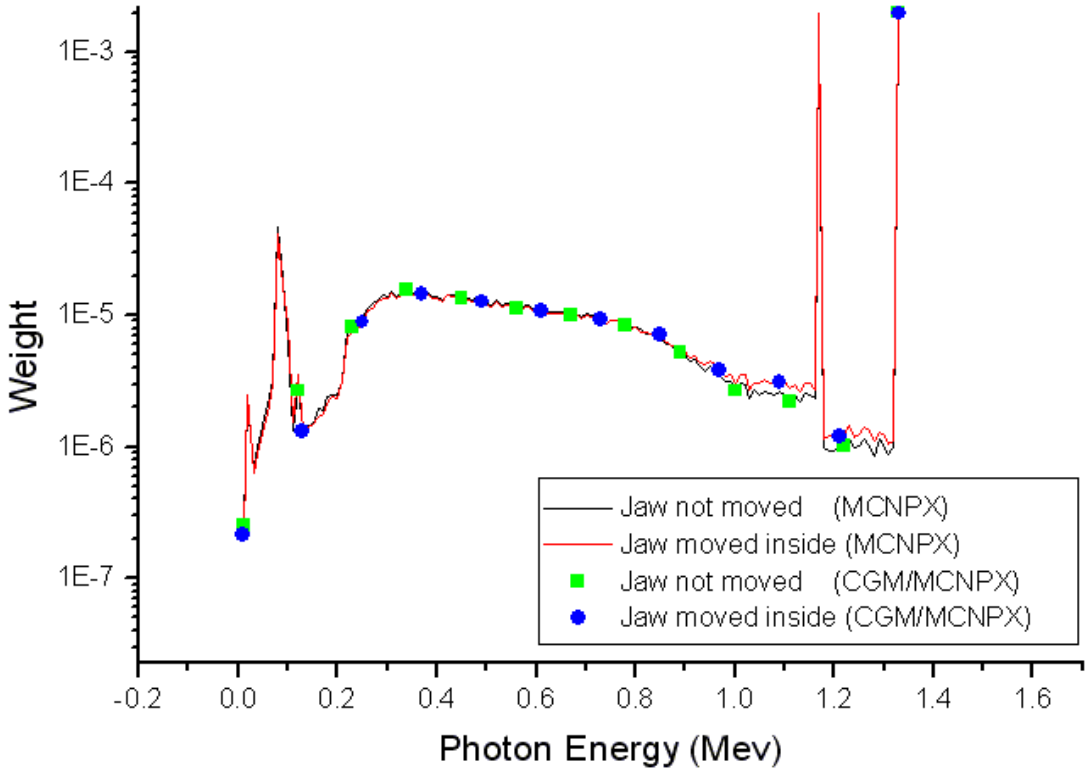


Figure 5.6. Tally spectrum of cobalt problem

As in the previous test case, the tally results for MCNPX/CGM are exactly the same as those from the original MCNPX. In Fig. 5.6, each spectrum curve is actually two spectra, one from each code, which totally coincide. Note the two source energy peaks at 1.17 and 1.33 MeV. Regarding the absence of the 0.511 MeV peak, to observe this peak the particle mode setting in the MCNPX run would have had to be Mode P E. The E signifies electron transport. With this option turned on the code would track the pair production electrons and the annihilation of the positrons, which leads to the production of the 0.511 MeV gammas. The absence of the E flag leads to an absence of the 0.511 MeV tally peak.

5.2 Advanced Test

The previous simple comparison tests demonstrated the correctness of MCNPX/CGM. This test will show that MCNPX/CGM can be applied to complicated geometries.

In Fig 5.7, a clothespin model with a match clinched between its jaws is depicted as an example of a complicated model test problem for MCNPX/CGM. Note that the spring has been purposely offset from its normal position to highlight its complicated geometry/structure (it was also shown in Chapter 1). This object would be quite hard to nearly impossible to model with the standard MCNPX code. Although there are only 4 parts/objects, this model has many plane surfaces and curved surfaces (112 surfaces in total). This model was provided by LANL. It was originally drawn with Pro/Engineer and was translated to ACIS.

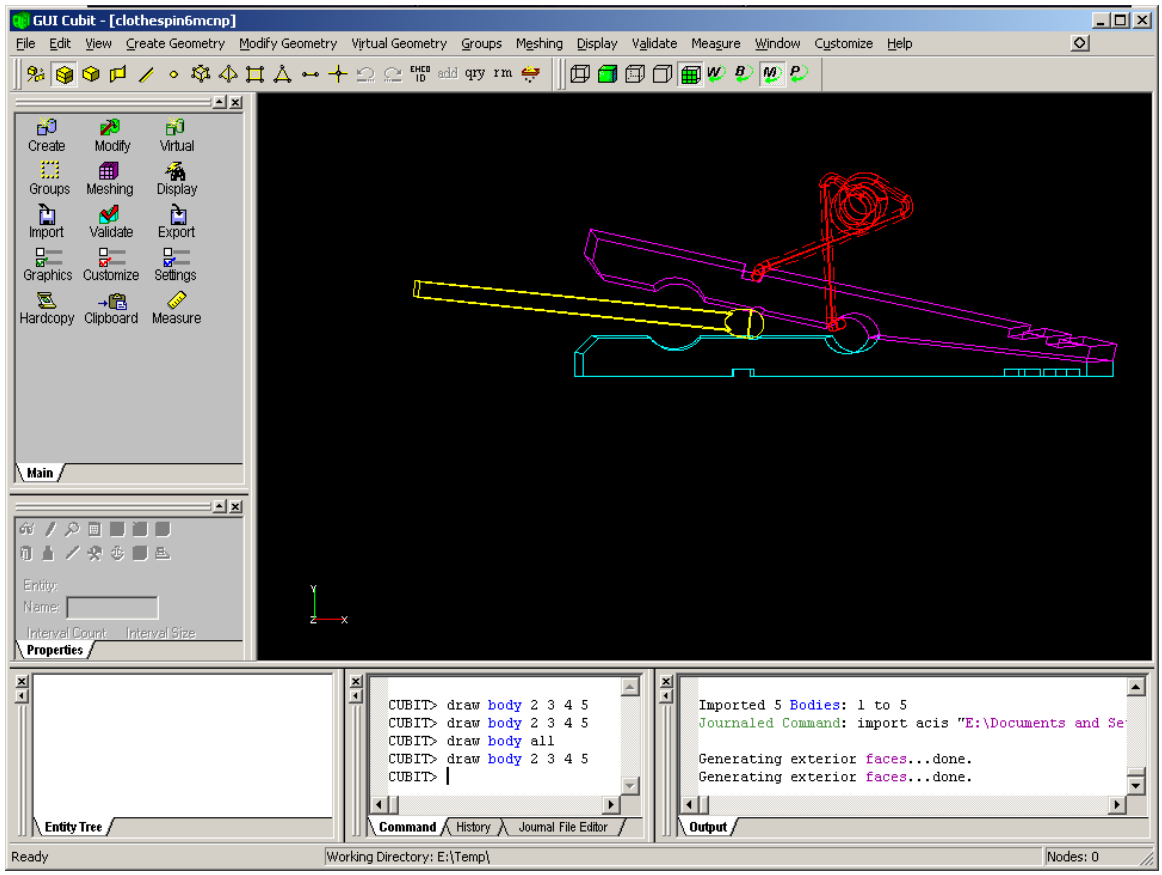


Figure 5.7. Clothespin model

The problem setup is that the clothespin is being imaged by a point gamma source located behind it. Figure 5.8 depicts the resulting image. We use the point source and the Fi5 (Pinhole image projection) tally features of MCNPX/CGM A pinhole image projection means a point is defined in space that acts like the hole in a pinhole camera. It is used to focus an image onto a grid, which acts like a film. This tally only exists in MCNPX [4].

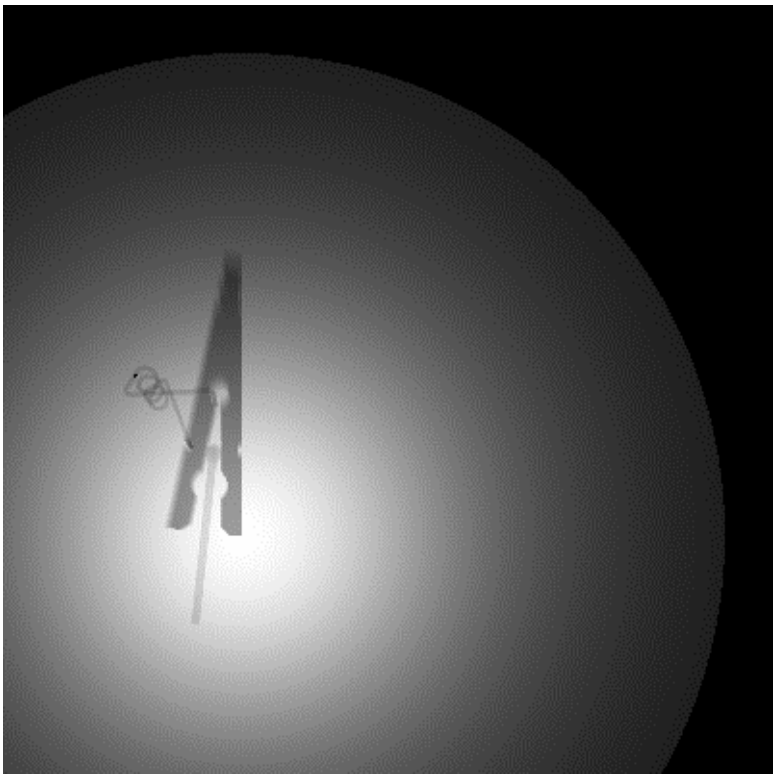


Figure 5.8. The image of pinhole projection

From this illustration we can see that MCNPX/CGM can be applied to complicated geometries. This increased geometric modeling can be important for neutron and photon simulations or nuclear reactor designs.

5.3 Real application (ARIES-CS Compact Stellerator)

A Stellerator is a plasma fusion energy device for future fusion power plants. The main components of the complicated toroidally shaped device are the vacuum vessel and large poloidal magnetic coils. Figure 5.9 depicts the complex magnetic coils and plasma shape. The vacuum vessel is surrounds a region called a blanket, which is composed of a tritium breeding zone, coolant region and neutron and gamma photon shield. A depiction of the outer layer of the blanket is given in Fig. 5.10 and in a CAD representation it is composed of many B-spline curve surfaces.

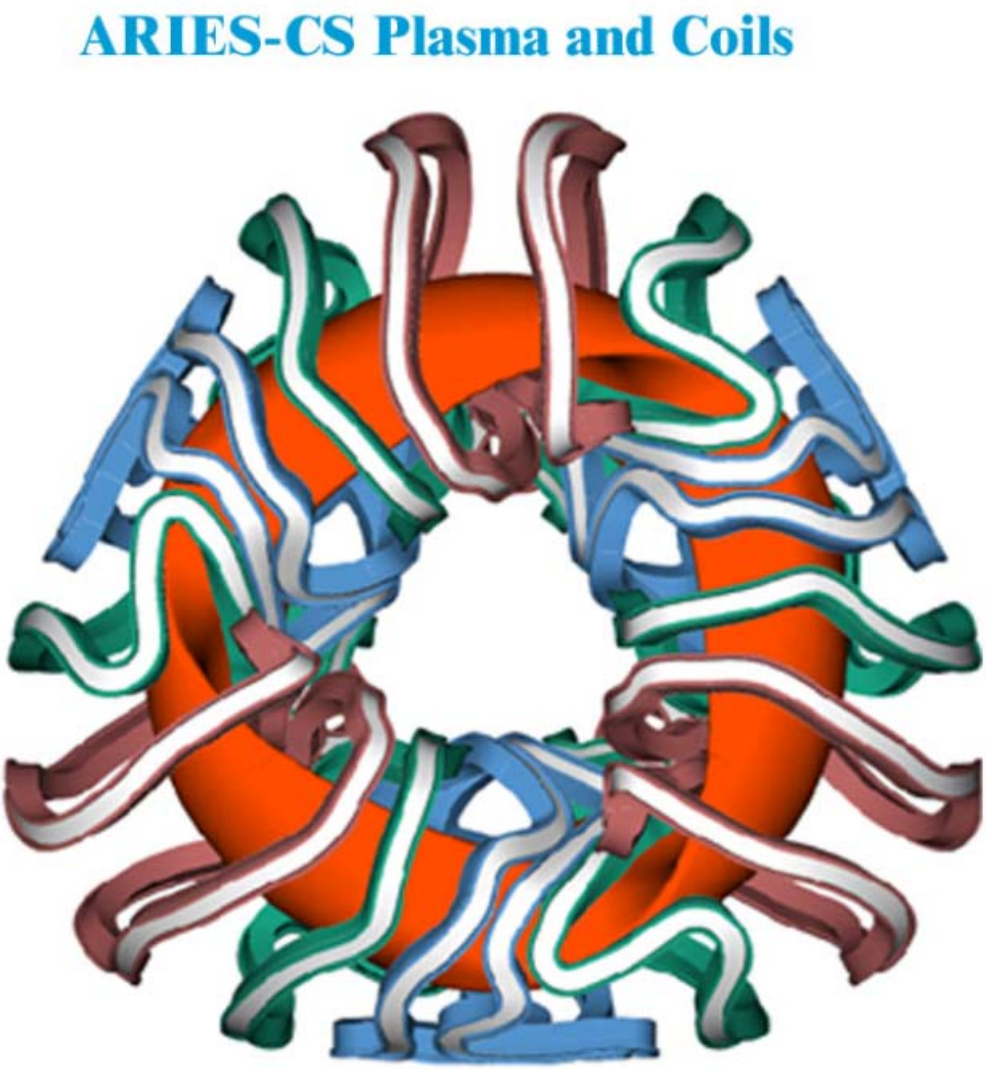


Figure 5.9. ARIES-CS Compact Stellerator and magnetic coils[31]

It would be quite hard to create a standard MCNPX geometry input file even if a CAD model were available. Before MCNPX/CGM, we could only use an approximate model, which is composed by planes or simple curve surfaces and the computational tally obtained would not be an accurate solution. In addition, it is quite difficult to construct such an approximate model by the user requiring many man-months of work. Even for the previous simpler design by ARIES team, the approximation model is the only choice to perform a Monte Carlo simulation. However, if we use MCNPX/CGM, we can directly use the existing solid model generated by the CAD program. Because the CAD model provides a more accurate representation, the computational results are also more dependable.

5.3.1 Modeling

The Stellerator model we used is a 7 layered torus shaped object. Figure 5.10 depicts the outer most blanket layer.

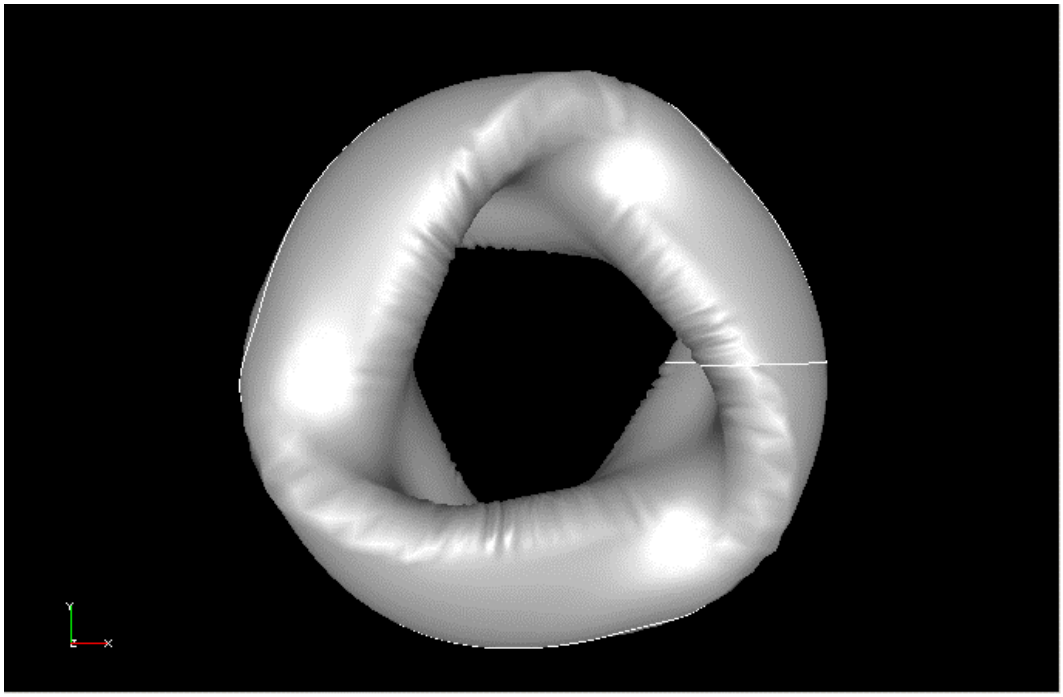


Figure 5.10. Seven layer Stellerator device

To depict the internal structure of the Stellerator, Fig 5.11 provides a cross sectional cut at a given toroidal position and shows only the silhouette lines of each layer.

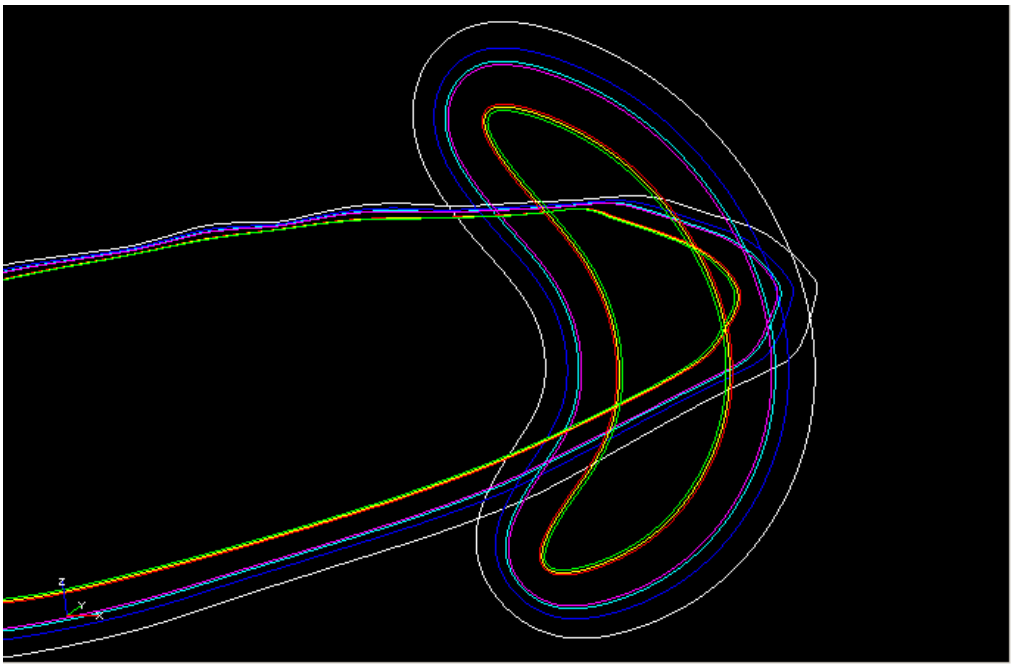


Figure 5.11. Seven layer Stellerator internal structure

To construct this geometry model, we have source data, which represents 72 cross sections of the inner most layer – the plasma surface. This is depicted in Fig 5.12.

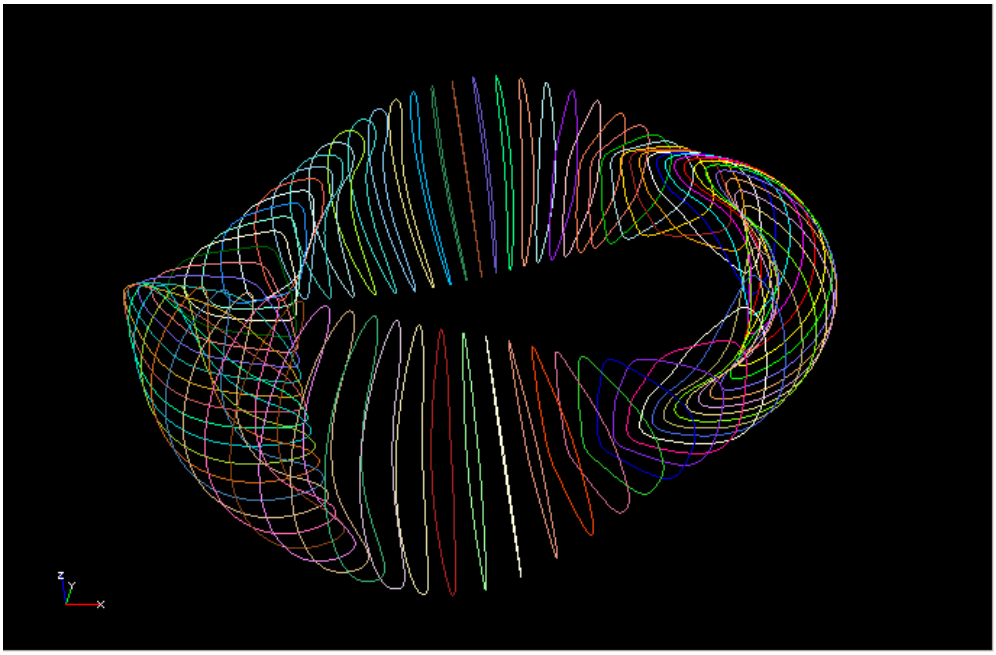


Figure 5.12. 72 cross sections of plasma surface

CUBIT is used to generate the B-spline curve boundary surface, which is the plasma surface. Then the "offset" feature of CUBIT is used to generate the cross sections for each layer. These cross sections are then used to generate the surface of each layer. The equation for the plasma cross sections was provided by Princeton Plasma Laboratory [32]. They are:

$$R = \sum rbc(n,m)\cos(m\theta - n_p n\phi) \quad (1)$$

$$Z = \sum zbs(n,m)\sin(m\theta - n_p n\phi) \quad (2)$$

where zbs and rbc are a series of fourier harmonics constant coefficient, n is from -6 to 6, m is from 0 to 10, θ is the poloidal angle, ϕ is toroidal angle and n_p is period number,. For thecurrent model, n_p =3.

Because we use a CAD geometry model directly as the calculational model in the Monte Carlo simulation, the CAD model should be carefully constructed and should be error free. If the CAD geometry configuration has errors in it, for example, regions of tiny gaps or overlaps, though outwardly the representation may look good, it will cause the Monte Carlo simulation to fail.

5.3.2 The Monte Carlo simulation

The first calculation is to find the peak neutron wall loading at the first wall surface. Figure

5.13 shows the computational model.

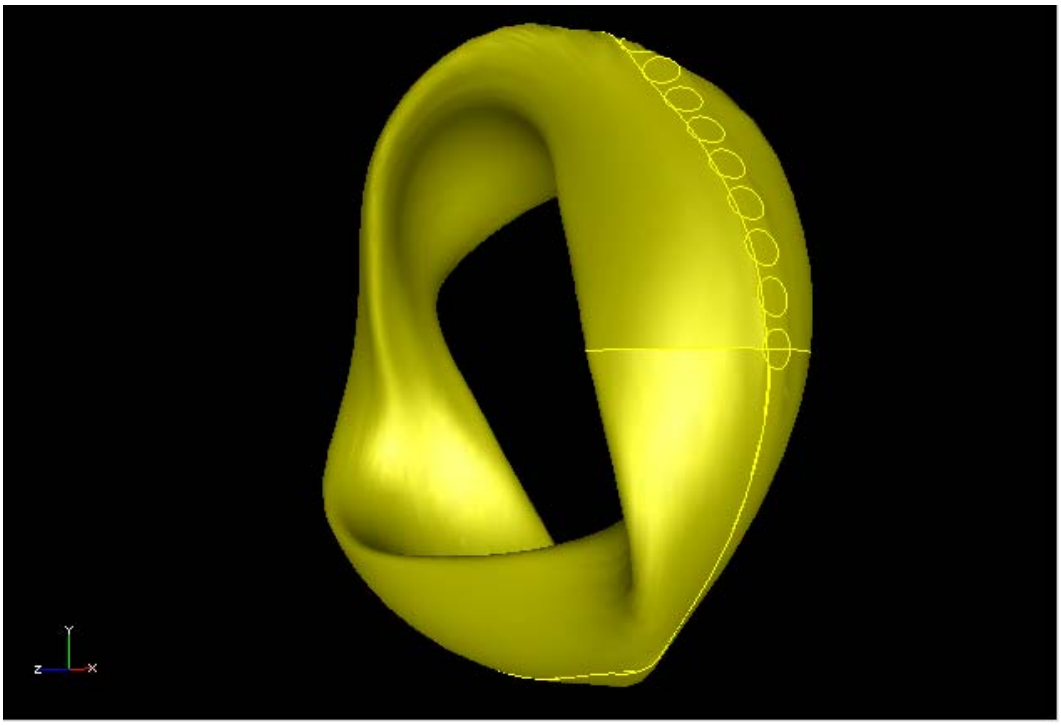


Figure 5.13. Peak neutron wall loading computational model

This model is just the first and the second layers of the model shown in Fig 5.10 and Fig 5.11. Layer 1 is the plasma source (not shown). Layer 2 is the first wall surface (yellow shape). There is no material in this model. The nine circles on first wall surface are tally surfaces. Their centres are located at midplane and they are spaced at intervals of 7.5 degrees toroidally. The F1 (surface flux) tally is used on these 9 circles. The results are shown in Fig 5.14.

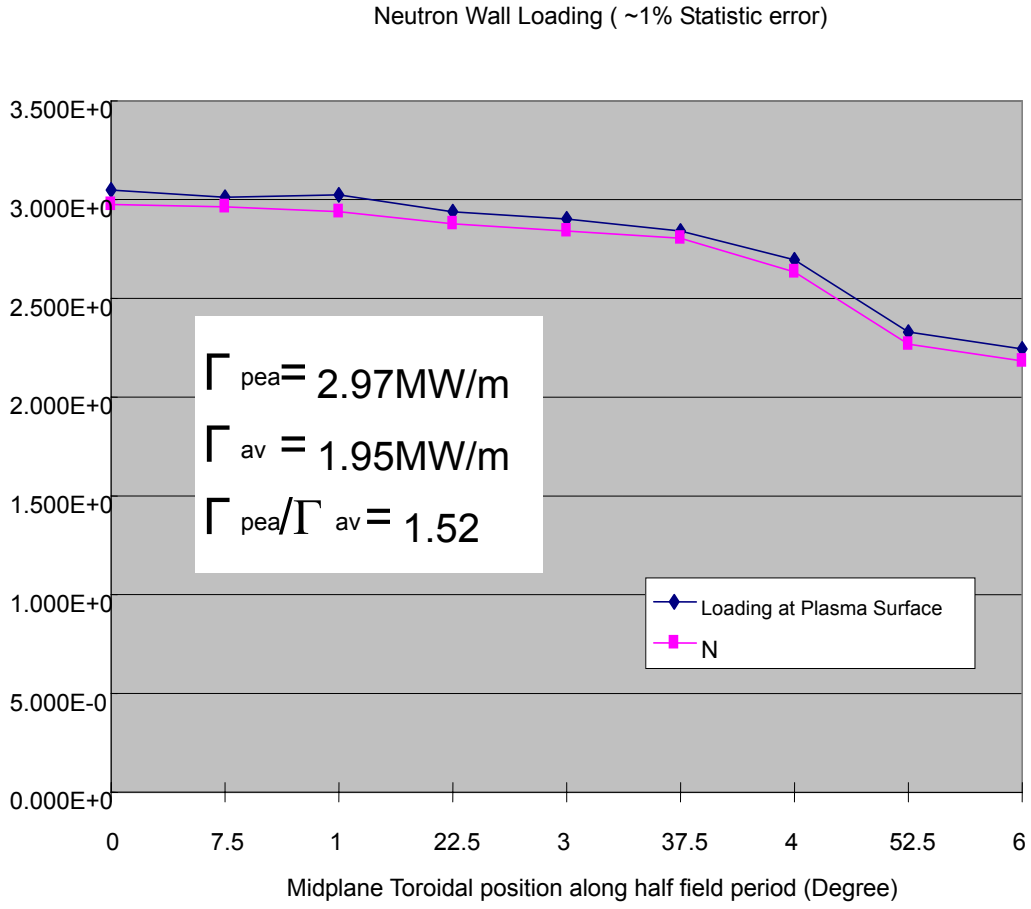


Figure 5.14. Neutron wall loading

The red (square) line in Fig 5.14 is the neutron wall loading at the tally circles. This graph clearly shows the peak wall loading is at the 0 degree toroidal position. The value is 2.97 MW/m². This result can be used to estimate the life span of first wall. It is also very important for the design of whole fusion reactor system.

The second calculation uses all 7 layers. Figures 5.15 and 5.16 depict the layers and name of each layer.

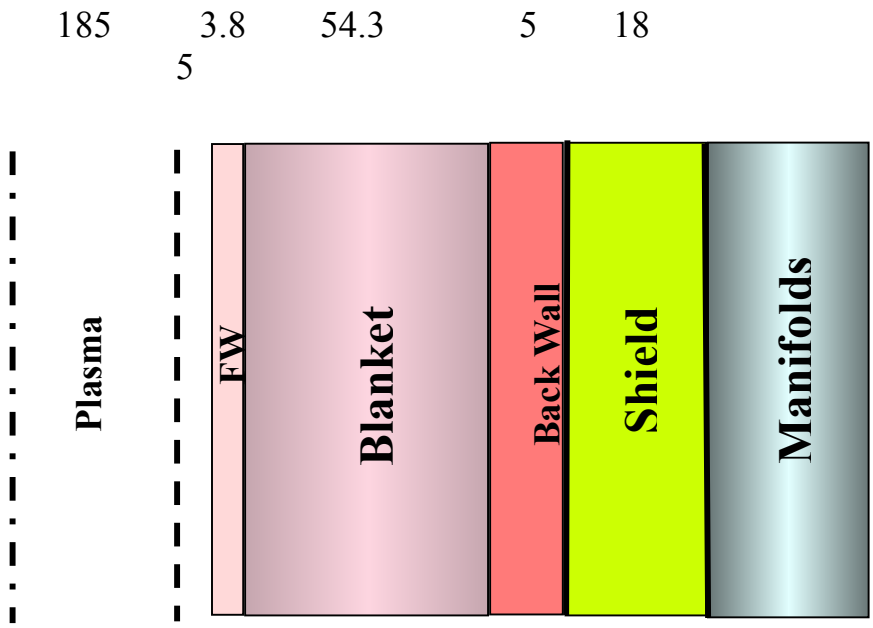


Figure 5.15. Name of each layer

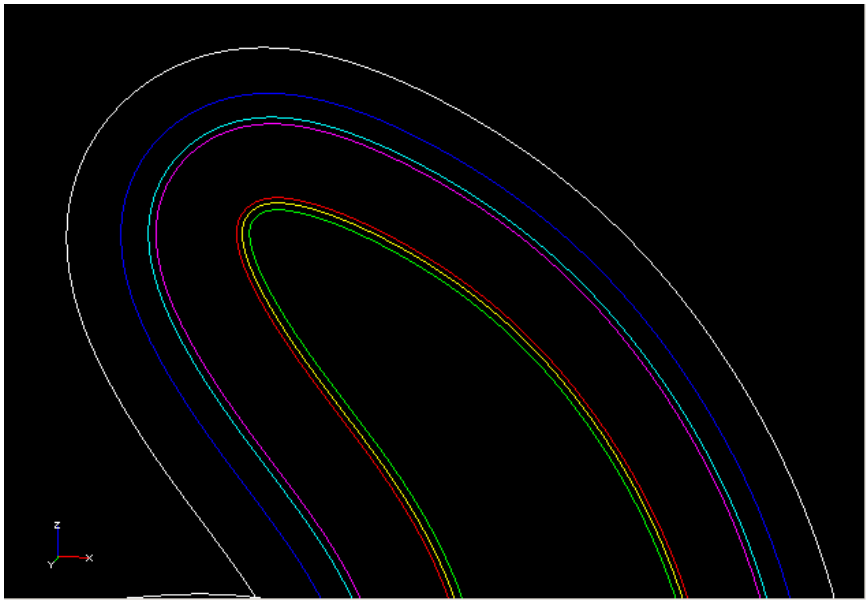


Figure 5.16. Seven layers of the computational model

The material composition of each layer is shown in Tab. 5.1[31]:

Table 5.1. Material of each layer

Homogeneous	Composition:
FW	34% FS Structure 66% He Coolant
Blanket	79% LiPb (90% enriched Li) 7% SiC Inserts (95% d.f.) 6% FS Structure 8% He Coolant
Back Wall	80% FS Structure 20% He Coolant
FS Shield	15% FS Structure 10% He Coolant 75% Borated Steel Filler
Manifolds	52% FS Structure 24% LiPb (90% enriched Li) 24% He Coolant

Multiple neutronics parameters are required for the Stellerator design and are calculated by MCNPX/CGM. Also the ARIES team performs a one-dimensional approximation calculation for these parameters [31]. A comparison of the one-dimensional results and our MCNPX/CGM three-dimensional simulation results is shown in Tab. 5.2:

Table 5.2. Comparison of the one-dimensional results and three-dimensional simulation results

	1-D	3-D	
Local TBR	1.285	1.316	± 0.61%
Energy multiplication (M_n)	1.14	1.143	± 0.49%
Average dpa rate (dpa/FPY)	26	29.5	± 0.66%
Peak dpa rate (dpa/FPY)	40	39	± 4.58%
FW/B lifetime (FPY)	5	5.1	± 4.58%
Nuclear heating (MW):			
FW	156	145	±1.33%
Blanket	1572	1590	±1.52%
Back wall	13	9.8	±6.45%
Shield	71	63	±2.73%
Manifolds	18	19	±5.49%
Total	1830	1820	±0.49%

From the table above we can see that the three-dimensional simulation matched well with the one-dimensional results. This is another proof of the correctness of our MCNPX/CGM implementation especially on a complicated geometry. One additional point, the parameters selected above can be simulated using a one-dimensional approximation. For other parameters like energy space distribution, the MCNPX/CGM code is the only tool available that can obtain the results.

5.4 Conclusion

Through the simple test problems, we have shown that the MCNPX/CGM's implementation yielded the same results as the standard MCNPX code. We have also shown that MCNPX/CGM can perform Monte Carlo simulations on complicated geometries, which validates our approach and algorithm implementation. Another question to consider is the performance issue. This will be discussed in Chapter 6.

Chapter 6

Performance Analysis

In this chapter we will focus on the computational performance of the MCNPX/CGM code. The performance analysis is based on the problems used in Chapter 5. First, we will show the effectiveness of the ray-tracing acceleration techniques on a simple geometry. Second, we will explore the validity and performance of a facet model. Finally we will analyze the effectiveness of other acceleration techniques and the execution time profile of the current code.

6.1. Ray-tracing acceleration effectiveness

First we will show the effectiveness of OBB compared to AABB using the simple 3 cylinders model. The geometry configuration and all other computational conditions are the same as described in Chapter 5. We will show the effectiveness of sorting distance and the distance limit. Finally we will apply the algorithms on the cobalt source test problem.

6.1.1. No acceleration case and AABB case

We begin with the no acceleration case. It will allow us to study the performance of CAD ray tracing function. This will be compared to the AABB case. This will demonstrate the

effectiveness of bounding box, and also show the limitation of the AABB.

Table 6.1. The Results of the AABB study

	MCNPX	MCNPX/CGM			
		Axis aligned bounding box (AABB)			
		No acceleration techniques	Bounding box only	Use bounding box and take advantage of distance to bounding box	Sorted distance to bounding box
Total Track ray tracing (Multiple of MCNPX time)	0.034min (1)	1.18min (34.7)	1.09min (32.1)	1.05min (30.9)	1.03min (30.3)
Number of calls to ray tracing	194198	194198	194198	194198	194198
Number of hits on bounding box		0	387095	387095	387095
Number of CAD surface tests		1370556	387095	377133	373421
Ratio of the number of CAD surface tests to the number of calls to ray tracing		7.06	1.99	1.94	1.92

In table 6.1, the second row is the total numbers of MCNPX calls to the ray tracing function. This function is given a particle position, direction and current cell and it will return which surface this particle will hit and the distance to hit surface. We can see the numbers in this row are same for all the cases. This is because MCNPX/CGM does not change the function calling the ray trace function. A particle in MCNPX/CGM will undergo exactly the same physics as in MCNPX. The first row is the total time spent on this function, which is the total ray tracing time. The third row is the total number of cases that a ray hits a bounding box.

The forth row is the total number of times that a ray CAD surface test is performed. We need to perform a ray CAD surface test only if the ray hits the bounding box. And for some cases using distance-limiting techniques, the ray CAD surface test number can be less than the number of ray hits to the bounding box. This is what is seen in row 4. The ratio in fifth row is the number of ray CAD surface tests to the number of ray tracing function calls. The last row is the ratio of the total run time of MCNPX/CGM to that of MCNPX.

We analyzed 5 cases. The first column contains the standard MCNPX results. For the second column, the CAD geometry engine was implemented, but no ray-tracing acceleration techniques were implemented. We can see that the ray CAD surface test lowers the performance considerably. At this stage the MCNPX/CGM code will be approximately 34 times slower than the standard MCNPX code. That is because the CAD software is concerned more with the functionality but not the performance of each function. This is the price that is paid for the added functionality and complexity. These results are not unexpected. In B.C.Franke's work [5], they experienced a larger slowdown with their CAD based Monte Carlo.

From the comparison of the standard MCNPX and the non-accelerated MCNPX/CGM, it is noted that the ray CAD surface test is a very low performance calculation. Therefore one of the goals of our acceleration techniques is to minimize the number of ray CAD surface test.

The third column is MCNPX/CGM with only the AABB algorithm. We can see in this algorithm, for all the cases a ray hits the bounding box, ray CAD surface test is performed. Thus the third row number 387095 is equal to fourth row number. The reason the number of CAD surface test is larger than number of ray tracing call is because each ray tracing call is actually a ray-body test. This requires multiple ray-surface tests to find the minimized distance to the surface. We have already obtained some acceleration by implementing the AABB. However, in contrast with the big difference in ratio (from 7.06 to 1.99), the performance improvement is rather small (from 1.18min to 1.09min). We believe this is because the AABB is a simple and well-known acceleration technique and that the AABB algorithm is already built into the basis of CGM – ACIS. The performance improvement we gained is primarily because our ray bounding box test algorithm is more efficient than the one in ACIS and we may not need to go through layers of code to reach ACIS's AABB test.

In fourth and fifth column, we take advantage of the distance to bounding box. The fourth column only compares the current distance to the surface with distance to bounding box. We can see there is an improvement in performance and the ratio is reduced to 1.94. For the fifth column, the sorting algorithm was used. We can see there is another small improvement and ratio is 1.92.

Further analysis on this case revealed that of the total 387095 times a ray hits the bounding box, 61.6% of the time the ray is inside the bounding box. This means that the AABB is not a tight

fit because for a tightly fitting bounding box on a surface, a particle will seldom be on the inside of the bounding box. In this case the bounding box algorithm will have no effect. For the remaining outside situation, only 3.5% of the cases are eliminated by our sorting distance algorithm. This is because in many cases, a ray hits the bounding box but does not hit the surface. We need to go through the sorted hit bounding box list to find a real hit. This also demonstrates that the AABB is not a tight fitting bounding box because if a ray were to hit a tightly fitted bounding box it will have a high chance to hit the real surface inside.

Our experiment is consistent with our analysis of the AABB in Chapter 4. The AABB is not a tightly fitted bounding box and we need a better performing bounding box algorithm. This leads to the computational experiment with the OBB tree.

6.1.2. OBB tree acceleration effectiveness

For this computational study the 3 cylinders test problem is also used. A tolerance of 10e-4 is used to generate the faceting surface. The computational conditions are the same as for the AABB calculation. Recall that the cylinder is 40 inches in height and 20 inches in radius, therefore a 10e-4 tolerance gives a fine facet. The computational results are presented in table 6.2.

Table 6.2. The results of OBB tree study

	MCNPX	MCNPX/CGM		
		Axis align bounding box (AABB)	Oriented bounding box tree (OBB tree)	
		Sorted Bounding Box	Sorted Bounding Box	Sorted Bounding Box And Distance Limit
Total ray tracing Time (OBB time)	0.035	1.03	0.83 (0.10)	0.72 (0.10)
Ratio of the ray tracing time to MCNPX	1	32.3	24.4	21.2
Number of calls to the ray tracing function	194198	194198	194198	194198
Number of hits to the bounding box		387095	258620	258620
Number of CAD surface tests		373421	202632	171088
Ratio of the number of CAD surface tests to the number of calls to ray tracing		1.92	1.04	0.88
Total run time (Multiple of MCNPX time)	0.04min (1)	1.36min (34)	1.16min (29)	1.05min (26.3)

Table 6.2 is similar to table 6.1. The last row is total run time of the MCNPX/CGM code and the ratio to MCNPX. Though surface faceting and OBB tree building require some computational time, when discussing performance, the ray tracing time will be used. In Section 6.3 I will discuss why the ray-tracing time is a more meaningful metric. In first column, The standard MCNPX is used. In second column, AABB algorithm with the best case

we achieved – sorting distance is referenced. In third column, a new case is added: the OBB tree is used to find the hit and we sort the distance to the OBB tree of each surface hit. We can see there is a dramatic difference with the AABB case. First, the number of bounding box hits drops from 387095 to 258620. This clearly demonstrates that the OBB tree is a tighter bounding box. Second, the number of CAD surface test drops to 202632 and the ratio is almost 1. This can be achieved only by: 1) having a very tight bounding box so that for almost all cases when a ray hits the bounding box, the ray does hit the surface and 2) our sorting distance algorithm is very effective and eliminates all the non-nearest hit cases. The ratio close to 1 is desirable, because for a ray originating from within a volume will need at least 1 hit of CAD surface to make it outside of the volume. The ratio is close to 1, which means our algorithm is very close to this lower limit.

In the fourth column, the distance limit algorithm has been implemented. We can see the number of CAD surface tests is even lower than the number of ray tracing function calls. The ratio drops down to 0.88. The effect of this algorithm relates the particle's mean free path to the cell's dimension. The mean free path of a particle is the average distance traveled to the next collision and is computed as follows:

$$\lambda = \frac{A}{(N_A \rho)\sigma} \qquad (1)$$

where A is the atomic weight, N_A is Avogadro's constant, ρ is the material density, σ is total interaction cross section and λ is mean free path.

If we have a high density material cell, the mean free path will be less than the cell's dimension,

which means this algorithm will have an even better performance improvement.

However, noting the total ray tracing time from row 1, even with the distance limit algorithm, we can only achieve a time of 0.72 minutes, which is 20 times slower than standard MCNPX. However, we also found the OBB time, which is the time spent to traverse the OBB tree and perform a ray bounding box test. This time is 0.10 minutes. Therefore the ray CAD surface test time is 0.72 – 0.10 equals 0.62 minutes. This shows that the ray CAD surface test is a very slow calculation but the ray OBB tree test has a very high performance, because for every surface in each cell, a ray OBB tree test is performed. Suppose we use a high accuracy facet model and do not perform any ray CAD surface test, we can still obtain a benefit. This idea will be explored and described in Section 6.2.

6.1.3. Acceleration effectiveness on the cobalt source test problem

Now we will test our acceleration techniques on a slightly more complicated test problem, the cobalt source problem. In this geometry configuration, there are more cells and larger variety of surfaces. The result of this study is presented in table 6.3.

Table 6.3. Computation result of the cobalt source problem

	MCNPX	MCNPX/CGM				
		Axis aligned bounding box (AABB)			Oriented bounding box tree (OBB tree)	
		Bounding box only	Sorted bounding box	Sorted bounding box and distance limit	Sorted bounding box	Sorted bounding box and distance limit
Total ray tracing time (OBB time)	0.03	0.63	0.54	0.54	0.47 (0.060)	0.33 (0.057)
Ratio of the ray tracing time to MCNPX	1	21	18	18	15.7	11
Number of calls to the ray tracing function	91700	91700	91700	91700	91700	91700
Number of hits to the bounding box		170643	170643	170643	101582	101582
Number of CAD surface tests		170643	168085	164658	98839	61418
Ratio of the number of CAD surface tests to the number of calls to ray tracing		1.86	1.83	1.80	1.08	0.67

Table 6.3 presents a similar content as tables 6.1 and 6.2. It also shows that the acceleration techniques are very effective. If we compare table 6.3 with table 6.2, there are two differences. First, one is for the AABB only algorithm, the -3-cylinder model is 34 times slower then the standard MCNPX. However, the cobalt source model case is only 0.63/0.03 = 21 times slower. This is because for the cobalt model case, there is a spherical surface, a

cylindrical surface and a cone surface. For this case the standard MCNPX will perform a bit slower on ray surface test, therefore the CAD geometry engine approach will perform relatively better for complicated geometry configurations. After implementation of the acceleration techniques, the performance will be universal better for the cobalt model than for the 3-cylinder model. Second, the ratio for the OBB tree with distance limit algorithm is different. The ratio of 0.67 for the cobalt model is better than the 0.88 for the 3-cylinder model. That is because in the cobalt model we have a large shield where the particle's mean free path is much less than the size of the shield cell. Our distance limit algorithm has a bigger benefit in this case.

6.2. Accuracy of facet VS CAD models

In section 6.1.2 we mentioned the idea of using a facet model as an approximation model and that it may have a huge improvement on computational performance. Table 6.4 presents the comparison of a facet model with CAD model. A tolerance of 10e-4 is used in generating the faceting surface.

Table 6.4. Results of a facet model for the 3 cylinder problem

	MCNPX	MCNPX/CGM		
		Axis align bounding box (AABB)	Oriented bounding box tree (OBB tree)	
		Sorted Bounding Box	Sorted Bounding Box	Facet model
Total ray tracing Time (OBB time)	0.034	1.03	0.83 (0.11)	0.11 (0.084)
Ratio of the ray tracing time to MCNPX	1	30.3	24.4	3.2
Number of calls to the ray tracing function	194198	194198	194198	194198
Number of hits to the bounding box		387095	258620	250150
Number of CAD surface tests		373421	202632	0
Ratio of the number of CAD surface tests to the number of calls to ray tracing		1.92	1.04	

In table 6.4, the last column presents the facet model results. First we note that the total ray tracing time is only 0.11 minutes, which is only 0.11/0.034 = 3.3 times slower than the original MCNPX calculation. This is a comparable performance to the original MCNPX. The

number of hits to the bounding box (third row) is 250150 now, which is a little less than the previous case. That is because we use a facet model, in the leaf node of OBB tree, the object to be enclosed by the bounding box is a triangle; we do not need to enlarge our bounding box to enclose the CAD surface. So for the facet model case, the bounding box is a little bit smaller than the bounding box in CAD model case; therefore the number of hits to the bounding box is also smaller.

Table 6.5. Computational results of the facet model of the cobalt source problem

	MCNPX	MCNPX/CGM		
		Axis aligned bounding box (AABB)	Oriented bounding box tree (OBB tree)	
		Sorted bounding box and distance limit	Sorted bounding box	Facet model
Total ray tracing time (OBB time)	0.030	0.54	0.47 (0.060)	0.074 (0.049)
Ratio of the ray tracing time to MCNPX	1	18	15.7	2.5
Number of calls to the ray tracing function	91700	91700	91700	91700
Number of hits to the bounding box		170643	101582	94146
Number of CAD surface tests		164658	98839	0
Ratio of the number of CAD surface tests to the number of calls to ray tracing		1.796	1.078	

Table 6.5 depicts the facet model results of the cobalt source problem. For this case we also find a huge improvement in the computational performance and fewer hits to the bounding boxes. If we compare these results to the 3-cylinder problem, we find that the cobalt source problem is 0.074/0.030 = 2.47 times slower than the original MCNPX. This is because for a more complicated geometry configuration, the MCNPX/CGM acceleration techniques have a better performance.

While there is a huge improvement in the computational performance, because a facet model, which is an approximation, is used, the correctness or accuracy of the model will be a concern. We find that if we use very coarse facet (tolerance is relatively big), the computational result will be slightly different from that of the CAD model. For example, for 10k source particles in the 3-cylinders problem, the source particles will experience 3194 collisions. If we use a very coarse facet (tolerance 0.1), those particles will experience 3156 collisions and some of the tally bins will have some differences compared to the CAD model tally bins. However if we shrink down the tolerance (1e-4 in our case), the particles will experience 3194 collisions again and the computational result will be exactly the same as for the CAD model.

The reason for this is that the difference between facet model and CAD model is very small. Only if a ray hits the area, which belongs to a cell in the facet model that is different than for the CAD model, will the computational results differ by 1 particle. We can conjecture the computational result as follows: if we allow for an infinite computational time and during this

time a ray experiences a hit in this area, the difference in the computational result between the facet model and the CAD model will be equal to shadow area divided by the total area (see Fig. 6.1). That is to say, the difference will be proportional to square of tolerance (in the two-dimensional case, for the three-dimensional case it will be the cube of tolerance). Given a limited computational time (which is the real case), the probability a ray will hit the shadow area is very small and hence it is highly unlikely that a ray will hit the shadow area (ray 1 in Fig 6.1). That is why we can still have identical computational results with CAD model.

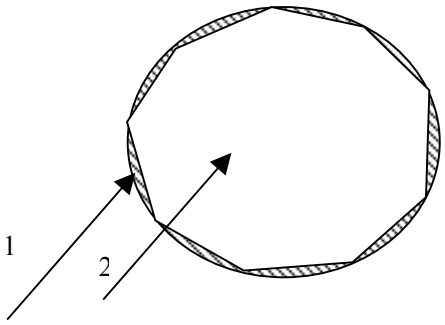


Figure 6.1. Accuracy of facet model

Even if the computational result is different from that of the CAD model, the computational result is second order convergent (or third order for the three-dimensional case) with tolerance. The above is just speculation. In the research, we just find for a certain amount of computational time, that a tolerance for which the computational result is identical to CAD case can always be found. The detailed research to investigate the convergence of the facet model results with tolerance to the CAD model will be part of the future work.

Because for a certain number of histories' computed, the computational result of the CAD

model will not change, however the computational result of the facet model could change with respect to different tolerances chosen. One way to handle this would be to shrink down the tolerance from a relative big number. Once the tolerance reachs a level for which the computational result does not change, this means that our calculational result is identical to the CAD model case. The benefit of this method is we do not need to perform a CAD model simulation.

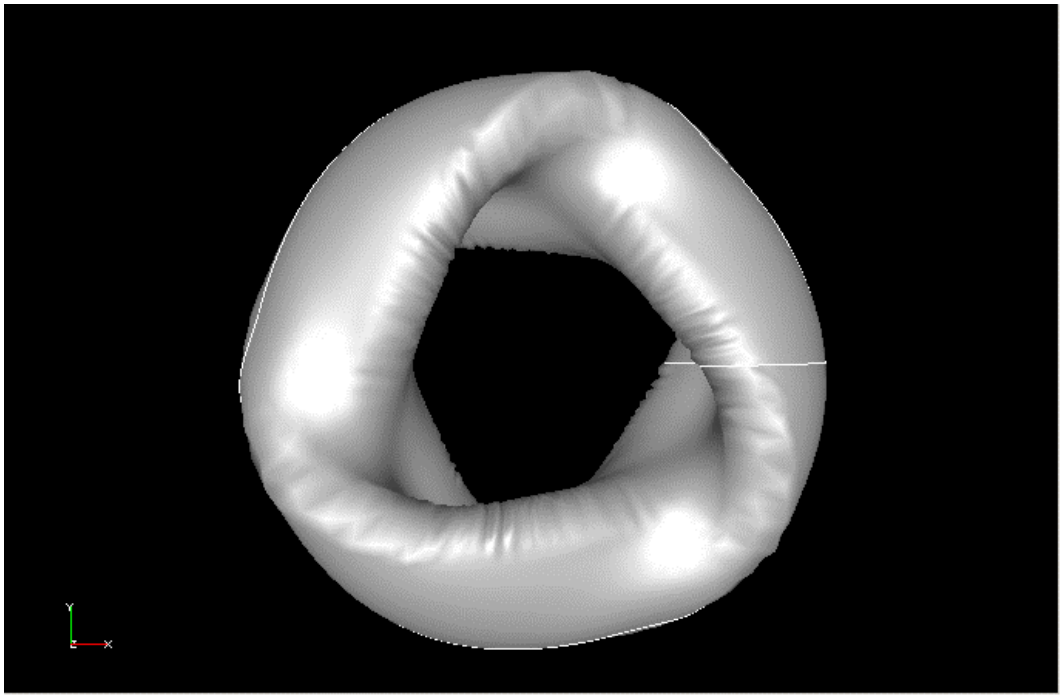


Figure 6.2. The ARIES-CS Compact Stellerator

The ARIES-CS Compact Stellerator is a torus shaped object with layers. The innermost layer has only 1 surface. The other layers have only two surfaces. Our OBB tree used with the CAD model algorithm will have no effect on the innermost layer because there is only one surface and will have a limited effect on other layers. The more serious problem is that the

ray CAD surface test is extremely expensive on this complicated curved surface. To perform a neutron wall loading calculation, it took 10 days' of computational time and only achieved a 10% statistic error. Therefore, the performance of any algorithm that still performs a ray CAD surface test is not satisfactory. In this case facet model is the only choice.

For the computation of ARIES-CS Compact Stellerator model, we choose a tolerance equal to 1e-2 because the Stellerator is a much larger object than the 3-cylinders and the cobalt source geometry. Still for the neutron wall loading calculation, the facet model with the OBB tree algorithm achieves a 1% statistical error in only a 1-hour calculation. That is about 24000 faster than if the CAD model is used!

6.3. Discussion and Benefit

There is some computational overhead with the faceting and OBB tree building algorithms. For the test problems considered, the total time in these algorithms range from a few seconds to 4 minutes depending on geometry and the tolerance. However as mentioned before, because our geometry and system are static, the faceting and OBB tree building operation are preformed once prior to the Monte Carlo simulation. For a complicated geometry system, if a low statistical error for the computational result is required, then a long computational time would be necessary. In this case, the fixed overhead of faceting and OBB tree building is relatively small. The ratio of the time spent on ray tracing of MCNPX/CGM to MCNPX's total time is approximately equal to the ratio of total running times of both codes.

Then overall, the CAD geometry engine level implementation has many benefits. First, the user will use existing CAD tools to generate the geometry configuration; for example CUBIT, in our case. Making use of the user-friendly interface of CAD tools, it is much easier to generate a geometry configuration than using MCNPX geometry package, especially for a complicated geometry. Second, because most CAD tools can usually read in or save to multiple CAD formats, the compatibility issue is simplified. Even if a new geometry format is developed and becomes available, we do not need to worry about it because the CAD software developers will implement the new format in the new version of the CAD tools (system). Third, complicated geometries can be simulated in Monte Carlo code without any approximation, the CAD model, or use the faceted model with a small approximation error. Fourth, in this approach, the physics portion is totally separated from the geometry portion allowing the nuclear engineer, the physicist or the scientist to focus on the physics and interaction science like new particles, new interactions, variance reduction techniques, and setup and problem modeling. The geometry modeling will automatically take advantage of the latest progress in CAD software.

The benefits of faceting the surface and applying the OBB tree are:

1. A Facet surface can be very close approximation to a CAD surface. Therefore we can obtain an approximate model with arbitrary accuracy. In the implementation side, the CAD geometry engine provides the surface faceting algorithm and the functionality. We do not

need worry about this or put an effort into this.

2. The Oriented bounding box on facet surface gives the tightest fitted bounding box. This means the ray is seldom inside the bounding box, which means the bounding box is almost always an effective ray tracing acceleration technique.

3. However, using one OBB for each facet will generate many bounding boxes for one surface. The finer the surface faceting, the larger the number of bounding boxes. Testing each bounding box for a possible hit will make computation quite inefficient. The OBB tree algorithm solves the computational cost problem. It only requires O(log(n)) tests to find a hit, where n is the number of facets, which means we can afford a very fine surface faceting.

For a convex object, using the surface faceting and OBB tree, we can expect that the ray surfaces test will find only one hit. The exception is the case where the ray exactly hits an edge or vertex. This is a trivial case. Although there may be many surfaces in a CAD convex cell, we only need to do one ray-CAD surface test to obtain the distance to a hit. Here we already see a large acceleration in comparison to the “minimal positive distance principal” used in MCNPX. MCNPX needs to test every surface in a cell to find the hit and the distance.

In the case a cell is concave, it is possible that the ray will hit more than one surface, or even that one surface is hit more than once.

The benefit of the sorted distance algorithm is that only one ray CAD surface test is required for convex and concave cells. That is the theoretical lower limit for a pure computational geometry acceleration algorithm, because a ray inside a cell will always hit a surface. To obtain an accurate hit distance to the surface, we require at least one ray CAD surface test.

Having one ray CAD surface test also means we do not need to implement other ray-tracing acceleration techniques from the computer graphics area, because there is no additional benefit to be gained.

The benefit of the distance limit algorithm is that in some cases we do not need to perform a ray-CAD surface test, that is, on average a particle will need less than one ray CAD surface test per cell. The amount of benefit obtained depends on the physics of the problem. In a cell whose dimensions are much larger than the simulated particle's mean free path, the algorithm will have a large benefit because most of time the particle will undergo an interaction.

The benefits of using a facet model are:

1. The computational performance is high because it eliminates the ray CAD surface test.
2. This approach can be applied to any kind of surface because it converts any CAD surface type into only one surface type – a faceting surface. Further research will only focus on the ray faceting surface test.

3. Because the OBB tree algorithm performance scales as O(log(n)) for the ray bounding box test to find a hit, a high accuracy faceting surface, will not face a large computational time increase. For the case of a complicated CAD surface, such as B-spline surface, the ray test of a high accuracy faceting surface will still have a better computational performance than the ray CAD surface test.
4. Because the faceting surface is controllable, we can easily generate approximate models with different levels of accuracy. The coarse approximation can be used as a fast estimation. The finest model can be used as a substitute of the CAD model. Also going from the coarse model to the finer model, the computational results will have fewer differences because the computational result converges to the CAD model case. We can know the computational accuracy without direct comparison with the CAD model case by observing the rate of convergence with finer and finer facets.

Chapter 7

Summary and Future Plan

In summary, this research addresses the problem that current Monte Carlo codes can not perform a high accuracy and high performance simulation on complex, complicated geometries. The research develops a CAD based Monte Carlo code that has an execution time that is competitive with the original code for the simple problems investigated. It involves the implementation of CGM into MCNPX and uses ray-object intersection acceleration techniques to speed up the execution of the Monte Carlo code. The origin of MCNPX/CGM includes:

1) Integrate a general-purpose solid model engine of CAD software into a general-purpose three-dimensional, multi-particle transport code. The MCNPX/CGM will have the broadest application field because it can be applied to a transport problem with any complex solid geometry model. Also, we can use the most recent geometry functions that will be updated by the CAD software developer instead of writing and maintaining our own geometry functions.

2) Review the ray-object intersection acceleration techniques in the area of computer graphics and select the one which can be applied to the Monte Carlo code, investigate the nature of the

coupling between the Monte Carlo calculation and geometry evaluation, and optimize and explore the best ray-object intersection acceleration techniques for the Monte Carlo code.

3) Take advantage of the advanced features of the CAD software; provide advanced geometry construction ability such as facet based geometry. The user can use facet-based geometry to construct different levels of approximation of the solid model. Also use of this feature, allows for an efficient implementation of the geometry evaluation for the ray-object intersection acceleration technique; the OBB tree.

However, the current MCNPX/CGM is still a research code. There are still things that can be done to make it better.

One research area was mentioned before --- the computational result of facet model converges to the CAD model when the tolerance becomes small. This requires further research.

Another useful upgrade would be to combine the CAD geometry and the MCNPX geometry. CAD tools are good at constructing complicated geometries. However MCNPX is good at generating simple geometries. If we can combine the benefits of both, the future MCNPX/CGM would have more features and would be even easier to use. Especially in cases where the geometry is only used to obtain a specific tally such as the surface tally subdivision. Combining the CAD geometry and the MCNPX geometry will be very useful for this case.

There are some features that are available in MCNPX but are not available in MCNPX/CGM. For example the F2 (surface current) tally requires surface area information. The current MCNPX/CGM code does not pass the surface area information to MCNPX. Therefore this feature is missing. Future work should make all features provided by MCNPX available in MCNPX/CGM.

Because MCNPX/CGM requires a non-manifold geometry, a user needs to imprint and merge the CAD geometry to make it compatible with MCNPX/CGM. If we could eliminate this requirement by letting MCNPX/CGM perform these operations inside the code, it would be much easier for users to construct complicated geometries.

Other upgrades to be considered include:

1) Pass material properties to the geometric model. This would make the user see the cell and then allow the user to input the material information directly into the cell.

2) Make a parallel version of the MCNPX/CGM code. Then MCNPX/CGM could take advantage of the computational power of a cluster.

(Rock: How would you do this? Would you algorithm be able to speed up more than linear? There is probably a lot more that you can say here.

Mengkuo:

References

1. Hammersley, J.M., and Handscomb D.C., 1964. Monte Carlo Methods. Fletcher & Son Ltd, Norwich, ISBN 0 412 15870 1
2. Briesmeister, Judith F., Editor March 1997. MCNP - A General Monte Carlo N-Particle Transport Code Version 4b, LA-12625-M, Version 4B Manual
3. http://mcnp-green.lanl.gov/manual.html
4. Waters, Laurie S., Editor November 14,1999. MCNPX USER'S MANUAL version 2.1.5 Los Alamos National Lab. TPO-E83-G-UG-X-00001
5. Franke, B.C., Kensek, R.P., Schriner, H.K., Lorence, L.J., Gelbard, F., and Warren, S., 2001 Adjoint Charge deposition and CAD Transport in ITS. M&C 2001 Salt Lake City, Utah, USA, September 2001.
6. Electron Gamma Shower (EGS) Monte Carlo Radiation Transport Code, Online manual, http://www.slac.stanford.edu/egs
7. FLUKA Online manual, http://www.fluka.org
8. Lewis, D., Pro/Engineer, Online manual, http://caesar.sdsu.edu/proe/
9. Planchard, D.C., and Planchard, M.P., Engineering Design with SolidWorks 2001
10. CAEN Technical Note, university of Michigan. Introduction to Unigraphics. Online manual http://www.engin.umich.edu/caen/technotes/unigraphics.pdf
11. ACIS http://www.spatial.com/
12. Parasolids http://www.theorem.co.uk/docs/fs_int.htm
13. Tautges, Timothy J., 2001. CGM: a Geometry Interface for Mesh Generatoin, Analysis and

Other Applications. Engineering with Computers, 17:299-314 (2001)

14. CUBIT http://sass1693.sandia.gov/cubit/

15. MCNP Vised http://www.mcnpvised.com

16. Liu X P, Tong L L, Luo Y T and Wu Y C, Development & Application of MCNP Auto-Modeling Tool: MCAM 2.0, 7th China/Japan Symp. on Materials for Advanced Energy Systems and Fission and Fusion Engineering, Lanzhou, July 31-Aug.2, 2

17. TopAct: Automated Translation from CAD to Combinatorial Geometry for Radiation Transport Analysis, Manson, Steven J; Williams, Eric K; Triggs, Brian P

Space 2005; Long Beach, CA; USA; 30 Aug.-1 Sept. 2005. 8 pp. 2005

18. Tsige-Tamirat H, Fischer U, CAD interface for Monte Carlo particle transport codes. The Monte Carlo Method: Versatility Unbounded in a Dynamic Computing World; Proc.of the Conf., Chattanooga, Tenn., April 17-21, 2005

19. A Serikov1, 3, U Fischer1, R Heidinger1, K Lang1, Y Luo2, H Tsige-Tamirat1,. 2005, Radiation shielding analyses for the ECRH launcher in theITER upper port, Journal of Physics: Conference Series 25 (2005) 181–188

20. J. A. Halbleib, R. P. Kensek, T. A. Mehlhorn, G. D. Valdez, S. M. Seltzer, and M. J.Berger, ITS Version 3.0: The Integrated TIGER Series of Coupled Electron/Photon Monte Carlo Transport Codes, Sandia report SAND91-1634 (1992).

21. Shirley□Peter., 2002. Fundamentals of Computer Graphics. A K Peters, Massachusetts.

22. Glassner, Andrew., Editor. 1989. An Introduction to ray tracing, Acadimic, London Arvo, J., and Kirk, D., Chapter 6: A survey of Ray Tracing Acceleration Techniques

P201-P262

23. Rubin, S. and Whitted, T., July 1980. A three-dimensional representation for fast rendering of complex scenes. Comput. Graph. 14(3), 110-116

24. Fujimoto, A., Tanaka, T. and Iwata, K., April 1986. ARTS: Accelerated Ray-Tracing System. IEEE Comput. Graph. Appl. 6(4), 16-26

25. Haines, E.A. and Greenbery, D.P., September 1986. The light buffer: a shadow testing accelerator. IEEE Comput. Graph. Appl. 6(9), 6-16

26. Ohta, M. and Maekawa M., 1987. Ray coherence theorem and constant time ray tracing algorithm. Computer Graphics (Proc. of CG International '87) (ed. T.L. Kunni), pp.303-314

27. Synder, J.M. and Barr, A.H., July 1987. Ray tracing complex models continning surface tessellations. Comput. Graph. 21(4), 119-126

28. S. Gottschalk , M. C. Lin , D. Manocha, OBBTree: a hierarchical structure for rapid interference detection, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, p.171-180, August 1996

29. Eric W. Weisstein. "Symmetric Matrix." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/SymmetricMatrix.html

30. Eric W. Weisstein et al. "Orthogonal Matrix." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/OrthogonalMatrix.html

31. L. El-Guebaly, R. Raffray, S. Malang, J. Lyon, and L.P. Ku (invited), "Benefits of Radial Build Minimization and Requirements Imposed on ARIES-CS Stellarator Design," Fusion Science & Technology, 47, No. 3, 432 (2005). Also, University of Wisconsin Fusion

Technology Institute Report, UWFDM-1233.

32. S. P. Hirshman, W. I. van Rij, and P. Merkel, "Three-Dimensional Free Boundary Calculations Using a Spectral Green's Function Method," Computer Physics Communications, 43(1986) 143-155.Ogawa, K., Takahashi, S., and Satori, Y., 1997. Description of an Object in Monte Carlo Simulations. IEEE Transactions on Nuclear Science, VOL 44 No.4 August 1997

33. Wang, H., Jaszczak, R.J., and Coleman, R.E., 1992 Solid Geometry-Based object Model for Monte Carlo Simulated Emission and Transmission Tomographic Imaging Systems. IEEE Transactions On Medical imaging, VOL.11 No.3 September 1992