NOTE: Use these buttons to print or save the form. DO NOT use the browser tool bar. SAVE PRINT

LA-UR-12-25478 Rev 1

Approved for public release;
distribution is unlimited.

Title:	The MCNP6 Book On Unstructured Mesh Geometry: Foundations (U)
Author(s):	Roger L. Martz David L. Crane
Intended for:	MCNP6 code release to RSICC, Oak Ridge, TN and general distribution.

Los Alamos
NATIONAL LABORATORY
EST.1943

Form 836 (7/06)

SAVE PRINT CLEAR FORM

The MCNP6 Book On Unstructured Mesh Geometry: Foundations

For use with MCNP6 Version 1.1 Beta or later.

Roger L. Martz
David L. Crane

March 2014

Abstract

Hybrid geometries with unstructured mesh embedded in a constructive solid geometry universe representation is a recently added (less than 10 years old) feature to MCNP6 and was present in its first production release as well as several prior beta versions. This capability requires methodologies previously not used in MCNP. After a brief introduction to the topic of finite elements this document discussions the methods used in implementing particle tracking on unstructured meshes in MCNP.

Chapter 1: Fundamentals

A mesh-tracking library has been developed for use with the MCNP6™ computer code [1]. Integration with this library enables MCNP to perform calculations on hybrid geometries that consist of unstructured mesh (UM) geometry representations embedded as universes using the legacy constructive solid geometry (CSG). This capability provides flexibility to the practitioner so that the strengths of each geometry type, UM or CSG, can be fully utilized. CSG provides fast and efficient particle tracking when the geometry is represented easily by first and second degree surfaces of analytical geometry. However, building models of complex geometries can be difficult, tedious, time consuming, and error prone [2,3,4]. An additional drawback of the CSG-only model is that this geometry does not integrate well with finite element codes for multi-physics analysis. Depending upon the geometry's complexity, particle tracking on an UM may be slower than with the equivalent CSG model because of more work that takes place in the code - the subject of this document. In some situations, particles may track as fast or faster in the UM model compared to the equivalent CSG model because of the UM tracking implementation strategy. Equivalent is in the eye of the beholder since the CSG implementation may be either a minimalist one or an attempted reproduction of the mesh using the CSG features. Whatever the situation, any slower computational time is a welcome trade off to some users since complex models can be constructed with less difficulty using the solid modeling tools present in state-of-the-art computer aided design (CAD) and computer aided engineering (CAE) software. An inherent benefit of the UM geometry is that it does integrate well with finite element codes since the CAE tools are themselves often integrated with finite element solvers.

The mesh-tracking library is written using modern Fortran and programming standards. The library was created with a defined application programer interface (API) so that it could easily integrate with other particle tracking codes. However, this work is evolving so that from time to time the interface does change. Please consult the latest documentation or look at the code for the latest description. As a library, all use of its features are through the API.

The purpose of this document is to describe in detail the methodologies that support tracking particles on an unstructured mesh representation of geometry with the goal that this would be the only source of background information that one would need in order to start writing computer code or to understand what has been implemented in MCNP. With this objective in mind, some might read a section and say "I already knew that." So be it. The overall motive is to be as complete as possible. Details on the API are presented elsewhere.

Some of the topics presented here can quite easily be presented in abstract ways that seem to appeal to computer scientists and mathematicians. It is the desire in this work to present the topics in a clear and simple fashion. These topics cover what is in the current MCNP implementation for tracking on an unstructured mesh. This is by no means an endorsement that these topics or this discussion describes the best or most efficient means for implementing unstructured mesh tracking. Maybe better methods will emerge as the technology matures.

However, researching these methods then implementing and testing them is no trivial task; considerable time and effort has been expended for the current version of the library. Whether the following is efficient or not, the most important goal is that the implementation yields an accurate result in a reasonable amount of time.

Finite Elements

In very simple terms, the finite element method (FEM) and its practical application, often known as finite element analysis (FEA), is a numerical technique for finding approximate solutions of partial differential and integral equations. The FEM is a numerical method like the finite difference method, but is more general and powerful in its application to real-world problems that involve complicated physics, geometry, and/or boundary conditions [5]. In reality, the term FEM actually identifies a variety of techniques that share common features. Over the last several decades, the advancement in computer technology has enabled solving even larger systems of equations, to formulate and assemble the discrete approximation, and to display the results quickly and conveniently. This has also helped the finite element method become a powerful tool.

No one individual can claim credit as the inventor of the finite element method. Its modern development can be traced back to the work of Alexander Hrennikoff in 1941 and Richard Courant in 1942 [5-7]. Courant's contribution is considered evolutionary, drawing on a large body of earlier results for PDEs developed by Rayleigh, Ritz, and Galerkin, and has earned him the title of "the father of finite elements". The method remained dormant for half a generation until the advent of modern computers [5]. Development of the FEM began in earnest in the 1950s for airframe and structural analysis. Its formal presentation is attributed to Turner, Clough, Martin, and Topp in 1965 [8,10] and to Argyris and Kelsey in 1960 [5,8]. The term “finite element” was first used by Clough in 1960 [5,9].

During the 1950s and 1960s the FEM technology transferred from the aerospace industry to a wider range of engineering applications such as heat transfer and fluid dynamics. Since then, the FEM has been used successfully in various engineering disciplines over the decades and continues to grow with the availability of more powerful and affordable computer systems. Not only do these computer systems permit the engineer to analyze all kinds of problems by solving larger systems of partial differential equations for example, they are generally coupled with interactive graphical user interfaces (GUI's) and solid modeling engines that allow easier construction of highly complex three-dimensional models, than what was possible in the past, along with quick and convenient display of results.

This chapter is not intended to contain an in-depth discussion of the FEM. One can find numerous materials on the internet or in the many text books that have been published in this area, not to mention the user's manual for both free and commercially available FEA software. However, this chapter does discuss those concepts borrowed from the FEM/FEA field for use in tracking particles on an unstructured mesh in a Monte Carlo code such as MCNP and does present relevant background information.

The Basic Features

The FEM is endowed with three distinct features that account for its superiority over other competing methods [5]:

- First, the problem's physical domain, either geometrically simple or complex, is represented as a collection of geometrically simple sub-domains, called finite elements. Other methods would refer to such a sub-domain as a cell, mesh-cell, zone, etc. In order to avoid confusion, the practice in this work is to use the terminology finite element or element when dealing with the unstructured mesh and to use appropriate terminology in discussing other paradigms. Each finite element is viewed as an independent domain by itself. Here the word "domain" refers to the geometric region over which the equations are solved. Today's CAE tools perform this function quite easily once the solid model is constructed. Various element shapes and sizes can be used depending on the needs of the analysis.
- Second, over each finite element, algebraic equations among the quantities of interest are developed using the governing equations of the problem. These will be different depending upon the type of problem to be solved.
- Third, the relationships from all elements are assembled using certain inter-element relationships to obtain a solution of the whole.

Approximations enter engineering analyses at several stages. The division of the whole domain into finite elements may not be exact, introducing error in the modeled domain. The second stage is when element equations are derived. Typically, the dependent unknown or state variable (u) of the problem is approximated using the basic idea that any continuous function can be represented by a linear combination of known functions (ϕ_i) and undetermined coefficients (c_i).

$$u \sim \sum_i c_i \phi_i \qquad \text{(Eq: 1-1)}$$

Often, algebraic relations among the undetermined coefficients c_i are obtained by satisfying the governing equations over each element where the solution is represented as a linear combination of terms evaluated at the nodal points. The approximation functions ϕ_i are often taken to be polynomials, and they are derived using concepts from interpolation theory. Therefore, they are termed interpolation functions; sometimes they are referred to as shape functions. Thus, approximation errors in the second stage are introduced both in representing the solution u as well as in evaluating the integrals. Finally, errors are introduced in solving the assembled system of equations.

A polynomial is the most common form of the functions ϕ_i for two principal reasons [10]: (1) It is easy to handle the mathematics of polynomials in formulating the desired equations for various elements and in performing digital computations. (2) A polynomial of arbitrary order permits a recognizable approximation to the true solution.

The subdivision of a whole domain into parts has two advantages: (1) Allows accurate representation of complex geometry and inclusion of dissimilar material properties. (2) Enables

easy representation of the total solution by functions defined within each element that capture local effects (e.g., large gradients of the solution).

Considerations & Contrasts

The geometry of the domain can be discretized, depending on its shape, into a mesh of more than one type of element (to be discussed later). Unless restrictions are place on the element type, this requires that the Monte Carlo particle tracking routines possess the ability to track on these different element types. Although textbooks discuss many different types of elements, in terms of three-dimensional elements, most CAE tools use meshing algorithms that generate at most three basic types of at most two orders (see below).

The number and the location of the nodes in an element depend upon (a) the geometry of the element, (b) the degree of the polynomial approximation, and (c) the weighted-integral form of the equations in the underlying FEM. By representing the required solution in terms of its values at the nodes, an approximate solution at the nodes is obtained. Generally, Monte Carlo estimation of results over the mesh would not be as localized. That is, path length estimators or collision estimators calculate results that are averaged over an element's volume and hence could be considered representative at the element's centroid. Naturally, point and surface estimators can be implemented on the mesh for specific locations, but could prove too costly, in terms of calculation time and computer memory, for all node locations and element faces.

There are three sources of error in a finite element solution: (a) those because of the approximation of the domain; (b) those because of the approximation of the solution; and (c) those because of numerical computation [5]. A similar statement can be made regarding the Monte Carlo method even though this method is viewed as more exact or accurate. If the Monte Carlo calculation uses the same unstructured mesh representation of the geometry as a corresponding finite element calculation, it will likewise suffer from the approximation of the domain by the elements. For relatively simple geometries that can effectively use the CSG capability, approximation of the domain is a non-issue.

The meshing algorithms that generate unstructured mesh representations of the geometric domain cannot guarantee meshes that are free of excessively distorted elements or have a sufficient number of elements in a region containing high gradients in the solution, both of which result in loss of accuracy in the finite element calculation or, in the case of nonlinear problems, non-convergence of solutions [5]. Because of some of the approaches borrowed from the FEM, tracking of particles through irregularly shaped elements is a problem for Monte Carlo codes as well. The degree of irregularity may be different between the two methods regarding where and when accurate results are obtained. To date, empirical evidence has shown that if the unstructured mesh passes the mesh quality checks from the CAE tools, then there is no problem tracking on that mesh as it relates to the element shape. However, the Monte Carlo calculation does not need a refined mesh in a region containing high gradients in the solution to obtain an accurate result. That is, it is not a necessary condition. What is more important to the Monte Carlo calculation is how accurate the elements represent the region through its volume - a more integral quantity. Whereas a sufficient number of elements may be necessary for the finite element calculation to converge to the correct result, a more refined mesh may only be needed in the Monte Carlo calculations to visualize the detail, particularly in a high gradient region.

Element Types

Finite element text books discuss many different types of one-, two-, and three-dimensional element types. Since MCNP is a generalized three-dimensional Monte Carlo code for radiation particle transport, our concern is with the three-dimensional element types that can be generated with a commercially available software product such as Abaqus/CAE [11]. The elements for consideration are then those with 4, 5, or 6 sides and commonly referred to as tetrahedra, pentahedra or prisms or wedges, and hexahedra. These element types are depicted in Figure 1-1.

For first-order elements there are nodes at the vertices so that a tetrahedron has 4 nodes, a pentahedron has 6 nodes, and a hexahedron has 8 nodes. Modern meshing algorithms do not guarantee the placement of four or more nodes associated with one face to lie within a plane. Hence, these faces may possess some degree of curvature and are termed bilinear with the corresponding volume termed trilinear. For second order elements, in addition to the vertex nodes there are nodes on the element edges midway between the vertices. A second-order tetrahedron has 10 total nodes, a second-order pentahedron has 15 total nodes, and a second-order hexahedron has 20 total nodes. Because each of the second order faces has six or more nodes, faces on all of these element types may result with curvature and are termed biquadratic with the corresponding volume termed triquadratic. Due to the extra edge-nodes, a higher degree of curvature can be obtained, making these second-order elements better suited to model objects with curved surfaces; that is, fewer second-order elements than first order elements are needed to accurately model these objects.

The element curvature for the trilinear and triquadratic elements pose a challenge when calculating volumes; obviously, simple formulas cannot be used. A special method has been devised for this purpose and is the subject of Chapter 10. Accurate volumes are required to calculate accurate results at the elemental level.

Requirements Summary

An examination of previous work in this field shows that particle tracking on a mesh, for the type of problems that are covered by this current work, has been rather limited and specialized. Computer codes exist that track particles on a Cartesian mesh or one comprised of first order tetrahedra as part of or separate from a CSG type environment. Generally, in these implementations a totally contiguous mesh is required; that is, one without gaps among elements and all space is uniquely defined. MCNP is considered a Monte Carlo code with generalized geometry. In this vane, the UM capability for MCNP needs to be as accommodating as reasonably possible.

The requirements for MCNP's UM capability stem in part from the modelling capability of the CAE tools. Generally, geometry models are assembled out of parts. In turn, a part may be sectioned into smaller regions with different *properties* in each region. One of those properties can be the element type. When the part is meshed, all of the elements will be connected into a contiguous object; all elements except surface elements will share all of their faces with a neighbor element. One or more instances of a part may be combined with instances of other parts to form an assembly. If the parts possess curved surfaces, then often during the assemblage, small user-controlled overlaps and gaps will exist in the final product. It is possible

with some CAE tools to eliminate these gaps and overlaps; but, it may not be worthwhile or desirable for the user to do this. Therefore, the tracking routines in the mesh library must be robust enough to handle reasonable gaps and overlaps. However, it is known that one element cannot be totally inside the other. At this time there is no quantitative measure of what is reasonable. Users always have the option of refining their models in the CAE tool to minimize or eliminate gaps and overlaps if their models behave poorly in the Monte Carlo code.

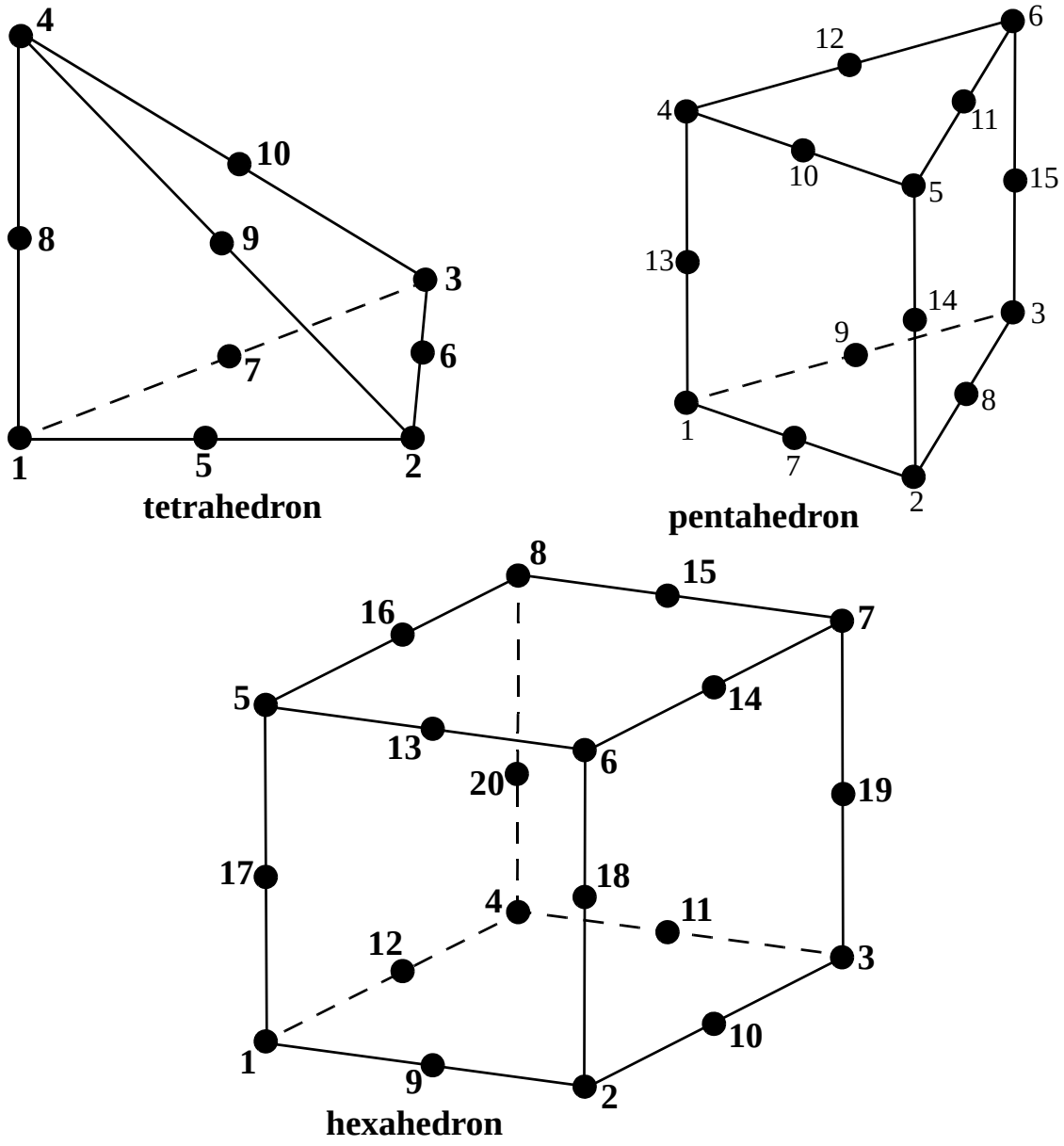

Figure: 1 - 1. Finite element types. First-order elements have nodes only at the vertices.

Particle tracking through CSG objects is basically a surface to surface (S2S) process. That is, the code tracks the particle from the surface of one object (part) to the surface of another unless interrupted because the tracking has exceeded the distance to collision or distance to a variance reduction event, for example. However, in order to accumulate results by mesh element, element to element (E2E) tracking is required so that path length estimates of the flux and various flux functionals can be accumulated as the particle tracks from one element face to another; these results are available for *ex post facto* analysis. The user should note that the terminology E2E tracking is a bit of a misnomer. Tracking is still from surface to surface of the element unless otherwise interrupted; this is generally on a much finer level than the traditional S2S tracking on a part. Obviously, for parts with high mesh densities, it will take the code longer to track through these in E2E mode.

Initial requirements for the UM library are summarized with the following list.

- accommodate first and second order 4, 5, and 6-sided polyhedra
- any combination of element types may appear in a single geometry model
- parts may not contain both tetrahedra and hexahedra or tetrahedra and pentahedra; pentahedra and hexahedra can be together in the same part
- robust handling of overlaps and gaps
- use E2E tracking to produce path length results at the element level
- produce a data file for post processing results analysis

Chapter 2: SKD-Trees

During the course of a Monte Carlo simulation on an UM model, many searches are required to determine in which polyhedral element a particle resides (containment search) or what face of an element is intersected (intersection search) during particle tracking. In the course of performing these tasks, it is desirable to use an effective, that is practical and efficient, data structure for the storage and manipulation of the many polyhedra which comprise a finite element mesh model. The *skd-tree* is such a data structure [12].

kd-Trees: General Discussion

The *k*-dimensional homogeneous binary search tree, *kd*-tree, was first proposed by Jon Louis Bentley [13]. The *kd*-tree has been the subject of intensive research and many variants have been proposed over the years, including the *skd*-tree variant. In principle, the *kd*-tree is a binary tree where the underlying space is partitioned on the basis of the value of just one attribute at each level of the tree instead of on the basis of the values of all *k* attributes.

In a traditional binary search tree, records are defined by only one key. In a *kd*-tree, records are define by *k* keys. However, the key that determines the next sub-tree (i.e., left or right branch) varies with the level in the tree. A node in the tree serves as a representation of the actual data point and the direction of a search. A discriminator whose value is between 1 and *k* inclusive, is used to indicate the key on which the branching decision depends. A node has two children or sub-nodes or branches or buckets, a low bucket (LOBUC) and a high bucket (HIBUC). If the discriminator value at the node is the j'th attribute (key), then the j'th attribute of any sub-node in the LOBUC is less than the j'th attribute of the node, and the j'th attribute of any sub-node in the HIBUC is greater than the j'th attribute of the node. This property enables the range along each dimension to be defined during a tree traversal and the ranges are smaller in the lower levels of the tree.

Restricting the number of attributes that are tested at each level of the tree has advantages: (1) one test is made at each level instead of *k* tests, (2) memory is only required for one discriminator at each node, (3) search algorithms are simpler at each recursive step because of only two options (LOBUC vs. HIBUC). The one attribute test per level is disadvantageous in that decomposition in *k*-dimensional space has now become a sequential process as the order in which various axes (attributes) are partitioned is important. Therefore, the *kd*-tree is often characterized as a superior serial data structure [13].

The spatial *kd*-tree, *skd*-tree, is a spatial access method for *k*-dimensional space where successive levels are split along different dimensions. That is, the *kd*-tree attribute referred to above is spatial. Objects are indexed by their centroid and a minimum axis-aligned bounding box (MAABB) for all objects in a node is stored. This data structure can be viewed as an extension to *kd*-trees and successfully handles finite sized objects without object duplication or object mapping [12]. However, the non-leaf nodes do not contain actual data points; all data

points of interest appear in the leaf nodes. This tree is a balanced binary tree with exactly 2N-1 nodes and $\log_2$ N levels, where N is the total number of elements stored in the tree, and each element is associated with a unique leaf. The tree is constructed in O(N $\log_2$ N) time and is searchable in O($\log_2$ N) time.

skd-Tree Construction

As mentioned in the introduction to this chapter, two types of searches are required for the MCNP implementation. Containment and intersection search terminology is used in this work and is not necessarily identical in meaning to the containment and intersection search terminology discussed in the literature by other authors.

The *skd*-trees for this work are needed to store information about the model geometry. Generically, the geometry can be constructed from many simpler objects. In the CAE paradigm, these objects are often referred to as parts or instances of a part (since a single part may be reused many times). It is not these objects that are stored *en masse* in the tree, but rather the elements from the mesh representation of these objects are the *objects* that are stored. From here on, elements are the objects that are stored in the trees.

Objects, whether they are solid parts or surfaces of a part, are tesselated with one of the basic element types. Various types of trees are constructed with these objects in the UM library. Separate containment and surface trees are needed for each part or instance of a part. Currently, a containment tree is needed for the global mesh model that is constructed from all parts, but this may soon be eliminated and replaced by the part trees. Elements in the parts and the global model are numbered consecutively from 1 to the maximum number in that part or global model.

The elements for the containment and intersection searches are slightly different. The containment search involves three-dimensional elements. The intersection search needs element faces for those faces on the exterior of a part and are referred to as surface faces. The faces may be flat (i.e., all nodes lie in one plane) or possess curvature (i.e., they are bilinear or quadratic). Either of these types (i.e., 3-D element or 2-D face) is generically termed an *element*. So, regardless of the type, the centroid of the element must be calculated and a MAABB must be defined for each before the tree is constructed.

If the surface face is flat and aligned parallel to one of the axes, the MAABB will be two-dimensional. In the same situation for a surface face with curvature, the MAABB will be nearly two-dimensional. As will be discussed later, neither of these situations cause problems for the intersection search. If the surface face is not parallel to one of the axes, the MAABB is three-dimensional.

In the tree construction, the key at each level or tree node is selected based on the maximum extents of the elements belonging to the bucket; the extents are determined using the elements' centroids. The key or direction chosen is referred to as the cut direction. It is along this direction that the elements in the current bucket are divided by a median point into low range and high range buckets with the elements in each bucket sorted from lowest to highest value along the chosen cut direction. After this, the process proceeds to the next tree level or sub-node.

At each level, a safety box is created. For the containment tree, the safety box is one-dimensional with the direction along the cut direction. This safety box represents the minimum and maximum extents of the MAABB's for all elements in the bucket. The maximum extent is associated with the LOBUC so that all elements in this bucket have a coordinate value in this cut direction that is less than the maximum. A similar statement exists for the minimum extent and the HIBUC. Note that this can produce overlapping buckets, but is a necessary condition when storing objects with extents; the search routines can adequately deal with this overlap condition. For the surface intersection tree, the safety box is three-dimensional and represents the minimum and maximum extents of the MAABB's for all surface faces in the bucket. The safety boxes are necessary for efficient search algorithms and are discussed when these algorithms are described later in this work.

The construction of the element and surface trees are nearly identical, except for the number of dimensions needed for the safety box at each node. The algorithm for constructing the *skd*-tree is summarized in Figure 2-1.

All trees are static in the UM library. That is, they are created during problem setup and are destroyed when the program terminates. Hence, there is no need for insertion or deletion operations to expand or contract trees during program execution.

for each element

- find the centroid
- construct the minimum axis-aligned bounding box, MAABB

end

do until the bucket size equals 1

for all elements in the bucket

- find the direction with the largest extents based upon the spread in the centroids; store this direction
- if a containment tree, find the min and max dimensions along this cut direction for the node safety box; if a surface intersection tree, find the min and max dimensions from all MAABB's along all directions for the safety box
- find the median point along the cut direction, creating low range and high range buckets
- sort the elements in each bucket solely along the cut direction
- repeat until the bucket size equals 1

when the bucket size equals 1

- set the flag indicating a leaf node
- safety boxes need not be set since they coincide with the axis-aligned bounding boxes for the element

Figure: 2 - 1. Algorithm outline for constructing an skd-tree

SKD-Tree Implementation

There exist several ways in which to implement binary trees and search algorithms for them. Two approaches were considered for this work: linked lists and link array. Initially, both were tried on simple problems, albeit not exhaustively. In the Monte Carlo transport, for which this implementation is intended, trees are constructed during problem initialization, are never modified during the calculation, and are destroyed at problem termination. Hence, the approach is to allocate memory once at the beginning of the problem, keep track of usable space in this allocation, and release the memory when done; hence, use an allocatable array. These methods have been used previously by others and are re-implemented with this unstructured mesh capability.

The heart of each *skd-tree* in this work is a link array, *linkp*, that has been allocated to the appropriate size during program initialization. The size of this array is twice the number of elements that it contains. Values stored in the array locations are either a positive integer, corresponding to the next left (non-leaf) node location in *linkp* or a negative integer representing a leaf node. The absolute value of the leaf node value is an element number for that tree. At each tree (non-leaf) node, branches occur in pairs. Positive array values in *linkp* correspond to the location of the next left branch node (LOBUC); the right branch node (HIBUC) is always assumed to be stored in the *linkp* array one beyond the left branch node location. A simple tree containing eight elements and its corresponding *linkp* array is illustrated in Figure 2-2. The cut direction and MAABB's at each tree node is stored in their respective arrays.

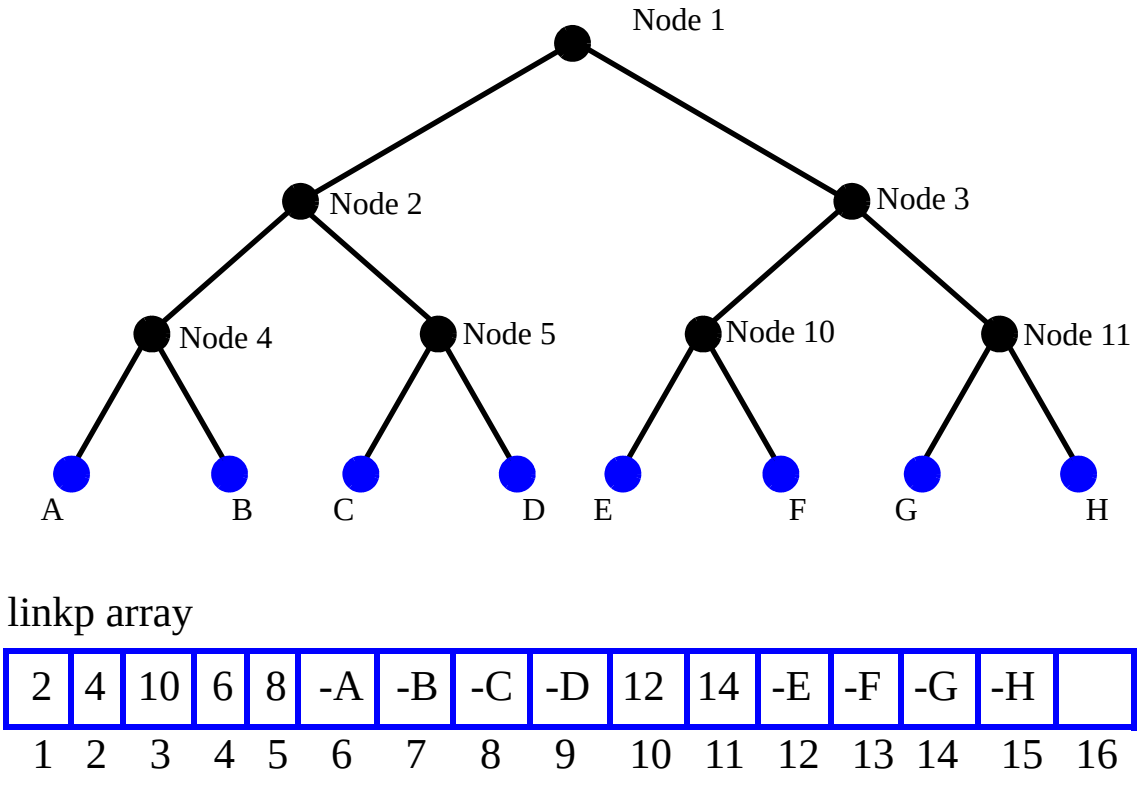

Figure: 2 - 2. Example binary tree with linkp implementation.

Tree Searches

The containment and intersection searches are both two part searches. The appropriate *skd*-tree is searched to find likely candidates that are added to a search list. The candidates are based on the MAABB's meeting a simple search criteria (e.g., is the point within a rectangular box) and is not the same as meeting the search criteria for the element itself. For the second part of the search, the candidates from the list are tested against more difficult search criteria (e.g., is the point within a tri-linear volume). The two search types are similar and both rely on recursive algorithms that are discussed in Figures 2-3 and 2-4. Essentials on the second part of the search, containment or intersection, are much more detailed and are relegated to their own chapters in this work.

for each point

do until a leaf node is reached or search is exhausted

- find the cut direction for the current node
- if the point position for the cut direction is <= the safety box value, call LOBUC node
- if the point position for the cut direction is >= the safety box value, call HIBUC node

when the leaf node is reached

- if the point lies within the element's MAABB, add the corresponding element number to the search list
- if the point does not lie within the element's MAABB, return

Figure: 2 - 3. Algorithm outline for containment search

for each particle track

do until a leaf node is reached or search is exhausted

- for the current node, determine the closest intersection point with the 3-D MAABB if the track intersection exists
- find the cut direction for the current node
  - -- if the intersection point for the cut direction is <= the safety box value, call LOBUC node
  - -- if the intersection point for the cut direction is >= the safety box value, call HIBUC node
- if no intersection with MAABB, return

when the leaf node is reached

- if intersection point lies on the surface of the element's MAABB, add corresponding element number to the search list
- if point does not lie on the surface of the element's MAABB, return

Figure: 2 - 4. Algorithm outline for intersection search with a surface tree

Chapter 3: Containment

One of the two main pillars for implementing UM tracking is containment -- a concept that is approachable in several ways. At times containment may mean, in what element does a 3-D location appear? In implementation terms this generally means searching at least a short candidate list and at most it could mean searching the entire list of elements comprising the global mesh model to find the appropriate element. At other times, containment may mean verification that the location is indeed inside a given element or residing on its surface or an edge.

Fundamental to implementing containment in this mesh library are the concepts of interpolation and mapping. These same concepts are also important to some of the intersection routines needed for the UM library. Therefore, since these topics are highly essential for what follows, an overview is provided in this chapter before discussing the details of containment. Much more depth on these subjects exist in finite element texts, but is basically overkill for what is required in the current work.

Transformation Basics

For simplicity sake, this discussion starts in the 2-D realm. The concepts presented here can easily be generalized to other dimensions.

From elementary calculus, a most familiar relationship is the one that exists between cylindrical or planar polar coordinates and Cartesian coordinates. Namely,

$$x = r\cos\theta \qquad y = r\sin\theta \qquad \text{(Eq: 3-1)}$$

This transformation is nothing more than a mapping from the (r,θ) space to the (x,y) space where every point in (r,θ) is represented by a one and only one point in (x,y). That is, the mapping is one to one. Figure 3-1 represents such a transformation where on the left is an element in an irregular domain (i.e., a domain with curved boundaries) and on the right is an element in a regular domain (i.e., a domain with straight boundaries). A non-rectangular region can't be represented as accurately using rectangular elements as it can by using non-rectangular ones. However, since the transformation functions from one system to the other are easily derivable and it is easier to perform mathematical evaluations and manipulations over rectangular geometries, make the necessary transformations to the rectangular system and perform the appropriate operations there. Finite element texts discuss making the transformations solely for the purpose of making it easier to numerically evaluate integrals. As will be seen in the discussion that follows in this chapter and others, the motivation for utilizing transformations in this work is more than evaluating integrals.

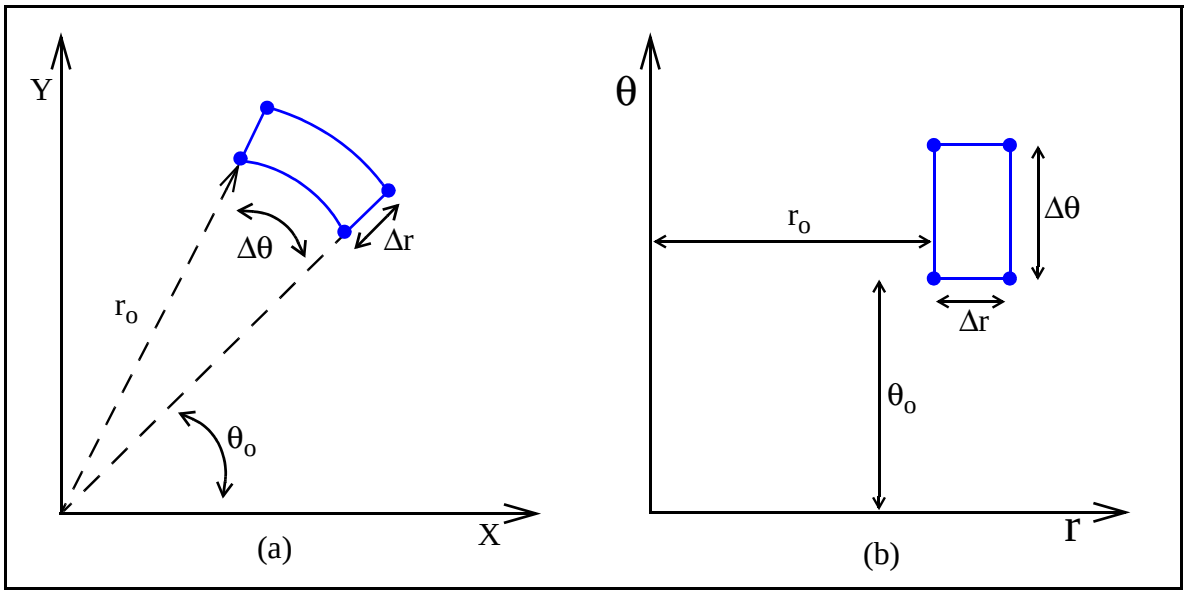

Figure: 3 - 1. Representative Cartesian coordinates (a) and cylindrical polar coordinates (b).

The functional relationship between these two coordinate systems can be generally expressed as

$$x = \phi_1(r, \theta) \qquad y = \phi_2(r, \theta) \tag{Eq: 3-2}$$

where ϕ_1 and ϕ_2 are the shape functions, (x,y) represents the global space, and (r,θ) represents the local or master space.

In the work that follows, it is necessary to establish shape function derivatives with respect to the (x,y) coordinates. As shown above, the rectangular element shape functions were defined in the master (r,θ) element coordinate system. However, by the chain rule of partial differentiation,

$$\begin{aligned} \frac{\partial \phi}{\partial r} &= \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial r} \\ \frac{\partial \phi}{\partial \theta} &= \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial \theta} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial \theta} \end{aligned} \tag{Eq: 3-3}$$

or, in matrix notation

$$\begin{bmatrix} \frac{\partial \phi}{\partial r} \\ \frac{\partial \phi}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial \theta} & \frac{\partial y}{\partial \theta} \end{bmatrix} \begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \end{bmatrix} = J \begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \end{bmatrix} \tag{Eq: 3-4}$$

where J is the Jacobian matrix of the transformation and is non-singular. A necessary and sufficient condition for the system of equations to be invertible is that the determinant |J| of the Jacobian matrix (also referred to as the Jacobian of the transformation) be nonzero for all (r,θ) of the master element.

The required derivatives $\partial\phi/\partial x$ and $\partial\phi/\partial y$ are obtained by inversion.

$$\begin{bmatrix} \frac{\partial\phi}{\partial x} \\ \frac{\partial\phi}{\partial y} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial\phi}{\partial r} \\ \frac{\partial\phi}{\partial\theta} \end{bmatrix} \qquad J = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial\theta} & \frac{\partial y}{\partial\theta} \end{bmatrix} \tag{Eq: 3-5}$$

For the transformation from polar to Cartesian coordinates,

$$J = \begin{bmatrix} \cos\theta & \sin\theta \\ -r\sin\theta & r\cos\theta \end{bmatrix} \tag{Eq: 3-6}$$

and

$$\begin{bmatrix} \frac{\partial\phi}{\partial x} \\ \frac{\partial\phi}{\partial y} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial\phi}{\partial r} \\ \frac{\partial\phi}{\partial\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & \frac{-\sin\theta}{r} \\ \sin\theta & \frac{\cos\theta}{r} \end{bmatrix} \begin{bmatrix} \frac{\partial\phi}{\partial r} \\ \frac{\partial\phi}{\partial\theta} \end{bmatrix} \tag{Eq: 3-7}$$

Finally for this example, the differential area, *dxdy*, in the (x,y) space can be replaced by an equivalent one in the (r,θ) space [14]

$$dxdy = |J|drd\theta \tag{Eq: 3-8}$$

yielding the well know expression

$$dxdy = rdrd\theta \tag{Eq: 3-9}$$

Note that $|J|$ may be regarded as a magnification factor that converts by multiplication the differential *dxdy* into the differential $drd\theta$. In the 2-D case, realize that it is the 2x2 Jacobian matrix J that plays, for two differentiable functions of two independent variables, the role played by the derivative of one differentiable function of one independent variable [14].

In the analysis of practical problems, complex geometries must be modeled for which the simple element shapes, at first glance, may seem impractical. This restriction is removed by *mapping* a simple element (i.e., rectangle in the previous example) in the master space coordinates into a more complex shape in the global coordinates. Mapping is understood to be a unique, one-to-one relationship between the master space coordinates and the global coordinates.

Once a particular form of mapping is adopted and the coordinates are chosen for every element so that these map into contiguous spaces, the shape functions written in the master element space can be used to represent the function variation over the element in the global space

without upsetting the inter-element continuity requirements [15]. The following section discusses the various types and mappings used in the UM library. These element types are both first and second order in nature. A variety of types is desired for a number of reasons. Obviously, as will be seen, the mappings for the higher order elements are more complex and costly to implement from an algorithmic point of view, but can be highly beneficial from an accuracy standpoint when modelling highly complex (i.e., curvature) objects. The higher degree of accuracy obtained with the higher order elements often means that a smaller number of such elements are needed to obtain an adequate solution.

Master Elements

The FEM uses elements of different shapes and order; each possesses a regularly shaped counterpart, called a master element, for which interpolation functions exist, These interpolation functions are used both to approximate the geometry and for numerical evaluation of integrals defined on the elements. Evaluation of these integrals requires transforming the geometry of the actual, irregularly-shaped element to the appropriate master element.

Finite element texts [5, 15-17] generally begin the discussion of elements with the Lagrange family of elements, which possess internal nodes for the higher order elements. These internal nodes do not contribute to the inter-element connectivity. Alternatively, serendipity elements are used instead since they have no internal nodes; all of the nodes are on the surface of the element and contribute to the inter-element connectivity. Serendipity elements are shown in Figure 1-1.

The master space coordinate system is a natural coordinate system which permits specification of a point within the element by a set of dimensionless numbers whose magnitudes never exceed unity. These systems are generally arranged so that some of the natural coordinates have unit magnitude at some nodes. Not only does such a coordinate system generalize and simplify the formulation, but it also simplifies any integration.

A mapping function, *u*, can be found that transforms from the master space to the global space.

$$u^d = \sum_{n=1}^{N} \phi_n(\bar{\theta}) u_n^d \qquad \text{(Eq: 3-10)}$$

where

u^d	coordinates in global space (d = 1, 2, or 3 representing x, y, or z)
$\bar{\theta}$	coordinates in master space (= $\theta(g, h, r)$)
u_n^d	the *d 'th* coordinate for node *n*
ϕ_n	interpolation (shape) function for the *n'th* node
N	total number of element nodes

and ϕ has the following properties

$$\phi_i(\bar{\theta}_j) = \delta_{ij} \qquad \text{(Eq: 3-11)}$$

$$\sum_{n=1}^{N} \phi_i = 1 \qquad \text{(Eq: 3-12)}$$

Equation (3-11) requires the shape functions to have a magnitude of unity at one node and zero values at all other nodes. From Equation (3-10), any point belonging in global space is a function of any one element's nodes in global space coordinates and the interpolation function (i.e., its position in master space).

Derivations of the interpolation functions for the serendipity elements are generally found in most finite element texts [5, 15-17]. They are presented as a matter of completeness at the end of this chapter, with the notation used here, for all elements of interest in this work.

As a means to illustrate the preceding discussion, consider the simple case of the 2-D rectangular master element shown in Figure 3-2 and the mapping to an arbitrary 2-D rectangular element.

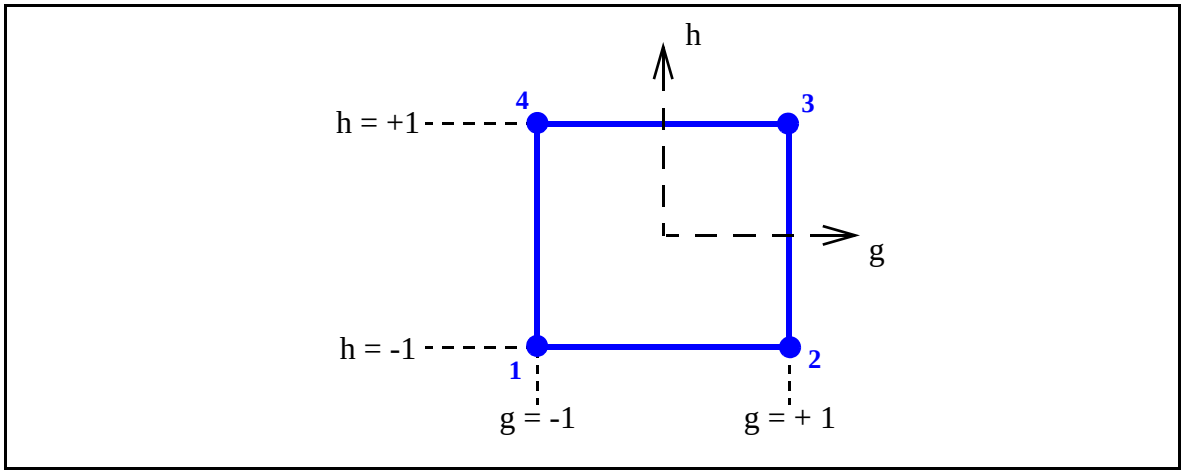

Figure: 3 - 2. Serendipity master element and coordinate system for the 2-D rectangular element.

Note that the origin of the master space coordinate system is at the center of the element and the extents of the sides are from -1 to +1 for a length of 2. The interpolation functions are

$$\phi_1 = (1-g)(1-h) \qquad \phi_2 = (1+g)(1-h)$$
$$\phi_3 = (1+g)(1+h) \qquad \phi_4 = (1-g)(1+h) \qquad \text{(Eq: 3-13)}$$

The mapping function is

$$u^d = \frac{1}{4}[(1-g)(1-h)u_1^d + (1+g)(1-h)u_2^d + (1+g)(1+h)u_3^d + (1-g)(1+h)u_4^d] \qquad \text{(Eq: 3-14)}$$

Extending rectangular elements from 2-D to 3-D is straightforward as can be seen when comparing Figures 3-2 and 3-3.

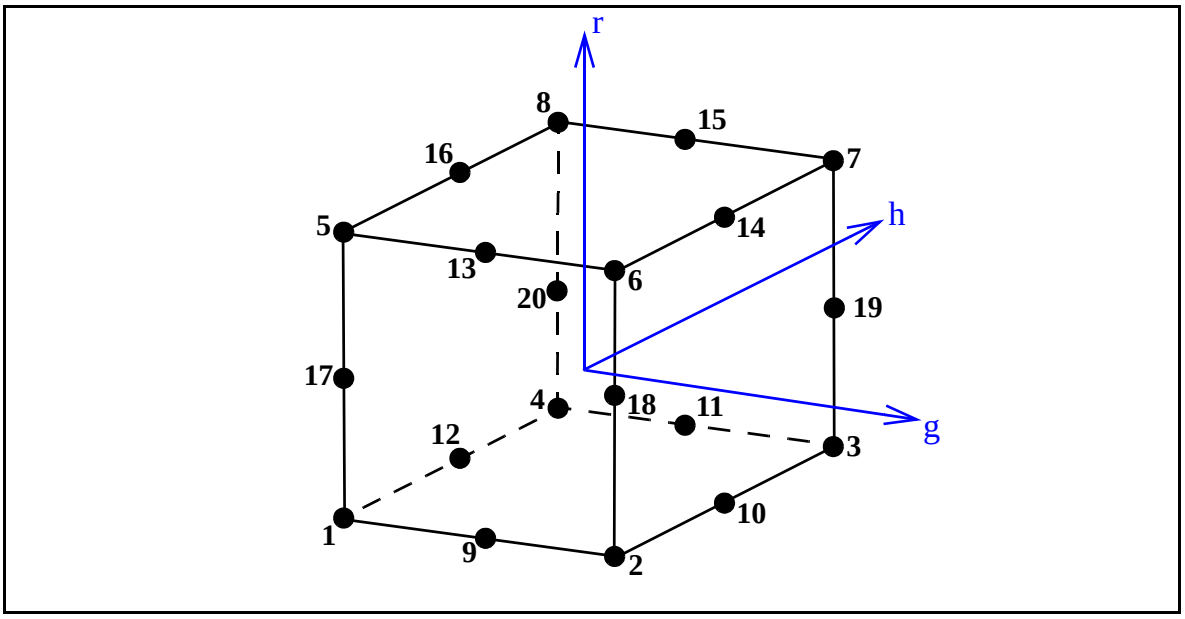

Figure: 3 - 3. Serendipity master element and coordinate system for hexahedra.

The master space axes are orthogonal and labeled *g*, *h*, and *r*. The origin of the coordinate system is still at the element's center. The extents of all sides are from -1 to +1 for a length of 2. For a point to be inside or on the surface of the master element, the following relationship among the master space coordinates must hold.

$$-1 \le g, h, r \le +1 \qquad \text{(Eq: 3-15)}$$

The master space system for tetrahedra is somewhat altered from that for the hexahedra discussed above.

The master space axes are still orthogonal with labels *g*, *h*, and *r*, but the origin of the coordinate system coincides with node 1 and the length of an edge parallel to a coordinate axis extends from 0 to +1, see Figure 3-4(a). Therefore, the master space coordinates for the vertices are

$$\theta_1(g, h, r) = 0, 0, 0 \qquad \theta_2(g, h, r) = 1, 0, 0$$
$$\theta_3(g, h, r) = 0, 1, 0 \qquad \theta_4(g, h, r) = 0, 0, 1 \qquad \text{(Eq: 3-16)}$$

The edge nodes are assumed to be located half-way between the vertices and their master space coordinates are obvious. For a point to be inside or on the surface of this master element, the following relationships among the master space coordinates must hold.

$$0 \le g, h, r \le +1$$
$$0 \le g + h + r \le +1 \qquad \text{(Eq: 3-17)}$$

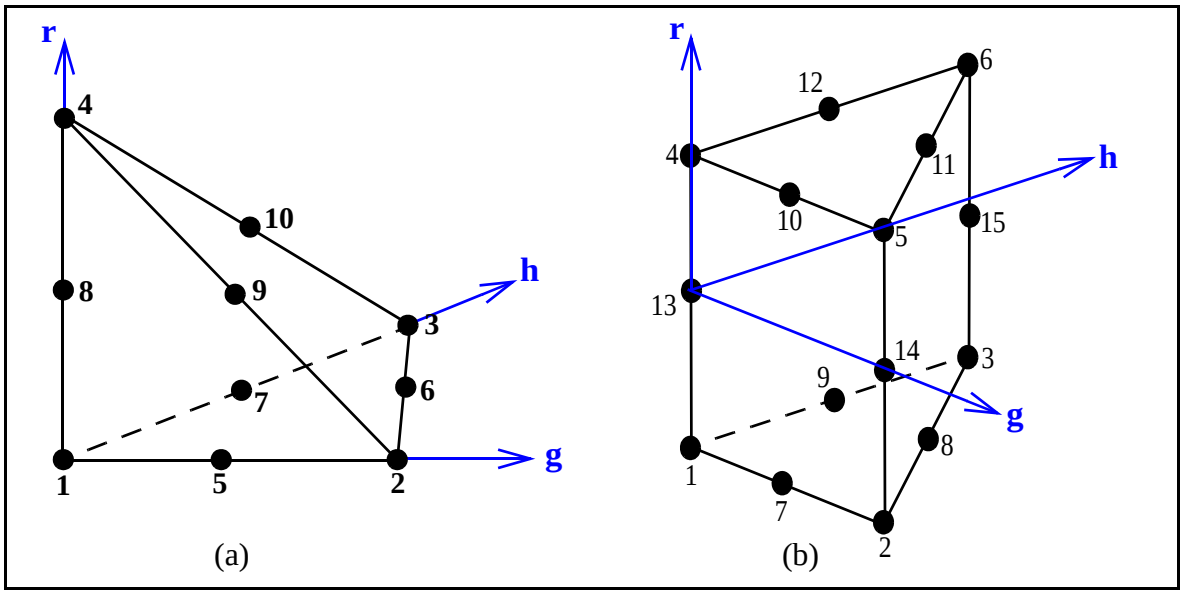

Figure: 3 - 4. Serendipity master element and coordinates systems for (a) tetrahedra and (b) pentahedra.

The master space system for pentahedra, Figure 3-4(b) is similar to that of the tetrahedra. Once again the axes are *g*, *h*, and *r*, but the origin of the coordinate system coincides to a point midway between nodes 1 and 4 (i.e., at node 13 if it is a second order element).

Edges parallel to the *g* and *h* axes extend from 0 to +1, but edges parallel to the *r* axis extend from -1 to +1. Therefore, the master space coordinates for the vertices are

$$\begin{aligned} \theta_1(g, h, r) &= 0, 0, -1 & \theta_4(g, h, r) &= 0, 0, 1 \\ \theta_2(g, h, r) &= 1, 0, -1 & \theta_5(g, h, r) &= 1, 0, 1 \\ \theta_3(g, h, r) &= 0, 1, -1 & \theta_6(g, h, r) &= 0, 1, 1 \end{aligned} \qquad \text{(Eq: 3-18)}$$

For a point to be inside or on the surface of this master element, the following relationships among the master space coordinates must hold.

$$\begin{aligned} 0 \le g, h \le +1 \\ -1 \le r \le +1 \\ 0 \le g + h \le +1 \end{aligned} \qquad \text{(Eq: 3-19)}$$

Polynomial Form

It should be noted that all of the mapping functions presented here are polynomial in form. For example, consider the 2-D rectangle presented earlier. Equation 3-14 can be rewritten as

$$u^d = \alpha_1 + \alpha_2 g + \alpha_3 h + \alpha_4 gh \qquad \text{(Eq: 3-20)}$$

where

$$\alpha_1 = \frac{1}{4}(u_1^d + u_2^d + u_3^d + u_4^d)$$

$$\alpha_2 = \frac{1}{4}(u_2^d + u_3^d - u_1^d - u_4^d)$$

$$\alpha_3 = \frac{1}{4}(u_3^d + u_4^d - u_1^d - u_2^d)$$

$$\alpha_4 = \frac{1}{4}(u_1^d + u_3^d - u_2^d - u_4^d)$$

Note that only terms with linear variation appear. The last term is bilinear since it is linear in both *g* and *h*. Therefore, it is possible for the four nodes in global space to not all appear in the same plane.

The other mapping functions are of the form

$$u^d = \alpha_1 + \alpha_2 g + \alpha_3 h + \alpha_4 r + \alpha_5 gh + \alpha_6 gr + \alpha_7 hr + \alpha_8 g^2 + \alpha_9 g^2 h + \ldots \qquad \text{(Eq: 3-21)}$$

Generally, for serendipity elements, this leads to one polynomial / coefficient for each node.

Other simple functions, such as the trigonometric functions, could have been chosen [10]. However, polynomials offer ease in mathematical manipulation and have been commonly employed in FEM's. First, with polynomials, it is easy to handle the mathematics in formulating the desired equations for various elements and in performing digital computations. In particular, polynomials can be differentiated and integrated with ease. Second, a polynomial of arbitrary order permits a recognizable approximation to the true solution. A polynomial of infinite order corresponds to an exact solution. However, practical purposes limit use to finite order.

Element Classification

For FEM's it is possible to employ different shape functions to approximate the geometry than those used to approximate the function of interest (i.e., dependent variable or state function) on that geometry. Therefore, the finite element formulations are classified into three categories:

- *Superparametric* -- the approximation used for the geometry is higher order than that used for the state function.
- *Isoparametric* -- equal degree of approximation is used for both.
- *Subparametric* -- the approximation used for the geometry is lower order than that used for the state function.

The functions needed in this work use the same shape functions that approximate the geometry since these functions are geometric in nature.

Element Containment

The chief problem of interest for this chapter is determining the state of a given point relative to a given element. Is the point inside that element, outside that element, or on its surface? If it is on the surface, on which of its faces does it reside may be important. Or perhaps, knowing that it is on an edge or corner may be important. Any of these questions and concerns are addressable by knowing the master space coordinates (*g*, *h*, *r*) and which of the master element conditions, equations 3-15, 3-17, or 3-19, to apply.

All the variables of interest are contained in equation 3-10, which gives rise to a system of three equations in three unknowns when considering a 3-D problem.

$$\begin{aligned} x &= \sum_{n=1}^{N} \phi_n(g, h, r) x_n \\ y &= \sum_{n=1}^{N} \phi_n(g, h, r) y_n \\ z &= \sum_{n=1}^{N} \phi_n(g, h, r) z_n \end{aligned} \qquad \text{(Eq: 3-22)}$$

The unknown *g*, *h*, and *r* can be easily found for a given *x*, *y*, and *z* by applying Newton's method to minimize the error terms, E_d , in the following system of equations which readily follow from Equation 3-22.

$$\begin{aligned} E_x &= \sum_{n=1}^{N} \phi_n(g, h, r) x_n - x \\ E_y &= \sum_{n=1}^{N} \phi_n(g, h, r) y_n - y \\ E_z &= \sum_{n=1}^{N} \phi_n(g, h, r) z_n - z \end{aligned} \qquad \text{(Eq: 3-23)}$$

Successive iterates of the error functions can be found from the previous system of equations by using a Taylor series expansion where the second order terms and higher are ignored. Thus,

$$
\begin{aligned}
E_x^{i+1} &= E_x^i + \frac{\partial E_x^i}{\partial g}\Delta g + \frac{\partial E_x^i}{\partial h}\Delta h + \frac{\partial E_x^i}{\partial r}\Delta r \\
E_y^{i+1} &= E_y^i + \frac{\partial E_y^i}{\partial g}\Delta g + \frac{\partial E_y^i}{\partial h}\Delta h + \frac{\partial E_y^i}{\partial r}\Delta r \\
E_z^{i+1} &= E_z^i + \frac{\partial E_z^i}{\partial g}\Delta g + \frac{\partial E_z^i}{\partial h}\Delta h + \frac{\partial E_z^i}{\partial r}\Delta r
\end{aligned}
\qquad \text{(Eq: 3-24)}
$$

Setting E_d^{i+1} to zero and re-writing in matrix form yields

$$
\begin{bmatrix}
\frac{\partial E_x}{\partial g} & \frac{\partial E_x}{\partial h} & \frac{\partial E_x}{\partial r} \\
\frac{\partial E_y}{\partial g} & \frac{\partial E_y}{\partial h} & \frac{\partial E_y}{\partial r} \\
\frac{\partial E_z}{\partial g} & \frac{\partial E_z}{\partial h} & \frac{\partial E_z}{\partial r}
\end{bmatrix}
\begin{bmatrix} \Delta g \\ \Delta h \\ \Delta r \end{bmatrix}
= -\begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix}
\qquad \text{(Eq: 3-25)}
$$

When Δg, Δh, and Δr are small such that each is within an epsilon (e.g., 10^{-10}) of zero, the resulting values of g, h, and r can be used with equations 3-15, 3-17, or 3-19 to make the appropriate containment determination.

It should be fairly obvious how Equations 3-15, 3-17, and 3-19 can be used to determine if a point lies on a particular face (see Table 3-1 for face definitions) or an edge. Consider a hexahedron for example. For a point to lie on the edge between nodes 1 and 5, the following must hold.

$$g = -1 \; ; \qquad h = -1 \; ; \qquad -1 \le r \le +1 \qquad \text{(Eq: 3-26)}$$

Table 3-1: Face Definitions By Vertex Number

Face Number	Tetrahedra	Pentahedra	Hexahedra
1	1 - 2 - 3	1 - 2 - 3	1 - 2 - 3 - 4
2	1 - 2 - 4	4 - 5 - 6	5 - 6 - 7 - 8
3	2 - 3 - 4	1 - 2 - 5 - 4	1 - 2 - 6 - 5
4	1 - 3 - 4	2 - 3 - 5 - 6	2 - 3 - 7 - 6
5	-	1 - 3 - 6 - 4	3 - 4 - 8 - 7
6	-	-	1 - 5 - 8 - 4

Element Quality

The mappings required by this work are acceptable if and only if they are one-to-one. That is, every point in the actual element is mapped uniquely into a point in the master element, and vice versa. For this requirement to be met, $|J|$ must be greater than zero everywhere in the element. Geometrically, J represents the ratio of an element's volume in the global element space to the corresponding volume in the master element space. If $J = 0$, then a nonzero volume element in the global element space is mapped into a zero volume in the master element -- a totally unacceptable arrangement. If $J < 0$, then a right-handed coordinate system is mapped into a left-handed coordinate system.

In general, the mapping transforms a distorted element into a regular one through a non-uniform mapping. Excessive distortion of elements should be avoided. The general guidance is that an element's interior angles should never be too small (i.e., approaching 0^o) or too large (i.e., approaching 180^o). A typical acceptable angle range is 15^o - 165^o [5].

Another measure of element distortion is its aspect ratio, defined as the ratio of the largest edge to the smallest. In finite element displacement calculations, for example, where the displacement varies at the same rate in each direction, the closer the aspect ratio is to unity, the better is the quality (i.e., error) of the solution [10]. Thus, long narrow elements are to be avoided.

Many CAE tools, like Abaqus/CAE, possess mesh quality checking features that are extensive and specific to each element type [11]. For example, a shape factor is available only for tetrahedra with an acceptable range from 0 to +1, with +1 representing the optimal element shape and 0 indicating a degenerate element.

$$\text{shape factor} = \frac{\text{element volume}}{\text{optimal element volume}} \qquad \text{(Eq: 3-27)}$$

Optimal volume is the volume of an equilateral tetrahedron with the same circumradius as the element. (The circumradius is the radius of the sphere passing through the four vertices of the tetrahedron.) While the set of mesh quality checks is extensive, some are specific to ensuring that the finite element calculations obtain a correct result. Experience to date has shown, that if an unstructured mesh passes all of the quality tests provided by Abaqus/CAE, then the UM tracking methods in MCNP function properly.

For particle tracking purposes in a Monte Carlo code, the mesh quality does not have the same implications, but a positive J must be maintained for the methods to work correctly. That is, a Monte Carlo particle tracking code may deal with highly distorted elements correctly, provided $|J|$ is positive, as compared to its FEM counterpart. The more important question for the Monte Carlo calculation to address is whether the elements are of the right shape and size in order to correctly represent the geometry and/or *visualize* the gradients through the region; it will calculate the correct answer in the element. If the answer is yes, then any element should do as long as $|J|$ is positive. If the answer is no, then the user needs to take steps to ensure that there are sufficient elements of the right size and, of course, possess $|J| > 0$.

Mapping Functions

The mapping function for first order tetrahedra is

$$u^d = (1-g-h-r)u_1^d + gu_2^d + hu_3^d + ru_4^d \qquad \text{(Eq: 3-28)}$$

The mapping function for first order pentahedra is

$$\begin{aligned} u^d = {} & \frac{1}{2}[(1-g-h)(1-r)u_1^d + g(1-r)u_2^d + h(1-r)u_3^d] \\ & + \frac{1}{2}[(1-g-h)(1+r)u_4^d + g(1+r)u_5^d + h(1+r)u_6^d] \end{aligned} \qquad \text{(Eq: 3-29)}$$

The mapping function for first order hexahedra is

$$\begin{aligned} u^d = {} & \frac{1}{8}[(1-g)(1-h)(1-r)u_1^d + (1+g)(1-h)(1-r)u_2^d] \\ & + \frac{1}{8}[(1+g)(1+h)(1-r)u_3^d + (1-g)(1+h)(1-r)u_4^d] \\ & + \frac{1}{8}[(1-g)(1-h)(1+r)u_5^d + (1+g)(1-h)(1+r)u_6^d] \\ & + \frac{1}{8}[(1+g)(1+h)(1+r)u_7^d + (1-g)(1+h)(1+r)u_8^d] \end{aligned} \qquad \text{(Eq: 3-30)}$$

The mapping function for second order tetrahedra is

$$\begin{aligned} u^d = {} & [2(1-g-h-r)-1](1-g-h-r)u_1^d + g(2g-1)u_2^d + h(2h-1)u_3^d \\ & + r(2r-1)u_4^d + 4g(1-g-h-r)u_5^d + 4ghu_6^d + 4h(1-g-h-r)u_7^d \\ & + 4r(1-g-h-r)u_8^d + 4gru_9^d + 4hru_{10}^d \end{aligned} \qquad \text{(Eq: 3-31)}$$

The mapping function for second order pentahedra is

$$\begin{aligned} u^d = {} & \frac{1}{2}\{(1-g-h)[2(1-g-h)-1](1-r) - (1-g-h)(1-r^2)\}u_1^d \\ & + \frac{1}{2}[g(2g-1)(1-r) - g(1-r^2)]u_2^d + \frac{1}{2}[h(2h-1)(1-r) - h(1-r^2)]u_3^d \\ & + \frac{1}{2}\{(1-g-h)[2(1-g-h)-1](1+r) - (1-g-h)(1-r^2)\}u_4^d \\ & + \frac{1}{2}[g(2g-1)(1+r) - g(1-r^2)]u_5^d + \frac{1}{2}[h(2h-1)(1+r) - h(1-r^2)]u_6^d \\ & + 2g(1-g-h)(1-r)u_7^d + 2gh(1-r)u_8^d + 2h(1-g-h)(1-r)u_9^d \\ & + 2g(1-g-h)(1+r)u_{10}^d + 2gh(1+r)u_{11}^d + 2h(1-g-h)(1+r)u_{12}^d \\ & + (1-g-h)(1-r^2)u_{13}^d + g(1-r^2)u_{14}^d + h(1-r^2)u_{15}^d \end{aligned} \qquad \text{(Eq: 3-32)}$$

The mapping function for second order hexahedra is

$$
\begin{aligned}
u^d = {} & \frac{1}{4}[(1-g)(1+g)(1-h)(1-r)u_9^d] + \frac{1}{4}[(1-h)(1+h)(1+g)(1-r)u_{10}^d] \\
& + \frac{1}{4}[(1-g)(1+g)(1+h)(1-r)u_{11}^d] + \frac{1}{4}[(1-h)(1+h)(1-g)(1-r)u_{12}^d] \\
& + \frac{1}{4}[(1-g)(1+g)(1-h)(1+r)u_{13}^d] + \frac{1}{4}[(1-h)(1+h)(1+g)(1+r)u_{14}^d] \\
& + \frac{1}{4}[(1-g)(1+g)(1+h)(1+r)u_{15}^d] + \frac{1}{4}[(1-h)(1+h)(1-g)(1+r)u_{16}^d] \\
& + \frac{1}{4}[(1-r)(1+r)(1-g)(1-h)u_{17}^d] + \frac{1}{4}[(1-r)(1+r)(1+g)(1-h)u_{18}^d] \\
& + \frac{1}{4}[(1-r)(1+r)(1+g)(1+h)u_{19}^d] + \frac{1}{4}[(1-r)(1+r)(1-g)(1+h)u_{20}^d] \\
& -\frac{1}{8}(1-g)(1-h)(1-r)(2+g+h+r)u_1^d - \frac{1}{8}(1+g)(1-h)(1-r)(2-g+h+r)u_2^d \\
& -\frac{1}{8}(1+g)(1+h)(1-r)(2-g-h+r)u_3^d - \frac{1}{8}(1-g)(1+h)(1-r)(2+g-h+r)u_4^d \\
& -\frac{1}{8}(1-g)(1-h)(1+r)(2+g+h-r)u_5^d - \frac{1}{8}(1+g)(1-h)(1+r)(2-g+h-r)u_6^d \\
& -\frac{1}{8}(1+g)(1+h)(1+r)(2-g-h-r)u_7^d - \frac{1}{8}(1-g)(1+h)(1+r)(2+g-h-r)u_8^d
\end{aligned}
\qquad \text{(Eq: 3-33)}
$$

Chapter 4: Intersection: Part I

Intersection of a particle ray with a surface is one of the key components of tracking on an unstructured mesh. Unless the mesh geometry is restricted to first order tetrahedra, the other supported polyhedra in MCNP will generally have bilinear or biquadratic surfaces and require special routines for finding intersection points. As will be seen in the next chapters, these special intersection routines for the curved surfaces possess some nice properties that are beneficial when applied to tracking on first order tetrahedra as well. However, since skd-trees with their axis-aligned bounding boxes are used for various reasons, the tried-and-true (TAT) methods for finding the intersection of a ray with a plane are needed when searching these trees for an intersection. Since only the intersection point with the plane is needed and no determination must be made as to whether the point lies on or within a face, the TAT method is used and this chapter provides a review as it relates to the mesh library implementation.

Plane and Line Equations

A Cartesian form of the equation of a plane, sometimes called the scalar equation of the plane [18], is given by

$$Ax + By + Cz = D \qquad \text{(Eq: 4-1)}$$

Alternatively, this equation can be written in the form of a dot product

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = D \qquad \text{(Eq: 4-2)}$$

The vector $\begin{bmatrix} A \\ B \\ C \end{bmatrix}$ is the plane's normal vector, $\bar{N}$. If $\bar{P}$ is any point in the plane, Equation 4-2 can be written

$$\bar{N} \cdot \bar{P} = D \qquad \text{(Eq: 4-3)}$$

The value $|D|/\|\bar{N}\|$ is the distance by which the plane is offset from a parallel plane passing through the origin. When the plane coefficients *A*, *B*, and *C* are normalize by $\|\bar{N}\|$, they are the direction cosines and the equation of the plane appears in what is referred to as the normal form [18].

A ray is a line possessing a single endpoint $\bar{s}$, a particle's source or collision location, and extending to infinity in a given direction $\bar{v}$. A ray can be expressed by a vector or parametric equation [18]

$$\bar{P} = \bar{P}(t) = \bar{S} + t\bar{V} \qquad \text{(Eq: 4-4)}$$

where *t* is the parameter and is allowed to be greater than or equal to zero.

Intersection

By substituting Equation 4-4 into Equation 4-3 for $\bar{p}$ and re-arranging, an expression for *t* can be obtained.

$$t = \frac{D - (\bar{N} \cdot \bar{S})}{\bar{N} \cdot \bar{V}} \qquad \text{(Eq: 4-5)}$$

If the denominator of Equation 4-5 is equal to zero, the ray is parallel to the plane and there is no intersection of the ray with the plane. If the numerator of Equation 4-5 is zero, the ray either lies entirely in the plane or only $\bar{p}$ lies in the plane and the ray points away from the plane with no intersection when $t > 0$.

The intersection point can then be found by substituting the value for *t* obtained from Equation 4-5 back into Equation 4-3. Obviously, when finding a ray intersection with an axis-aligned bounding box from the skd-trees, the plane normal vectors are either $\begin{bmatrix}1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\end{bmatrix}$, or $\begin{bmatrix}0\\0\\1\end{bmatrix}$ depending on whether the plane is aligned with the x-, y-, or z-axis, respectively. Therefore, the vector equation for intersection, Equation 4-5, reduces to a scalar one and the value of *t* can be substituted into Equation 4-4 to find the appropriated x-, y-, or z-location.

Chapter 5: Intersection: Part II

The TAT method that was discussed in the previous chapter can not be used to find the intersection point of a particle ray and a bilinear surface. The approach implemented in the UM library for the intersection of a particle ray and a bilinear surface was originally documented in Reference [19], but is reproduced here for completeness and because it appears that an electronic version is not readily available; the current authors have only an earlier draft of Reference [19] at their disposal that was published as a LANL internal report. So, with apologies to the authors of Reference [19], the appropriate work on intersections with bilinear surfaces is recreated here in a form that is consistent and appropriate for the current work.

The first part of this chapter discusses the methodology from Reference [19] as it relates to an intersection with a surface defined by four nodes. Because of the nice properties inherent in this method, the later part of this chapter discusses how the method is used with a surface defined by three nodes as are prevalent in tetrahedra and pentahedra.

A Four-Node Surface

A bilinear surface arises when the four nodes of an element face are non-coplanar.

$$u_1 = (x_1, y_1, z_1) \qquad u_2 = (x_2, y_2, z_2)$$
$$u_3 = (x_3, y_3, z_3) \qquad u_4 = (x_4, y_4, z_4) \qquad \text{(Eq: 5-1)}$$

The four nodes represent a distorted rectangle in 3-D global space that, for the purpose of this method, are mapped to a square in master space, similar to the 2-D example in Chapter 3 (see Figure 5-1).

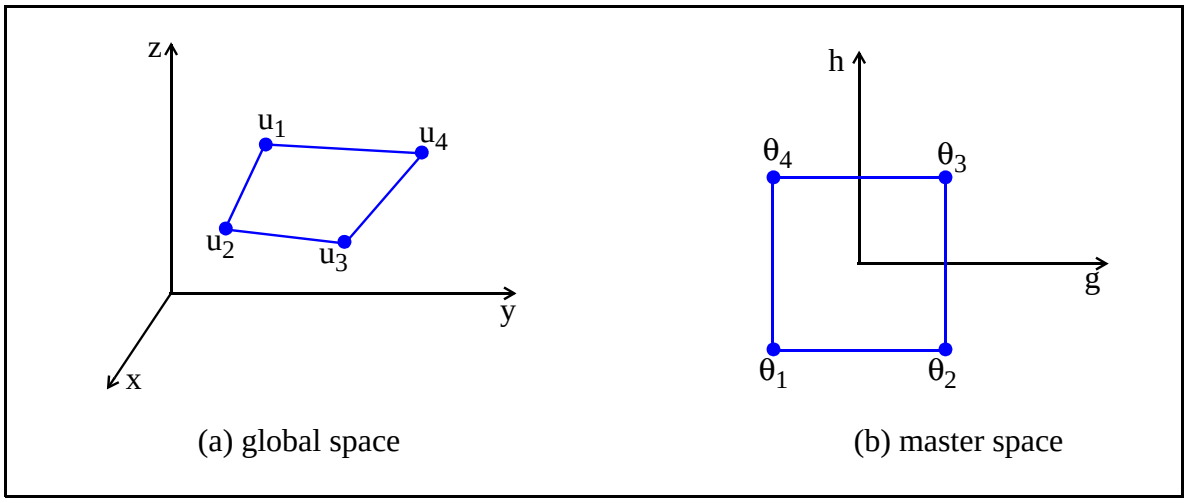

Figure: 5 - 1. Two-dimensional first order rectangular elements.

As discussed in Chapter 3, the origin of the master space coordinate system is at the center of the master element and the extents of the master space element sides are from -1 to +1. If a ray intersects this 2-D element, the values of *g* and *h* must both be between -1 and +1 or equal to -1 or +1. Any values of *g* or *h* outside of this range imply a miss (i.e., no intersection with the face).

Since the master element contains nodes on the edges and none in the interior, this is a serendipity element and the correct mapping function between the global space and the master space is given by Equation 3-14 and reproduced as equation 5-2:

$$u^d = \frac{1}{4}[(1-g)(1-h)u_1^d + (1+g)(1-h)u_2^d + (1+g)(1+h)u_3^d + (1-g)(1+h)u_4^d] \quad \text{(Eq: 5-2)}$$

where $u^d(g,h)$ is a coordinate or point in the global space expressed as a function of master space coordinates *g* and *h*.

When all three global space dimensions are included and like terms are collected, Equation 5-2 is a system of three equations that can be represented as a matrix equation in *g*, *h*, and *gh* for the surface.

$$\bar{F}(g,h) = \frac{1}{4}[\bar{A} + \bar{B}g + \bar{C}h + \bar{E}gh] \quad \text{(Eq: 5-3)}$$

where A, B, C, and E are 3x1 matrices defined as follows:

$$\bar{A} = \begin{bmatrix} x_1 + x_2 + x_3 + x_4 \\ y_1 + y_2 + y_3 + y_4 \\ z_1 + z_2 + z_3 + z_4 \end{bmatrix} \qquad \bar{B} = \begin{bmatrix} -x_1 + x_2 + x_3 - x_4 \\ -y_1 + y_2 + y_3 - y_4 \\ -z_1 + z_2 + z_3 - z_4 \end{bmatrix}$$

$$\bar{C} = \begin{bmatrix} -x_1 - x_2 + x_3 + x_4 \\ -y_1 - y_2 + y_3 + y_4 \\ -z_1 - z_2 + z_3 + z_4 \end{bmatrix} \qquad \bar{E} = \begin{bmatrix} x_1 - x_2 + x_3 - x_4 \\ y_1 - y_2 + y_3 - y_4 \\ z_1 - z_2 + z_3 - z_4 \end{bmatrix} \quad \text{(Eq: 5-4)}$$

Any point in the master space face can be expressed by this equation and $F(g,h)$ is synonymous with $u^d(g,h)$.

Intersection Equations

Consider a particle at position $\bar{R}$ traveling in direction $\bar{\Omega}$ where $\bar{R}$ and $\bar{\Omega}$ are 3x1 matrices defined as follows:

$$\bar{R} = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} \qquad \bar{\Omega} = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{Eq: 5-5}$$

The equation for the particle at a distance *d* along its trajectory is

$$\bar{P}(d) = \bar{R} + \bar{\Omega}d \tag{Eq: 5-6}$$

By setting the respective equations equal, the point where the particle's trajectory intersects the bilinear surface is found.

$$\bar{P}(d) = \bar{F}(g, h) \tag{Eq: 5-7}$$

$$\bar{R} + \bar{\Omega}d = \frac{1}{4}[\bar{A} + \bar{B}g + \bar{C}h + \bar{E}gh] \tag{Eq: 5-8}$$

$$0 = (\bar{A} - 4\bar{R}) + \bar{B}g + \bar{C}h + \bar{E}gh - 4\bar{\Omega}d \tag{Eq: 5-9}$$

$$0 = (\bar{A} - 4\bar{R}) + \bar{B}g + \bar{C}h + \bar{E}gh + \bar{D}d \tag{Eq: 5-10}$$

where

$$\bar{D} = -4\bar{\Omega} = -4\begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{Eq: 5-11}$$

Thus, a matrix of three equations in three unknowns *(g,h,d)* results. Unfortunately, the nonlinear *gh* term means that this matrix equation can't be solved with a simple 3x3 matrix inversion.

In global space, this problem is one of a straight line intersecting a curved surface. The transformation to master space makes this a problem of a curved line intersecting a flat plane. Consequently, the solution of the system of equations can produce two roots. Using a Newton iterative method on these equations could require extra work to check that the correct intersection was found [19]. An analytic solution is preferred and the reminder of the next section shows how this is achieved.

The Analytic Solution

Equation 5-10 can be written as an equation in linear terms.

$$\overline{M}\begin{bmatrix} g \\ h \\ d \end{bmatrix} = -\overline{E}gh + (4\overline{R} - \overline{A}) \qquad \text{(Eq: 5-12)}$$

where

$$\overline{M} = [[\overline{B}][\overline{C}][\overline{D}]] = \begin{bmatrix} B_1 & C_1 & D_1 \\ B_2 & C_2 & D_2 \\ B_3 & C_3 & D_3 \end{bmatrix} \qquad \text{(Eq: 5-13)}$$

Solving for the linear terms of the unknowns gives

$$\begin{bmatrix} g \\ h \\ d \end{bmatrix} = -\overline{M}^{-1}\overline{E}(gh) + \overline{M}^{-1}(4\overline{R} - \overline{A}) \qquad \text{(Eq: 5-14)}$$

where each linear term is a function of the term (*gh*) and a constant.

With the following definitions

$$Q_i = -\sum_{j=1}^{3} M_{ij}^{-1} E_j \qquad \text{(Eq: 5-15)}$$

$$S_i = \sum_{j=1}^{3} M_{ij}^{-1}(4R_j - A_j) \qquad \text{(Eq: 5-16)}$$

g and *h* are given by

$$g = Q_1(gh) + S_1 \qquad h = Q_2(gh) + S_2 \qquad \text{(Eq: 5-17)}$$

Multiplying these two equations together yields an equation that is quadratic in *gh*.

$$(gh) = Q_1Q_2(gh)^2 + (Q_1S_2 + Q_2S_1)(gh) + S_1S_2 \qquad \text{(Eq: 5-18)}$$

$$0 = Q_1Q_2(gh)^2 + (Q_1S_2 + Q_2S_1 - 1)(gh) + S_1S_2 \qquad \text{(Eq: 5-19)}$$

From this quadratic equation, there are two solutions for (*gh*) that can be used in Equation 5-14 to find the values of *(g, h, d)*. The final desired intersection solution must have real val-

ues of $(gh) \in [-1, 1]$. If both values of (*gh*) meet the criteria, then the final solution for this particular surface is the one with the smallest, positive value of *d* given by

$$d = Q_3(gh) + S_3 \qquad \text{(Eq: 5-20)}$$

If only one value of (*gh*) meets the criteria, the choice is obvious provided the value of *d* is positive. If both values fail the criteria, there is no intersection.

A Three-Node Surface

An analytic intersection approach similar to that used for four-noded bilinear surfaces can be implemented for faces with three nodes. The three-noded face is linear and other, more conventional methods could be used. However, using a procedure similar to that for the four-noded bilinear faces not only finds any intersection point in the plane containing the three-noded face, but easily confirms that the intersection point is on or within the face.

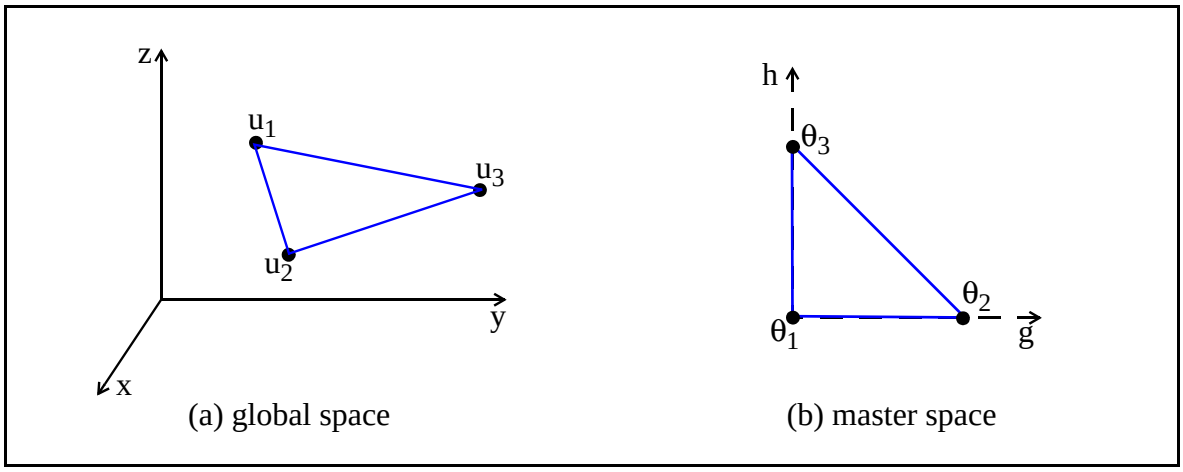

Figure: 5 - 2. Two-dimensional triangular elements.

An arbitrary three-noded triangle face in global space is mapped to an isosceles triangle in master space, Figure 5-2, where node θ_1, that is shared by the equal edges of the isosceles triangle, coincides with the origin. Node θ_2 is located on the *g*-axis a distance of 1 unit from the origin. Node θ_3 is located on the *h*-axis a distance of 1 unit from the origin. For a point to be on or inside the triangular master element, the following relationships among the master space coordinates must hold.

$$\begin{aligned} 0 \le g, h \le 1 \\ 0 \le g + h \le 1 \end{aligned} \qquad \text{(Eq: 5-21)}$$

The mapping function between the global space and the master space system is

$$u^d(g, h) = (1 - g - h)u_1^d + gu_2^d + hu_3^d \qquad \text{(Eq: 5-22)}$$

As before, when all three dimensions are included and like terms are collected, a system of three equations results that can be represented as a matrix equation in *g* and *h*.

$$\bar{F}(g, h) = \bar{A} + \bar{B}g + \bar{C}h \qquad \text{(Eq: 5-23)}$$

where A, B, and C are 3x1 matrices defined as follows:

$$\bar{A} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \bar{B} = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} \qquad \bar{C} = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{bmatrix} \qquad \text{(Eq: 5-24)}$$

Intersection Equations

Likewise, consider a particle at position $\bar{R}$ traveling in direction $\bar{\Omega}$ where $\bar{R}$ and $\bar{\Omega}$ are 3x1 matrices defined as follows:

$$\bar{R} = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} \qquad \bar{D} = \Omega = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \qquad \text{(Eq: 5-25)}$$

The equation for the particle at a distance *d* along its trajectory is, as before,

$$\bar{P}(d) = \bar{R} + \bar{\Omega}d \qquad \text{(Eq: 5-26)}$$

Setting Equation 5-23 equal to Equation 5-26 yields

$$\bar{R} + \bar{D}d = \bar{A} + \bar{B}g + \bar{C}h \qquad \text{(Eq: 5-27)}$$

$$0 = (\bar{A} - \bar{R}) + \bar{B}g + \bar{C}h - \bar{D}d \qquad \text{(Eq: 5-28)}$$

$$(\bar{R} - \bar{A}) = \bar{M} \begin{bmatrix} g \\ h \\ d \end{bmatrix} \qquad \text{(Eq: 5-29)}$$

where

$$\bar{M} = [[\bar{B}][\bar{C}][-\bar{D}]] = \begin{bmatrix} B_1 & C_1 & -D_1 \\ B_2 & C_2 & -D_2 \\ B_3 & C_3 & -D_3 \end{bmatrix} \qquad \text{(Eq: 5-30)}$$

Solving for the linear terms of the unknowns.

$$\begin{bmatrix} g \\ h \\ d \end{bmatrix} = \overline{M}^{-1}(\overline{R} - \overline{A}) \qquad \text{(Eq: 5-31)}$$

The solution for the three unknowns is straight forward and only involves a matrix multiplication after the 3x3 M matrix is inverted and a normalized vector is found by subtracting A from R.

Chapter 6: Part III Intersection

Quadratic element faces often appear when second order polyhedra are created by CAE meshing tools. Finding intersections with these surfaces is a must for the Monte Carlo tracking algorithms, but the analytic approach of the preceding chapter has not been pursued in this work. Rather, due to the complexity of the mapping functions, a Newton iterative method is used and will be discussed in this chapter for both six- and eight-noded faces.

An Eight-Node Face

A quadratic surface arises when the eight nodes of an element face are non-coplanar.

$$\begin{aligned} u_1 &= (x_1, y_1, z_1) & u_2 &= (x_2, y_2, z_2) \\ u_3 &= (x_3, y_3, z_3) & u_4 &= (x_4, y_4, z_4) \\ u_5 &= (x_5, y_5, z_5) & u_6 &= (x_6, y_6, z_6) \\ u_7 &= (x_7, y_7, z_7) & u_8 &= (x_8, y_8, z_8) \end{aligned} \qquad \text{(Eq: 6-1)}$$

The eight nodes (see Figure 6-1b) represent a quadratic face in 3-D global space (Figure 6-1a), that for the purpose of this method are mapped to a square in master space (see Figure 6-1c), in a manner similar to that discussed in Chapter 5.

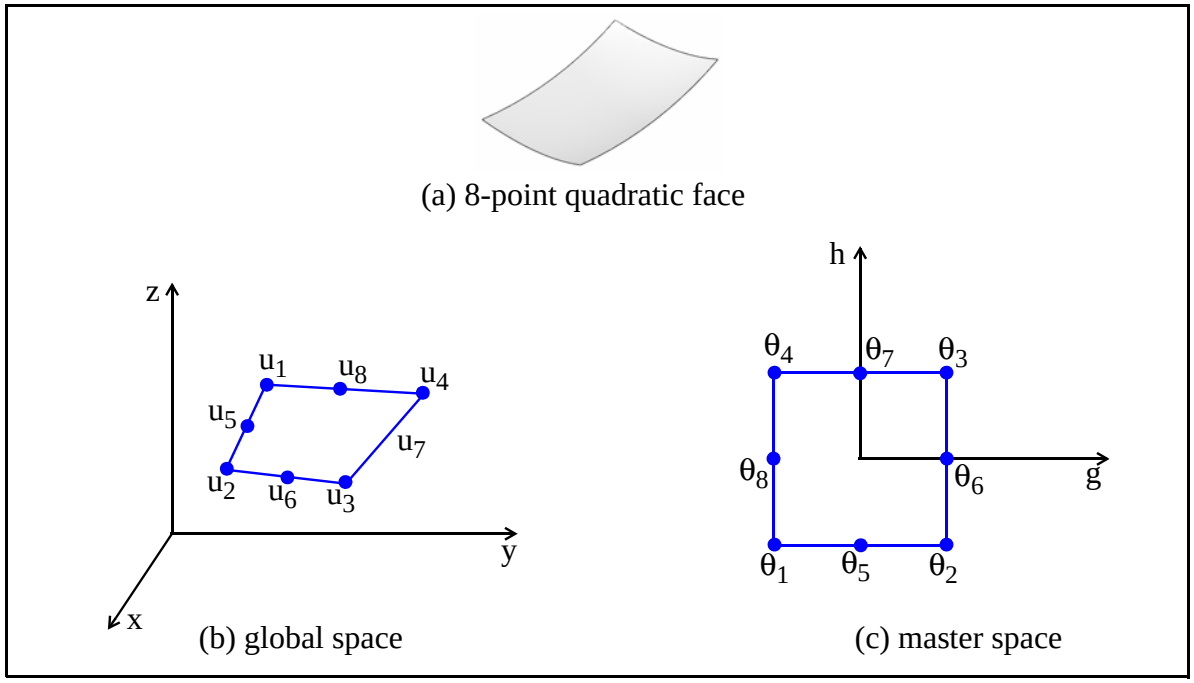

Figure: 6 - 1. Two-dimensional second-order quadrilateral elements.

As discussed in Chapters 3 and 5, the origin of the master space coordinate system is at the center of the master element and the extents of the master space element sides are from -1 to +1. If a ray intersects this 2-D element, the values of *g* and *h* must both be between -1 and +1 or be equal to -1 or +1. That is,

$$-1 \le g, h \le +1 \qquad \text{(Eq: 6-2)}$$

Any values of *g* or *h* outside of this range imply a miss (i.e., no intersection with the faces).

Since the master element contains nodes on the edges and none in the interior, this is a serendipity element and the correct mapping function between the global space and the master space is given as

$$\begin{aligned} u^d(g,h) \;=\; & \frac{1}{2}[(1-g)(1+g)(1-h)u_5^d + (1-h)(1+h)(1+g)u_6^d] \\ & + \frac{1}{2}[(1-g)(1+g)(1+h)u_7^d + (1-h)(1+h)(1-g)u_8^d] \\ & - \frac{1}{4}[(1-g)(1-h)(1+g+h)u_1^d + (1+g)(1-h)(1-g+h)u_2^d] \\ & - \frac{1}{4}[(1+g)(1+h)(1-g-h)u_3^d + (1-g)(1+h)(1+g+h)u_4^d] \end{aligned} \qquad \text{(Eq: 6-3)}$$

where $u^d(g,h)$ is a coordinate in the global space expressed as a function of master space coordinates *g* and *h*.

This equation is nothing more than an expansion of the generic mapping function of Equation 3-10.

$$u^d = \sum_{n=1}^{N} \phi_n(\bar{\theta}) u_n^d \qquad \text{(Eq: 6-4)}$$

When all three dimensions are included, this becomes a system of three equations that can be represented as a matrix equation. Like terms can be collected in Equation 6-3 to yield an equation of the form

$$F(g,h) \;=\; \alpha_1 + \alpha_2 g + \alpha_3 h + \alpha_4 gh + \alpha_5 g^2 + \alpha_6 h^2 + \alpha_7 g^2 h + \ldots \qquad \text{(Eq: 6-5)}$$

Any point in the master space surface containing the element's face can be expressed by either equation and $F(g,h)$ is synonymous with $u^d(g,h)$

A Six-Node Face

A quadratic surface also arises when the six nodes of a triangular element face are non-coplanar.

$$u_1 = (x_1, y_1, z_1) \qquad u_2 = (x_2, y_2, z_2)$$
$$u_3 = (x_3, y_3, z_3) \qquad u_4 = (x_4, y_4, z_4) \qquad \text{(Eq: 6-6)}$$
$$u_5 = (x_5, y_5, z_5) \qquad u_6 = (x_6, y_6, z_6)$$

The six nodes (see Figure 6-2b) represent a quadratic face in 3-D global space (Figure6-2a), that for the purpose of this method are mapped to an equilateral triangle in master space (see Figure 6-2c), in a manner similar to that presented in previous discussions.

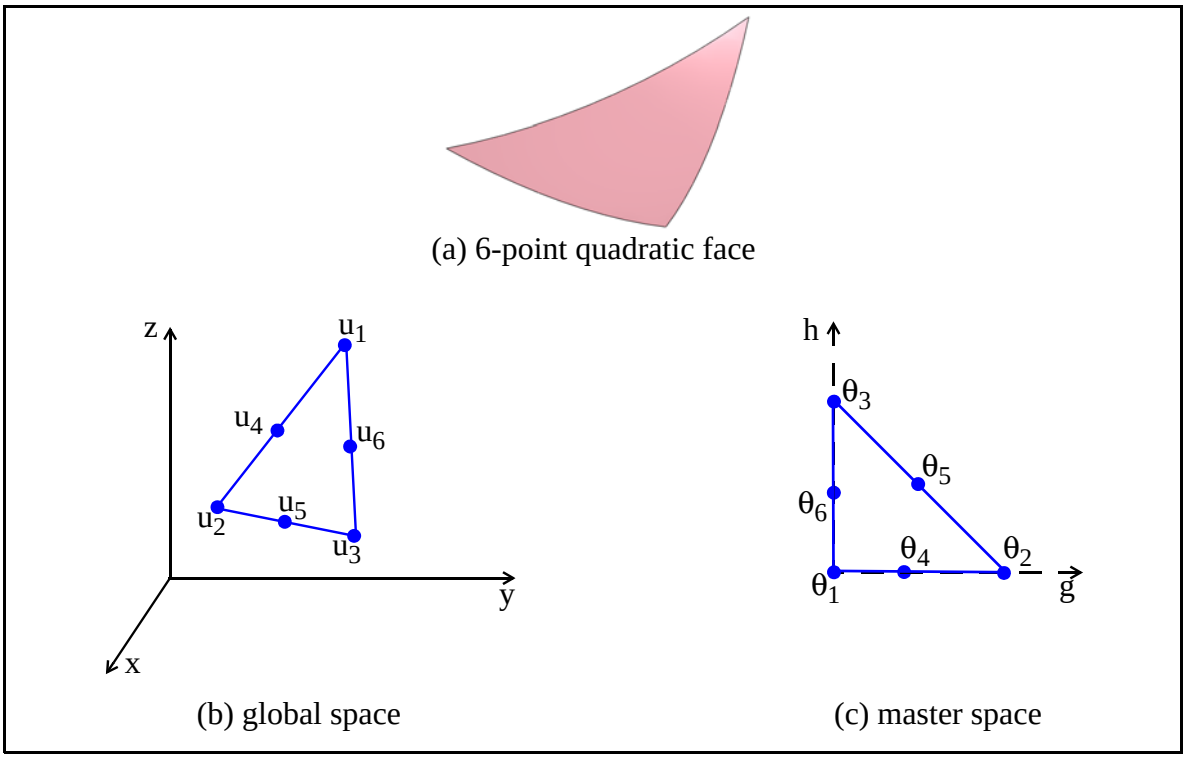

Figure: 6 - 2. Two-dimensional second-order triangular elements.

The origin of the master space coordinate system coincides with node θ_1 and the length of an edge parallel to a coordinate axis extends from 0 to +1. If a ray intersects this 2-D element, the values of *g* and *h* must meet the following requirements.

$$0 \le g, h \le +1$$
$$0 \le g + h \le +1 \qquad \text{(Eq: 6-7)}$$

Any values of *g* and *h* outside of this range imply no intersection with the element's face.

Since the master element contains nodes on the edges and none in the interior, this is a serendipity element and the correct mapping function between the global space and the master space is given as

$$u^d(g,h) = [2(1-g-h)-1](1-g-h)u_1 + g(2g-1)u_2 + h(2h-1)u_3 + 4g(1-g-h)u_4 + 4ghu_5 + 4h(1-g-h)u_6 \quad \text{(Eq: 6-8)}$$

Like terms can be collected to yield an equation of the form shown with Equation 6-5.

Intersection Methodology

The problem to be solved for an eight-noded face is depicted in Figure 6-3. The corresponding problem to be solved for a six-noded face is shown in Figure 6-4,

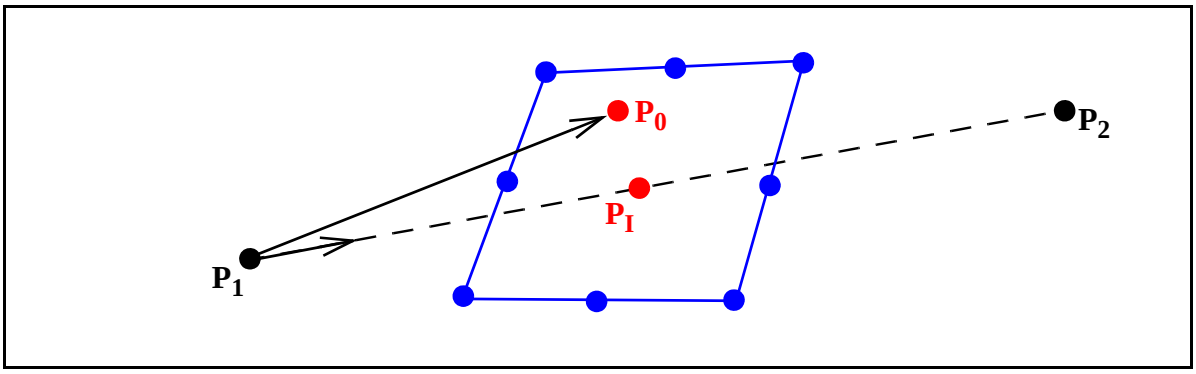

Figure: 6 - 3. Particle ray intersecting a second-order quadrilateral element face.

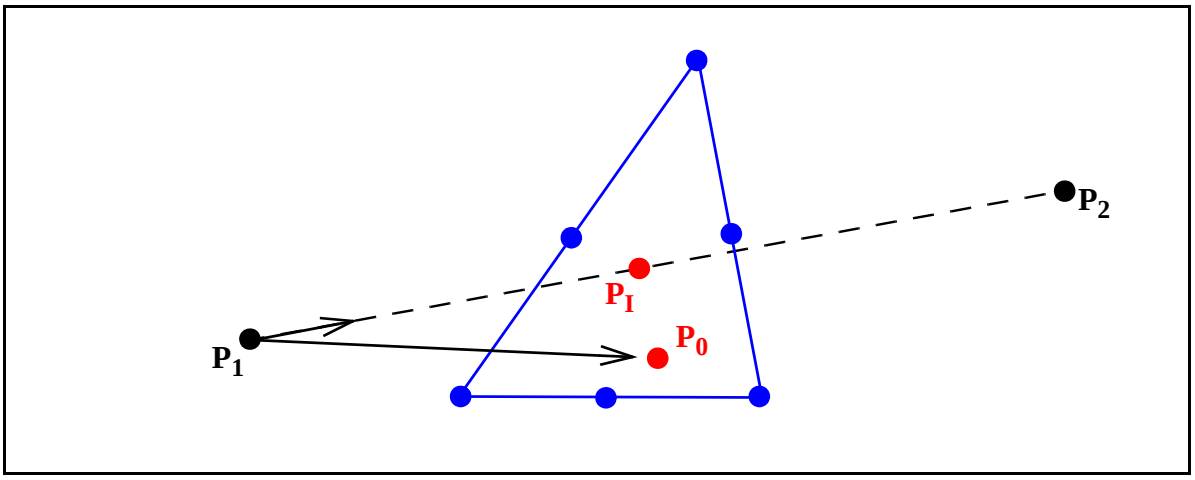

Figure: 6 - 4. Particle ray intersecting a second-order triangular element face.

In global space, a particle travels in a straight line from $\overline{P}_1$ to $\overline{P}_2$ intersecting the quadratic face at $\overline{P}_I$, Figures 6-3 and 6-4. When transformed to the master space coordinate system (see Chapter3), the quadratic face becomes planar and the straight line becomes somewhat curved. The previous methods (Chapters 4 and 5) for finding $\overline{P}_I$ won't produce the correct result in this situation. Instead the following approach, referred to as the d-squared or cross-product method is used.

The cross product of the particle's flight vector, $\overline{P_{12}}$, into a vector $\overline{P_{10}}$ from the particle's current position, $\overline{P_1}$, to some arbitrary point, $\overline{P_0}$, constrained to be on the face, creates a vector $\bar{d}$ that is perpendicular to both vectors, $\overline{P_{12}}$ and $\overline{P_{10}}$. Finding $\overline{P_I}$ requires minimizing the angle θ between $\overline{P_{10}}$ and $\overline{P_{12}}$; hence, finding a cross product with a magnitude of zero. Since $\overline{P_0}$ is constrained to be a point on the quadratic face that is arbitrarily oriented in 3-D global space, $\overline{P_0}$ is a function of the master space coordinates: g, h, r. Knowing which element face that the particle is attempting to intersect is equivalent to fixing either g, h, or r. Since the 3-D mapping function is regular in g, h, and r, choose r and fix its value at +1. The 3-D mapping function simplifies and the corresponding nodes numbers can be re-mapped to 1 through 8, Equation 6-3, or 1 through 6, Equation 6-8.

In either case, the cross product is given as

$$\bar{P}_{10} \times \bar{P}_{12} = \left|\bar{P}_{10}\right|\left|\bar{P}_{12}\right| \sin\theta = \left|\bar{d}\right| \qquad \text{(Eq: 6-9)}$$

with

$$\bar{d} = f_1\hat{i} + f_2\hat{j} + f_3\hat{k} \qquad \text{(Eq: 6-10)}$$

where

$$\begin{aligned} f_1 &= (y_0 - y_1)(z_2 - z_1) - (y_2 - y_1)(z_0 - z_1) \\ f_2 &= (z_0 - z_1)(x_2 - x_1) - (z_2 - z_1)(x_0 - x_1) \\ f_3 &= (x_0 - x_1)(y_2 - y_1) - (x_2 - x_1)(y_0 - y_1) \end{aligned} \qquad \text{(Eq: 6-11)}$$

and

$$x_0 = x_0(g, h) \qquad y_0 = y_0(g, h) \qquad z_0 = z_0(g, h) \qquad \text{(Eq: 6-12)}$$

The magnitude of $\bar{d}$ is given by

$$\left|\bar{d}\right| = \sqrt{f_1^2 + f_2^2 + f_3^2} \qquad \text{(Eq: 6-13)}$$

However, minimizing $\left|\bar{d}\right|$ is equivalent to minimizing $\left|\bar{d}\right|^2$ and makes the resulting equations less cumbersome to solve.

$\left|\bar{d}\right|^2$ is a function of two unknowns, g and h, and will have its minimum where the derivatives with respect to g and h are zero. Similar to solving for containment in Chapter 3, Newton's method can be used to solve a system of two equations in the two unknowns.

$$\begin{bmatrix} \frac{\partial^2 |\bar{d}|^2}{\partial g^2} & \frac{\partial^2 |\bar{d}|^2}{\partial g \partial h} \\ \frac{\partial^2 |\bar{d}|^2}{\partial g \partial h} & \frac{\partial^2 |\bar{d}|^2}{\partial h^2} \end{bmatrix} \begin{bmatrix} \Delta g \\ \Delta h \end{bmatrix} = - \begin{bmatrix} \frac{\partial |\bar{d}|^2}{\partial g} \\ \frac{\partial |\bar{d}|^2}{\partial h} \end{bmatrix} \qquad \text{(Eq: 6-14)}$$

When Δg and Δh are small such that each is within an epsilon (e.g., 10^{-10}) of zero, the resulting values of *g* and *h* can be used to find $\overline{P_I}$ with either equation 6-3 or 6-8. If the values of g and h satisfy Equation 6-2 or 6-7, the face containing $\overline{P_I}$ is the face the particle intersects.

Regression Test Example

Material in this section demonstrates the difficulty of finding the intersection point on a quadratic face. An example is taken from one of the MCNP regression test problems and is shown in Figure 6-5. While the element is a second-order hexahedron, it is rendered as a first-order element for visualization purposes. A particle is sitting on the bottom face of the element labeled 101 in the figure and needs to travel along the black line through the face with the green dots on its four vertices. The mid-point nodes on each of these edges is some small distance off of the straight line edges shown in the figure. The reader can gain a better understanding of this by noting that the red dot on the black line is the actual intersection point with the quadratic face and should imagine that face bowing outward to include that point.

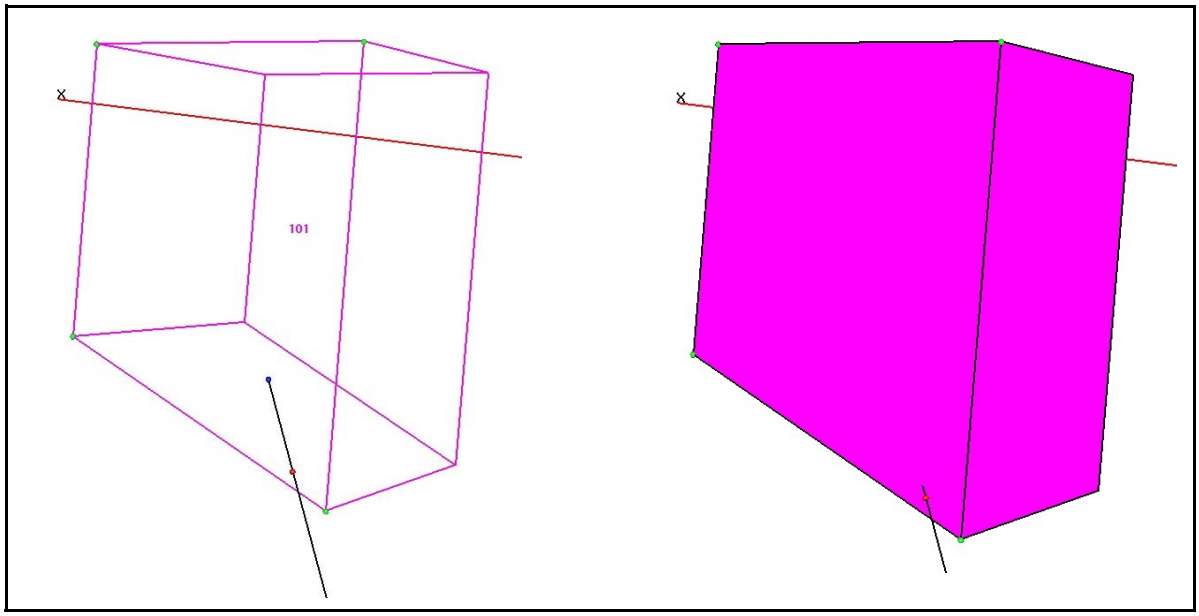

Figure: 6 - 5. Particle intersection with a quadratic face on a second-order hexahedron.

Equation 6-13 for the intersection with this face is plotted as a function of g and h, Figure 6-6, and shows that $|\bar{d}|$ is quite complex with several local minimum in addition to the minimum corresponding to the intersection point at roughly g = 0.825 and h = -0.885. $|\bar{d}|^2$ is shown as a function of g and h in Figure 6-7. The local minima are in the same places in these two figures, but the $|\bar{d}|^2$ surface is smoother. It would be preferable to use $|\bar{d}|$ instead of $|\bar{d}|^2$ in Equation 6-14, but the radical in Equation 6-13 leads to highly complex derivatives that easily double the number of floating point operations in the Newton's method solution.

The first and second derivatives of $|\bar{d}|^2$ are shown in Figures 6-8 and 6-9, respectively. From an examination of these figures, it should be no surprise that an inappropriate initial guess for g and h could cause the Newton's method to converge to a local minimum instead of the intersection minimum.

This section's example was taken from one of MCNP's regression test problems when the calculation stalled because the intersection routine could not find the correct intersection point (red point in Figure 6-5). This was traced to an inappropriate initial guess for g and h. Currently, the second-order intersection routine is structured to take the four vertices of the face and use the first-order (analytic) intersection routine to estimate where the intersection point is on the face. The final pair of g and h values from the first-order routine can then be used as the initial guess for the second-order intersection routine.

As a last resort, if the second-order intersection routine fails, then a more computationally intensive, liner bisection method is used. In this method two points are found to be on opposite sides of the face, with one point inside the element and the other outside the element to which the face belongs. Then by bisecting the distance between the two points to find a middle point, this point can be checked using the containment routines of Chapter 3 to see whether it is inside or outside of the element. The result of the containment routine determines which point is replaced before the bisection process is repeated. When the two points are within an epsilon (10^{-14}) of each other, the midpoint should be on the surface of the face. The value of 10^{-14} is needed to produce a result of sufficient precision that is consistent with the other intersection methods. From the small amount of empirical data observed to date, this bisection procedure generally takes on the order of 25 iterations to find the intersection point on the face where each iteration calls the containment routines which are in turn iterative. For any particular second-order element, this method will produce the correct intersection point with that element no matter which face is specified. Therefore, the code is set to do this operation only one time per element for the current particle's flight segment.

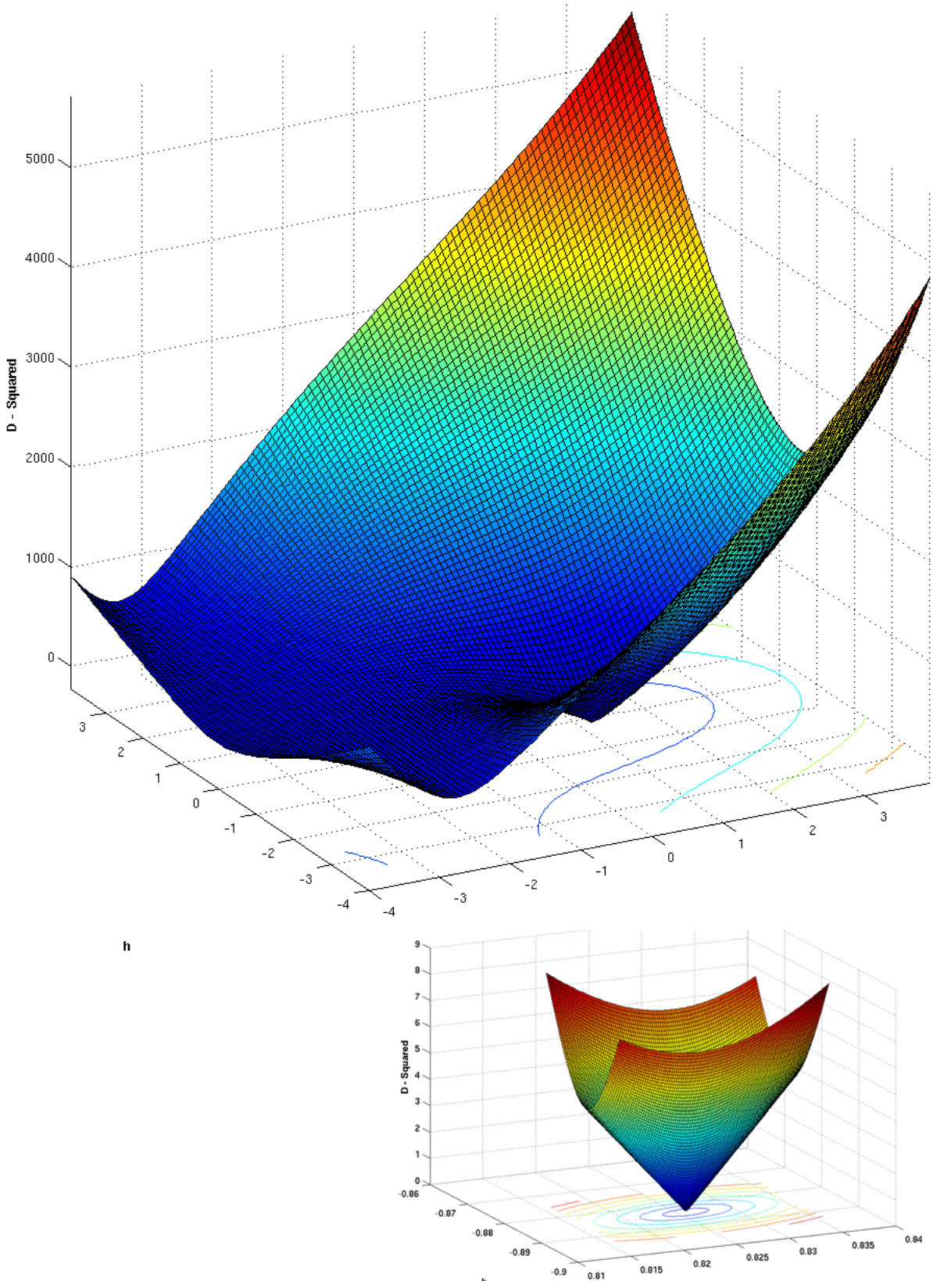

Figure: 6 - 6. $|\bar{d}|$ as a function of g and h for example quadratic face intersection.

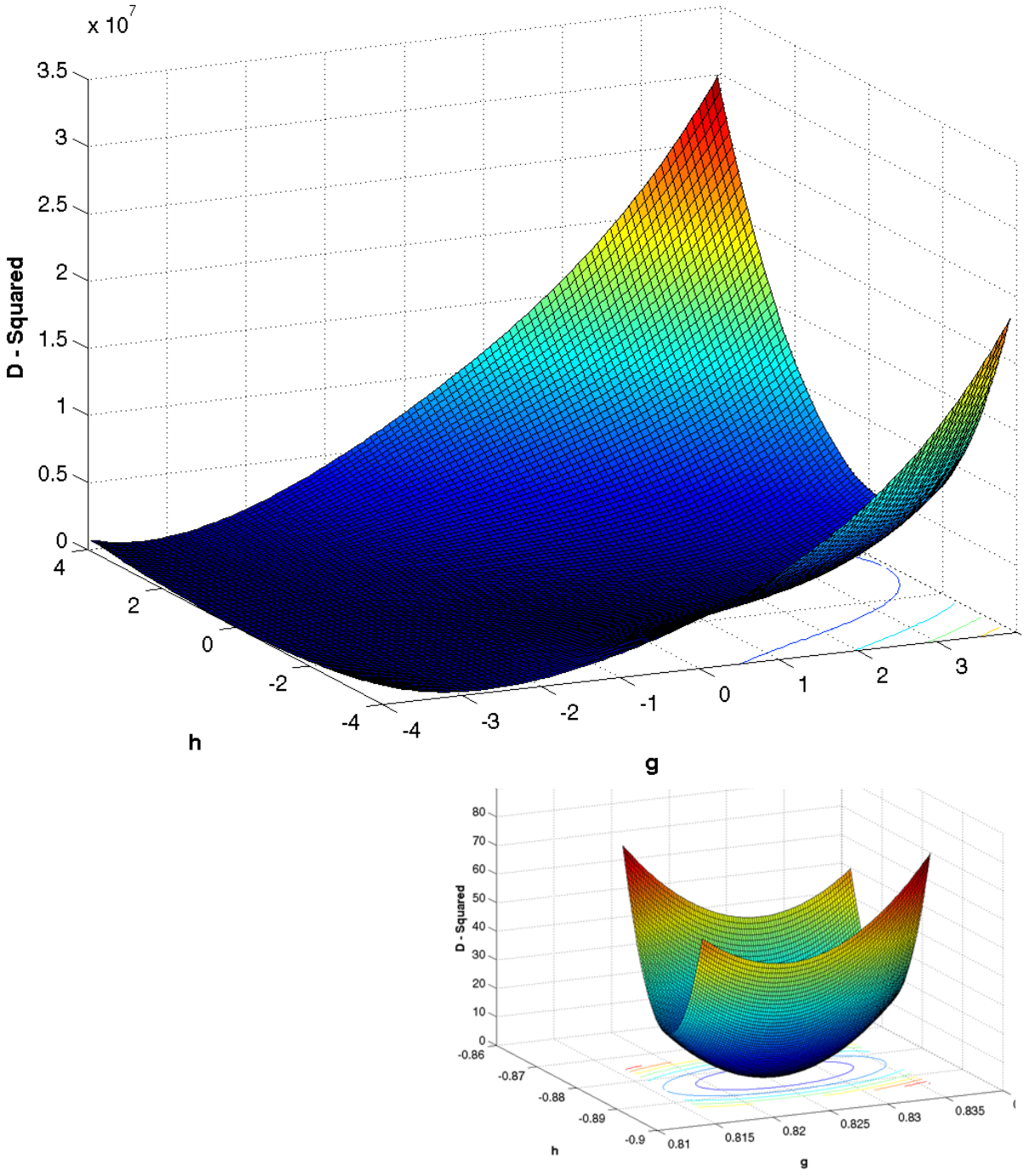

Figure: 6 - 7. $|\bar{d}|^2$ as a function of g and h for example quadratic face intersection.

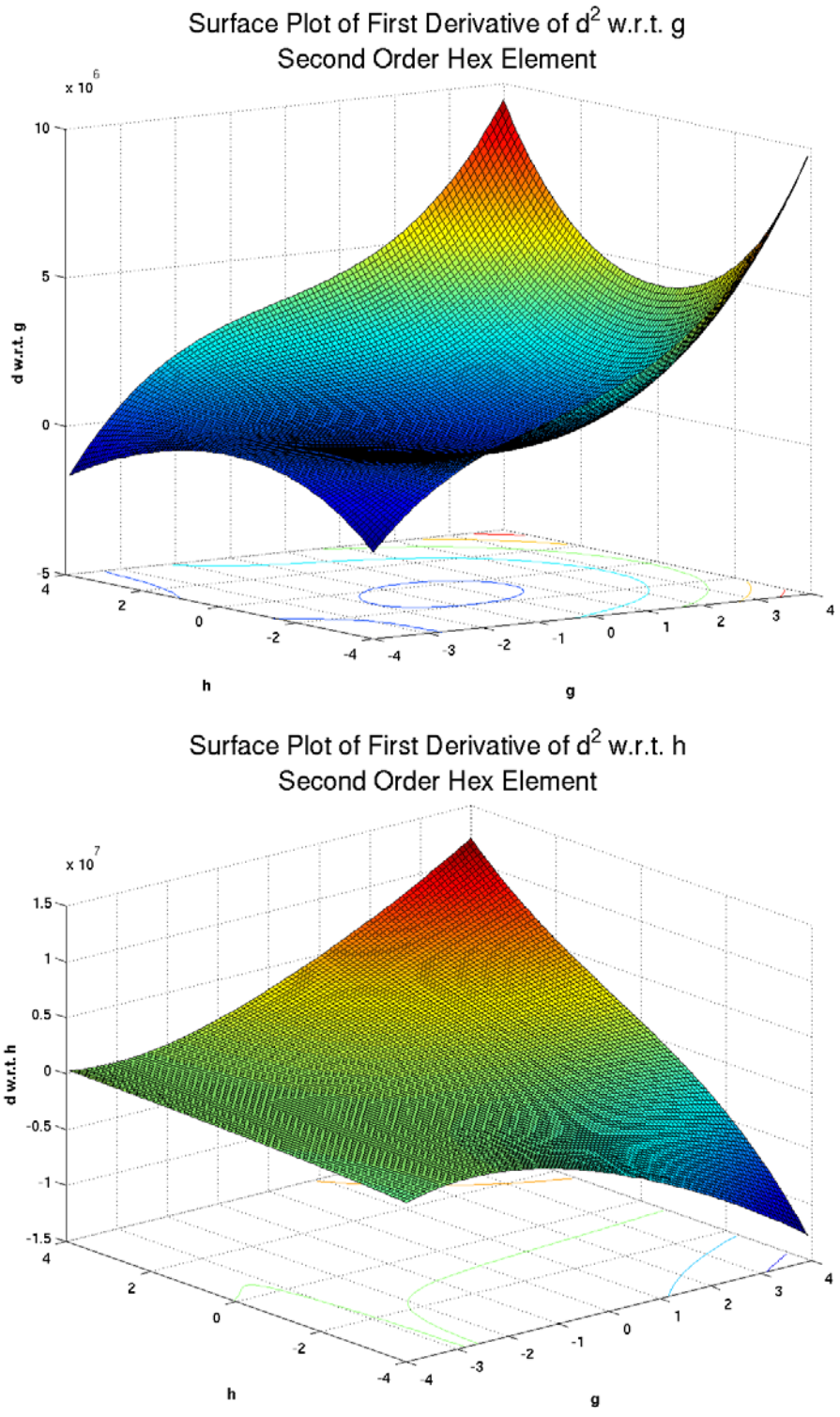

Figure: 6 - 8. First derivatives of $|\bar{d}|^2$ as a function of g and h.

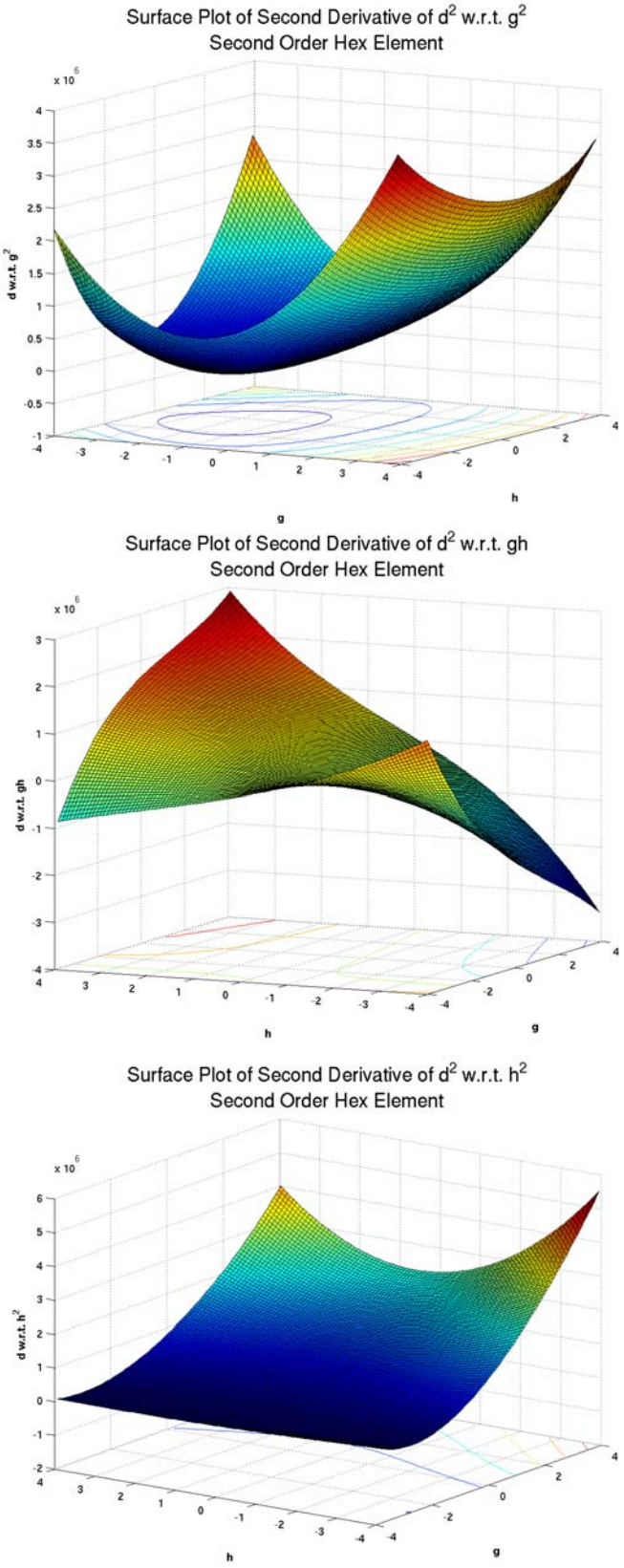

Figure: 6 - 9. Second derivatives of $|\bar{d}|^2$ as a function of g and h.

Chapter 7: Part IV Intersection

The intersection methods of the pervious chapters were not the only ones explored for use in this work. This chapter presents the *tgh-method* or otherwise known as the parameterized line intersection method; one that was initially invented and later discarded because of its larger floating point operation count. It is included in this work for completeness.

Intersection Methodology

The problem to be solved is the same one that has been described in previous chapters and depicted by Figures 6-3 and 6-4. A particle travels in a straight line from point $\overline{P_1}$ to point $\overline{P_2}$ and intersects some surface of unstated curvature at point $\overline{P_I}$. The surface may be defined by 3, 4, 6 or 8 nodes in a form similar to Equations 5-3, 5-23, or 6-5. The straight line equation between the two points can be parameterized with parametric coordinate *t* as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x_{P_1} + x_{P_2} \\ y_{P_1} + y_{P_2} \\ z_{P_1} + z_{P_2} \end{bmatrix} + \frac{t}{2}\begin{bmatrix} x_{P_1} - x_{P_2} \\ y_{P_1} - y_{P_2} \\ z_{P_1} - z_{P_2} \end{bmatrix} \qquad \text{(Eq: 7-1)}$$

When *t* is set to -1, Equation 7-1 yields point $\overline{P_2}$ and when *t* is set to +1 point $\overline{P_1}$ is the result. A value of *t* between -1 to +1 yields a point between these points.

Equation 7-1 can be set equal to any of the surface equations (5-3, 5-23, or 6-5) to yield a system of three equations in three unknowns that can be solved using Newton's method as described in Chapters 3 and 6. For example, consider Equation 5-3 for the intersection surface. The system of equations become

$$f_k = \frac{k_{P_1} + k_{P_2}}{2} + \frac{k_{P_1} - k_{P_2}}{2} - A_k - B_k g - C_k h - D_k gh \qquad \text{(Eq: 7-2)}$$

where *k* is either *x, y,* or *z* and A_k, B_k, C_k, and D_k are given by the appropriate rows in the Equation 5-4 definitions for *A, B, C,* and *D,* respectively.

When Newton's method is applied, the result is

$$\begin{bmatrix} \Delta t \\ \Delta g \\ \Delta h \end{bmatrix} = \begin{bmatrix} \frac{\partial f_x}{\partial t} & \frac{\partial f_x}{\partial g} & \frac{\partial f_x}{\partial h} \\ \frac{\partial f_y}{\partial t} & \frac{\partial f_y}{\partial g} & \frac{\partial f_y}{\partial h} \\ \frac{\partial f_z}{\partial t} & \frac{\partial f_z}{\partial g} & \frac{\partial f_z}{\partial h} \end{bmatrix}^{-1} \begin{bmatrix} -f_x \\ -f_y \\ -f_z \end{bmatrix} \qquad \text{(Eq: 7-3)}$$

When Δt, Δg, and Δh are small such that each is within an epsilon (e.g., 10^{-10}) of zero, the resulting values of t, *g*, and *h* can be used to find $\overline{P_I}$ with either the parametrized line equation or the surface equation.

When this methodology is used with the quadratic faces from second-order elements, good values for *g* and *h* are needed as was discussed with the d-squared method at the end of Chapter 6.

Chapter 8: Tracking

Tracking a particle from one element to another in an UM representation of a geometry is the key feature (i.e., in-mesh tracking) of the UM library. Implementing this key feature given the requirements of Chapter 1 is a complex task. This chapter introduces the basic concepts needed to achieve this goal.

Tracking Scenarios

In order to construct a successful UM tracking algorithm that meets the general requirements presented in Chapter 1, the expected tracking scenarios that can be encountered under these circumstances must be understood. Broadly, these scenarios can be broken into two categories: 1) inside an instance or part, and 2) between instances or parts.

In-Part Tracking

At the part level, all elements are assumed to be connected so that each face and its nodes are shared by two and only two elements, except for those faces on the part's surface where some nodes are not shared. The sharing of faces results in an element possessing nearest neighbors with the total number of nearest neighbors for an element equal to the number of faces, except for those elements on a surface. As shown in Figure 8-1 for a 2-D mesh, element 5 is an interior element and shares four faces, one each with elements 2, 4, 6, and 8. Element 3 is a surface element and shares only two faces, one with element 2 and one with element 6. Nearest neighbors sharing an edge but not a face (elements 1, 3, 7, and 9 for element 5) are not determined ahead of time and stored since this would require more memory, particularly for tetrahedra, and the tracking routines can determine these on-the-fly.

Five potential tracking scenarios are shown in Figure 8-1 for in-part tracking (IPT). Scenario A shows a particle inside element 5 tracking to the face that it shares with element 8. This is the most frequent scenario that occurs while tracking within a part because of the large amount of area associated with the internal faces. Since this is the most frequent scenario, it is beneficial to create nearest neighbor lists for each element so that elements from these lists are the first ones that are checked during the IPT. If the nearest neighbor check is successful in yielding the next element to which a particle tracks, the other more expense checks, described elsewhere in this chapter, need not be done and the E2E tracking mode is fairly quick. Therefore, if E2E tracking is required, large blocks of contiguous mesh are highly desirable to take advantage of the nearest neighbor lists. The nearest neighbor lists can be created during problem initialization or constructed on the fly when any particle first enters an element. This later approach has the advantage of using computer time in constructing the lists for only those elements that are seen in the problem by the simulation's particles. However, creating the lists during problem initialization can take advantage of dedicated processors during parallel input processing.

Scenario B in Figure 8-1 shows a particle from element 5 tracking through a corner or edge to element 1. This scenario can be handled with the nearest neighbor list search if the faces are permitted fuzzy boundaries. That is, the extents of the faces are taken to be an epsilon larger (10^{-10}) in all directions so that there is essentially a small overlap of the faces at an edge. Then the intersection routines will find an intersection with either element 2 or element 4, depending upon which it encounters first. So, the particle in Figure 8-1 actually tracks from element 5 to either element 2 or 4, with a very tiny to negligible path length attributed to the chosen element, before tracking to element 1. While this approach may not be entirely appealing and may only be a real concern if this negligible path length really matters, the alternatives can be costly. For example, it may be possible to create next neighbor lists to keep track of an element's neighbors associated with an edge or corner. While this may be reasonable for hexahedra, the amount of storage needed to remember all of the tetrahedra that can share one single corner can be significant since it is possible for hundreds of tetrahedra to share a corner. If this were carried out for all tetrahedra in a model, the storage could be quite large and associated with this is a large amount of time to sort through the list to find the appropriate candidate. In fact, this is not a fool-proof method in dealing with tetrahedra when the collision or source point is very near a vertex that is shared by numerous elements.

Scenario C in Figure 8-1 arises when the surface faces are bilinear or biquadratic and, hence, possess curvature. This scenario can be referred to as the re-entrant element scenario and often leads to a gap situation if there is no element encountered from another part (to be discussed later). A similar situation arises with scenario D, but this is because of the part's surface curvature. This scenario can be referred to as the re-entrant part scenario. Likewise, the treatment of this track becomes more complicated if there is an element from another part lying along the path from element 3 to element 9.

Because of the re-entrant element scenario, faces can't be eliminated from intersection checking as they are with MCNP's csg tracking routines.

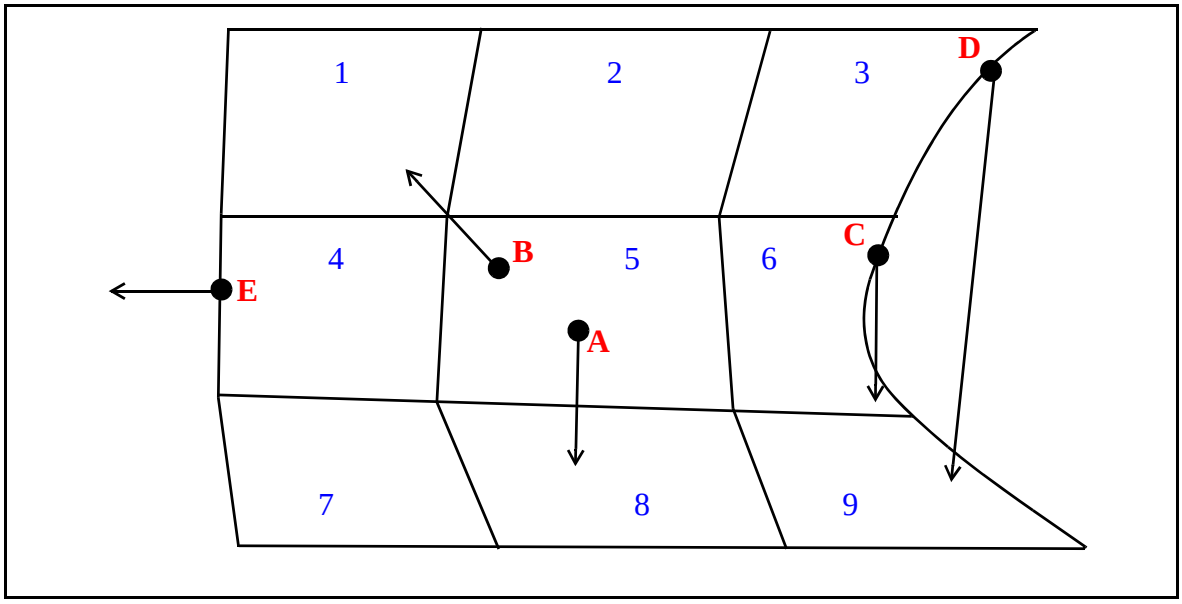

Figure: 8 - 1. IPT scenarios

Scenario E in Figure 8-1 is technically not tracking within the part, but rather represents particles leaving the part to either eventually enter another part or leave the mesh body entirely. This scenario is discussed more fully in the between-part tracking section.

Between-Part Tracking

When transiting from one part to another, the complexity of the tracking algorithm increases since parts may be in contact and essentially share a surface but not element faces or nodes (i.e., redundant faces and nodes), parts my overlap, or gaps may exist between parts. Eventually, a particle will leave the mesh entirely so that the transition is from the part to the background region. Six potential tracking scenarios are shown in Figure 8-2 for between-part tracking (BPT) that drive the algorithms for determining if the tracking continues in the unstructured mesh or in the background material.

The main objective of BPT is to find the element in the next part, if the particle is not leaving the mesh body, to which it will track and from which IPT will take over. Therefore, it is only necessary to know what elements and their faces comprise a part's surface. For this reason, special *skd*-trees are created at program start up that contain the necessary surface information. Unlike Figure 8-1, the components shown in the Figure 8-2 diagrams are surfaces for the parts that they represent. The diagrams in Figure 8-2 are the result of observing tracking behavior in early implementations of the mesh tracking routines while working with some rather large and complex models that possessed both overlaps and gaps. These diagrams represent a simplification of what was observed and serve to illustrate the issues that must be considered in order to meet the tracking requirements outlined in Chapter 1.

A key to understanding how the BPT works in conjunction with determining overlaps and gaps is recognizing that during an intersection search of a surface tree, if the particle ray enters a part then it must exit the part. That is, expect the intersection search on a part to find both entering and exiting elements (points) except for two special cases: 1) the particle ray grazes a corner or edge, and 2) the parts overlap and the starting point is already in the part that is searched; there will only be an exit intersection point. The first of these special cases is basically a *don't care* situation since the particle track makes at most a negligible contribution. The second of the special cases results in only one positive intersection point (i.e., in the forward direction) and can be readily used as discussed later in this chapter.

In addition, because of the re-entrant element and re-entrant part scenarios discussed in the previous section, there is the likelihood that a particle path will have more than two intersection points with a part, indicating multiple "entrance" and "exit" points.

The key to BPT is creating a list of intersection distances and points when considering the particle's projected trajectory from its current location through the remaining parts. With information from this list, the tracking algorithm can decide which of the Figure 8 cases exist and how the code should continue with the tracking.

Shown in Figure 8-2(a), BPT scenario I is a two parts in contact, no overlap, no gap case. It illustrates a particle tracking from part A to part B where the parts are in contact, sharing a surface but not sharing nodes or faces. This most likely occurs when the surface is flat. It is highly unlikely for two curved surfaces to be in full contact (i.e., without gaps and overlaps) unless special measures are taken in the CAE tool to merge the faces. In this scenario, the dis-

tance between point 1 in part A and point 1 in part B is zero (i.e., $d_{11}(AB) = 0$) and the distance between point 1 in part A and point 2 in part B is greater than zero (i.e., $d_{12}(AB) > 0$). It should be clear that the next element along the particle's path is the element in part B that contains point 1. In theory, this should be an easy case for the tracking routine to handle, but depends to a degree upon how complicated the geometry is and the totality of the intersection list for which other scenarios must be eliminated.

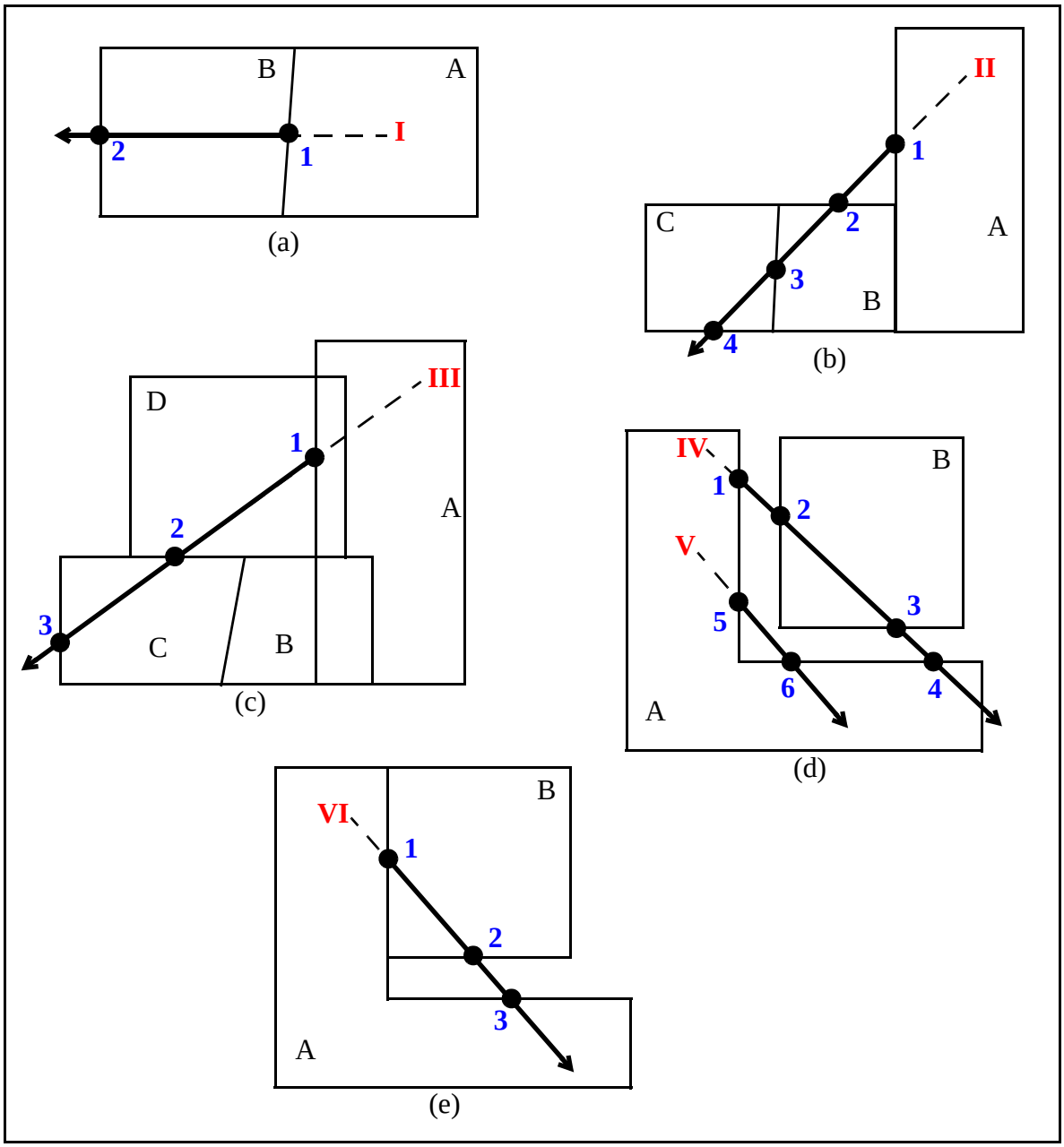

Figure: 8 - 2. BPT tracking scenarios

Figure 8-2(b) presents scenario II and is a no overlap, multiple parts with single gap case. In this scenario the following relationship among the intersection distances hold:

$$0 < d_{12}(AB) < d_{13}(AB) = d_{13}(AC) < d_{14}(AC)$$

From this list and a negative overlap test, the algorithm determines that the particle exits part A at point 1. The gap is next treated as part of the background until the particle reenters the mesh.

Figure 8-2(c) presents scenario III and is a no gap, overlap, multiple parts case where there is the potential for more than two parts to overlap at the point the particle exits the old part, A. Multiple simultaneous overlaps may seem highly unlikely or even non-existent, but it is possible and has been seen in practice; it can be easily handled by the tracking algorithms, as will be discussed shortly. It should be quite clear that if part D is not present, then an arrangement much like scenario II exists with a gap distance of $d_{12}(AC)$ which in this case equals $d_{12}(AD)$. However, this scenario demands that an overlap test take place (as previously discussed) to either confirm or deny an overlap condition before the gap test is performed. The overlap check involves a containment search of all possible new (remaining) parts where a list is constructed of all elements that contain the exiting location from the old part. If the list is empty, there is no overlap. If the list contains only one element, that is the element along the particle's path. If there are multiple elements in the list, indicating overlap of more than two parts, there are several possible approaches. The easiest is to select the first in the list and continue the particle track with it. Another is to compute the distances from the exit location in the old part to the exit locations of all potential new parts from the list and select the part showing the shortest distance.

Scenario IV is shown in figure 8-2(d) and represents a case of multiple gaps and no overlapping parts. When the IPT routine finishes in this case, it is known that the particle leaves part A at point 1 and re-enters part A at point 4 (i.e., IPT scenario D) with a potential gap distance of $d_{14}(AA)$. This triggers the BPT routine which returns intersections with surface elements in part B at points 2 and 3. With $0 < d_{12}(AB) < d_{14}(AA)$, the algorithm selects the element containing point 2 in part B as the next potential element along the path since $d_{12}(AB)$ is the shortest distance to travel. Since a gap is encountered, control exits the unstructured mesh tracking routine for proper consideration of the background material.

Scenario V is also shown in figure 8-2(d) and is the same as IPT scenario D. Because $d_{56}(AA) > 0$, the BPT routine is triggered. However, since intersections are not found with any of the potential new parts, the reentrant part scenario is confirmed and the next potentiall element along the particle's path is that occurring at point 6 in part A. Since a gap is encountered, control exits the unstructured mesh tracking routine.

Scenario VI is shown in figure 8-2(e) and can be considered a no overlap, contact, gap case. This is a variation of scenarios IV and V and like them, it is triggered by the IPT routine indicating a potential reentrant part scenario, $d_{13}(AA) > 0$. After the BPT routine searches the other parts, a list of intersection distances exist with the following relationship:

$$0 = d_{11}(AB) < d_{12}(AB) < d_{13}(AA)$$

$d_{11}(AA) = 0$ implies that the old part and new part are in contact, the next element along the flight path is that in part B at point 1, and there is no gap to consider.

Gap Tracking

In versions of MCNP6 prior to version 1.1 beta, gaps were treated as voids in a rather convoluted manner. This has been corrected starting with version 1.1 beta so that all gaps are treated as regions of the background material. When a particle tracks to a gap, the unstructured mesh tracking stops and the legacy tracking routines take over.

Overlap Tracking

From the BPT discussion previously in this chapter, it should be recognized that the UM library keeps a particle in a part until it can determine a new one. In the case of overlaps, it assumes the new part starts where the old part ends. This is the `EXIT` overlap model and is the default overlap model in use since it was the first one developed and implemented.

One of the initial requirements for the unstructured mesh implementation in MCNP was to permit multiple, non-contiguous, meshed parts instead of requiring one contiguous mesh. This naturally leads to the possibility of overlapping parts, particularly when two parts attempt to share a curved surface. If it is crucial to the model that the integrity of any curved surface be maintained, the user should then consider, using Abaqus/CAE terminology, merging the two separate parts into a single part, try second-order elements, or try refining the mesh. Significant overlapping regions are never a good idea. Users should never rely on any of the following models to correctly produce the same results as a model where the boundary between two regions is definitely defined so that there is no overlap.

The program can accommodate a small amount of overlap in one of several ways. For the initial implementation there was no correction for tracking through overlapping elements. A particle tracks in an element until it finds a definite transition point in phase space (i.e., another element, or gap). Of the three overlap models currently in place (see Figure 8-3 below), this is known as the `EXIT` model, meaning that in an overlap situation the exit point of the overlap is used as the transition point from one part to the other and results are accumulated accordingly.

The second overlap model, `ENTRY`, is the one that uses the entry point of the overlap in an overlap situation and the results are accumulated accordingly. The third and last overlap model is called `AVERAGE` and results in averaging the entry and exit points in an attempt to find the midpoint of the overlap in the direction the particle is tracking; the particle's path length in the overlap is then divided between the two parts instead of being assigned to one or the other. In actuality, the particle's position is advanced only to the location of the `AVERAGE` point.

Although the code defaults to the `EXIT` model, ultimately the choice of which model to use is left to the user. If both parts are important and the particle flux through this region is fairly isotropic, the `AVERAGE` model is probably the best choice. If the flux is somewhat more directional and one part is deemed more significant than the other, a better choice might be `ENTRY` or `EXIT`; the user must decide.

Recent modification to the code permit model selection by part.

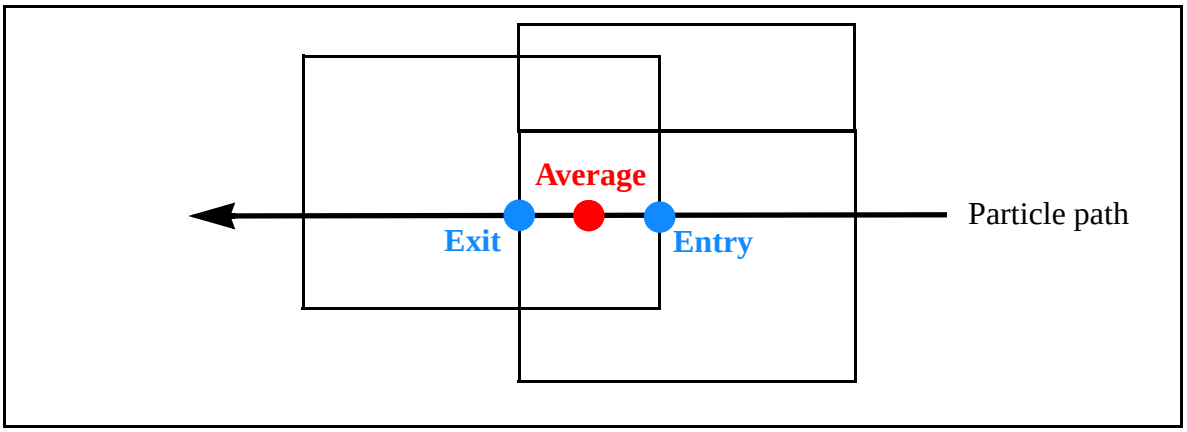

Figure: 8 - 3. Illustration of the three critical points that define the overlap models.

Chapter 9: Volumes

Accurate volumes are a necessity in a Monte Carlo code such as MCNP not only to serve as a check on the geometry modeling, but also to calculate volume averaged results such as energy deposition per cubic centimeter. This is complicated because of the fact that first and second order elements may possess bilinear or quadratic faces giving rise to trilinear and triquadratic volumes. While numerical integration methods exist for calculating such volumes, an analytic approach is more desirable since this approach eliminates any numerical integration error. This chapter discusses the adoption of a methodology for an analytic approach.

Methodology

The trilinear and triquadratic elements require different analytic formulas for finding their volumes. These formulas are more complex than the familiar ones used in this regard for linear polyhedra. Once again the finite element methodology of transforming from a curvilinear coordinate system (global space) to a Cartesian coordinate system (master space) is used in this endeavor.

The mapping functions from one space to the other depend upon the element type and are presented elsewhere in this work. All six of the different functions of interest can be factored, Equation 9-1, so that there exists one term f_m, Equation 9-2, for each node where there are M total nodes in an element.

$$u^d = \alpha_1 + \alpha_2 g + \alpha_3 h + \alpha_4 r + \alpha_5 gh + \alpha_6 gr + \alpha_7 hr + \alpha_8 g^2 + \alpha_9 g^2 h + \ldots \qquad \text{(Eq: 9-1)}$$

$$f_m \in \{1, g, h, r, gh, gr, hr, ghr, g^2, g^2 h, g^2 r, h^2, \ldots\} \qquad \text{(Eq: 9-2)}$$

Mapping of a node w in master space to its corresponding node w in global space (excessively twisted or inverted elements are not permitted for reasons discussed elsewhere in this work) requires evaluation of the f_m terms at the w node in the master space. Hence, $f_{wm}(g, h, r)\big|_w$ denotes the evaluation at node w with the corresponding values of g, h, and r so that the terms evaluate to either `-1`, `0`, or `+1`. Then in general, the mapping for each dimension between the two spaces for each node w can be written as

$$\left[u_w^d\right] = \left[f_{wm}\right] \cdot \left[\alpha_w^d\right] \qquad \text{(Eq: 9-3)}$$

α_w^d are coefficients for each dimension and can be found from

$$\left[\alpha_w^d\right] = \left[f_{wm}\right]^{-1} \cdot \left[u_w^d\right] \qquad \text{(Eq: 9-4)}$$

To calculate an element's volume, the expression for the differential volume [16] is used

$$dV_{global} = |J|dgdhdr = |J|dV_{master} \qquad \text{(Eq: 9-5)}$$

Where *J* is the Jacobian and

$$|J| = \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{M}|A_{ijk}|\frac{\partial f_i}{\partial g}\frac{\partial f_j}{\partial h}\frac{\partial f_k}{\partial r} \qquad \text{(Eq: 9-6)}$$

With

$$|A_{ijk}| = \begin{bmatrix} \alpha_i^x & \alpha_j^x & \alpha_k^x \\ \alpha_i^y & \alpha_j^y & \alpha_k^y \\ \alpha_i^z & \alpha_j^z & \alpha_k^z \end{bmatrix} \qquad \text{(Eq: 9-7)}$$

The indexes i, j, and k extend over the three dimensions.

The volume is found by integrating Equation 9-5 in the master space where the limits of integration are generally `-1` to `+1` or `0` to `+1`, depending on the element shape and dimension.

The triple summation of Equation 9-6 implies a large number of terms for the volume evaluation. In reality, a great many of these terms are zero. Table 9-1summaries the maximum number of partial terms and the number of non-zero terms by mesh element type.

Table 9-1: Number of partial terms by element type

Faces	Nodes	Max # of Partial Terms	Non-Zero Partial Terms
4	4	64	1
5	6	216	6
6	8	512	8
4	10	1000	64
5	15	3375	222
6	20	8000	222

Note that there is one non-zero term for the first order tetrahedron; this is the familiar formula for the volume of a (linear) tetrahedron. It should be recognized that this methodology holds for area calculations once the appropriate simplifications have been made in Equations 9-1 to 9-7.

Application

Volumes are examined for several primitive objects that posses curvature: a sphere and a right circular cylinder.

A sphere with a radius of 3 is meshed using first and second order hexahedra with various mesh seeds. The seed parameter is an Abaqus/CAE parameter used to set the element size. Abaqus attempts to make elements with edges of this length; no attempt is made to force edges to this length. The number of total elements in this model is dictated by the seed number. The volume results using the current method along with the actual volume are presented in Table 9-2.

Table 9-2: Volume comparison for a sphere (r = 3) with different numbers of hexahedra

Seed	Number of Elements	Mesh Volume	% Difference*
1st Order Hexahedra			
1.5	56	95.0270	19.0
1.0	224	110.2924	2.54
0.5	1320	111.6583	1.29
0.4	3456	112.2451	0.76
0.3	7168	112.6199	0.42
2nd Order Hexahedra			
2.0	32	111.4489	1.50
1.5	56	112.4322	0.59
1.0	224	113.0342	0.056
0.5	1320	113.0941	0.003
0.4	3456	113.0960	0.001
0.3	7168	113.0968	0.0004
Actual		113.0973	n/a

* from actual volume

A right circular cylinder with radius 5 and height 5 is meshed using first and second order tetrahedra with various meshing seeds to obtain representations with various number of mesh elements. The volume results using the current method along with the actual volume are presented in Table 9-3.

Table 9-3: Volume comparisons for a right circular cylinder (r-5, h=5) with different numbers of tetrahedra

Seed	Number of Elements	Mesh Volume	% Difference*
1st Order Tetrahedra			
4.0	123	719.709	9.0
3.0	345	754.154	4.0
2.0	621	762.589	3.0
1.5	2170	778.478	0.9
1.0	5295	780.139	0.7
0.5	33369	784.023	0.2
2nd Order Tetrahedra			
5.0	38	784.635	0.1
4.0	123	784.955	0.056
3.0	345	785.296	0.013
2.0	621	785.343	0.007
1.5	2170	785.393	0.0006
1.0	5295	785.395	0.0004
0.5	33369	785.398	0.0
Actual		785.398	n/a

* from actual volume

This exercise shows that fewer numbers of second order elements by several orders of magnitude are needed to accurately reproduce the intended volumes. This should have a direct impact on particle tracking times, accuracy of results and the number of elements needed, and their associated computer memory allocation.

Chapter 10: Acronyms In This Work

API - - - - - - - - - - - - application programmer interface
BPT - - - - - - - - - - - - between-part tracking
CAD - - - - - - - - - - - - computer aided design
CAE - - - - - - - - - - - - computer aided engineering
CSG - - - - - - - - - - - - constructive solid geometry
E2E - - - - - - - - - - - - element to element tracking
FEA - - - - - - - - - - - - finite element analysis
FEM - - - - - - - - - - - - finite element method
GUI - - - - - - - - - - - - graphical user interface
HIBUC - - - - - - - - - high range bucket
IPT - - - - - - - - - - - - in-part tracking
LANL - - - - - - - - - - Los Alamos National Laboratory
LOBUC - - - - - - - - - low range bucket
MAABB - - - - - - - - minimum axis-aligned bounding box
MCNP - - - - - - - - - - Monte Carlo N-Particle (computer code)
PDE - - - - - - - - - - - - partial differential equation
S2S - - - - - - - - - - - - surface to surface tracking
TAT - - - - - - - - - - - - tried and true
UM - - - - - - - - - - - - unstructured mesh

Chapter 11: Acknowledgements

We would like to thank Brain Jean for his recommendation on the skd-trees and providing reference material. We would like to acknowledge Todd J. Urbatsch, Thomas M. Evans, and Jim Morel for their Reference [19] work.

Chapter 12: References

1. T. Goorley, M. James, T. Booth, F. Brown, J. Bull, L.J. Cox, J. Durkee, J. Elson, M. Fensin, R.A. Forster, J. Hendricks, H.G. Hughes, R. Johns, B. Kiedrowski, R. Martz, S. Mashnik, G. McKinney, D. Pelowitz, R. Prael, J. Sweezy, L. Waters, T. Wilcox, T. Zukaitis, "Initial MCNP6 Release Overview", Nuclear Technology, 180, 1-36, Dec 2012.
2. Timothy J. Tautges, Paul P. H. Wilson, Jason A. Kraftcheck, Brandon M. Smith, Douglass L. Henderson, "Acceleration Techniques For Direct Use Of CAD-Based Geometries In Monte Carlo Radiation Transport," International Conference On Mathematics, Computational Methods & Reactor Physics, Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009).
3. Yican Wu, Qin Zeng, and Lei Lu, "CAD-Based Modeling For 3D Particle Transport," International Conference On Mathematics, Computational Methods & Reactor Physics, Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009).
4. D. Groβe and H. Tsige-Tamirat, "Current Status of the CAD Interface Program McCad for MC Particle Transport Calculations," International Conference On Mathematics, Computational Methods & Reactor Physics, Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009).
5. J.N. Reddy, *An Introduction to the Finite Element Method*, 3rd Edition, McGraw-Hill, Boston, 2006.

6. G. Pelosi, "The Finite-Element Method, Part I: R.L. Courant," IEEE Antennas and Propagation, Vol. 49, No. 2, 180-182, April 2007.

7. R. Courant, "Variational Methods for the Solution of Problems of Equilibrium and Vibrations," Bulletin of the American Mathematical Society, 49, 1-23, 1943.

8. M.J. Turner, R.W. Clough, H.C. Martin, and L.J. Topp, "Stiffness and Deflection Analysis of Complex Structures," Journal of Aeronautical Science, 23, 805-823, 1956.

9. R.W. Clough, "The Finite Element Method in Plane Stress Analysis," Journal of Structures Division, ASCE, Proceedings of 2nd Conference on Electronic Computation, 345-378, 1960.

10. Chandrakant S. Desai and John F. Abel, *Introduction to the Finite Element Method A Numerical Method for Engineering Analysis,* Van Nostrand Reinhold Co., New York, 1972.

11. Dessault Systemes Simulia, Inc., "ABAQUS USER MANUALS, Version 6.9," Providence, RI (2009).

12. Beng Chin Ooi, Efficient Query Processing in Geographic Information Systems, Springer-Verlag, Berlin, Germany (1987) in the Lecture Notes in Computer Science series, G. Goos and J Hartmanis, editors.

13. Jon Louis Bentley, “Multidimensional binary search trees used for associative search,” Comm. ACM 18, 9, 509-517, 1975.

14. Francis D. Murnaghan, *Finite Deformation of an Elastic Solid*, John Wiley & Sons, Inc., New York, 1951.

15. O.C. Zienkiewicz and K. Morgan, *Finite Elements and Approximations*, John Wiley and Sons, New York, 1983.

16. O. C. Zienkiewicz, *The Finite Element Method in Engineering Science*, McGraw-Hill, London, 1971.

17. Eric B. Becker, Graham F. Carey, and J. Tinsley Oden, *Finte Elements An Introduction*, Volume I, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

18. Mary P. Dolciani, Edwin F. Beckenbach, Alfred J. Donnelly, Ray C. Jurgensen, and Willaim Wooton, *Modern Introductory Analysis*, Houghton Mifflin Co, Boston, 1967.

19. Todd J. Urbatsch and Thomas M. Evans, Monte Carlo Particle Transport on a General Hexahedral Mesh, LA-UR-00-1065, February 29, 2000.