SKI Report 2008:57

Research

Discrete Feature Model (DFM) User Documentation

Joel Geier

June 2008

ISSN 1104-1374
ISRN SKI-R-08/57-SE

SKI

SKI Perspective

Background

SKI engages consultants to perform scientific and technical assessments in order to obtain a good scientific and technical basis for monitoring the Swedish nuclear fuel and waste management company's (SKB) site investigations and reviewing SKB's long term safety analyses for a spent nuclear fuel repository. The hydrogeologic conditions at the investigated candidate repository sites are a part of the site descriptive modelling and an input to the long term safety analyses. Due to the importance and complexity of the hydrogeolocical conditions and models SKI aims at establishing independent hydrogeological modelling capabilities. SKI has therefore funded Clearwater Hardrock Consulting to develop a code for discrete feature modelling (DFM) that can calculate site specific water flow and particle transport in fractured rock. This code has been and will be used in several projects. For instance was flow and particle transport modelled for the Forsmark and Laxemar sites as part of SKI's and SSI's review of SKB's long term safety analysis SR-Can (SKI report 2008:11).

Objective

The objective of this SKI report is to provide a documentation and comprehensive user manual for the DFM code developed by Clearwater Hardrock Consulting. The manual refers to the version used in the review of the SR-Can assessment.

Future work

It is planned to use the DFM code in several projects to support the review of SKB's site characterisation programme and license application for a spent nuclear fuel repository. One ongoing project, in which the DFM code is used, is an independent analysis of SKB's single well injection withdrawal tests performed at the Forsmark and Laxemar sites. Further code development may be part of the future modelling projects.

Project information

Project manager:	Georg Lindgren
Project reference:	SKI 2007/92
Project number:	200710205

SKI Report 2008:57

Research

Discrete Feature Model (DFM) User Documentation

Joel Geier
Clearwater Hardrock Consulting
Corvallis, Oregon, USA

June 2008

Table of Contents

1 Introduction

This manual describes the Discrete-Feature Model (DFM) software package for modelling groundwater flow and solute transport in networks of discrete features.

A **discrete-feature conceptual model** represents fractures and other water-conducting features around a repository as discrete conductors surrounded by a rock matrix which is usually treated as impermeable. This approximation may be valid for crystalline rocks such as granite or basalt, which have very low permeability if macroscopic fractures are excluded.

A **discrete feature** is any entity that can conduct water and permit solute transport through bedrock, and can be reasonably represented as a piecewise-planar conductor. Examples of such entities may include individual natural fractures (joints or faults), fracture zones, and disturbed-zone features around tunnels (*e.g.* blasting-induced fractures or stress-concentration induced "onion skin" fractures around underground openings).

In a more abstract sense, the effectively discontinuous nature of pathways through fractured crystalline bedrock may be idealized as discrete, equivalent transmissive features that reproduce large-scale observations, even if the details of connective paths (and unconnected domains) are not precisely known.

A discrete-feature model explicitly represents the fundamentally discontinuous and irregularly connected nature of systems of such systems, by constraining flow and transport to occur only within such features and their intersections. Pathways for flow and solute transport in this conceptualization are a consequence not just of the boundary conditions and hydrologic properties (as with continuum models), but also the irregularity of connections between conductive/transmissive features.

The DFM software package described here is an extensible code for investigating problems of flow and transport in geological (natural or human-altered) systems that can be characterized effectively in terms of discrete features.

With this software, the geometry of discrete features and their hydrologic properties are defined as a mesh composed of triangular, finite elements. Hydrologic boundary conditions are prescribed as a simulation sequence, which permits specification of conditions ranging from simple, steady-state flow to complex situations where both the magnitude and type of boundary conditions may vary over time.

The essential components of a discrete-feature model (DFM) are:

(1) the geometry of the features, which determines their interconnections,
(2) the hydrologic (hydraulic and transport) properties of the features, and
(3) the boundary conditions.

In the DFM package, a **discrete feature** is represented as a planar or piecewise-planar surface, described at each point ξ on its surface by effective 2-D parameters of transmissivity $T(\xi)$, storativity $S(\xi)$, and transport aperture $b(\xi)$. Typically these parameters are taken to be uniform over all segments of a given feature, although variable properties may also be modeled.

The boundaries of a discrete-fracture network model are in the form of polyhedra. In general the boundaries may include an external boundary, which bounds the domain to be modelled, and an arbitrary number of internal boundaries which represent tunnels, segments of boreholes, *etc*. Boundary conditions are imposed at intersections between discrete features and the external or internal boundaries.

Groundwater flow and transport through the discrete-fracture network are specified by 2-D equations that apply locally within each planar segment, by conditions of continuity which apply at the intersections between segments, and by the external and internal boundary conditions. The groundwater flow field is defined only on this network, and boundary conditions are specified only along the intersections between the network and the internal and external boundaries.

1.1 Geometrical representation of discrete features

The geometry of a discrete-feature model is represented in terms of triangular elements which approximate the (possibly curviplanar and/or tabular) geometry of the features in terms of piecewise-planar conductors. Each planar section in this representation is modelled as one or more triangular elements, often referred to simply as “elements.” The vertices of the triangular elements, which can be common to two or more elements, are referred to as "nodes." The line segments connecting pairs of nodes in a given element are often referred to as "edges" since these are the edges of the element.

Connections between different features and between adjoining planar sections of a given feature are represented by the fact that nodes (and edges) are shared in common between elements. Thus an element representing a portion of one feature, say "Fracture Zone X" may share nodes with elements representing a single, smaller-scale fracture. Hydrologic connections between different features are represented by requiring continuity of state variables (hydraulic head and/or concentration) at the nodes, or (in the case of transport modelling by the method of particle tracking) by allowing particles to move between elements belonging to different features, across shared edges.

The geometry and hydrologic connectivity of a discrete-feature model is thus represented in terms of triangular elements and the nodes which are at the vertices of (possibly one or more) elements. Edges are implicitly defined, given a list of nodal coordinates and the nodes belonging to each element. For the DFM program, this information is provided as a mesh file, the contents and format of which are described later in this manual.

Hydrologic properties are also defined in the mesh file, as properties (transmissivity, storativity, and transport aperture) assigned to each of the individual elements.

1.2 Software modules

The DFM software includes the following main modules (presented in the order in which they might be applied in a typical modelling project for repository safety assessment):

- ***fracgen****:* generates stochastic (random) realizations of a fracture population, based on a statistical description, with option to represent parts of the model domain as a regular grid of features with equivalent block-scale continuum properties;
- ***repository****:* produces discrete features to represent the excavation-disturbed zone (EDZ) around transport tunnels, deposition tunnels, and deposition holes within a repository, based on a specified tunnel layout and (optionally) conditional upon a realization of the fracture population, applying the full-perimeter intersection criterion and other criteria for deposition-hole acceptance or rejection as have been defined for the Swedish spent-fuel repository program (Munier, 2006);
- ***meshgenx****:* produces a triangular finite-element mesh for a model consisting of discrete features that represent surface topography, large-scale deformation zones, smaller-scale discrete fractures, and EDZ within a repository;
- ***dfm****:* solves steady-state and/or transient finite-element flow equations for a given mesh geometry and set of boundary conditions; and
- ***meshtrkr****:* tracks advective-dispersive or advective-diffusive motion of discrete particles through a flow field defined on a discrete-feature network, and calculates integral properties for transport paths originating from a given source location (e.g., spent-fuel canister position).

These modules are described in sequence in Chapters 2 through 6. Supplementary utilities are described in Appendix 1.

1.3 Sequence of steps in a DFM modelling application

In a typical application of the DFM package, the different modules and supplementary utilities are applied in sequence. The exact sequence may vary depending on the modelling application, but a specific example may help to illustrate how the different modules can be used together.

Table 1.1 lists the main steps for an application of the DFM package to repository safety assessment as an illustrative example. See Geier (2008) for a full description of this application. Very briefly, flow and solute transport simulations were carried out for a model of a hypothetical, spent-nuclear-fuel repository at a coastal site in Sweden. Water-conducting features that were represented in the model included deformation zones on scales of 1 to 10 km (treated as deterministic features), discrete fractures on smaller scales down to 1 m radius (treated stochastically), topography and conductive strata at the ground surface (represented deterministically as a single transmissive layer), tunnels and associated disturbed-rock zones within the repository (treated as deterministic based on a design layout), and deposition holes for waste emplacement (based on the tunnel layout, and conditioned on the stochastic fracture population). The DFM package was used to calculate groundwater flows through the model, including flows around the deposition holes, and to characterize pathways for radionuclide transport from the deposition holes, by calculating integrals of the transport properties along paths traversed by particles representing conservative (nonreactive, non-sorbing) solute.

Table 1.1 Example of the main steps in application of the DFM package to calculate groundwater flows and characterize pathways for radionuclide transport from deposition holes in a hypothetical radioactive waste repository in Sweden (see Geier, 2008 for a full description of the application).

Operation	**DFM module used**	**Refer to Section(s)**
Prepare **panel file** to define the geometry of large-scale deterministic features (deformation zones).	other[1]	Appendix 2
Prepare **panel file** to define the geometry of the external boundary of the model domain.	other[1]	Appendix 2
Prepare **domain files** to define the geometry of rock domains, within each of which the fracture population is considered to be statistically homogeneous.	other[1]	2.5
Prepare **generation sites file** to define portions of the model within which more small and/or low-transmissivity fractures should be retained during stochastic simulations.	other[1]	2.6
Prepare **shells file** to define portions of the model within which more small and/or low-transmissivity fractures should be retained during stochastic simulations.	other[1]	2.7
Prepare **grid file** to define geometry for block-scale representation of the relatively small and low-transmissivity fractures.	other[1]	2.3.1
Prepare **fracture set definition files** for each domain.	other[1]	2.4
Generate a realization of the stochastic fracture population in the rock around the repository, with smaller and lower-transmissivity fractures treated in aggregate as equivalent block-scale features.[2]	***fracgen***	2
Parse the panel data representing retained fractures and block-scale features from the fracgen output, into two separate panel files.[2]	***parsepanels***	Appendix 1
Prepare **tunnel parameters file** to specify geometric parameters of the repository tunnel system and criteria for deposition-hole placement.	other[1]	3.3
Prepare **tunnel axes file** with plan-view coordinates of access tunnels and deposition tunnels within the repository.	other[1]	3.4
Generate discrete features to represent conductive pathways due to excavation-damaged rock along the repository tunnels and deposition holes, conditioned on the realization of the stochastic fracture population.[2]	***repository***	3
Reduce size contrast among fractures prior to mesh generation.[2]	***presplit_fracs***	Appendix 1
Combine panel files for model boundary, deterministic deformation zones, stochastic fractures, block-scale features and repository elements into a single panel file.	other[1]	

Table 1.1, ctd.

Operation	**DFM module used**	**Refer to Section(s)**
Perform first stage of mesh generation to find all intersections among features (using the ***meshgen*** -s option).[2]	***meshgenx***	4
Perform second stage of mesh generation by triangulating all features.[2]	***tripostx***	4.2.1
Consolidate output from trinagulation to produce a single mesh file.[2]	***consolidate_ triangulation***	4.2.2
Prepare a ***simulation sequence file*** to define hydraulic boundary conditions and method for solving the flow equations, for a given calculation case.[3]	other[1]	5.1
Apply edits to feature hydraulic properties depending on calculation case.[2,3]	other[1]	
Solve the flow problem as defined on the mesh.[2,3]	***dfm***	5
Post-process ***dfm*** output to extract flows to deposition holes.	other[1]	
Prepare **particle file** for particle-tracking.[2]	other[1]	6.3
Prepare **dispersivity file** for particle-tracking.[3]	other[1]	6.4
Prepare **DCX mesh file** for partcle-tracking.[2]	other[1]	Appendix 5
Perform advective-dispersive particle tracking in the calculated flow fields.	***meshtrkr***	6
Postprocess output of particle-tracking to characterize transport pathways.	***extractarrivals, processarrivals, formattracks***	7

[1] Input files for DFM modules were prepared with a variety of digitization software, text editors, and special-purpose scripts.
[2] These steps were performed for each realization of the stochastic fracture population.
[3] These steps were performed for each calculation case considered (*e.g.* for different hydraulic boundary conditions).

1.4 Conventions used in this manual

Each module of the DFM software package is designed to be executed from a Unix-style command line (*i.e.*, terminal emulator), within a shell such as ***tcsh*** or ***bash*** (which are supported by Linux as well as Unix operating systems). This permits the use of shell scripts to automate steps in the modeling process.

Names of executable codes or commands mentioned within the text are given in ***bold italic serif*** font, for example ***bash***, ***fracgen***, ***dfm***, or ***grep***.

Commands to be typed on the command line (or in a shell script) are indicated by a # sign which represents the shell prompt, and are shown in `sans serif` font. For example:

```
# fracgen -h
```

runs the ***fracgen*** module (in this case, with command-line argument -h to indicate "help," so that an explanation of ***fracgen*** command-line options will be printed, as explained further in Section 2.1).

The following conventions are used in descriptions of the syntax for text that should be typed either on the command line or in an input file to one of the DFM modules:

`roman sans serif`	font indicates text that should be typed as shown.
`italic sans serif`	font indicates variable options that are defined later in the text.
`[...]`	bracket a list of optional arguments
`...|...`	separates different options
bold roman serif	font indicates specific file formats which are described later in the document (note that this font is also used to highlight key words, when used within the main text).

As is conventional for Linux/Unix shells, the standard output stream (*stdout*) from a given command may be redirected to a file by use of the > sign. For example:

```
# fracgen -h > help.log
```

will print the ***fracgen*** information to a file named *help.log*.

2 *fracgen* module for fracture statistical simulation

The ***fracgen*** module is used to generate stochastic realizations of a fracture population comprising one or more fracture sets, in one or several "generation domains." A generation domain can be a rectangular box, a right circular cylinder, or an arbitrary polyhedron. As an option, fractures in parts of the model domain can be represented as a regular grid of features with equivalent block-scale continuum properties; this can be used to reduce the numerical complexity of flow and transport simulations.

The ***fracgen*** module is run with the command-line syntax:

```
# fracgen [OPTIONS] nsides seed domains fsets_file1 domains_file1 [...]
```

where:

nsides	Number of sides for fracture polygons;
seed	Seed value for random number generator (any positive integer);
domains	Number of fracture generation domains;
fsets_file1	Name of file containing **fracture set definitions** for Fracture Set 1;
domains_file1	Name of file defining the **generation domain** for Fracture Set 1; (and optionally for domains 2 through N = *nsides*):
fsets_file2	Name of file containing **fracture set definitions** for Fracture Set 2;
domains_file2	Name of file defining the **generation domain** for Fracture Set 2;
.	
.	
.	
fsets_fileN	Name of file containing **fracture set definitions** for Fracture Set N;
domains_fileN	Name of file defining the **generation domain** for Fracture Set N;

and where *OPTIONS* may be any of the following:

`-s` *sites_file shells_file*

where:

sites_file	Name of file listing (x,y,z) coordinates for sites with higher resolution;
shells_file	Name of file defining nested shells;

`-np` No panels (suppress output of panel data); usually used with -K option;

`-lp panfile` Load deterministic/prior panel definitions from *panfile* (the name of a ***fracgen* panel definitions file**) before generating fractures;

`-Kf gridfile` Calculate **K** tensor values for 3-D grid points that are defined in a ***fracgen* grid file** named *gridfile*.

`-K dx dy dz` Calculate **K** tensor values for 3-D grid of points:

$$\{X_o + i*dx,\ Y_o + j*dy,\ Z_o + k*dz\}$$

where:

$\{X_o, Y_o, Z_o\}$ = the center of the domain specified in domains_file,

$\{i, j, k\}$ = integers (positive or negative) spanning the domain.

and where the grid spacings (*dx, dy, dz*) are in meters.

`-Ksh` Add only fractures excluded from ***fracgen* generation shells** to the **K** tensors (other fractures are retained as distinct features).

`-KT` Convert **K** tensor values to an equivalent network of orthogonal panels.

The last four options involve calculation of equivalent hydraulic conductivity (**K**) tensors from the fractures that are generated by the stochastic model (either all of the fractures or a subset, if the `-Ksh` option is invoked). Note that the `-Kf` option and the `-K` option are exclusive. Thus only one of them, not both, may be invoked in a given run. The `-Ksh` and `-KT` options are effective only when either the `-Kf` option or the `-K` option has been invoked.

Modeling Tip: The command:

```
# fracgen -h
```

will yield a reminder of the command-line syntax for ***fracgen***.

Output from ***fracgen*** is written to the stream *stdout*, and thus should be redirected to a file that saves the results, e.g.

```
# fracgen 6 1 1 test.fset test.domain > test.panels
```

Output is in the form of a **DFM panel** file which gives a list of vertex coordinates, followed by a list of indices to the vertices that form the panel for each fracture. The format of panel files is described in Appendix 2. When the ***fracgen*** option `-KT` is used to produce orthogonal features to represent the equivalent **K** tensors, the list of vertices and panels corresponding to these features is appended to the list of vertices and panels for any retained fractures.

2.1 *fracgen* fractures and fracture sets

Each individual **fracture** is idealized as a circular disc characterized by the following attributes (see Figure 2.1):

$\boldsymbol{x}_c$	**location** (3-D coordinates of the disc center),
r	**radius** of disc,
$\boldsymbol{n}$	**orientation** (unit vector normal to the plane of the disc),
T	**transmissivity**,
S	**storativity**, and
b_T	**transport aperture** (effective aperture for solute transport).

Note that the **transport aperture** b_T is not necessarily equivalent to the **hydraulic aperture** b_h which is related to the fracture's transmissivity by the **cubic law** (see Section 2.4).

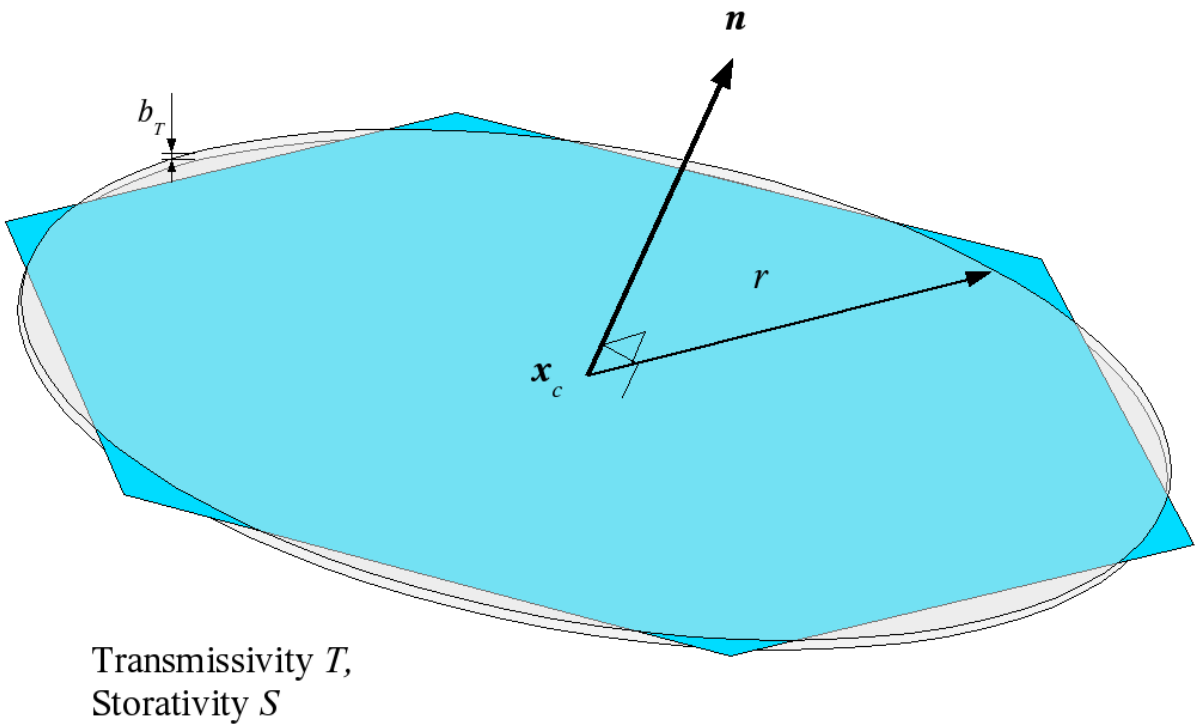


Figure 2.1 *Attributes of an idealized disc-shaped fracture and its representation as an n-sided regular polygon of equal area (in this case, n = 6).*

For practical purposes in modeling, the idealized disc-shaped fractures are represented by *n*-sided regular polygons with side lengths chosen such that each polygon has the same area as the disc that it represents (see Figure 2.1). The value $n = 6$ (yielding hexagons) is recommended as practical for most cases. Lower values of n (yielding triangles, squares or pentagons) give poorer approximations to the disc shape, while higher values of n (*e.g.* octagons or higher-order polygons) tend to increase the complexity of the mesh-discretization problem.

A **fracture set** is a portion of the fracture population which have similar properties, in a statistical sense. For a given fracture set, fracture locations are determined by a 3-D *stochastic process*, the simplest of which is the Poisson process (uniformly random location in three dimensions). Orientation is described by a *directional probability distribution* (for example, a Fisher distribution). The other attributes are described either by *univariate probability distributions* (if treated as independent attributes) or by *correlation functions* (if treated as dependent attributes). The number of fractures in a given 3-D volume is limited by an *intensity measure*. Options for stochastic processes, probability distributions, correlation functions, and intensity measures are described below.

2.2 *fracgen* thinning of fractures

A DFN submodel could potentially contain many millions of fractures if it were explicitly represented over the entire domain of the kilometer-scale model. This would lead to an intractably large network problem for numerical solution.

In order to reduce the complexity of the problem, block-scale features are used to represent the contribution of smaller-scale fractures to large-scale flow. Fractures are selectively thinned from the population, depending on their properties and their distance from areas where higher resolution is needed, such as around tunnels or sources of solute. These fractures may in some circumstances simply be ignored, or (preferably) represented in a simplified fashion by block-scale discrete features (as described in Sections 2.3).

The way that this is done is based on the concepts of ***fracgen* generation sites** and **generation shells**. A generation site may be either a point or a 3-D polygon; multiple sites of either type may be defined in a generation site file as described in Section 2.6.

A **generation shell** is the volume bounded by two surfaces, each of which is at a uniform distance from the nearest generation site (Figure 2.2). In mathematical terms, define **G** as the set of all points ***y*** belonging to the generation sites that are defined either as single points or as polygons, and for any given point ***x***, let $R(\boldsymbol{x})$ be the distance from ***x*** to the nearest point $y \in G$. Then a set of N generation shells $\{\boldsymbol{S}_1, \boldsymbol{S}_2, ..., \boldsymbol{S}_N\}$ may be defined in terms of a monotonically increasing set of values $R_0 < R_1 < R_2 < R_N$ with $R_0 = 0$, such that:

$$R_{i-1} < R(x) \leq R_i, \quad x \in S_i$$

Fractures are associated with a given shell $\boldsymbol{S}_i$ based on the distance from the fracture to the nearest generation sites. Specifically, this distance is evaluated as the minimum three-dimensional distance d_{min} from the nearest point in the set of generation sites **G** to the nearest point ***x*** on the fracture. This is illustrated in Figure 2.3, for an example in which **G** comprises a single polygonal site that outlines the deposition tunnels in a repository. A fracture is assigned to the shell $\boldsymbol{S}_i$ if $R_{i-1} < d_{min} \leq R_i$.

Each shell S_i of shell radius R_i is associated with a threshold transmissivity T_i and a threshold fracture size (disc radius) r_i as assigned in the **generation shell file** (see Section 2.7). These are used to determine which fractures should be retained explicitly, versus which should be represented in aggregate form as block-scale hydraulic conductivity.

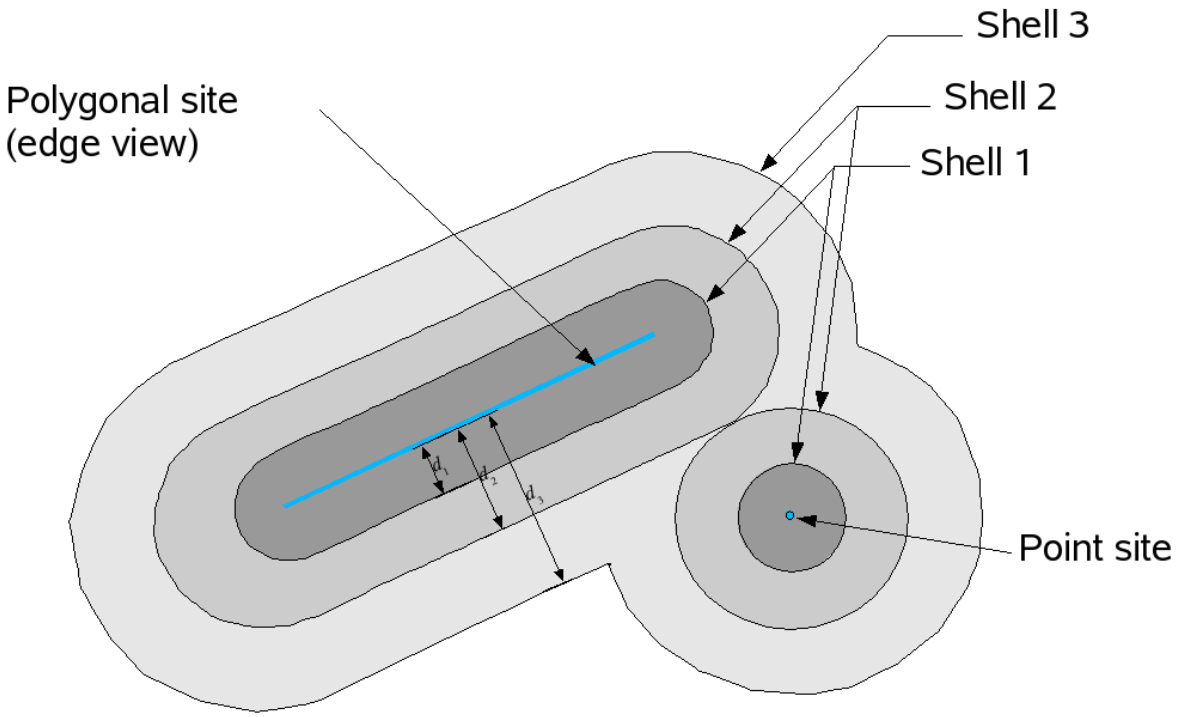


*Figure 2.2 Concept of **generation shells** as used in the fracgen module to define volumes at a given range of distances from **generation sites** (either points or polygons), as indicated in blue in the figure.*

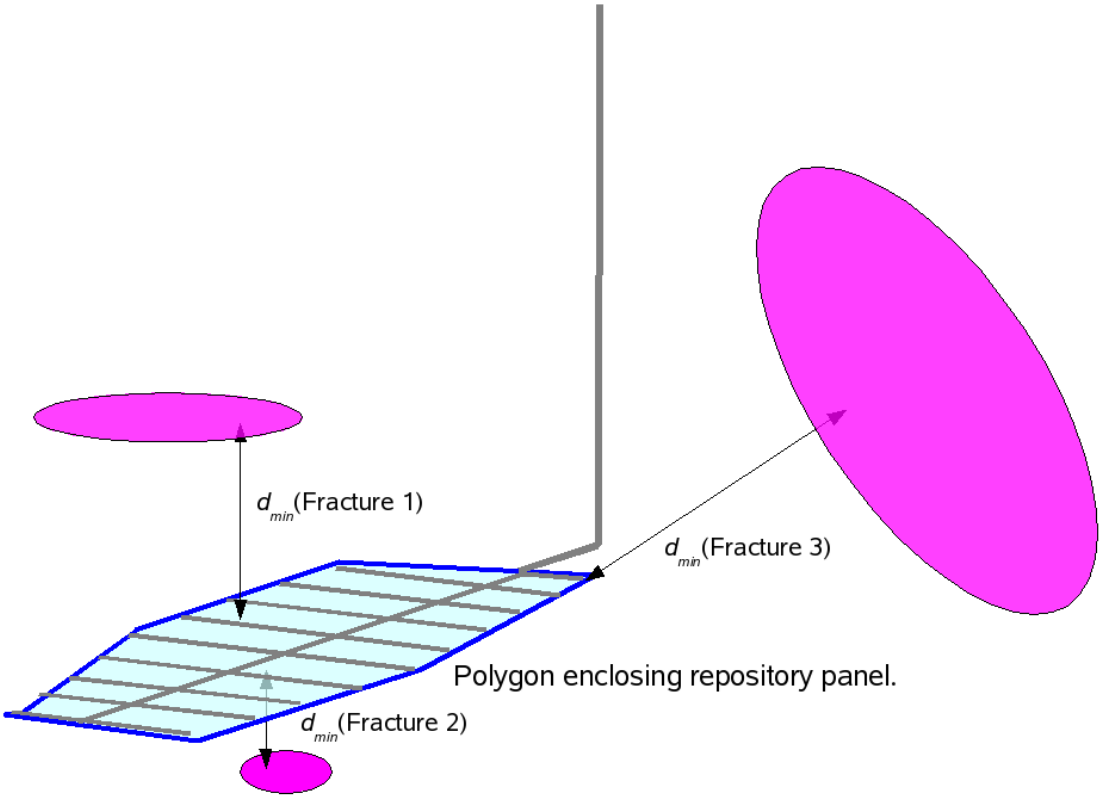


Figure 2.3 *Minimum distance d_{min} from three different fractures to a single polygonal* ***generation site*** *that outlines the deposition tunnels in a section of a repository.*

2.3 *fracgen* block-scale representation of fractures

When ***fracgen*** is run using either the `-K` or `-Kf` option, an effective 3-D hydraulic conductivity tensor **K** in calculated for each block in the grid (as defined by the arguments that follow these options), to represent the potential contribution of the fractures to large-scale flow. Effective block-scale porosity and specific storage values are also calculated.

When either the `-K` or `-Kf` option is used, by default the block-scale hydrologic properties are calculated based on all of the fractures (including both stochastically generated fractures and any deterministic features that were loaded with the `-lp` option). However, if the `-Ksh` option is used together with one of these options, the block-scale properties are calculated based only on the subset of fractures that are not retained explicitly (based on rules specified by the arguments of the `-s` option).

The following subsections describe how these grids are defined, and how the block-scale properties are calculated and (optionally) represented by a set of orthogonal discrete features.

2.3.1 Grid specification

The 3-D grid used to define the blocks can be specified as a regular grid by using the `-K` option with parameters as documented above. Alternatively, the grid may be defined by a **grid file** by using the `-Kf` option. The latter option is more generally useful as it permits variable spacing of the grid (although in either case, the grid needs to be orthogonal). A grid file consists of a series of lines, each of which lists the (x,y) coordinates of a vertical column of grid points:

$x_0\ y_0 : z_0\ z_1\ z_2 \ldots z_N$
$x_1\ y_0 : z_0\ z_1\ z_2 \ldots z_N$
:
$x_L\ y_0 : z_0\ z_1\ z_2 \ldots z_N$
$x_0\ y_1 : z_0\ z_1\ z_2 \ldots z_N$
:
$x_L\ y_1 : z_0\ z_1\ z_2 \ldots z_N$
:
$x_0\ y_M : z_0\ z_1\ z_2 \ldots z_N$
:
$x_L\ y_M : z_0\ z_1\ z_2 \ldots z_N$

where L, M, and N are the number of grid points in the x, y, and z directions, respectively.

2.3.2 Calculation of block-scale properties

The contribution of a single fracture *i* to the block-scale tensor **K** is calculated from Snow's law (Snow, 1969) which can be written in matrix form as:

$$K_i = \frac{T_i}{s_i}\left[I - n \otimes n\right]$$

where:

T_i = fracture transmissivity

s_i = effective fracture spacing

I = the identity matrix with components $I_{ii} = 1$; $I_{ij} = 0$ for $i \neq j$; $i,j = 1,2,3,.$

n = unit normal vector to fracture plane

and where $n \otimes n$ denotes the outer (tensor) product with components $n_i n_j$, for $i, j = 1, 2, 3$. The effective fracture spacing s_i is taken as V/A_i where A_i is the area of the fracture that lies within the volume V of the rock block (the entire area of the fracture, if the fracture is entirely within the rock block).

The block-scale hydraulic conductivity tensor is then approximated as the sum of the contributions of each fracture that has some portion within the block volume V:

$$K = \sum_{i \in V} K_i$$

Note that this approximation generally overestimates the block-scale hydraulic conductivity that would be obtained by an explicit block-scale DFN calculation, since not all fractures within a given volume will form part of the conductive "backbone" of the through-flowing network, and the effects of network tortuosity are neglected. The approximation is used to reduce the computational burden.

Block-scale porosity is calculated as a scalar property:

$$\Theta = \sum_{i \in V} \frac{b_i}{s_i}$$

where b_i is the effective transport aperture of the *i*th fracture. Note that this does not take into account possible directional dependence of block-scale porosity, which would require further development of the algorithms to evaluate.

Block-scale specific storage is calculated by an analogous formula:

$$S_s = \sum_{i \in V} \frac{S_i}{s_i}$$

where S_i is the storativity of the *i*th fracture.

2.3.3 Representation by block-scale discrete features

In some modeling circumstances the block-scale properties themselves are of interest, for example, for export to an equivalent-continuum model to simulate large-scale groundwater flow. In other cases it may desirable to represent these block-scale properties as large-scale, equivalent discrete features, for a multi-scale discrete-feature model. The remainder of this section concerns the case where equivalent, block-scale discrete features are required.

Each rock block can be represented in a discrete-feature model by a set of three orthogonal features, which are divided into rectangular patches (panels) with different transmissivities T_1, T_2, and T_3, as illustrated in Figure 2.4. The coordinate axes x_1, x_2, and x_3 are chosen to be parallel to the diagonal components of the hydraulic conductivity tensor K_{11}, K_{22}, and K_{33}, ordered such that $K_{11} \geq K_{22} \geq K_{33}$.

For this configuration of orthogonal features and transmissivities, it may easily be shown that the directional conductivities K_1, K_2, and K_3 (parallel to the directions x_1, x_2, and x_3, respectively) are related to the panel transmissivities T_1, T_2, and T_3 as:

$$K_1 = \frac{T_1 + T_3}{2}\left(\frac{1}{w_2} + \frac{1}{w_3}\right)$$

$$K_2 = \frac{T_2 + T_3}{2w_1} + \frac{T_1 T_3}{(T_1 + T_3) w_3}$$

$$K_3 = \frac{T_1 T_3}{(T_1 + T_3) w_1} + \frac{T_2 T_3}{(T_2 + T_3) w_1}$$

where w_i is the width of the block in the ith dimension, i=1,2,3. The transmissivities of the panels on the features are calculated by Newton-Raphson iteration to obtain $K_1 = K_{11}$, $K_2 = K_{22}$, and $K_3 = K_{33}$.

Note that the off-diagonal components of the tensor **K** are not reproduced, so this representation results in some under-representation of anisotropy and reduction of the overall conductivity. These effects are counter to the effects of the Snow's law approximation which is used to estimate K, so to some degree these are offsetting effects, but the net consequences have not been investigated.

Note also that equivalent features may influence the connectivity of the model, when these are used to represent “background” fracturing (fractures with size and/or transmissivity below some threshold values) in combination with explicit representations of the more “significant” fractures that have transmissivities and/or sizes above the threshold values. As discussed above, the use of the Snow’s law approximation for finite fractures results in an exaggeration of the component of hydraulic conductivity due to background fractures, and this conductivity is concentrated in the equivalent features.

Hence the connections between the “significant” fractures and the equivalent features representing background fractures are less broadly distributed than in the actual fracture network. These effects of the representation should be kept in mind when constructing models, just as the effects of an equivalent continuum representation also need to be kept in mind when combined with a discrete-fracture-network representation, in alternative modelling approaches.

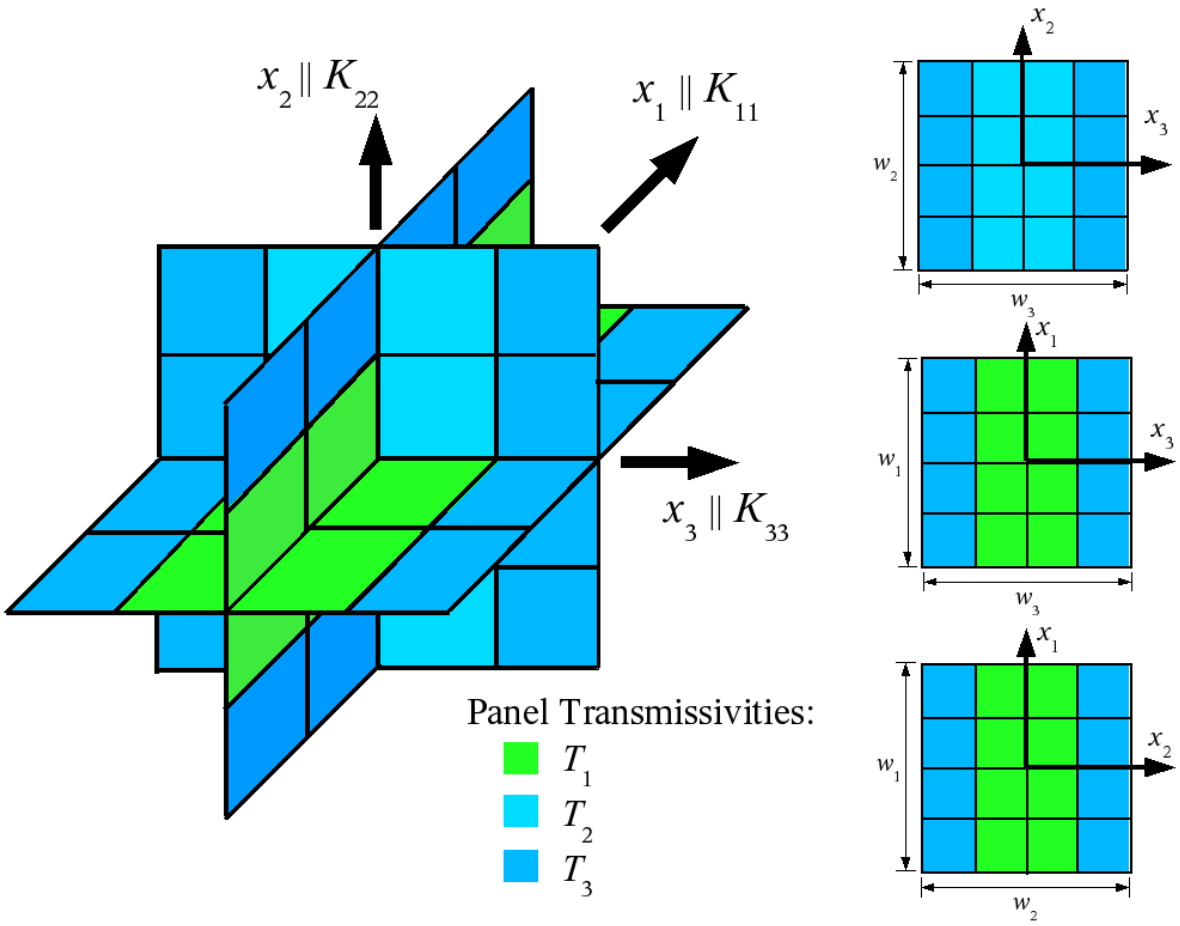


Figure 2.4 *Representation of a rock block by three orthogonal features, each comprising sixteen rectangular panels of equal dimensions, to represent block-scale hydrologic properties in the discrete-feature conceptual model. The rock block has dimensions* $w_1 \times w_2 \times w_3$*. Transmissivities assigned to the individual panels are indicated by the color of the panels, with* $T_1 \geq T_2 \geq T_3$*. Note that in this illustration the block is approximately equidimensional (*$w_1 = w_2 = w_3$*), but in general the block may be any rectangular parallelepiped with* $w_1 \neq w_2$ or $w_2 \neq w_3$.

2.4 *fracgen* fracture set definitions

A **fracture set definitions** file defines a stochastic model for each fracture set in a ***fracgen*** simulation. This type of file should list the fracture set definitions in sequence, using the following syntax for each set:

```
Set N
Transmissivity scalarmodel
Storativity scalarmodel
Aperture [scalarmodel|cubiclaw]
Radius scalarmodel
Orientation dirmodel
Location locprocess
Intensity intensity_measure
```

where the last five definitions for properties of the set can be in any order, and where:

N	=	Unique integer to identify set (1,2,3,...)
scalarmodel	=	Scalar model (*i.e.,* univariate statistical model).
dirmodel	=	Directional model (*i.e.,* vector statistical model) for the vector normal to fractures in the set.
locproc	=	[Poisson|...|help] [*parameter_list*]
intensity_measure	=	[P32|...|help] [*parameter_list*]
parameter_list	=	List of parameters (depends on type of submodel, see below).

Modeling Tip: Information on the syntax for these submodels is described in the Sections 2.4.1 through 2.4.4. A condensed version of this information can also be obtained by giving the word "help" in place of the key word for the type of submodel.

The keyword cubiclaw (one of the options that can follow the keyword Aperture) is used to impose a cubic-law relationship between transport aperture b_T (dependent variable) and transmissivity T (treated as independent variable). In this case, b_T is treated as equivalent to the hydraulic aperture b_h which is related to the transmissivity as:

$$b_h = \left(\frac{12 \mu_w T}{\rho_w g} \right)^{1/3}$$

where:

ρ_w = density of water (taken to be 1000 kg/m^3);
μ_w = viscosity of water (taken to be 0.01 kg/m•s);
g = gravitational acceleration (9.81 m^2/s).

Note that this is a special case of the loglinear form of scalar model, which is described below.

The # character is used to demarcate comments in the file which will be ignored, for example:

```
Set 1 # This is a comment and can contain any information that we want.
```

The following is an example of a **fracture set definitions** file for a fracture set with power-law distributed size (disc radius), Fisher-distributed orientation (fracture poles), and transmissivity loglinearly correlated to fracture radius (see below for further explanation):

```
Set 1   # NE
Transmissivity Loglinear r 1.791 8.55e-12
Radius         Powerlaw  3.97 5 limits 5 500
Location       Poisson
Orientation    Fisher    trend 319.9 plunge 1.3 kappa 19.7
Intensity      P32       0.0040364       # 0.04 scaled
```

2.4.1 Scalar models

A scalar model is either a loglinear correlation model or a scalar probability distribution.

Loglinear correlations

A loglinear correlation specifies a loglinear correlation $y = f(x)$ between an independent variable x and a dependent variable y:

$$\log_{10} y \;=\; \mu_{\log y} \;+\; m \log_{10} x \;+\; \sigma_{\log y} N(0,1)$$

or equivalently:

$$y \;=\; x^m 10^{\mu_{\log y} + \sigma_{\log y} N(0,1)}$$

where $N(0,1)$ signifies the standardized normal distribution with zero mean and unit standard deviation. The parameter $\sigma_{\log y}$ signifies the standard deviation of the "random noise" with respect to a log-linear correlation, if values of the dependent variable y are plotted vs. the independent variable x on a log-log graph. If $\sigma_{\log y} = 0$ the loglinear correlation is perfect (i.e., no random "noise").

The syntax used to specify a loglinear model is:

```
loglinear x ybar m ysdev
```

where:

`x` = the name of the independent variable:
- `T` for transmissivity,
- `r` for fracture radius, or
- `b` for fracture transport aperture;

`ybar` = $\mu_{\log y}$ is the mean value of $\log_{10} y$ for $x = 0$;

m = the log-slope of the correlation;

`ysdev` = $\sigma_{\log y}$ is the standard deviation of the random "noise" on a log-log plot.

Currently the choice of independent variable is not flexible. The following are allowed:

Transmissivity $T = f(r)$
Storativity $S = f(T)$
Aperture $b_T = f(T)$

Scalar probability distributions

A scalar probability distribution used as a scalar model may take any of the following forms:

- Constant $y = \bar{y}$
- Uniform $U(y_{min}, y_{max})$: $p(y) = \dfrac{1}{y_{max} - y_{min}}$; $y_{min} \le y \le y_{max}$
- Exponential $E(\bar{y})$: $p(y) = \dfrac{1}{\bar{y}} e^{y/y}$
- Normal $N(\bar{y}, \sigma_y)$: $p(y) = \dfrac{1}{\sqrt{2\pi\sigma_y^2}} e^{-(y-\bar{y})^2/2\sigma_y^2}$
- Lognormal $L(\mu_{\log y}, \sigma_{\log y})$: $y = 10^x$
  where x is normally distributed as $N(\mu_{\log y}, \sigma_{\log y})$
- Power-law $P(y_{min}, b_y)$: $p(y) = Cy^{-b_y}$, $y \ge y_{min}$
  where C is a normalization constant.

Any of these except the constant or uniform model can be modified by imposing limits (y_{min}, y_{max}) to give a truncated distribution:

$$p_{trunc}(y) = \begin{cases} \dfrac{p(y)}{C_{trunc}}; & y_{min} \le y \le y_{max} \\ 0 \quad \text{otherwise} & \end{cases}$$

where:

$$C_{trunc} = \int_{y_{min}}^{y_{max}} p(y)\,dy$$

The syntax for specifying one of these scalar models is (respectively):

`Constant`	*`ybar`*
`Uniform`	*`ymin ymax`*
`Normal`	*`ybar ysdev`* `[limits` *`ymin ymax`*`]`
`Lognormal`	*`ylogbar ylogsdev`* `[limits` *`ymin ymax`*`]`
`Exponential`	*`ybar`* `[limits` *`ymin ymax`*`]`
`Powerlaw`	*`yexp ymin`* `[limits` *`ymin ymax`*`]`

where:

`ybar` = $\bar{y}$
`ysdev` = σ_y
`ylogbar` = $\mu_{\log y}$

`ylogsdev` = $\sigma_{\log y}$

`yexp` = b_y

as used in the foregoing mathematical definitions of the scalar distributions.

Note: If a truncated power-law distribution is specified, the first instance of y_{min} is ignored.

2.4.2 Directional models

A directional model is based on a directional probability distribution (*i.e.* a vector distribution constrained to the unit sphere). Expressed in terms of spherical polar coordinates (θ, ϕ) (see Figure 2.5), the directional probability distribution may take any of the following forms:

- Constant $\theta = \bar{\theta}; \phi = \bar{\phi}$
- Uniform $p(\theta, \phi) = \frac{1}{4\pi}$
- Fisher $p(\omega, \psi; \kappa, \bar{\theta}, \bar{\phi}) = \frac{\kappa}{2\sinh\kappa} \sin\omega\, e^{\kappa\cos\omega}$

  where ω is the polar angle of the direction vector measured from the mean direction $(\bar{\theta}, \bar{\phi})$ and ψ is a uniformly random angular rotation from 0 to 2π about an axis through the mean direction $(\bar{\theta}, \bar{\phi})$.
- Bingham $p(\omega, \psi; \kappa, \bar{\theta}, \bar{\phi}) = \frac{1}{4\pi\, d(\kappa_1, \kappa_2)} \sin\omega\, e^{(\kappa_1\cos^2\psi + \kappa_2\sin^2\psi)\sin^2\omega}$

  where (ω,ψ) are angles measured relative to a reference direction $(\bar{\theta}, \bar{\phi})$ as above and the normalizing factor is given by:

  $$d(\kappa_1, \kappa_2) = \frac{1}{\sqrt{4\pi}} \sum_{i,j=0}^{\infty} \frac{\Gamma(i+1/2)\Gamma(j+1/2)}{\Gamma(i+j+3/2)} \frac{\kappa_1^i \kappa_2^j}{i!\, j!}$$
- Bootstrap $(\theta, \phi) =$ random sample from $\{(\theta_1, \phi_1), (\theta_2, \phi_2), \ldots, (\theta_N, \phi_N)\}$

The Fisher and Bingham distributions are further explained by Mardia (1972) and by Mardia et al. (1979).

The syntax for specifying a directional model is:

```
[discrete] model
```

where the optional keyword `discrete` is used to specify a discrete distribution (see below) and *`model`* may be any of the following (corresponding to the directional distributions defined above):

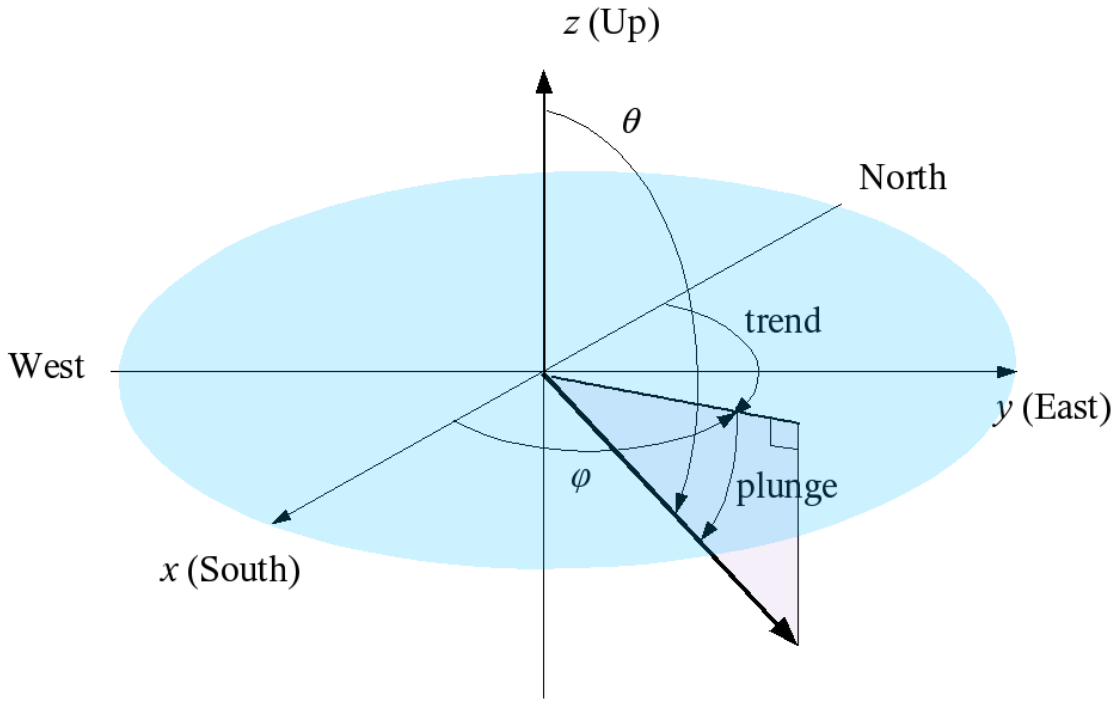
```
Constant      trend mean_tr plunge mean_pl

Uniform

Fisher        trend mean_tr plunge mean_pl kappa k

Bingham       trend mean_tr plunge mean_pl kappa k1 k2

Bootstrap     bootstrap_file
```

where:

mean_tr = trend of mean direction (in degrees)
mean_pl = plunge of mean direction (in degrees)
k = κ (Fisher concentration parameter)
k1 = κ_1 (Bingham distribution parameter)
k2 = κ_2 (Bingham distribution parameter)

as used in the foregoing mathematical definitions of the directional distributions. For the Constant, Fisher, and Bingham cases, the order of the phrases `trend` *`(value)`*`, plunge` *`(value)`*, *and* `kappa` *`(value(s))`* is arbitrary.

If the optional keyword **`discrete`** is prepended to a directional distribution, the simulated directions are constrained to a discrete, icosahedral set of 20 directions which are uniformly spaced on the unit sphere. This can help to constrain the range of angles at which intersections among fractures occur, and thus lead to better-conditioned meshes for flow and transport calculations, although at the cost of resolution of the fracture orientation distribution.



2.4.3 Location processes

A location process is a statistical process by which points (generally fracture centers, in the context of ***fracgen***) are generated within the region of 3-D space delimited by a generation domain. Two options are supported:

- Poisson process
- Lévy process

Poisson process

A 3-D isotropic Poisson process generates points that are uniformly random in three dimensions, with no correlation between successive points. For the simplest case of a domain defined as a rectangular box aligned with the coordinate axes:

$$x_{min} \le x \le x_{max}$$
$$y_{min} \le y \le y_{max}$$
$$z_{min} \le z \le z_{max}$$

each point $\boldsymbol{x}_i$ generated by a Poisson process has coordinates (x_i, y_i, z_i) drawn from independent, uniform distributions on these intervals:

$$x \sim \mathrm{U}(x_{min}, x_{max})$$
$$y \sim \mathrm{U}(y_{min}, y_{max})$$
$$z \sim \mathrm{U}(z_{min}, z_{max})$$

For a domain of arbitrary shape, points are in effect generated from the same 3-D distribution as above, but restricted to the volume of the domain.

The syntax used to specify a Poisson process is simply the keyword:

```
Poisson
```

since no additional parameters are needed to define an isotropic Poisson process (anisotropic Poisson processes are not supported in the present version of ***fracgen***).

Lévy process

A 3-D Lévy process (also known as Lévy-Lee process by correspondence to the Lévy-Lee model, as implemented in the FracMan code described by Dershowitz *et al.*, 1996) generates a sequence of points as a type of random walk in 3-D space. For the case of an isotropic Lévy process (the anisotropic case is not supported in this version of ***fracgen***), each step in the random walk has a random direction (*i.e.*, uniformly distributed on the unit sphere), and the probability of a step of length l decreases with l as:

$$P[l] \quad \propto \quad l^{-D_L}$$

where:

D_L = fractal dimension of the Lévy process.

The syntax used to specify a Lévy process is:

`Levy-Lee D` D_L `lambda` λ

where:

D_L = the fractal dimension.

λ = scale factor.

For a 3-D Lévy process, the fractal dimension should be in the range $0 < D_L \leq 3$. A value of 3 results in a distribution of points equivalent to a Poisson process. Values closer to zero imply stronger clustering. Values outside this range may lead to unpredictable results.

2.4.4 Intensity measures

Intensity measures are used to limit the number of fractures that are generated for a particular fracture set and generation domain. Fractures are generated one at a time until the intensity measure reaches or exceeds the specified value.

Two types of intensity measure are supported:

- Number of fractures in the generation region.
- Total area of fractures per unit volume,
- To use the first type of measure, the syntax is simply:

`N` *n*

where:

n = the number of fractures to simulate.

To use the second type of measure, the syntax is:

`P32` P_{32} `[unscaled]`

where:

P_{32} = threshold value for total area of fractures per unit volume,

following the nomenclature of Dershowitz (1985). The optional keyword `unscaled` is used to indicate if this measure should be scaled to compensate for truncation of the fracture size and/or transmissivity distribution..

The measure is calculated by ***fracgen*** as:

$$P_{32} = \frac{\sum_{i=1}^{n} A_i}{V_\Omega}$$

where:

n = the number of fractures,

V_Ω = the volume of the generation domain Ω,

A_i = the area of the part of the *i*th fracture that is inside Ω.

For example, if the *i*th fracture is a square fracture 1 m on a side, entirely inside the domain Ω, the area used in this calculation is $A_i = 1$ m^2. If only half of the fracture is inside Ω, the area would be $A_i = 0.5$ m^2.

Note that the definition of fracture area used for intensity calculations is different from the definition of fracture surface area as used for transport calculations. The fracture surface area for transport is potentially twice as much, since both faces of the fracture are counted.

2.5 *fracgen* generation domains

This file defines a generation domain for the fracture population using the syntax:

```
Domain [gendomain [parameters]|help]
```

where *gendomain* is one of the following:

```
Box
Cylinder
Polyhedron
help
```

and where the list of parameters depends on the type of generation domain, as described below. The types of generation domains are illustrated in Figure 2.6.

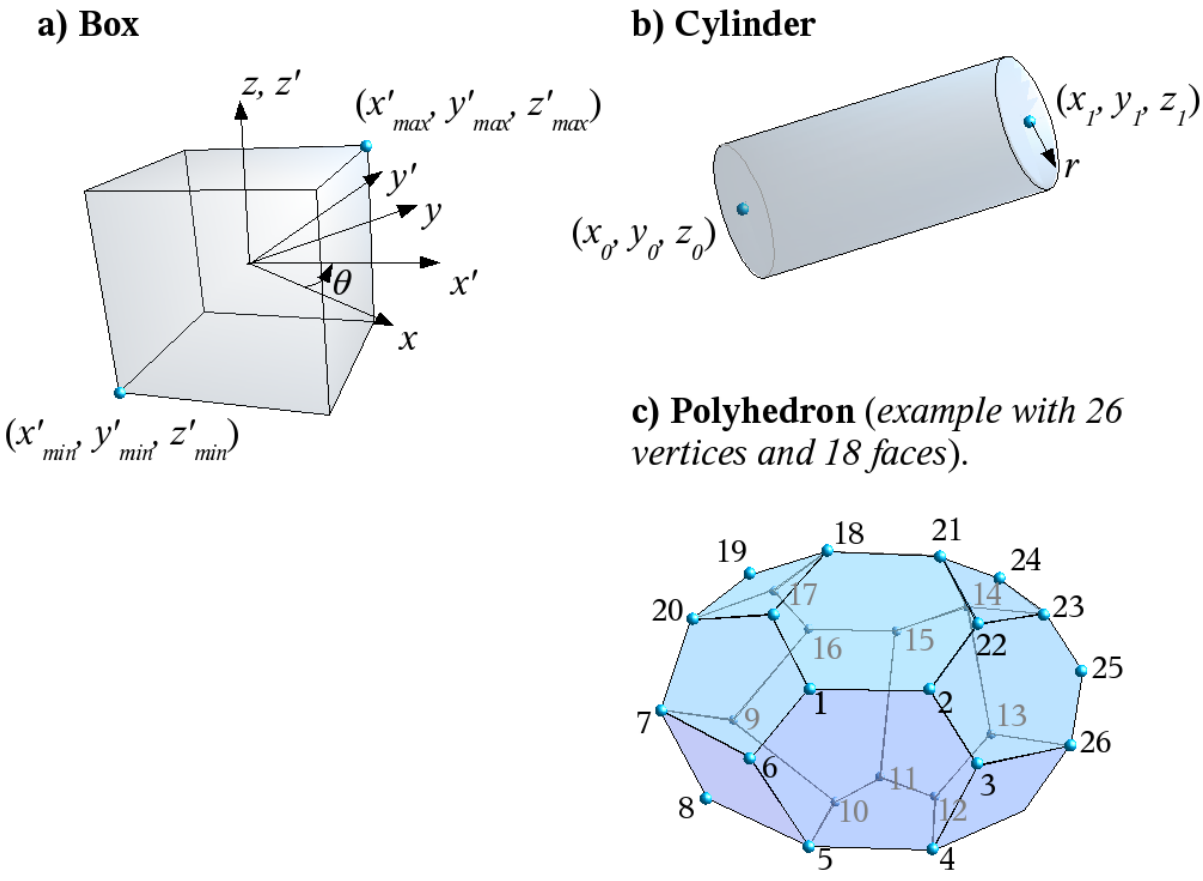


Figure 2.6 Types of fracgen generation domains: a) box; b) cylinder; c) polyhedron.

If multiple domains are listed in sequence, the domains are treated together as subdomains of a compound domain, with identical fracture statistics in each volume. Note that ***fracgen*** does not check for overlaps between subdomains, so this must be done by the modeler.

Modeling Tip: The syntax for generation regions can be obtained by giving the word "help" in place of the key word for the type of region.

Box domains

A box domain is defined by minimum and maximum coordinates in three dimensions, and a rotation angle which (if non-zero), specifies a rotation of the box in the horizontal plane. The syntax is:

`Domain Box` x_{min} y_{min} z_{min} x_{max} y_{max} z_{max} *theta*

where:

theta	rotation angle θ from reference coordinates (x,y,z) to a rotated coordinate system (x',y',z').
$(x_{min},\ y_{min},\ z_{min})$	minimum coordinates $(x'_{min},y'_{min},z'_{min})$ in the rotated system.
$(x_{max},\ y_{max},\ z_{max})$	maximum coordinates $(x'_{max},y'_{max},z'_{max})$ in the rotated system.

If $\theta = 0$ the box is aligned with the reference coordinates (x,y,z). For non-zero θ, the box is rotated around the z-axis using the transformation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} cos\,\theta & -sin\,\theta & 0 \\ sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Modeling Tip: It is generally easier and more transparent to use polyhedral domains to model box-shaped domains that are not aligned with the reference coordinates. The possibility for a box domains with rotation angle $\theta \neq 0$ is a ***fracgen*** feature that was developed before polyhedral domains were implemented. This feature is potentially confusing to use, since the rotation is about the origin in the (x,y) plane which might be well outside the modeling area. Furthermore, in some cases where the box is far from the origin, round-off errors in box corner coordinates may result if not enough digits are specified on input. When a polyhedral domain is used to specify a box-shaped generation region, the coordinates of the corners can be input directly.

Cylindrical domains

A cylindrical domain (properly a right, circular cylinder) is defined by the endpoints of the cylinder and the radius of the cylinder. The syntax is:

Domain Cylinder *radius* x_0 y_0 z_0 x_1 y_1 z_1

where:

radius	radius of the cylinder;
(x_0,y_0,z_0)	coordinates of one end of the cylinder's axis;
(x_1,y_1,z_1)	coordinates of the other end of the cylinder's axis.

Polyhedral domains

A polyhedral domain is defined by a series of *V* vertices and *F* faces with the following syntax:

Domain Polyhedron V F
1 x_1 y_1 z_1
...
V x_V y_V z_V
$face_1$
$face_2$
...
$face_F$

Each *face* of the polyhedron is defined as a list starting with the number of vertices on the face (this number must be 3 or more), followed by the vertex indices, on a single line of the file. The vertex indices should appear in clockwise order if viewed from inside the polyhedron.

Examples of polyhedral domains for the cases of a simple cube and a tetrahedron are given in Tables 2.1 and 2.2, respectively.

Table 2.1 Example of the syntax for a polyhedral domain in the shape of a simple cube.

```
Domain Polyhedron 8 6  # Simple cube with faces toward NW, NE, SE, and SW
1     150     50   -200
2     200    100   -200
3     150    150   -200
4     100    100   -200
5     150     50   -129
6     200    100   -129
7     150    150   -129
8     100    100   -129
4 4 3 2 1   # Bottom
4 2 3 7 6   # Southeast face (x axis is toward east; y axis toward north)
4 3 4 8 7   # Northeast face
4 4 1 5 8   # Northwest face
4 1 2 6 5   # Southwest face
4 5 6 7 8   # Top
```

Table 2.2 Example of the syntax for a polyhedral domain in the shape of a tetrahedron.

```
Domain Polyhedron 4 4  # Tetrahedron
1      50     50   -200
2     250     50   -200
3     150    150   -200
4     150    100   -200
3 3 2 1  # Base
3 1 2 4  # South face (x axis is toward east; y axis toward north)
3 3 1 4  # Northwest face
3 2 3 4  # Northeast face
```

2.6 *fracgen* generation sites

This file defines **generation sites** (used with **generation shells**) for reduced level of detail in nested models. Each line in the file defines one site, which may be either a single point or a polygon.

For a point site, the syntax is:

ID x y z

where:

ID = Integer to identify the site.
(x,y,z) = 3-D coordinates of the point (assumed to be in meters).

For a polygonal site, the syntax is:

ID x_1 y_1 z_1 x_2 y_2 z_2 ... x_n y_n z_n

where:

ID = Integer to identify the site.
(x_i,y_i,z_i) = 3-D coordinates of the *i*th vertex of the polygon (assumed to be in meters).

In the current version of ***fracgen***, points and polygons are distinguished only by the number of parameters on each line. Comments are not supported.

Example

The following lines define two sites. Site 1 is a triangle at 500 m depth; Site 2 is a single point:

```
1 9204.4 5604.9 -500 9618.5 5994.7 -500 9829.7 6417.0 -500 9521.1 7017.9
2 7409.7 6205.9 -500
```

2.7 *fracgen* generation shells

This file defines shells for reduced level of detail in nested models, using the syntax:

```
Shell 1 shellrad rmin Tmin
  .
  .
  .
Shell N shellrad rmin Tmin
```

where:

shellrad = shell radius (radius within which rules for this shell are applied),
rmin = minimum fracture radius (smaller fractures will be discarded from the given shell),
Tmin = minimum fracture transmissivity (tighter fractures will be discarded from the given shell).

Note shells must be in decreasing size, *i.e.* *shellrad*[1] > ... > *shellrad*[N]

A fracture is considered to be within the *i*th "shell radius" R_i if any part of the fracture is within distance R_i of one of the sites defined in the *sites* file. See Section 2.2 for a mathematical definition of shell radii.

Modelling Tip: Due to a quirk in the current version of fracgen, fractures that are entirely outside of the largest-radius shell (Shell N) are considered to belong to the "null" shell, and are not discarded regardless of fracture radius or transmissivity. This can be avoided by making the Nth shell large enough to encompass the entire modeling region.

3 *repository* module for simulation of tunnels

The DFM module ***repository*** produces discrete features to represent the transmissive excavation-disturbed zone (EDZ) around transport tunnels, deposition tunnels, and deposition holes within a KBS-3 type repository. This is done based on a specified tunnel layout, and conditional upon a realization of the fracture population, by applying geologic and/or hydrogeologic criteria for deposition-hole acceptance or rejection.

The command-line syntax for ***repository*** is:

```
# repository [OPTIONS] parfname tunfname nsides tol seed
```

where:

parfname	Tunnel parameters file;
tunfname	Tunnel axes file;
tol	Tolerance for checking if two lines are colinear (in meters);
seed	Seed value for random number generator;

and where *OPTIONS* may be any of the following:

`-lp` *panfile*	Load deterministic/prior panel definitions from *panfile* (in **DFM panel file** format) before generating fractures;
`-h`	Print this message with additional info on file formats.

Output is printed to *stdout,* in the form of a **DFM panel** file. The format of panel files is described in Appendix 2.

Modeling Tip: Typing `repository -h` on the command line will yield a summary of the syntax.

The current version of ***repository*** only handles tunnels that are within the repository's deposition horizon, so all must have the same elevation for tunnel floor. Hence features to represent vertical shafts or inclined access ramps need to be developed by other means (*e.g.* piecing together the coordinates by hand or with commercial computer-aided design software).

3.1 *repository* tunnel EDZ

The hydraulic conductivity of backfilled tunnels and the transmissive excavation-damaged zone (EDZ) in the wall rock along repository tunnels are represented by a group of transmissive features configured as a tube of rectangular cross-section – one feature for each of the four sides of the tube – along the length of each tunnel segment (Figure 3.1). Repository access tunnels (main tunnels and transport tunnels) as well as deposition tunnels are represented in this fashion.

These tubes are slightly larger than the actual tunnels (by 1 m on each side), to account for the extent of the excavation-disturbed zone (EDZ) into the wall rock. Transmissivity and aperture values are assigned to the features on each side of the tube, such that the total conductance and porosity, respectively, of the tunnel cross section are reproduced.

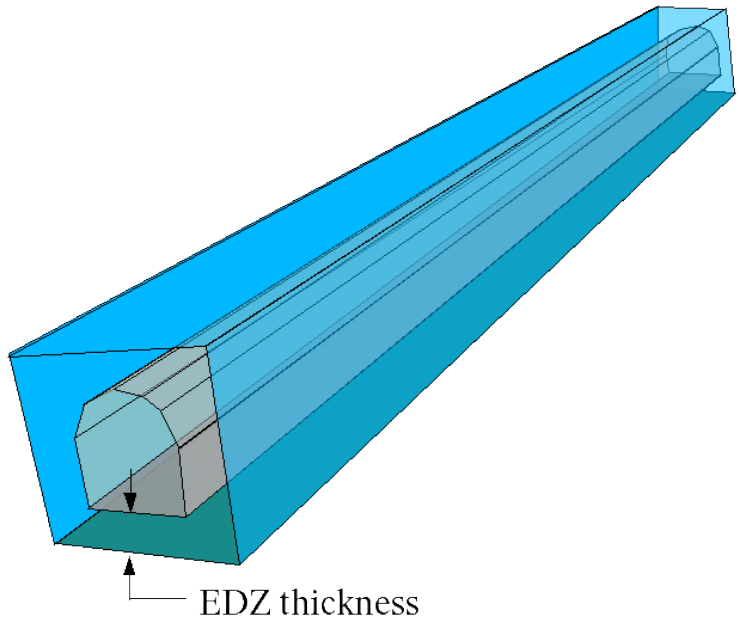


Figure 3.1 *Illustration of discrete-feature representation of tunnel in terms of a rectangular tube of features representing the excavation-disturbed zone (EDZ) around the tunnels.*

3.2 *repository* deposition-hole criteria

Canister positions along the deposition tunnels are chosen for each realization of the discrete-fracture network, according to the full-perimeter intersection criterion (FPC) as described in the SR-Can Main Report (SKB, 2006) and by Munier (2006). This is done with the program ***repository*** which is part of the ***dfm*** toolkit.

For each deposition tunnel, full-perimeter intersections (FPIs) are identified as the simulated fractures that cross all surfaces (top, bottom, and sides) of the tunnel. Deposition hole positions are then chosen sequentially by the following procedure, avoiding positions in which the canister would be intersected by an FPI fracture:

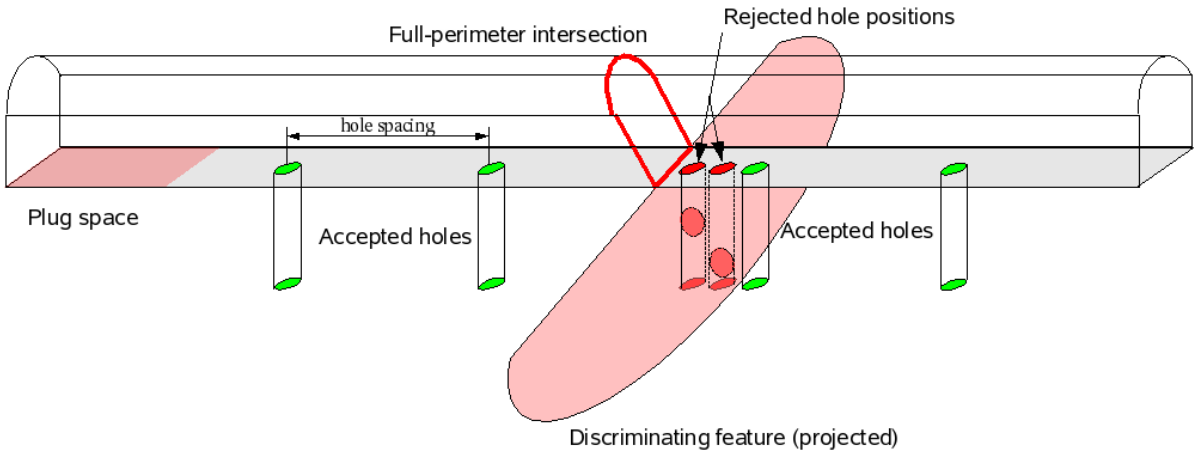
Starting from the entrance of the deposition tunnel, the first part of length l_{plug} is avoided (see Figure 3.2) in order to allow room for a sealing plug, such as specified in repository designs for the Swedish repository programme (Janson *et al.*, 2006; Brantberger *et al.*, 2006).

A trial position is tested to see if:

- It meets respect-distance criteria for any nearby deterministic deformation zones,
- It meets the FPC criterion (i.e., no intersections with a FPI fracture) and,
- (optionally) the total transmissivity of fractures intersected by a pilot hole would be less than the allowable transmissivity.

If the trial position is acceptable, a deposition hole is created at the position and a new trial position is chosen a distance $l_{spacing}$ further along the tunnel, where $l_{spacing}$ is the design spacing between canisters, based on thermal criteria.

If the trial position is rejected, a new trial position is chosen by advancing a small distance l_{step} along the tunnel and repeating the tests, until an acceptable position is found.


Full-perimeter intersection
Rejected hole positions
hole spacing
Plug space
Accepted holes
Accepted holes
Discriminating feature (projected)

3.3 *repository* tunnel parameters files

A ***repository* tunnel parameters file** consists of a series of lines with the basic format:

```
parameter_name value(s) [units]
```

where:

parameter_name	is the name of the parameter (see Table 3.1);
value(s)	is the value to be assigned to the parameter (or values in some cases as specified below);
units	is an optional string to indicate the units of the assigned value (see note below).

Lines beginning with the character # are ignored and thus can be used as "comment lines" to document the choices of parameters.

Note: The current version of ***repository*** does not make use of the *units* specifier. Values of parameters therefore need to be given in SI units as indicated in the list of parameters in Table 3.1. Units may be indicated for the sake of documentation, but the user should be aware that these currently have no effect.

The deposition holes for accepted positions are represented by vertical, internal boundaries of polygonal (usually hexagonal) cross-section, starting from the floor of the tunnel and extending to the depth specified in the design.

Table 3.1 List of repository parameters used in tunnel parameters files.

Parameter name	Number of values	SI units	Definition of parameter
`Scale`	1	m	Scaling factor s for tunnel axes.
`Origin`	2	m	Location of origin (x_0, y_0) for scaling tunnel axis coordinates on input, using formulae: $x' = s(x - x_0)$ $y' = s(y - y_0)$ Scaling of tunnel axis coordinates can be convenient for combining input files based on different coordinate systems (for example if the tunnel coordinates are given in a regional system in kilometers while the rest of the problem is formulated in terms of a local system in meters).
Tunnel parameters:			
`Tunnel sides`	1	-	Number of sides for tunnels. Specify a value of 4 for a rectangular tunnel cross-section. (***Note:*** In the current version of ***repository***, algorithms have not been tested for other numbers of sides). Deposition tunnels and access tunnels are required to have the same number of sides.
`Tunnel floor`	1	m	Vertical coordinate z_{floor} of the floor of the tunnels (typically the elevation above sea level or other regional datum, with z negative for tunnels that are below sea level).
`Deposition tunnel height`	1	m	Height H_d of each deposition tunnel (see Figure 3.3).
`Deposition tunnel width`	1	m	Width W_d of each deposition tunnel (see Figure 3.3).

Parameter name	Number of values	SI units	Definition of parameter
`Deposition tunnel spacing`	1	m	Spacing S_d between deposition tunnels (see Figure 3.3). Note this has no effect in the present version of ***repository***.
`Access tunnel height`	1	m	Height H_a of each access tunnel (see Figure 3.3).
`Access tunnel width`	1	m	Width W_a of each access tunnel (see Figure 3.3).
`DRZ thickness`	1	m	Thickness W_{EDZ} of disturbed-rock zone (DRZ , also called EDZ) around deposition and access tunnels (see Figure 3.4).
`DRZ transmissivity`	1	m^2/s	Transmissivity of DRZ (EDZ) features around tunnels.
`DRZ storativity`	1	-	Storativity of DRZ (EDZ) features around tunnels.
`DRZ aperture`	1	m	Effective transport aperture of DRZ (EDZ) features around tunnels.
Deposition hole parameters:			
`Deposition hole sides`	1	-	Number of sides on deposition holes (a value of 6, for hexagonal deposition holes, gives a reasonable approximation to a circular hole for most purposes).
`Deposition hole radius`	1	m	Radius r_{dep} of deposition hole ((see Figure 3.4).
`Deposition hole depth`	1	m	Depth L_{dep} of each deposition hole (see Figure 3.4).
`Canister radius`	1	m	Radius r_{can} of waste-containment canister (see Figure 3.4).
`Canister length`	1	m	Length L_{can} of waste-containment canister (see Figure 3.4).

Parameter name	Number of values	SI units	Definition of parameter
`Canister top`	1	m	Distance H_{can} from floor of deposition tunnels to top of waste-containment canister (see Figure 3.4).
Deposition hole criteria:			
`Utilization fraction`	1	-	Fraction of deposition hole sites in the layout that should be utilized. This is set to a value less than one if a simplified, random rejection criterion is to be used; if set to one, more sophisticated criteria are used instead.
`Pilot hole diameter`	1	m	Diameter of pilot hole for trial deposition holes (not utilized in current version).
`Pilot hole transmissivity`	1	m^2/s	Maximum total transmissivity of fractures that intersect a trial deposition hole position. Position will be rejected if this value is exceeded.
`Pilot hole seepage`	1	m^3/s	Maximum allowable rate of seepage into a pilot hole for a deposition hole (not utilized in current version).
`Pilot hole bradius`	1	m	Effective constant-pressure boundary radius for calculating flows to pilot holes (not utilized in current version).
`Pilot hole bhead`	1	m	Assumed head at boundary radius for calculating flows to pilot holes (not utilized in current version).
`Distance between holes`	1	m	Minimum allowable distance between deposition holes (Note: The current version of ***repository*** supports just a single value; future versions may allow for adapting this distance to the thermal properties of the rock type).

Parameter name	Number of values	SI units	Definition of parameter
`Distance from drift end`	1	m	Minimum allowable distance from the edge of a deposition hole to the blind end of a deposition tunnel (drift).
`Distance from drift start`	1	m	Minimum allowable distance l_{plug} from the edge of a deposition hole to where a deposition tunnel (drift) starts at the access tunnel.
`Minimum step distance`	1	m	Distance l_{step} to move between trial positions for a deposition hole, if a given trial position is found to be unsuitable.
`Maximum intersected holes`	1	-	Maximum number of deposition holes that can be intersected by a feature, without rejecting those deposition holes according to the extended full-perimeter intersection criterion (EFPC). This parameter is not used in the present version of ***repository***, so it has no effect.

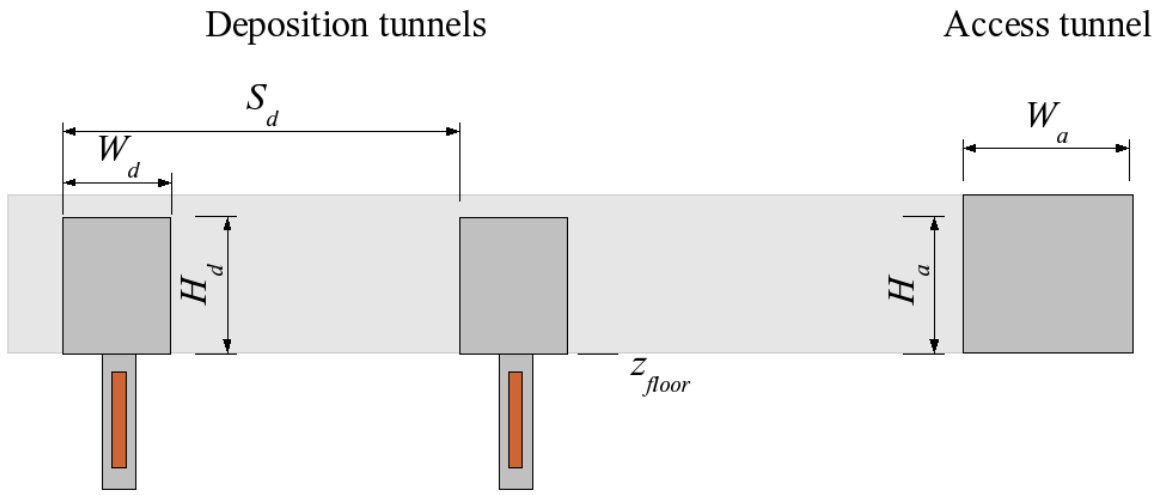


Figure 3.3 Definition of tunnel layout parameters for deposition and access tunnels. Refer to Table 3.1 for explanation of parameters.

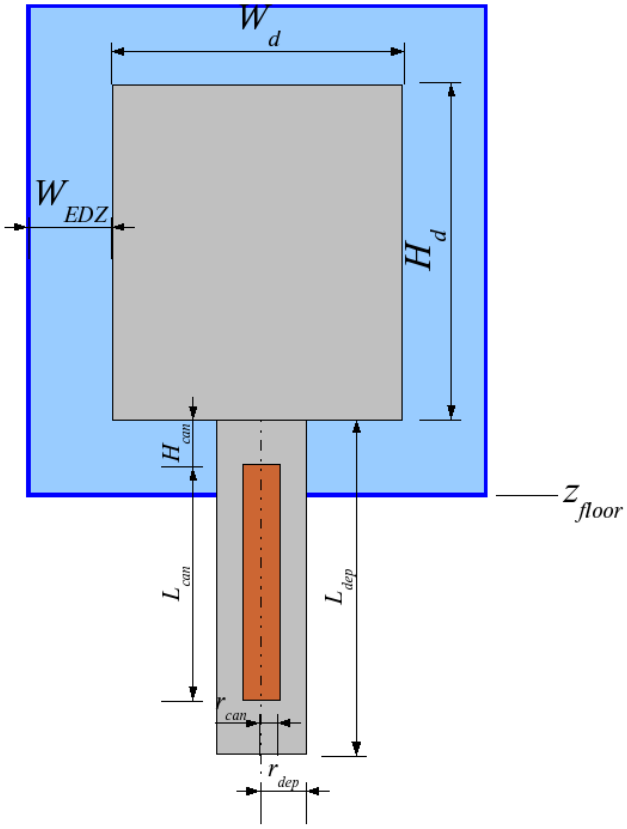


Figure 3.4 Definition of parameters for canister, deposition-hole and excavation-disturbed-zone (EDZ) geometry in a KBS-3 type repository. Refer to Table 3.1 for explanation of parameters.

Table 3.2 Example of tunnel parameters file.

```
# Tunnel system parameters:
Scale 1 m
Origin 0 0
Tunnel sides 4
Tunnel floor -500 m
Deposition tunnel height 4.2 m
Deposition tunnel width  3.5 m
Deposition tunnel spacing 40 m
Access tunnel height 5 m
Access tunnel width  5 m
```

3.4 *repository* tunnel axes files

Tunnel axes are specified in terms of the plan-view coordinates of their endpoints, using a DFM-DXF file format. See Appendix 3 for an explanation of this format.

A tunnel axes file consists of two sections: The first section contains DXF header information (as explained in the appendix) which is simply scanned over by ***repository***. This header information may be useful for other programs (*e.g.* plotting software) but is not used by *repository*. The second section gives the tunnel axis coordinates, as a series of labeled polylines.

An **access tunnel** is introduced by a label statement of the form:

`LABEL Access Tunnel [`*name*`]`

where *name* is an optional text string that describes the deposition tunnel. This is followed by a polyline:

```
BEGIN POLYLINE
 x0 y0
 x1 y1
 ...
 xN yN
END POLYLINE
```

where N is the number of segments in this access tunnel, and $\{(x_0, y_0), (x_1, y_1),...(x_N, y_N)\}$ are the coordinates of successive vertices (segment endpoints).

A **deposition tunnel** (or deposition drift) is introduced by a label statement of the form:

`LABEL Deposition Tunnel [`*name*`]`

where *name* is an optional text string. This is followed by a polyline with just one segment:

```
BEGIN POLYLINE
 x0 y0
 x1 y1
END POLYLINE
```

where (x_0, y_0) and (x_1, y_1) are the coordinates of the two endpoints. The first point (x_0, y_0) should be the starting point of the drift (where it starts from an access tunnel), while the second point (x_1, y_1) should be at the end of the drift.

Note: All tunnels are assumed to have the same floor elevation, as specified by the "tunnel floor" parameter as defined in Table 3.1. Thus the current version of ***repository*** only handles tunnels that are within the repository's deposition horizon. The input format may be revised in future versions, in order to represent vertical shafts or inclined access tunnels which are likely to be part of a complete repository design.

Table 3.3 Example of tunnel axes file.

```
BEGIN SECTION
BEGIN HEADER
$EXTMIN  -143.5000  2772.5000
$EXTMAX  3026.1250  4801.0000
$LIMMIN  -143.5000  2772.5000
$LIMMAX  3026.1250  4801.0000
END HEADER
END SECTION
BEGIN SECTION
LABEL Access Tunnel 1
BEGIN POLYLINE
 2436.0000  4200.5000
 2199.6250  4514.0000
 2173.8750  4540.0000
 1921.3750  4585.5000
  716.3750  4382.5000
  393.2500  4127.5000
  379.3750  4058.5000
  507.5000  3822.0000
  547.0000  3805.5000
  583.7500  3810.5000
  851.5000  4016.0000
  945.1250  3961.5000
END POLYLINE
LABEL Deposition Tunnel 1-01
BEGIN POLYLINE
  419.0000  4149.0000
  562.0000  3938.0000
END POLYLINE
LABEL Deposition Tunnel 1-02
BEGIN POLYLINE
  454.8750  4173.0000
  595.1250  3960.5000
END POLYLINE
LABEL Deposition Tunnel 1-03
BEGIN POLYLINE
  482.6250  4204.0000
  627.2500  3990.0000
END POLYLINE
END SECTION
```

4 *meshgenx* module for mesh generation

The ***meshgenx*** module is used to produces a triangular finite-element mesh from one or more files listing the panels that define discrete features which may include any of the following:

- surface topography,
- large-scale deformation zones,
- smaller-scale discrete fractures,
- EDZ within a repository,
- external boundary (limits of model domain),
- internal boundaries (*e.g.* boreholes or underground openings).

The command-line syntax for ***meshgenx*** is:

```
meshgenx [-h] tinylen panelfile [-r][-s trifile][-m meshfile][-f maxangle]
```

where:

tinylen Tolerance for nodes to be regarded as identical (assumed to be in meters).
panelfile Input file in panel format.

Options:

`-r` Add random points within panels to improve triangulations *(recommended)*.
`-s` Save pre-triangulation panel data to *trifile* and skip triangulation.
`-m` Save mesh to *meshfile* (default is stdout).
`-f` Set maximum allowable obtuse angle on elements.
Note: The flag for this option will be changed in future versions of ***meshgenx***.

Modeling Tip: Typing "`meshgenx -h`" on the command line will yield a summary of the syntax.

When the `-s` option is used, an intermediate file is saved that defines the intersections among all panels. This simplifies the global mesh-generation problem to a series of smaller, independent problems, one for each of the panels. The stand-alone triangulation program ***tripost*** (see Section 4.2) can then be used to discretize the lines for each panel one at a time, in batch mode.

Modeling Tip: The `-s` option is recommended when discretizing large files for complicated network geometry. A bug in the triangulation software can occasionally lead to infinite loops that cause the program to "hang." This tends to happen for about one out of every thousand

panels, depending on the geometric complexity as well as the raw number of panels. Normally these cases can be solved simply by choosing a different random number seed, but if the `-s` is not used, the entire run can be lost. Hence it is strongly recommended to use this option for large network problems.

4.1 *meshgenx* discretization options

The primary ***meshgenx*** option that can be used to control panel discretization and the quality of the resulting mesh is the command-line option `-r`, which results in random points being added to each panel before forming triangles. The effect of choosing this option is illustrated in Figure 4.1.

Modeling Tip: The random-point algorithm can be very slow in some cases, particularly when a single panel has several hundred or more vertices. If very slow run-times are encountered for a given model, this option might need to be skipped. The next version of ***meshgenx*** is expected to have a significantly more efficient algorithm.

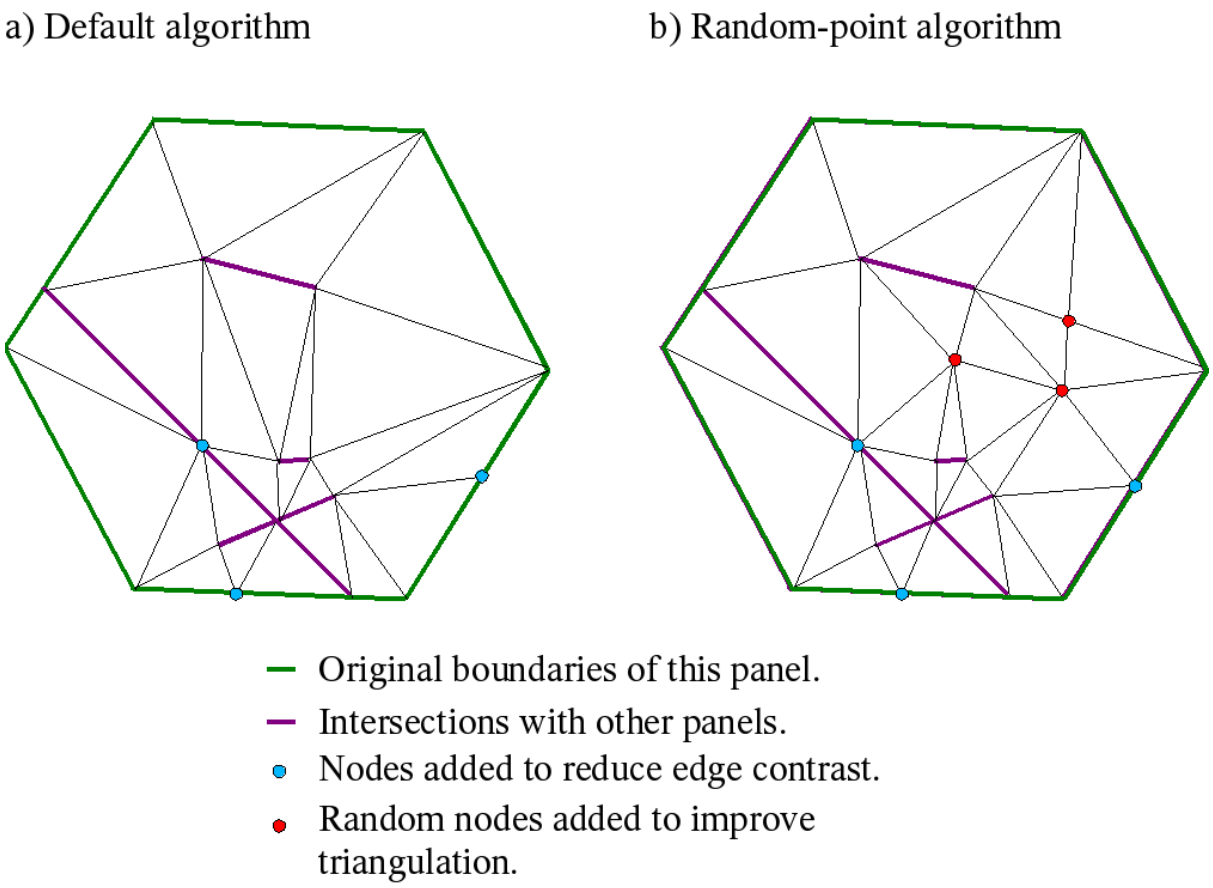


Figure 4.1 Example of discretization of a single hexagonal panel (representing one fracture) into triangular finite elements, using (a) the default meshgenx algorithm, or (b) the random-point algorithm which yields a more well-conditioned mesh. Note that in either case, a few nodes (vertices) may added deterministically, to reduce contrast in length between adjoining line segments that are formed either by the original panel boundaries, or by intersections with other panels.

4.2 *tripost* module and associated shell scripts

The stand-alone triangulation program ***tripost*** is used in conjunction with partial discretizations as produced by the ***meshgenx*** `-s` option. Normally this program is used in batch mode, via the shell scripts ***tripostx*** and ***consolidate_triangulation*** as described below.

The shell script ***tripostx*** (located in the directory *~/bin/dfmscripts*, see Appendix 1) is used to automate this process of running ***tripost***, by trying up to 20 different random number seeds in the rare cases where the triangulation algorithm gets stuck on a particular panel. Occasionally manual intervention may occasionally be needed if one of the ***tripost*** runs stalls.

4.2.1 *tripost*: shell script *tripostx*

The shell script ***tripostx*** is used to post-process triangulation data produced by ***meshgenx*** (if `-s` option is specified) to produce a triangulation file. This script is installed in *~/bin/dfmscripts*. The command-line syntax is:

```
# tripostx trifile
```

where:

trifile name of the triangulation input file in the form *casename.suffix*, usually *casename*`.tri`.

The triangulation output file will be named *casenamesuffix*`.triangulation`, and must be consolidated using the shell script ***consolidate_triangulation*** to produce a ***dfm*** **mesh file**.

The script ***tripostx*** triangulates the points and line segments that are defined in *trifile*, parsing the data for one panel one at a time and then running ***tripost*** (automatically). Progress is indicated as a list of panel numbers that have been processed, which are printed to the terminal.

If a ***tripost*** run fails, ***tripostx*** will print a warning message, then attempt to run ***tripost*** with a different random number seed. In some cases more than two attempts are needed; a parameter in ***tripostx*** sets the maximum number of attempts to 20.

Modeling Tip: Each ***tripost*** run executed by the ***tripostx*** script normally takes just one to several seconds to either succeed or fail, so the monitor should show a nearly continuous progression of panel numbers. In a few cases, a ***tripost*** run may "hang" (this apparently results from infinite loops in an algorithm that steps through adjoining simplexes that are

formed in the trial triangulation, resulting from finite numerical precision), so no progress is seen after 5 or 10 seconds. In such cases, it is advisable to terminate the ***tripost*** run. In a Linux or Unix shell, the command:

```
# ps -f -a
```

will list the process identities of all processes belonging to the user. The ***tripost*** run can then be terminated with the command:

```
# kill -9 PID
```

where *PID* is the process identity number for the ***tripost*** run. Once ***tripost*** is terminated, the ***tripostx*** script will automatically attempt a restart with a different value of the random number seed. It is seldom necessary to terminate ***tripost*** more than once for a given panel.

4.2.2 *tripost*: shell script *consolidate_triangulation*

The shell script ***consolidate_triangulation*** is used to consolidate the triangulation file produced by ***tripostx***, into a mesh file that is in a suitable format for ***dfm*** input. This script is installed to the directory *~/bin/dfmscripts*. The command-line syntax for running this script is:

```
# consolidate_triangulation casename suffix
```

where:

casename	name of the calculation case (should correspond to a **case file** named *casename*`.case` which contains header information for the mesh, see below);
suffix	suffix to identify a unique run for the calculation case (should correspond to an input file *casenamesuffix*`.triangulation`; the output file will be named *casenamesuffix*`.msh`).

A **case file** consists of a mesh title and a list of names for the enumerated boundaries. The syntax should conform to that for ***dfm*** **mesh files** as described in Chapter 5. An example is given in Table 4.1. See Section 5.3.1 for further explanation of this file.

Table 4.1 Example of a case file for consolidate_triangulation

Mesh title: Laxemar SR-Can Base Case
Boundary 1: Top
Boundary 2: Bottom
Boundary 3: South side
Boundary 4: East side
Boundary 5: North side
Boundary 6: West side
Boundary 7: Surface

5 *dfm* module for flow and solute transport simulation

The main module of the DFM software package, ***dfm***, solves steady-state and/or transient finite-element flow equations for a given mesh geometry and set of boundary conditions. This module is also designed for solute transport calculations. However, for many applications it is more practical to use the stand-alone particle-tracking module, ***meshtrkr***, as described in the following chapter.

The command-line syntax for running ***dfm*** is:

```
# dfm simseqfile
```

where:

simseqfile dfm simulation sequence definitions file

A **dfm simulation sequence definitions file** (which customarily has the suffix `.dfm`) entirely specifies the **simulation sequence** of modeling operations to be performed by the ***dfm*** module. The structure of this type of file is described in Section 5.1. It includes a designation of the **mesh file** which defines the geometry and properties of the discrete-feature network for a particular calculation case (see Section 5.2). The most important aspect of a simulation sequence is the specification of **boundary conditions** which determine flow (and optionally solute transport) in the discrete-feature network. Section 5.3 describes the types of boundary conditions that can be specified. Section 5.4 gives a detailed list of ***dfm*** **keywords** that are used to define simulation sequences.

Output from the dfm code is printed to the stream *stdout*. This should be redirected to a file in order to save the output, *i.e.*:

```
# dfm simseqfile >  output_file
```

While running, the ***dfm*** module also prints a series of status messages (and possibly error messages or warning messages) to the terminal, which in general are self-explanatory.

5.1 *dfm* simulation sequences

The set of modeling operations to be performed by the DFM code in a given run are specified in terms of a **simulation sequence** which is defined as the sequence of operations that the program should apply for a given physical representation (i.e., a given **mesh file**).

A simulation sequence is defined in a ***dfm* simulation sequence definition file**, which for brevity is referred to as a **dfm file** (since by convention its file name has suffix .dfm). A dfm file consists of an arbitrary number of **simulation stages**, which ordinarily correspond to changes in the types of boundary conditions that are imposed. Time-varying boundary conditions (for example, time-varying meteoric flux to the surface, or time-varying head imposed in a borehole during a well test) can be accomodated within a single simulation stage, but changes in the type of boundary condition (for example, switching from no-net-flow to specified head in a borehole) must be treated in separate simulation stages.

The overall structure of a **dfm file** is shown in Table 5.1. It consists of a series of statements to set options that apply to either:

- the **simulation sequence** as a whole, or
- specific **simulation stages** which are to be performed in sequence.

The geometry for a simulation sequence is constant throughout the simulation sequence. This geometry, and associated hydrologic properties, are defined in a **mesh file** which is specified for the simulation sequence as a whole (see Section 5.2). The hydrologic properties may under certain circumstances be modified between simulation stages, but the geometry is kept constant. As a realistic example, Table 5.2 shows the overall structure of a **dfm file** for simulation of a constant-head well test, including equilibration and recovery stages.

Table 5.1 *Overall structure of simulation sequence definitions (dfm) file.*

```
Simulation Sequence Definitions
Simulation Stage 1
Simulation Stage 2
...
Simulation Stage N
```

Table 5.2 *Example of the structure of a* ***dfm*** *file for simulation of a fixed-head injection test, including equilibration and recovery stages. Note that this is just to illustrate the overall structure, and does not exemplify the actual detailed syntax that would be used. The detailed syntax is documented in the remainder of this chapter.*

```
Mesh File
Initial head
Simulation Stage 1: Equilibration stage, Net flux = 0
Simulation Stage 2: Injection stage, head = H
Simulation Stage 3: Recovery stage, Net flux = 0
```

5.1.1 *dfm* file general rules

Each simulation stage involves solving a flow and/or transport problem for a specific mesh geometry, with a particular set of steady-state or time-dependent boundary conditions.

Within a simulation stage, values of particular boundary conditions (for example, the head in a borehole) may vary with time, but the type of boundary condition (*e.g.* specified head vs. specified net flux) cannot change. Hydrological properties of the features must also remain constant. Complex problems involving changing types of boundary conditions and/or changing feature properties can handled by simulation sequences consisting of multiple simulation stages.

In multi-stage simulations, by default it is assumed that the boundary conditions at the end of the previous stage apply as constant boundary conditions through the following stage, if boundary conditions are not otherwise specified.

The statements in a DFM file can be in a quite flexible order, as they are recognized by **keywords** rather than position in the file. A list of the keywords that are recognized is given in Section 5.4.

The following conventions affect the interpretation of these statements:

- Any line beginning with a pound sign (#) is ignored. This allows inclusion of explanatory text (comments) in the ***dfm*** file.
- An ampersand (&) at the end of a line means that the next line is to be read as a continuation of the current line.
- Continued lines can be up to 1000 characters total length, which amounts to approximately 20 lines for an average line length of 50 characters. ***Note:*** *This limit is set by the compile-time parameter* `MAXLONGLINE` *in the* ***dfm*** *source code, and could easily be increased, though at the cost of increased memory usage.*
- Certain characters (including spaces, tabs, colons, and equality signs) are ignored, but may be included to make files more readable.

The following restrictions must be observed regarding the order of statements:

1. The number of simulation stages should be specified using the keyword **`simulation stages`** before defining the individual stages (This requirement may be removed in future versions of ***dfm***).

2. Statements beginning with a **sequence level keyword** (see Section 5.4) can appear anywhere in the dfm file, and affect the simulation sequence as a whole, regardless of where they appear in the dfm file.

3. Statements beginning with a **stage level keyword** (see Section 5.4) affect the simulation stage identified by the most recent **`simulation stage id`** statement to appear in the dfm file. Thus the influence of these statements depends on their position relative to **`simulation stage id`** statements in the DFM file. However, within a given block of statements between two **`simulation stage id`** statements, the ordering of **stage level keyword** statements is arbitrary.

4. Certain keywords require additional lines of input. These additional lines must be given sequentially as a single block within the dfm file. For example, the **`time steps`** keyword is used to indicate the number of time steps in a simulation stage. The data for the specified number of time steps needs to follow on the lines immediately after this statement.

5.2 *dfm* mesh files

Mesh files are ordinarily created by the ***meshgenx*** module, rather than by typing by hand. However, it may be useful to understand the format of these files, in case there is a need to make minor modifications. The syntax for mesh files is shown in Table 5.3, with variables explained in Table 5.4, and other terms explained in the text below.

Table 5.3 Format for mesh files. Variables (indicated by italic text) are explained in Table 5.4. Square braces [] enclose lists of optional arguments which are separated by vertical pipe (|) characters. An ampersand (&) represents a line that is continued on the next line of the file; the use of ampersands as shown here is optional but serves to improve readability.

```
Mesh title: mesh_title

Coordinates in           L
Aperture values in       L
Transmissivity values in L2/T

[Initial heads specified]
[Initial fluxes specified]

Element type: [constant|variable] transmissivity  &
              [constant|variable] storativity     &
              [constant|variable] aperture

Boundary 1: boundary_ID1
 :
Boundary B: boundary_IDB

Number of nodes     n
Number of elements  e
Number of features  f

List of nodes:
# ID   x(m)  y(m) z(m) NdGrp [head] [flux]
   1   x1    y1    z1     g1    [h1]   [q1]
   :
   n   xn    yn    zn     gn    [hn]   [qn]

List of elements:
# ID  N(1) N(2) N(3)    T      S     A   Feature  Set
   1   N11  N21  N31     T1     S1    b1     F1     s1
   :
   e   N1e  N2e  N3e     Te     Se    be     Fe     se
```

Table 5.4 Definition of variables used to define mesh file syntax in Table 5.3.

mesh_title	A text string to describe this mesh (up to 100 characters long).
L	Units of length (normally `m` for meters; see Appendix 4 for other options).
T	Units of time (normally `s` for seconds; see Appendix 4 for other options).
$boundary\_ID_i$	Name of *i*th **boundary segment**, i = 1, 2, ..., B where B is the number of boundary segments that are represented in the mesh file.
n	Number of nodes in mesh.
e	Number of elements in mesh.
f	ID of the highest-numbered feature in the mesh. This will be equal to the number of features in the mesh, if the features are numbered in sequence as 1,2,3,...
x_i	x-coordinate of *i*th node (units should be as specified at top of file).
y_i	y-coordinate of *i*th node.
z_i	z-coordinate of *i*th node.
g_i	Nodal-group number of *i*th node. Each nodal group must correspond to a named **boundary segment** with the same identification number, as described above.
h_i	Initial head value for *i*th node (units assumed to be in meters).
q_i	Initial flux value for *i*th node (units assumed to be in m^3/s).
$N1_i$	Node number of 1st node in *i*th element.
$N2_i$	Node number of 2nd node in *i*th element.
$N3_i$	Node number of 3rd node in *i*th element.
T_i	Transmissivity of *i*th element (units should be as specified at top of file).
S_i	Storativity of *i*th element (note this is dimensionless).
b_i	Transport aperture of *i*th element (units should be as specified at top of file).
F_i	Identification number of the feature to which the *i*th element belongs.
s_i	Feature set number to which the *i*th element belongs.

In Table 5.3, the portion of the mesh file that precedes the line:

```
Number of nodes      n
```

is referred to below as the **mesh file header**. In simulations of discrete-feature networks that include a stochastic component, the header will generally be identical between mesh files for different realizations.

In the list of nodes, the columns labels `head` and `flux` and the corresponding values h_i and q_i , i = 1, 2, ..., n) are optional. The `head` column (to specify initial values of head at each node, for transient simulations) should be included only if the line:

```
Initial heads specified
```

is included in the mesh file header. Similarly the `flux` column (to specify initial values of flux at each node) should be included only if the line:

```
Initial fluxes specified
```

is included in the mesh file header.

The ***dfm*** solver includes mathematical functions to work with finite elements that have either constant (uniform) or linearly varying element properties (transmissivity, storativity, and/or transport aperture). The type (**`constant`** or **`variable`**) for each of these properties can be specified in the **`element type:`** statement in the mesh file header. The following paragraphs give mathematical background and describe modifications to the mesh file which are required if one or more **`variable`** element properties are specified.

Important Note: *The current version of the **dfm** module has not been tested for cases with variable element properties. Also, other modules of the DFM package do not support this feature. Therefore this option is not recommended, and should be used, if at all, with caution. This feature is documented here, mainly with the expectation that further development and testing will be performed for future versions of the DFM package.*

Following the mathematical development given by Geier (2005), for a linearly varying element, the transmissivity T is defined at each point $\boldsymbol{x}$ on a given triangular element e as:

$$T_e(x) = \sum_{i=1}^{3} T_{e_i} w_{ei}(x)$$

where:

T_{ei} = element transmissivity at the ith node belonging to element e.

$w_{ei}(\boldsymbol{x})$ = linear weighting function as a function of the 3-D coordinates $\boldsymbol{x}$.

The weighting function $w_{ei}(\boldsymbol{x})$ is defined for points on the element e as:

$$w_{ei}(x) = \begin{Bmatrix} 1 \quad for\, x = x_i \\ 0 \quad for\, x = x_j, \quad i \neq j \\ w \quad with\ 0 < w < 1 \quad varying\ linearly\ for\ other\ x\ on\ e \end{Bmatrix}$$

Note that the transmissivity does not need to be continuous between adjoining elements, although one possible application of this option could be to examine the consequences of continuously varying aperture within single, variable-aperture fractures (as an alternative to discrete, stepwise variation which is commonly used to represent such cases in the literature).

The cases of linearly varying storativity S and transport aperture b_T are defined analogously:

$$S_e(x) = \sum_{i=1}^{3} S_{e_i} w_{ei}(x)$$

$$b_{Te}(x) = \sum_{i=1}^{3} b_{Te_i} w_{ei}(x)$$

If a given element property is specified as **`variable`** rather than **`constant`**, then three values should be given in place of the single value for T_i, S_i, or b_i for the ith element, in Table 5.3. For example, if transmissivity is specified as **`variable`** but storativity and aperture are specified as **`constant`**, then the following data are needed for each node i:

i $N1_i$ $N2_i$ $N3_i$ $T1_i$ $T2_i$ $T3_i$ S_i b_i F_i s_i

where:

$T1_i$ Transmissivity value at the 1st node of the ith element,
$T2_i$ Transmissivity value at the 2nd node of the ith element,
$T3_i$ Transmissivity value at the 3rd node of the ith element,

Or, if all three element properties (transmissivity, storativity and aperture) are specified as **`variable`** the following data are needed for each node i:

i $N1_i$ $N2_i$ $N3_i$ $T1_i$ $T2_i$ $T3_i$ $S1_i$ $S2_i$ $S3_i$ $b1_i$ $b2_i$ $b3_i$ F_i s_i

where $S1_i$, $S2_i$, and $S3_i$ and $b1_i$, $b2_i$, and $b3_i$ are defined analogously as the point values as each node, which are then used to determine the values of each property over the remainder of the element, by linear interpolation.

5.3 *dfm* boundary conditions

Boundary conditions describe the far-field driving forces and processes (hydraulic heads and fluxes of groundwater and/or solute) that affect the directions and magnitudes of flow and transport through the discrete-feature network. This section describes the way that boundary conditions are specified with the DFM package.

5.3.1 *dfm* boundary groups

Boundary conditions (BCs) are applied to named **boundary groups**. A boundary group is a group of nodes (and by implication, the discrete-feature element edges connecting between pairs of those nodes). Ordinarily a boundary group is a set of nodes that belong either to a given physical boundary (for example, a section of a borehole, a portion of the ground surface), or to a modeling boundary (usually the lateral and/or lower boundary of the modeled region).

Boundary groups are given text names which can be chosen for mnemonic purpose, e.g. "Borehole KLX 01" or "East Face" or "Deposition Hole 39." This avoids the need to refer to boundaries by numerical codes which have the potential to be confusing, particularly if particular boundaries are used in simulations representing certain scenarios or discrete-feature network geometries, but not others.

The ***dfm*** module requires only that the name, *i.e.* the **boundary ID**, be the same in the simulation sequence definitions (dfm) file that defines the boundary conditions, as in the mesh file which defines the problem geometry and the hydrologic properties of the features.

Boundary groups with no specified boundary conditions

Named boundary groups for which conditions are not specified in a simulation sequence (by the start of a given simulation stage) are treated as no-flow (zero-flux) boundaries. The program will print an informational message, indicating that the boundary is being treated as a no-flow boundary because the boundary condition is not otherwise specified.

Quirk: *In the current version of **dfm**, nodes belonging to such boundary groups will lose their boundary segment IDs on output. This should be corrected in future versions.*

Boundary groups with boundary conditions specified but no corresponding nodes

A warning message will be printed if a named boundary group has a boundary condition specified, but no nodes belonging to this boundary are present in the mesh file for the given simulation. In this case, the specified boundary condition has no effect on the simulation. Sometimes this may occur due to an error by the modeler. However, it can also occur if the discrete-feature network has no intersections with the physical boundary for which the simulation sequence defines boundary conditions. This may arise, for example, if only stochastically generated fractures have a chance of intersecting a given boundary, such as a borehole or tunnel segment.

5.3.2 Types of hydraulic boundary conditions

The major types of hydraulic boundary conditions that can be applied are **`specified head`**, **`specified flux`**, and **`specified net flowrate`**. A topographically constrained **`infiltration`** boundary condition is expected to be implemented in a future version and is under development, but not available at present. Boundary conditions for solute transport can be either **`specified concentration`** or **`specified solute flux`**.

For a given type of boundary condition, the values as a function of time and spatial coordinates are specified in terms of functions referred to here as **temporal/spatial functions**, or **tfuncs** for short. These may be simple functions of one to three parameters, or compound functions which are more complex. The syntax for specifying **tfuncs** is described in detail in Section 5.4.4.

5.4 *dfm* keywords

This section lists the syntax for the keywords that may be used in dfm files. There are two main categories of keywords:

- **sequence-level keywords** which affect the simulation sequence as a whole, and can appear anywhere in the dfm file.
- **stage-level keywords** which affect the simulation stage identified by the most recent **`simulation stage id`** statement to appear in the dfm file, and thus affect simulations differently depending on their position in the dfm file; however, within a given block of statements between two **`simulation stage id`** statements, the ordering of stage-level keywords is arbitrary.

These two categories of keywords are presented in Sections 5.4.1 and 5.4.2, respectively. **Temporal/spatial functions**, used as arguments to some stage-level keywords affecting boundary conditions, are described in Section 5.4.3.

5.4.1 *dfm* sequence-level keywords

The following is a list of sequence-level keywords, with a brief description of their syntax.

Keyword: **`mesh file`**
Purpose: Specify name of mesh data file.
Syntax: `mesh file[:] filename`
Arguments:
`filename` path and name of file containing mesh data.
Example:
`Mesh file: test1.msh`
Default: null string (will cause error)

Keyword: `simulation title`
Purpose: Specify title for the simulation.
Syntax: `simulation title[:] title`
Arguments:
`title` Text of title.
Example:
`simulation title: Test problem to illustrate simple case.`
Default: null string (no title for simulation).

Keyword: `simulation stages`
Purpose: Specify number of stages in this simulation.
Syntax: `simulation stages n`
Arguments:
`n` Number of simulation stages in this problem.
Example:
`simulation stages: 2`
Default: 0 (no simulation stages so run will terminate).

Keyword: **`fluid compressibility`**
Purpose: Specify fluid compressibility constant C_f.
Syntax: `fluid compressibility[:]` C_f
Arguments:
C_f Fluid compressibility in units of T^2L/M (inverse pressure).
Example:
`fluid compressibility: 1.4e-12 1/Pa`
Default: $C_f = 0$ (incompressible fluid)

Note: This parameter currently has no effect, but is reserved for future use.

Keyword: **fluid density**
Purpose: Specify fluid density parameter ρ_f.
Syntax: `fluid density[:] rho`
Arguments:

`rho` fluid density in units of M/L^3

Example:

```
fluid density: 1000 kg/m3
```

Default: ρ_f = 1000 kg/m^3 (density of standard water)
Note: This parameter currently has no effect, but is reserved for future use.

Keyword: **molecular diffusion coefficient**
Purpose: Specify molecular diffusion coefficient D_m for use in calculating diffusion of solute in stagnant fractures, and for calculation of dispersion coefficients in pore space of features with nonzero fluid velocity (although effect of D_m typically becomes negligible with increasing fluid velocities).
Syntax:

`molecular diffusion coefficient Dm`

Arguments:

`Dm` molecular dispersion coefficient in units of L/T^2

Example:

```
Molecular dispersion coefficient: 3.0e-6 m/s2
```

Default: $D_m = 10^{-6}$ m/s^2

Keyword: **longitudinal dispersivity**
Purpose: Specify longitudinal dispersivity α_L for calculation of dispersion coefficients for solutes in features.
Syntax: `longitudinal dispersivity alpha_L`
Arguments:

`alpha_L` longitudinal dispersivity in units of L.

Example:

```
Longitudinal dispersivity: 1 m
```

Default: α_L =1 m

Keyword: **transverse dispersivity**
Purpose: Specify transverse dispersivity α_T for calculation of dispersion coefficients for solutes in features.

Syntax: `transverse dispersivity alpha_T`

Arguments:

`alpha_T` transverse dispersivity in units of L.

Example:

```
Transverse dispersivity: 0.1 m
```

Default: $\alpha_T = 0.1$ m

Keyword: **simulation stage id**

Purpose: Heading for **simulation stage**. Assigns name to simulation stage and indicates start of a block of statements that apply to this simulation stage.

Syntax: `simulation stage id name`
`{simulation stage definitions}`

Arguments:

`name` Name of simulation stage (single word or phrase).

{simulation stage definitions}

Block of lines following the line on which this keyword appears, up until the next sequence-level keyword, are treated as applying to this simulation stage.

Example:

```
Simulation Stage ID: Equilibration prior to pumping test
{additional statements to define equilibration BCs etc.}
```

Keyword: **random aperture model**

Purpose: Define a stochastic model for assigning variable aperture to fractures. Intended for use in particle-tracking simulations where effects of fracture aperture variation on a fine scale are to be considered. Not fully implemented in this version of ***dfm***.

Syntax: *Not fully defined in this version.*

Example:

```
Random Aperture Model
Fracture Sets:        All
Mean value:           Use initial value
Grid resolution:      5 cm
Covariance Model:         Exponential &
                      Isotropic   &
                      Variance    0.0001 &
                      Range       0.1
```

Default: Random apertures not assigned.

Keyword: **node renumbering**

Purpose: Specify whether or not to perform automatic renumbering of nodes to optimize bandwidth.

Syntax: `node renumbering [on|off]`

Options:

`off`	Node renumbering turned off, i.e. no automatic renumbering of nodes.
`on`	Currently has no effect, reserved for addition of renumbering algorithm in future version.

Example: `Node renumbering OFF`

Default: `off` (node renumbering turned off).

Keyword: **matrix storage type**

Purpose: Defines type of computer storage to use for finite-element matrices.

Syntax: `matrix storage type [full matrix|banded|skyline|compact]`

Options:

`full matrix`	Full matrix is stored (least efficient, only used for small problems, primarily debugging matrix solvers).
`banded`	Banded storage
`skyline`	Skyline matrix storage
`compact`	Compact (Yale) storage for sparse matrices (most efficient).

Default: `full matrix`

5.4.2 *dfm* simulation-stage level keywords

The following is a list of stage-level keywords, with a brief description of their syntax.

Keyword: **`flow solution mode`**
Purpose: Defines mode for flow solution.
Syntax: `flow solution mode` *`option`*
Argument:

`option` may take one of the following values:

`off`	No flow solution will be computed for this stage.
`steady-state`	Steady-state flow solution.
`transient`	Transient flow solution.
`laplace-galerkin`	Transient solution using Laplace-Galerkin method.
`backward-difference`	Transient solution using backward-difference method.

Default: `off` (no flow solution will be computed for this stage).

Keyword: **`transport solution mode`**
Purpose: Defines mode for flow solution.
Syntax: `flow solution mode` *`option`*
Argument:

`option` may take one of the following values:

`off`	No flow solution will be computed for this stage.
`particle tracking`	Transport solution by particle-tracking.
`laplace-galerkin`	Transport solution by Laplace-Galerkin method.

Default: `off` (no transport solution will be computed for this stage).

Keyword: **`boundary id`**
Purpose: Specifies name for boundary group corresponding to the boundary condition definition statement that follows.
Syntax: `boundary id` *`name`*
`{boundary condition specification}`
Argument:

`name` Name of boundary group (must match a boundary group in mesh file).
(see Section 5.4.3 for a list of possible boundary-condition specifications)
Example:

Boundary ID: Bottom
Specified flux Constant(0 m3/s)

Keyword: **boundary type**

Syntax: `boundary type` *`btype`*

Argument:

`btype`	Type of boundary

Note: This keyword is not used in the present version, and is ignored.

Keyword: **matrix solver**

Syntax: `matrix solver` *`solver_type`*

Argument:

`solver_type` may take one of the following values:

`gauss-jordan`	Gauss-Jordan solver
`lu decomposition`	LU decomposition
`conjugate-gradient`	"Naive" conjugate-gradient
modified `conjugate-gradient`	Modified conjugate-gradient solver
`complex gaussian`	Gaussian solver for complex matrices (used with Laplace-Galerkin method).
`biconjugate-gradient`	Biconjugate-gradient solver for complex matrices (not currently implemented).
`orthomin`	ORTHOMIN solver (not currently implemented).

Keyword: **solver settings**

Syntax: `solver settings solver_type [{`*`solver_settings`*`}|{`*`tracker-settings`*`}]`

Arguments:

`solver_type`	See keyword **`matrix solver`** (above) for list of options.
`{`*`solver_settings`*`}`	See topic "solver settings list" below.
`{`*`tracker_settings`*`}`	See topic "tracker settings list" below.

Example:

```
Solver settings:   Modified conjugate-gradient &
                   Precondition diagonal &
                   Maximum iterations   100000 &
                   Convergence tolerance 1e-8
```

Defaults: No method defined for flow or transport solution.

Keyword: **time step**

Purpose: Identifies a single time step. Not needed if time steps follow sequentially after the keyword **`time steps`**.

Note: This keyword does not have a practical use and is potentially confusing. It should be eliminated in future ***dfm*** versions.

Keyword: **time steps**

Purpose: Initiates a block of **timesteps** to specify the steps, substeps, and output options for a transient flow simulation.

Syntax:
```
time steps N
t_i [substeps n_i] [output_option_1]
t_2 [substeps n_2] [output_option_2]
:
t_N [substeps n_N] [output_option_N]
```

Arguments:

N	number of time steps.
t_i	time of *i*th time step, in units of T.
n_i	(optional) number of substeps for intermediate calculations (see note below);
output_option$_i$	[full output|summary output|no output]

Example:
```
Time steps = 6
5 s
10 s
20 s
1 min
2 min
10 min : full output
```

Notes: The optional substeps are used to improve accuracy of backward-difference solution schemes, by using a finer discretization in time than is desired for output. The optional flags [full output], [summary output] and [no output] are used to specify type of output to be printed for each (full) time step.

5.4.2.1 *dfm* solver settings lists

The following keywords are used only within solver settings lists:

Keyword: **`precondition`**
Purpose: Used for conjugate-gradient solvers only. Specifies the type of matrix preconditioning method that will be used.
Syntax: `precondition [diagonal|choleski [1|2]]`
Options:

`diagonal` Simple diagonal scaling.
`choleski` Incomplete Choleski preconditioning (first-order or second-order depending on optional argument 1 or 2).

Default: Preconditioner not specified.

Keyword: **`maximum iterations`**
Purpose: Used for iterative solvers only. Specifies the maximum number of iterations that are allowed to achieve convergence, before giving up and terminating the solver.
Syntax: `maximum iterations` *`n`*
Argument:

`n` Number of iterations to allow for convergence

Default: *n* =1

Keyword: **`convergence tolerance`**
Purpose: Used for iterative solvers only. Specifies the absolute tolerance for testing convergence.
Syntax: `convergence tolerance` *`epsilon`*
Argument:

`epsilon` Maximum allowed change in absolute error measure in units of head (L) for flow solution, or units of concentration (M/L3) for transport solution.

Default: *epsilon* = 10^{-10} in SI units (m or kg/m^3 depending on context)

Keyword: **`projection factor`**
Purpose: Used for transient forward difference solvers only. Specifies projection factor to use as initial guess for next time step.
Syntax: `projection factor` *`proj`*
Argument:

`proj` Factor for projection of heads as an initial guess for next time step.

Default: *proj* = 1

5.4.2.2 *dfm* tracker settings list

The following keywords are used only within tracker-settings lists:

Keyword: **`random seed`**
Purpose: Specifies seed value for random number generator.
Syntax: `random seed` *`seed`*
Argument:

`seed` Integer seed for random-number generator (used in particle-tracking).

Default: An arbitrary random seed value will be used.

Keyword: **`particle mass`**
Purpose: Specifies mass that is represented by each discrete particle, for the discrete-particle random-walk (DPRW) algorithm.
Syntax: `particle mass` m_p
Argument:

m_p Mass of solute represented by each tracked particle, in units of M.

Default: No value set.

Keyword: **`sink radius`**
Purpose: Sets the capture radius for solute sink nodes in particle-tracking simulations (i.e. the radius within which a particle is assumed to have reached the sink).
Syntax: `sink radius` *`rsink`*
Argument:

`rsink` Capture radius, in units of L.

Default: No value set.

5.4.2.3 *dfm* boundary condition specifications

The following keywords are used only in boundary-condition specifications, which follow a **`boundary id`** statement in a stage-level block. Boundary conditions are specified using temporal/spatial functions (**tfunctions**), the syntax of which is explained in Section 5.4.3.

Keyword: **`specified head`**
Purpose: Prescribes head values for the specified nodal group, as a function of time and/or space.
Syntax: `specified head[:] head_function`
Argument:

`head_function` **tfunction** in units of head (L)

Examples:

```
Specified Head: Constant( 20 m )
Specified Head: Constant( 20 m ) + Step( 1 m, 5 min )
Specified Head: Constant 10 m + &
          Ramp( 1 cm/min, 5 min ) - Ramp( 1 cm/min, 10 min )
```

Keyword: **`specified flux`**
Purpose: Prescribes flux values for the specified nodal group, as a function of time and/or space.
Syntax: `specified flux[:] flux_function`
Argument:

`flux_function` **tfunction** in units of L3/T

Examples:

```
Specified flux: Constant( 1e-5 m3/s )
```

Keyword: **`specified net flux`**
Purpose: Prescribes the net flux to the specified nodal group, as a function of time and/or space.
Syntax: `specified net flux[:]  net_flux_function`
Argument:

`net_flux_function` **tfunction** in units of L3/T

Note: Using **tfunctions** with a spatial component is not recommended for net-flux groups, since the net flux will be calculated using the mean coordinates of the nodes in the group (which might not necessarily represent the centroid of the physical boundary).

Keyword: **`specified concentration`**

Purpose: Prescribes the concentration for inflows to the specified nodal group, as a function of time and/or space.

Syntax: `specified concentration[:] conc_function`

Argument:

`conc_function` **tfunction** in units of M/L3

Keyword: **`infiltration`**

Purpose: Intended to support topographically constrained infiltration. This keyword is disabled in the present version.

Syntax: `infiltration[:]` *`infilt_function`*

Argument:

`infilt_function` **tfunction** in units of L3/(TL2)

5.4.3 *dfm* temporal/spatial functions

A temporal/spatial function (referred to as **tfunction** or **tfunc** for short) is used to specify constant or time-varying boundary conditions.

5.4.3.1 *dfm* tfunction syntax

The syntax of a **tfunction** is defined as follows:

tfunc : one of the following:
```
tfunction + tfprod
tfunction - tfprod
tfprod - tfunction
tfprod
```

tfprod : one of the following:
```
tfprod * tfelem
tfelem * tfprod
tfprod / tfelem
tfelem
```

tfelem : one of the following elementary function types with meanings as indicated:

`constant(v)`	$f(t) = v$
`step(v,t0)`	$f(t) = vH(t-t_0)$ *where* $H(t)=\begin{cases} 0, & t<0 \\ 1, & t\geq 0 \end{cases}$
`ramp(r,t0)`	$f(t) = rR(t-t_0)$ *where* $R(t) = \begin{cases} 0, & t<0 \\ t, & t\geq 0 \end{cases}$
`exponential(v,tau)`	$f(t) = ve^{t/\tau}$ where τ = *tau*.
`sinusoidal(v,tau,ts)`	$f(t) = v\,sin\,\frac{2\pi(t+t_s)}{\tau}$ where τ = *tau*.
`sinusoidal(v,tau,ts)`	$f(t) = v\,sin\,\frac{2\pi(t+t_s)}{\tau}$ where τ = *tau*.
`plane(a0,ax,ay,az)`	$f(x,y,z) = a_0 + a_x x + a_y y + a_z z$

where:

t	time at a given timestep in the simulation.
(x,y,z)	coordinates of a given node.

and where the units of the ***tfelem*** arguments are as follows:

v, a_0	boundary-condition units V (see below).
t, t_0, t_s , τ	units of time (T).
r	units of V/T (see below).
a_x, a_y, a_z	units of V/L (see below).

The boundary-condition units V depend on the type of boundary condition:

V = L for specified head,

V = L^3/T for specified net flux,

V = L/T for specified flux (note this is a flux density),

V = M/L^3 for specified concentration.

5.4.3.2 *dfm* tfunction usage and restrictions

The elementary **tfunctions** as defined above can be assembled additively to make compound functions of the form:

$$f(t) = \sum_i f_i(t)$$

For example the tfunc:

```
Constant( 1 m ) + Ramp( 10 cm/min, 5 min ) - Ramp( 10 cm/min, 10 min )
```

describes a time-varying function (in units of head) which begins at the level h = 1 m, then beginning at t = 5 min increases linearly at the rate of 10 cm/min (as specified in the first ramp function), then levels off at h = 1.5 m from t = 10 min onward (at which point the second ramp function goes into effect, offsetting the continuing effect of the first ramp function).

Ramp, exponential, and sinusoidal tfuncs are not permitted as boundary-condition specifiers for steady-state simulation stages, since these tfuncs do not unconditionally approach a finite value as time approaches infinity (ramp and exponential functions can increase/decrease indefinitely, while sinusoidal functions fluctuate indefinitely). In special cases (for example, exponential functions with $\tau<0$, or offsetting ramp functions used as part of compound tfuncs) these tfuncs could yield finite results which would be permissible in steady-state stages. However, more sophisticated logic would be needed to recognize these special cases. This may be possible in future versions, but for the present, these tfuncs are permitted only in transient simulation stages.

Support for tfuncs in Laplace-Galerkin simulations is limited to linear combinations of elementary tfuncs for which Laplace transforms can be calculated in a straighforward manner by the ***dfm*** tfunc parsing algorithm. Error messages may sometimes be generated for tfuncs for which the parser cannot resolve the Laplace transform, even in some cases where the Laplace transform could be obtained by mathematical manipulations. Sometimes such cases can be resolved by re-writing the tfunc in a simpler form.

5.4.3.2 *dfm* table tfunctions (disabled feature)

An alternative **table tfunction** for specifying time-varying boundary conditions has also been implemented on the mathematical side of the program, but lacks full input/output support so is disabled in the current version of the code. It is presented here to maintain continuity of documentation, for when it can be reintroduced in a future version.

The syntax for a table tfunction is:

```
Table[:] ( t1, v1, t2, v2, ..., tn, vn )
```

or making use of ampersands (&'s) for continuation lines:

```
Table: ( t1, v1, &
         t2, v2, &
         :
         tn, vn )
```

The time-dependent value of the specified boundary condition is:

$$f(t) = \begin{Bmatrix} v_{1,} & t \le t_1 \\ v_i(1-s_i)+v_{i+1}s_i, & t_i \le t_{i+1}, \quad i=1,\ldots n-1 \\ v_n, & t \ge t_n \end{Bmatrix} \quad where \quad s_i = \frac{t-t_i}{t_{i+1}-t_i}$$

Note that this would be equivalent to (but more compact than) a series of tfuncs, *e.g.*:

```
Constant( v1 ) - Step( v1,t1 ) + &
Ramp( r1,t1 )  - Ramp( r1,t2 ) + &
:
Ramp( rn-1,tn-1 ) - Ramp( rn-1,tn ) + Constant( vn )
```

6 *meshtrkr* particle-tracking module

The ***meshtrkr*** module of the DFM toolkit tracks advective-dispersive or advective-diffusive motion of discrete particles representing solute.

The command for running meshtrkr is of the form:

```
# meshtrkr [-h] partfile dispfile meshfile t [T dt|L dl] zmax [ > logfile ]
```

where:

`-h`	Prints a summary of the command syntax.
`partfile`	Particle file listing the boundary groups to launch particles from.
`dispfile`	Dispersivity file defining dispersivity values for different classes of features.
`meshfile`	Mesh file in DCX format (see Appendix 5).
`t`	Maximum time [s] to track a given particle.
`T`	Specifies option to report particle positions at fixed time intervals *`dt`*.
`dt`	Time interval [s] between reports of a particle's position (used with option `T`).
`L`	Specifies option to report particle positions at fixed distance intervals *`dl`*.
`dl`	Distance interval [m] between reports of a particle's position (used with option `L`).
`zmax`	*z* coordinate [m] above which particles are assumed to have discharged to the surface environment.
`logfile`	Log file to record output of the particle tracker (otherwise this will be printed to *stdout*.

6.1 *meshtrkr* particle-tracking algorithm

Advective-dispersive transport of nonsorbing solute in the 3-D network (for the case of no matrix diffusion) is modeled by the discrete-parcel random walk (DPRW) method (Ahlstrom *et al.*, 1977). This approach represents local, 2-D advective-dispersive transport within each fracture plane. 3-D network dispersion, due to the interconnectivity among discrete features, arises as the result of local dispersion in combination with mixing across fracture intersections.

Particles are initiated at source locations, which are typically internal boundaries to the mesh. Particles are initiated along the line segments where fractures intersect the holes, at randomly distributed locations along the segments.

Once inside the mesh, the motion of a particle within a fracture element (triangular finite element) is modeled as a random walk within the fracture plane. In the random walk, each step $\Delta \boldsymbol{x}$ consists of a deterministic, advective component plus a random, dispersive component:

$$\Delta x = v\Delta t + r$$

where Δt is a locally-specified time step, $v = -\frac{T}{b_T}\nabla h$ is the (local) fluid velocity within the plane of the element, T is the local transmissivity, b_T is the local effective transport aperture, ∇h is the 2-D head gradient within the plane of the element, and $\boldsymbol{r}$ is the vector sum of random components representing longitudinal and transverse dispersion:

$$r = r_L u_L + r_T u_T$$

where $u_L = v/\|v\|$ is the unit vector parallel to the local velocity and $u_T = n \times u_L$ (where n is the unit normal to the element plane) is a unit vector transverse to the local velocity, within the plane of the element. The random scalars r_L and r_T (with dimensions of length) are drawn from normal distributions with zero mean:

$$r_L \sim N\left(0, \sqrt{2\mathrm{D}_L \Delta t}\right)$$
$$r_T \sim N\left(0, \sqrt{2\mathrm{D}_T \Delta t}\right)$$

where D_L is the longitudinal dispersion coefficient and D_T is the transverse dispersion coefficient.

The local dispersion coefficients depend on the magnitude of the local velocity as:

$$D_L = \alpha_L |v| + D_m$$
$$D_T = \alpha_T |v| + D_m$$

where α_L and α_T are the longitudinal and transverse dispersivities, respectively, within a given fracture.

Application of the DPRW model in a network requires an assumption regarding the degree of mixing within each fracture intersection, due to molecular diffusion across streamlines as water passes through the intersection. The degree of mixing in an intersection is governed by the Peclet number for flow through the intersection:

$$Pe = \frac{vb}{D_d}$$

where b is the fracture aperture, v is the mean fluid velocity and D_d is the coefficient of molecular diffusion. Berkowitz *et al.* (1994) showed that for an idealized intersection, mixing is negligible for Pe > 0.1, but significant for Pe = 0.0001, which corresponds to $v \approx 3$ cm/yr in a 0.1 mm fracture. For natural gradients that are expected within a radioactive-waste repository, v can be on the order of 1 m/yr or less, and substantial mixing will occur at most fracture intersections.

Hence complete mixing is assumed as a reasonable approximation for repository time scales. When a particle arrives at an intersection edge, the particle is randomly assigned to one of the elements sharing that edge. The probability of assignment to the ith connected element is:

$$P[i] = Q_i \; / \; \sum_e Q_e$$

where Q_e is the inflow to the eth connected element along the edge (zero if there is outflow), and the summation is taken over all elements connected to the edge. This reassignment technique does not allow for particles to move between adjacent elements in the absence of net advection. Hence the model may under-represent the actual diffusion and/or transverse dispersion (due to small-scale heterogeneity within fracture planes) that takes place in the physical system.

6.2 Calculation of pathway parameters

Advective-dispersive particle trajectories are traced for multiple particles for each deposition hole in the repository. For each release-path trajectory τ consisting of discrete segments $\{\tau_1, \tau_2, \ldots\}$ the following quantities are calculated by summing over the segments τ_i :

$$
\begin{aligned}
F_r &= \sum_{\tau_i} \frac{a_w(\tau_i)\,\Delta l}{v(\tau_i)} = \sum_i \frac{2\,\Delta t}{b_T(\tau_i)} \\
L_r &= \sum_{\tau_i} \Delta l \\
t_r &= \sum_{\tau_i} \Delta t \\
I_a &= \sum_{\tau_i} a_w(\tau_i)\,\Delta l = \sum_{\tau_i} \frac{2\,\Delta l}{b_T(\tau_i)} \\
I_b &= \sum_{\tau_i} b_T(\tau_i)\,\Delta l \\
I_c &= \sum_{\tau_i} T(\tau_i)\,\Delta l
\end{aligned}
$$

where Δl and Δt are the increments of distance and time for each step, $a_w = 2/b_T$ is the local wetted surface per unit volume water, T is the local transmissivity, and $v = \Delta l/\Delta t$ is the magnitude of the local advective velocity.

The equivalent quantities are also calculated for each feature set Φ (as defined in the mesh file) along each path, where:

$$
\begin{aligned}
F_{r\Phi} &= \sum_{\tau_i \in \Phi} \frac{2\,\Delta t}{b_T(\tau_i)} \\
L_{r\Phi} &= \sum_{\tau_i \in \Phi} \Delta l \\
t_{r\Phi} &= \sum_{\tau_i \in \Phi} \Delta t \\
I_{a\Phi} &= \sum_{\tau_i \in \Phi} \frac{2\,\Delta l}{b_T(\tau_i)} \\
I_{b\Phi} &= \sum_{\tau_i \in \Phi} b_T(\tau_i)\,\Delta l \\
I_{c\Phi} &= \sum_{\tau_i \in \Phi} T(\tau_i)\,\Delta l
\end{aligned}
$$

The location local fluid velocity, and aperture at the source are also recorded, along with the exit location which can subsequently be related to the biosphere receptor (lake, sea, mire etc.) in the landscape for risk calculations. For detailed models of transport along streamlines, the properties of features traversed by each particle are also recorded.

In some calculation cases the excavation-disturbed zone (EDZ) around the deposition tunnels forms an important path for transport. Hence particles released from a source S_i at one deposition hole may travel along the tunnel and arrive another deposition hole S_j before they continue along the way to the surface. The properties of the release paths represented by such particles can be found by convolution of the distributions of properties for paths from S_i to S_j with the distributions of paths S_j from to the surface.

6.3 *meshtrkr* particle file format

A ***meshtrkr*** particle file is simply a list of boundary groups from which particles should be launched. By convention, the names for particle files have suffix "`part`".

A particle file has the following format:

```
Particles Np
Source Group G1
:
Source Group Gn
```

where:

- N_p Number of particles to launch from each group.
- G_i Boundary ID for the *i*th boundary group to track particles from.
- n Number of groups to track particles from.

Example:

```
Particles 100
Source Group 310
Source Group 450
Source Group  12
```

This will track 100 particles from each of the three specified source groups (with boundary ID numbers 310, 450, 12). Note that the source groups need not be in sequence.

The current version of ***meshtrkr*** does not allow referencing source groups by the text-string names of the corresponding boundaries (e.g. "Deposition Hole A-120"). This limitation can be an inconvenience, and will hopefully be alleviated in future versions of ***meshtrkr***.

6.4 *meshtrkr* dispersivity file format

A ***meshtrkr*** dispersivity file defines dispersivities and other transport-specific values for different classes of features. By convention, the names for dispersivity files have suffix "`disp`". An example of the ***meshtrkr*** dispersivity file format is given in Table 6.1.

A dispersivity file contains statements of the following form:

```
Molecular diffusion      Dm
Taylor dispersion [on|off]
Transverse dispersion ratio rDT
Dispersivity range si aLi
```

where:

D_m	Molecular diffusion coefficient [units of L/T^2] .
r_{DT}	Ratio of transverse dispersivity to longitudinal dispersivity [dimensionless].
s_i	Maximum **feature set** number for the *i*th class of features.
a_{Li}	Longitudinal dispersivity α_L for the *i*th class of features [units of L].

The current version of ***meshtrkr*** does not check units, so all parameters should be in the default SI units.

The **`dispersivity range`** lines specify dispersivity values for **feature classes** which are groups of **feature sets** (the latter are defined in the mesh file). The specified value of dispersivity will be assigned to all of the features that belong to a feature set which in turn belongs to the feature class. The feature classes must be defined in order of increasing feature set numbers s_i:

Feature Class 1: $0 \le s \le s_1$
Feature Class 2: $s_1 < s \le s_2$
:
Feature Class N: $s_{N-1} < s \le s_N$

where s is the feature set number and N is the total number of feature classes defined.

If **`Taylor dispersion`** is set to **`off`**, (the default), then the DPRW algorithm described in Section 6.1 is used to model advective dispersion. If **`Taylor dispersion`** is set to **`on`**, then the explicit Taylor dispersion (advection-diffusion) algorithm described by Geier (2005) is used instead.

Table 6.1 Example of a dispersivity file for a calculation case in which major deformation zones belong to Feature Set 1, features representing a surficial layer of Quaternary deposits belong to Feature Set 2, features representing the disturbed-rock zone around repository tunnels belong to Feature Set 3, and 5 sets of stochastically defined fractures make up the remainder of the feature sets.

```
Molecular diffusion       2.0e-9
Taylor dispersion            off
Transverse dispersion ratio  0.1
Dispersivity range  1       10.0 # Major deformation zones
Dispersivity range  2        1.0 # Quaternary deposits
Dispersivity range  3        1.0 # Repository tunnels
Dispersivity range  8        1.0 # Single fractures
```

7 References

Ahlstrom, S. W., Foote, H. P., Arnett, R. C., Cole, C. R., and Serne, R.J., 1977. Multicomponent mass transport model. Theory and numerical implementation (discrete parcel random walk version), Battelle report BNWL for ERDA, Columbus, Ohio.

Berkowitz, B., Nauman, C., and Smith, L., 1994. Mass transfer at fracture intersections: An evaluation of mixing models. *Water Resources Research*, v. 30, p. 1765-1773.

Brantberger, M., Zetterqvist, A., Anbjerg-Nielsen, Olsson, T., Outters, N., and Syrjänen, P., 2006. Final repository for spent nuclear fuel: Underground design Forsmark, Layout D1. SKB Report R-06-34, Swedish Nuclear Fuel and Waste Management Co., Stockholm.

Dershowitz, 1984. *Rock Joint Systems.* Ph.D. dissertation, Massachusetts Institute of Technnology, Cambridge, Massachusetts.

Dershowitz, W. S., Lee, G., Geier, J., Foxford, T., LaPointe, P., and Thomas, A., 1996. FracMan™: Interactive discrete feature data analysis, geometric modeling, and exploration simulation: User Documentation, Version 2.5, Golder Associates Inc., Redmond, Washington.

Geier, J.E., 2005. *Groundwater flow and radionuclide transport in fault zones in granitic rock.* Ph. D. dissertation, Geosciences Department, Oregon State University, Corvallis, Oregon. Published as SKI Report 05:33, Swedish Nuclear Power Inspectorate, Stockholm.

Geier, J., 2008. Discrete-Feature Modelling of Groundwater Flow and Solute Transport for SR-Can Review. SKI Report 2008:11, Swedish Nuclear Power Inspectorate, Stockholm.

Janson, T., Magnusson, J., Bergvall, M., Olsson, R., Cuisiat, F., and Skurtveit, E., 2006. Final repository for spent nuclear fuel: Underground design Laxemar, Layout D1. SKB Report R-06-36, Swedish Nuclear Fuel and Waste Management Co., Stockholm.

Mardia, K.V., 1972. *Statistics of Directional Data*, Academic Press, London, 339 p.

Mardia, K.V., Kent, J.T., and Bibby, J.M., 1979. *Multivariate Analysis*, Academic Press, London, 521 p.

Munier, R., 2006. Using observations in deposition tunnels to avoid intersections with critical fractures in deposition holes. SKB Report R-06-54, Swedish Nuclear Fuel and Waste Management Co., Stockholm.

SKB, 2006. Long-term safety for KBS-3 repositories at Forsmark and Laxemar – a first evaluation. Main report of the SR-Can project, SKB report TR-06-09, Swedish Nuclear Fuel and Waste Management Co., Stockholm.

Snow, D. T., 1969. Anisotropic permeability of fractured media. *Water Resources Research* v. 5, no 6, p. 1273-1289.

Appendix 1: Supplementary utilities

This appendix gives brief descriptions of supplementary utilities that are included with this version of the DFM package.

Two of these utilities are provided as C-language source code, which must be compiled in order to produce executable programs:

showxpm	Simple digitization program that works with XPM images (useful for digitization of fracture traces from images).
splinter	Program for analysis of fracture maps (see Geier, 2005 for details).

Both programs make use of the open-source ***gtk*** toolkit, which is provided as part of most Linux releases. However, these codes make use of several older features of ***gtk*** that have been deprecated in newer versions of ***gtk***. Hence difficulties may be encountered, in trying to compile these codes using more recent Linux releases.

The remaining utilities are provided as Linux/Unix-compatible scripts. These scripts should be copied into a subdirectory named *~/bin/dfmscripts*, where ~ signifies the user's home directory. This directory should be appended to the Linux/Unix environment variable PATH.

Some of these scripts have been developed for specific applications of the ***dfm*** package in a particular modelling project, and may not necessarily be directly useful for other projects. Others have been developed primarily for debugging and checking of code functionality.

In general these scripts have not been designed for the general user of the DFM package. However, they may still serve as useful templates for customized scripts that the user might wish to develop for other modelling projects. Knowledge of C-shell (***csh*** or ***tcsh***) and AWK language programming techniques will be required. This is beyond the scope of the current document. However, resources on C-shell and AWK programming can readily be found on the Internet.

The following pages contain a list of these scripts with brief descriptions.

General-purpose utility scripts

cleanup	C-shell script to clean up carriage-return and line-feed characters which sometimes cause problems in text (data) files transferred from DOS/Windows to Linux operating systems.
lines	C-shell script to extract a range of lines from output.
pop	C-shell script used to print the first line a text file into *stdout*, and then delete this line from the file. This script is utilized by ***tripostx*** and used by numerous other scripts.
postscript	C-shell script to parse a file in DFM-DXF format into postscript-viewer format.

Mesh assembly scripts

rectify_topography.awk	AWK script used to enforce sign convention on ordering of topographic nodes (used in preparing topography files prior to mesh assembly).
parsepanels	C-shell script used to parse panel data from ***fracgen*** output.
presplit_fracs	C-shell script used to reduce size contrast among features in a panel file, by subdividing features larger than a specified radius, prior to mesh generation.
presplit_fracs.awk	AWK script used by ***presplit_fracs***.
tripostx	C-shell script to post-process triangulation data (see Section 4.2.1).
tripostx_finish	C-shell script to post-process triangulation data, used in cases where it is necessary to terminate a ***tripostx*** run prematurely (for example if a run is terminated to free up CPU capacity for other processes running on the same computer).
consolidate_triangulation	C-shell script used to consolidate triangulations produced by ***tripostx***.
consolidate_triangulation.awk	AWK script used by ***consolidate_triangulation***.

Particle-tracker pre- and post-processing scripts

getpartgrps.awk	AWK script to create source-group input for ***meshtrkr***.
extractarrivals	C-shell script to extract data on particle arrivals at the geosphere/biosphere interface, from a ***meshtrkr*** log file.
extractarrivals.awk	AWK script used by ***extractarrivals***.
formattracks	C-shell script to format particle-tracking data after extracting arrival data.
formattracks.awk	AWK script used by ***formattracks***.
processarrivals	C-shell script to post-process particle arrival data produced by ***meshtrkr***.
processarrivals.awk	AWK script used by ***processarrivals***.

Miscellaneous scripts for debugging and mesh diagnosis

countpanels.awk	AWK script to count the number of panels in a panel definition file.
dcxelement	C-shell script used to extract information on the nodes belonging to a specified finite element in a dcx file.
dcxelement.awk	AWK script used by ***dcxelement***.
dcxelemneighbors	C-shell script used to extract information on the neighbors of a specified finite element in a dcx file.
dcxnodeneighbors	C-shell script used to extract information on the neighbors of a specified node in a dcx file.
dcxnodeneighbors.awk	AWK script used by ***dcxnodeneighbors***.
dxfmeshplot.awk	AWK script used to produce color plots representing head and flux values from a ***dfm*** simulation.
extractpanel	C-shell script to extract panel vertex data for a single numbered panel, from a panel file.
extractpanel.awk	AWK script used by ***extractpanel***.
getpanel	C-shell script to plot the triangulation of a panel.
memchk	C-shell script used to check memory allocation and deallocation (debugging only).
meshcomplexity.awk	AWK script used to characterize the complexity of a mesh in terms of the maximum number of nodes connected to a given node.
nodes_used.awk	AWK script used to check for nodes in a mesh file that are not referenced for any elements.

parse_restart_mesh	C-shell script used to parse ***dfm*** input file from ***dfm*** output when convergence of the flow solution is judged to be inadequate.
parse_restart_mesh.awk	AWK script used by ***parse_restart_mesh***.
plotmesh	C-shell script used to produce a postscript-format plot from a triangulation file.
plotmesh.awk	AWK script used by ***plotmesh***.

Appendix 2: DFM Panel Files

Panel files are the primary way to represent the geometry and properties of discrete-features, prior to discretization into finite-element mesh files. They are the main form of output from the ***fracgen*** and ***repository*** modules, and the main form of input to the ***meshgenx*** module.

A panel file consists of two principal sections:

- Vertex list
- Panel list

The **vertex list** is simply a list of vertex coordinates headed by the keyword `Vertices`. The syntax is:

```
Vertices
1 x1 y1 z1
2 x2 y2 z2
 :
n xn yn zn
```

where (x_i, y_i, z_i), $i = 1, 2, ..., n$ are the three-dimensional coordinates of the n vertices in the list, listed in order of their vertex ID (the number in the first column). White space (space or tab characters) between these numbers are ignored. A pound sign (#) can be used to append a comment to one of the lines, but is not strictly necessary for that purpose, provided that the comment comes after the expected number of fields on a line.

The **panel list** consists of a list of panels (polygons) which, depending on the context, may represent portions of **boundary segments**, **features**, or **boundary faces** (a special class of boundary segments which also are treated as features). The basic structure of a panel list is:

```
Panels
[boundary definition(s)]
[feature definition(s)]
```

where the *boundary definition(s)* and *feature definition(s)* may be interchanged in order, or even interspersed.

A single **panel** $panel_i$ is represented by a list of vertex IDs on a single line:

$v_1\ v_2\ \ldots\ v_{Ni}$

where v_j is the vertex ID of the *j*th vertex belonging to this panel (in cyclic order), and N_i is the number of vertices in the panel. The number N_i must be at least 3 (since a panel with fewer vertices than a triangle would be degenerate), but otherwise can be arbitrarily large.

The cyclic order of the vertices (clockwise or counterclockwise) is arbitrary if the panel is part of (or an entire) discrete feature. If the panel is part of a boundary segment or boundary face, the order is significant as described below.

A ***feature definition*** has the form:

```
Feature f T Tf S Sf b bTf  [# label]
panel1
[panel2]
 :
[paneln]
```

where each $panel_i$ is a list of vertex IDs on a single line, as indicated above, and where:

f	feature ID
T_f	feature transmissivity [L^2/T]
S_f	feature storativity [-]
b_{Tf}	feature transport aperture [L]
label	optional text string to describe feature

Note that a feature may consist of just one panel, or more than one. If the feature consists of multiple panels, these need not have any vertices in common. However, in practice multiple-panel features (representing *e.g.* piecewise planar segments of curved features, or large features which have been prediscretized as multiple panels) will commonly share vertices along the edges where they join.

The feature properties defined on the line that begins with the keyword **`feature`** (i.e., transmissivity, storativity, and transport aperture) are assigned to each of the panels that follow this statement, until another **`feature`** or **`boundary`** statement is encountered. These properties may appear in any order, provided that the values of these properties are paired with their respective labels (**`T`**, **`S`**, and **`b`**). This format is designed to allow for extension of panel files to account for other feature properties that may be needed in some models (*e.g.* rock mechanics properties).

If a feature definition is preceded by the keyword **`Topography`**, it is treated as a topographic feature forming an upper boundary to the model. In this case, the ordering of vertices for panels belonging to this feature should follow the rules for boundary segments/faces, as described below.

A ***boundary definition*** has the form:

```
Boundary B
boundary [segment|face] definition(s)
```

where B is a unique positive integer to identify the boundary and each ***boundary [segment|face] definition*** is either a ***boundary segment definition*** or a ***boundary face definition***.

A ***boundary segment definition*** has the form:

```
Boundary Segment s
panel1
[panel2]
 :
[paneln]
```

where *s* is a segment identifier, and each each $panel_i$ is a list of vertex IDs on a single line, as indicated above.

A ***boundary face definition*** has the same form:

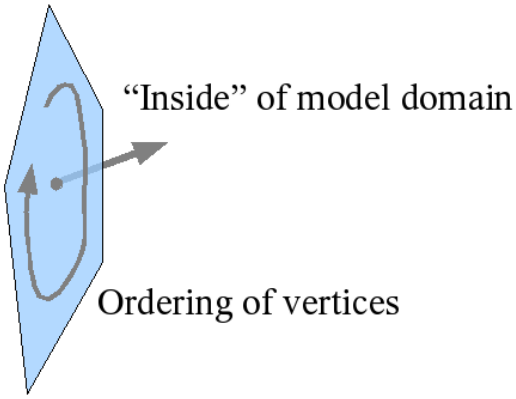
```
Boundary Face s
panel₁
[panel₂]
 :
[panelₙ]
```

Both **boundary faces** and **boundary segments** are used (in ***meshgenx***) to clip the portions of intersecting features that are "outside" the boundary that they represent, and to assign boundary group IDs to the resulting nodes in the mesh file. The convention used to determine which side of a boundary panel (whether part of a boundary face or segment) is as follows:

If viewed from the "inside" of the boundary (the domain within which features should be retained), the panel vertices are ordered in a *counterclockwise* direction (Figure A2.1). Another way to think of this is in terms of the "right-hand rule": If your right hand is oriented so that your fingers curl in the same direction as which the vertices on a panel are ordered, then your right thumb points toward the "inside" of this boundary.

The difference between **boundary faces** and **boundary segments** is that the polygon representing a **boundary face** is preserved as a transmissive feature (in ***meshgenx***), whereas the polygon representing a **boundary segment** is discarded after mesh generation.

a) Ordering of vertices on a single boundary panel.



b) Ordering of vertices on an internal boundary (smaller cube) within an external boundary (larger cube).

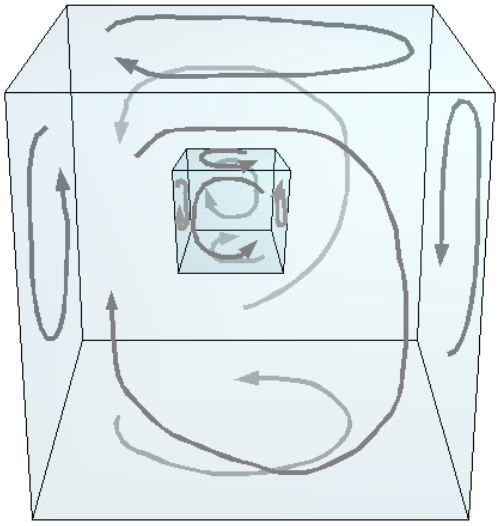


Figure A2.1 Ordering of vertices on boundary panels

Table A2.1 Example of a panel list.

```
Panels
Topography
Feature 1 T 1e-5 S 1e-6 b 0.2
Boundary Segment 1
11 50 17
60 66 43
66 78 43
104 98 12
  :
Boundary 1
Boundary Segment 1 # Top
4 39 43 42
Boundary Segment 2 # Bottom
5 59 63 62
59 14 60 63
  :
Boundary 2 # Deposition Hole
Boundary Face 8
7383 7382 7381 7380 7379 7378
7384 7385 7386 7387 7388 7389
7378 7379 7385 7384
7379 7380 7386 7385
7380 7381 7387 7386
7381 7382 7388 7387
7382 7383 7389 7388
7383 7378 7384 7389
  :
Feature Set 1
Feature 2 T 1e-05 S 1e-06 b 0.2 # ZFMNE0061
1676 2 1677 1680
1677 634 1678 1680
1678 637 1679 1680
1679 633 1676 1680
  :
```

Appendix 3: DFM-DXF Files

DFM-DXF files used in the DFM package are a modified form of the DXF format that was developed for older versions of the commercial AutoCAD™ program. With the DFM package, these are used principally for defining tunnel layouts for the ***repository*** module.

A DFM-DXF file as used here ordinarily consists of two main sections: a header (which specifies the *x-y* limits of the data as well as various line styles etc.), and a data section which defines a series of **polylines** (polygons, not necessarily closed, consisting of one or more line segments). Polylines are defined in just two dimensions. The basic format of a DFM-DXF file is illustrated in Table A3.1.

Table A3.1 Abbreviated example of a DFM-DXF file used to define access tunnels and deposition tunnels for a repository model.

```
BEGIN SECTION
BEGIN HEADER
$EXTMIN  -143.5000  2772.5000
$EXTMAX  3026.1250  4801.0000
$LIMMIN  -143.5000  2772.5000
$LIMMAX  3026.1250  4801.0000
FILLMODE
END HEADER
END SECTION
BEGIN SECTION
LABEL Access Tunnel 1
BEGIN POLYLINE
 2436.0000  4200.5000
 2199.6250  4514.0000
 2173.8750  4540.0000
 1921.3750  4585.5000
  716.3750  4382.5000
  393.2500  4127.5000
  379.3750  4058.5000
  507.5000  3822.0000
  547.0000  3805.5000
  583.7500  3810.5000
  851.5000  4016.0000
  945.1250  3961.5000
END POLYLINE
LABEL Deposition Tunnel 1-01
BEGIN POLYLINE
  419.0000  4149.0000
  562.0000  3938.0000
END POLYLINE
LABEL Deposition Tunnel 1-02
BEGIN POLYLINE
  454.8750  4173.0000
  595.1250  3960.5000
END POLYLINE
LABEL Deposition Tunnel 1-03
BEGIN POLYLINE
  482.6250  4204.0000
  627.2500  3990.0000
END POLYLINE
END SECTION
```

Appendix 4: Physical Units

The ***dfm*** module of the DFM package requires that physical parameters be specified in dimensionally consistent units. Input in non-SI units is automatically converted to SI units. This helps to avoid common errors from use of incompatible units.

Note: *Other modules in the package are less consistent in this regard, but more explicit checking of units is planned for future versions.*

In their most simple form, units may be expressed as one of the basic unit identifiers *u* recognized by the program, such as *u* = `Pa`, `ft` or `mm` (see below for a full list).

A positive integer *i* immediately following a basic unit identifer, in the form *ui*, indicates exponentiation of the unit to the indicated power. For example, `s2` indicates seconds squared, and `km3` indicates cubic kilometers.

Unit identifiers with or without exponents can be combined in more complex expressions, by using an asterisk (*) or a space between identifiers to indicate multiplication or a slash (/) to indicate division. For example, either `N*m` or `N m` indicates units of Newtons times meters, and `kg m/s2` indicates kilograms-meters per second squared.

Parentheses can also be used to define subexpressions which are interpreted by the usual rules of algebra. For example, a flux could be specified as `(m3/s)/m2` which is equivalent to units of `m/s`.

Indeterminate expressions arising from use of multiple slashes, or multiplication signs following a slash, without clarifying parentheses, are resolved by treating everything after the first slash as part of the divisor. For example `ft/min/kg` and `ft/min*kg` would both be interpreted as meaning `ft/(min*kg)`.

Output by default is in SI units. Conversions from user-specified units to the SI units used in calculations can thus be checked by inspecting the input parameters which are recorded as part of the output.

Table A4.1 List of basic unit identifiers. For each category, the SI units are indicated.

Time		
`s`	seconds	SI
`min`	minutes	
`hr`	hours	
`day`	days	
`week`	weeks	
`year`	years (365.25-day solar year)	
`yr`	years (365.25-day solar year)	
Mass		
`kg`	kilograms	SI
`g`	grams	
`slug`	slugs	
`lbm`	pounds mass	
Length		
`m`	meters	SI
`mm`	millimeters	
`cm`	centimeters	
`km`	kilometers	
`in`	inches	
`ft`	feet	
`yards`	yards	
`yard`	yards	
`miles`	miles	
`mile`	miles	
Area		
`m2`	square meters	SI
`hectare`	hectares	
`acre`	acres	

Table A4.1 List of basic unit identifiers. For each category, the SI units are indicated.

	`sqft`	square feet	
Volume			
	`m3`	cubic meters	SI
	`l`	liters	
	`ml`	milliliters	
	`cc`	cubic centimeters (= ml)	
	`gal`	U.S. gallons	
Force			
	`N`	newtons	SI
	`lbs`	pounds (force)	
	`lbf`	pounds (force)	
Energy			
	`J`	joules	SI
	`kWhr`	kilowatt-hours	
	`Btu`	British thermal units	
Power			
	`W`	watts	SI
	`kW`	kilowatts	
	`hp`	horsepower	
Pressure			
	`Pa`	pascals	SI
	`kPa`	kilopascals	
	`MPa`	megapascals	
	`psi`	pounds per square inch	
	`ksi`	kilopounds per square inch	
Viscosity			
	`Ns/m2`		SI
	`P`	Poise	

Table A4.1 List of basic unit identifiers. For each category, the SI units are indicated.

`cP`	centiPoise	
Flowrate		
`m3/s`	cubic meters per second	SI
`gpm`	U.S. gallons per minute	
Velocity or volumetric flux		
`m/s`	meters per second	SI
`kph`	kilometers per hour	
`mph`	miles per hour	
`fps`	feet per second	

Appendix 5: DCX Mesh File Format

The ***meshtrkr*** module of the DFM package requires that mesh data be provided in an older format known as DCX format, rather than the ***dfm*** mesh file format (as described in Section 5.2). The DCX format is defined as three header lines (the contents of which are ignored in ***meshtrkr***) followed by a list of node data, then two more header lines followed by the element data, as follows:

```
3 1.0
************************ NODE COORDINATES ************************
 Node #  x    y    z   type   h   q   grp
  1      x1   y1   z1  type1  h1  q1  g1
  .
  .
  M      xM   yM   zM  typeM  hM  qM  gM
***************************** FRACTURES ****************************
 Elem # n1   n2   n3    Set    T   S   b
  1     n11  n21  n31   set1   T1  S1  b1
  .
  .
  N     n1N  n2N  n3N   setN   TN  SN  bN
```

where:

M = number of nodes;
x_i = x coordinate of ith node [m], i = 1, 2, ..., M;
y_i = y coordinate of ith node [m];
z_i = z coordinate of ith node [m];
$type_i$ = nodal type for ith node;
h_i = head value at ith node [m];
q_i = nodal flux at ith node [m/s];
g_i = nodal group identification number for the ith node;
N = number of elements;
n_{1j} = first node of jth element, j = 1, 2, ..., N;
n_{2j} = second node of jth element;
n_{3j} = third node of jth element;
set_j = feature set to which jth element belongs;
T_j = transmissivity of jth element [m^2/s];
S_j = storativity of jth element [-];
b_j = transport aperture of jth element [m];

In practice, the DCX mesh file for ***meshtrkr*** input is typically parsed from the ***dfm*** mesh file that was used as input for flow calculations, plus the head and nodal flux values from ***dfm*** output, using scripts that the user writes for the specific application. The method is cumbersome and difficult to document for the general user. Therefore in future versions of the DFM package, the DCX format for ***meshtrkr*** input is planned to be replaced by an extended version of the ***dfm*** mesh file format.

www.ski.se

STATENS KÄRNKRAFTINSPEKTION
Swedish Nuclear Power Inspectorate

POST/POSTAL ADDRESS SE-106 58 Stockholm
BESÖK/OFFICE Klarabergsviadukten 90
TELEFON/TELEPHONE +46 (0)8 698 84 00
TELEFAX +46 (0)8 661 90 86
E-POST/E-MAIL ski@ski.se
WEBBPLATS/WEB SITE www.ski.se