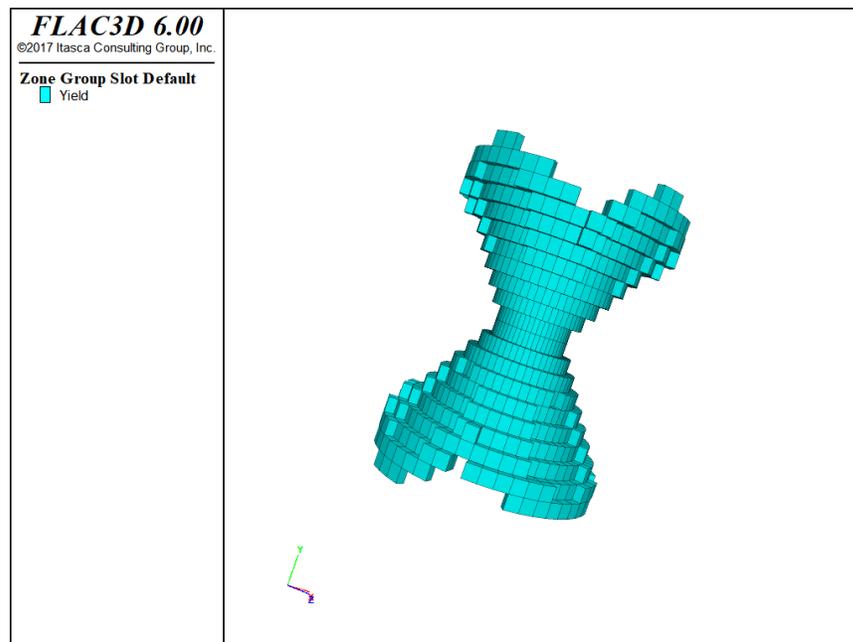


FLAC3D 6.0

FLAC3D Modeling

(an excerpt from *FLAC3D Help*)



First Edition (*FLAC3D* Version 2.1) April 2002
Second Edition (*FLAC3D* Version 3.0) September 2005
Third Edition (*FLAC3D* Version 3.1) December 2006
Fourth Edition (*FLAC3D* Version 4.0) December 2009
Fifth Edition (*FLAC3D* Version 5.0) October 2012
Sixth Edition (*FLAC3D* Version 6.0) April 2017

Precis

This document is a reproduction of the section “*FLAC3D Modeling*” from the *FLAC3D Help*. It is provided as a convenience to *FLAC3D* users who would prefer access to the documentation in a printable form.

There is *no substantial difference* between the content presented here and that which appears in the *FLAC3D Help*. In cases of variance, the difference is either a change made to accommodate format in going from a screen-based media to one intended for paper, or, since the *Help* is updated concurrent to code revisions, that the *Help* file content is more up-to-date than this document. In any case of variance between the two, precedence should *always* be given to the *Help*.

Printing Tip

This document uses a facing pages layout. Users who wish to print the document are advised that printing double-sided, if possible, will produce the best result. A double-sided copy can be bound or inserted into a ring-binder for ease of use.

This Document and *Help* Hyperlinks

The *FLAC3D Help*, from which this document is excerpted and produced, has many cross-reference hyperlinks. These are not reproduced here. Elements in the body-text of the *Help* that are hyperlinks are colored **dark green** in this document. This convention has been adopted to allow for link identification within the text of this PDF. To see the matter referenced, however, readers will need to access the equivalent link in the *Help* file.

FLAC3D Modeling: Table of Contents

INTRODUCTION	3
Overview	5
Why A Command-Driven Interface?	6
Why FISH?	9
The Lagrangian Finite Volume Grid	18
Nomenclature	19
FLAC3D Features	27
Features	27
Optional Features	28
Modeling Physical Processes and Interactions	29
New Features in Version 6.0	31
Comparison with Other Methods	43
Fields of Application	45
Modeling in Concept	49
Numerical Analysis, Geotechnical Engineering	49
Modeling Methodology	54
General Solution Procedure Illustrated	59
The Solution Procedure as a <i>FLAC3D</i> Project	60
About <i>FLAC3D</i> Commands	61
Command Scope & Overwriting	64
Targeting Commands with Ranges and Groups	65
Command Tools	67
Syntax Changes from Version 5.0 to 6.0	68
<i>FLAC3D</i> Program Layout	71
Files	73
<i>FLAC3D</i> Projects Introduced	76
Guide to the <i>FLAC3D</i> Help File.....	79
User Support	81
About Itasca Consulting Group, Inc.	83

TUTORIALS	84
Tutorial: Quick Start	84
Tutorial: Illustrative Model — Mechanics of Using <i>FLAC3D</i>	103
Create a New Project	107
Discussion	108
Projects: In the Interface	108
Generate the Grid	111
Discussion	111
Further Discussion: Meshing With Primitives	112
Meshing: In the Interface	118
Create Model Groups	121
Discussion	122
Further Discussion: Model Groups	123
Grouping: In the Interface	124
Specifying Models, Boundaries, and Initial Conditions	127
Discussion	127
Further Discussion: Assigning a Constitutive Model	129
Further Discussion: Applying Boundary Conditions	130
Further Discussion: Specifying Initial Conditions	134
Step to Equilibrium	137
Discussion	137
Further Discussion: Step to Equilibrium	138
Stepping: In the Interface	143
Alter Model: Excavation	145
Discussion	145
Further Discussion: Performing Alterations	147
Examine Model: Plotting	149
Discussion	150
Plotting: In the Interface	151
Further Alterations: Support	153
Discussion	154
Further Discussion: Using Save/Restore and Staged Modeling	154

Project Finishing	159
Discussion	159
Finishing: In the Interface	160
Tutorial: Working with FISH	163
Defining a FISH Function	163
Variables	164
Using FISH for Custom Calculations	166
FISH Rules, Syntax, and Statements, Illustrated	169
Arrays and Maps	178
Further Information	179
PROBLEM SOLVING WITH <i>FLAC3D</i>	181
Approach and Project Setup	187
Modeling on a Spectrum	188
Recommended Steps	189
Start a Project	192
Grid Generation	195
Primitive-Based Grids	197
Overview of the Grid Primitives	198
Connecting Adjoining Primitive Shapes	205
Fitting the Grid to Simple Shapes	208
Grid Generation with FISH	213
Extrusion-Based Grids	215
Create a Set	216
Define the 2D Geometry	216
Zoning and Other Operations	218
Define the Extrusion	218
Building Blocks-Based Grids	220
Create a Building Blocks Set	221
Snap Blocks Together	221
Refinement Operations and Utilities	224
Zoning	225

Grids from Outside <i>FLAC3D</i>	226
Grids from Rhino/Griddle	227
Grid Generation: Additional Facilities	230
Densifying Grids	230
Geometry-Based Densification: Octree Meshing	236
Surface Topography and Layering	239
Identifying Regions of the Model	247
Groups and Slots	247
Using the Group Range Element	249
Select and Hide	250
Using the Model Pane	253
Objects in the Model Pane	253
Visualizing Objects	253
Selection	255
Groups	258
Additional Commands	258
Working with Geometric Data	261
Geometric Data	261
Geometry Visualization	264
Geometry Painting	264
Geometric Filtering - Geometry Range Elements	265
Geometry Data and Group Assignment	268
Choice of Constitutive Model	271
Overview of Constitutive Models	271
The Constitutive Models in <i>FLAC3D</i>	274
Selection of an Appropriate Model	277
The Effect of Water	281
Ways to Implement Constitutive Models	282
Material Properties	285
Intrinsic Deformability Properties	285
Intrinsic Strength Properties	289
Post-Failure Properties	292

Extrapolation to Field-Scale Properties	302
Spatial Variation and Randomness of Property Distribution	308
Boundary Conditions	309
Stress Boundary	309
Applied Stress Gradients	312
Changing Boundary Stress	313
Cautions and Advice	317
Displacement Boundary	319
Local System and Applied Velocities	320
Surface Corners and Velocity Boundaries	322
Fix vs. Apply	324
Reaction Forces	324
Nonuniform Velocities	324
Real Boundaries — Choosing the Right Type	326
Artificial Boundaries	326
Symmetry Planes	326
Boundary Truncation	327
Initial Conditions	329
Uniform Stresses — No Gravity	330
Stresses with Gradients — Uniform Material	331
Stresses with Gradients — Nonuniform Material	334
Stress Initialization in a Nonuniform Material	335
Compaction within a Nonuniform Grid	339
Initial Stresses following a Model Change	342
Stress and Pore-Pressure Initialization with a Phreatic Surface	344
Initialization of Velocities	346
Reaching Equilibrium	349
Convergence Criteria	349
Maximum Out-of-Balance Force	350
Local Maximum Force Ratio	350
Average Force Ratio	351
Maximum Force Ratio	351

Ratio	351
Convergence	352
Choosing Convergence Criteria	353
Evaluating Equilibrium	354
Effects of Large Stiffness Differences	357
Loading and Sequential Modeling	361
Example: Loading on Three Tunnels	362
Structural Support	371
Interfaces	373
Tips and Advice	377
1. Check Model Runtime	377
2. Effects on Runtime	377
3. Considerations for Zoning Density	378
4. Automatic Detection of an Equilibrium State	378
5. Considerations for Selecting Damping	379
6. Check Model Response	379
7. Initializing Variables	380
8. Minimizing Transient Effects on Static Analysis	380
9. Changing Material Models	381
10. Running Problems with In-Situ Field Stresses and Gravity	381
11. Determining Collapse Loads	381
12. Determining Factor of Safety	381
13. Use Bulk and Shear Moduli	384
<i>FLAC3D</i> Runtime Benchmark	385
Interpretation	387
Unbalanced Force and Convergence	387
Gridpoint Velocities	388
Plastic Indicators	389
Histories	390
Project Completion	391
References	393

FLAC3D Modeling

This section serves the purpose of generally introducing *FLAC3D* and providing an overview of working with the program. Many of the subjects introduced here are more exhaustively documented in later parts of the *Help*. It is divided into three major parts.

- **Introduction**

This section, as its name implies, gives the user a first look at the program, its features, tools, and utilities, as well as its design and its uses.

- **Tutorials**

This section is optimal for the user who wants to dive right in. It contains tutorials that will provide step by step guidance on the most fundamental tasks involved in working with *FLAC3D*.

- **Problem Solving with *FLAC3D***

This section provides an up-close look at the methodology of sequential modeling that is used to construct the `flac3d model`.

With regard to the sections following this one, their contents may be summarized briefly as follows:

FLAC3D Elements - is a descriptive and programmatic reference for *all* *FLAC3D* objects, and the commands and *FISH* functions for working with them.

FLAC3D Program Interface - serves as a reference for how to perform tasks in *FLAC3D*, whether at the command line or in the interactive components of the interface.

FLAC3D Theory and Background - provides background information and descriptions of the theoretical basis for the basic components of *FLAC3D*.

FISH Scripting Reference - supplied a reference on the *FISH* scripting language in general. Note *FLAC3D*-specific *FISH* functions are described in the “elements” section.

Examples - provides access to example applications, verification problems, and all other examples that appear in the Help documentation.

Options - provides complete documentation on options.

FLAC3D Install & Resources - covers the information necessary to set up *FLAC3D*, how to obtain support, and a full revision history of this version of the program.

Introduction

Material in this section provides users with a first look at *FLAC3D*, covering a range of introductory topics about the program, such as its origin, basic use, its general design and capabilities, how it compares to other numerical methods, how it may be applied, and more. It is composed of the following sections.

- Overview
 - Why A Command-Driven Interface?
 - Why *FISH*?
 - The Lagrangian Finite Volume Grid
 - Nomenclature
- *FLAC3D* Features
 - Features
 - Optional Features
 - Modeling Physical Processes and Interactions
 - New Features in Version 6.0
- Comparison with Other Methods
- Fields of Application
- Modeling in Concept
 - Numerical Analysis, Geotechnical Engineering
 - Modeling Methodology
 - General Solution Procedure Illustrated
 - The Solution Procedure as a *FLAC3D* Project
- About *FLAC3D* Commands
 - Command Scope & Overwriting
 - Targeting Commands with Ranges and Groups
 - Command Tools
 - Syntax Changes from Version 5.0 to 6.0
- *FLAC3D* Program Layout
 - Files
 - *FLAC3D* Projects Introduced
- Guide to the *FLAC3D* Help File
 - User Support
- About Itasca Consulting Group Inc.

Overview

FLAC3D is a three-dimensional explicit Lagrangian finite-volume program for engineering mechanics computation. The basis for this program is the well-established numerical formulation used by our two-dimensional program, *FLAC*.^[1] *FLAC3D* extends the analysis capability of *FLAC* into three dimensions, simulating the behavior of three-dimensional structures built of soil, rock, or other materials that exhibit path-dependent behavior.

Materials are represented by polyhedral elements within a three-dimensional grid that is adjusted by the user to fit the shape of the object to be modeled. Each element behaves according to a prescribed linear or nonlinear stress/strain law in response to applied forces or boundary restraints. The material can yield and flow, and the grid can deform (in large-strain mode) and move with the material that is represented.

The explicit, Lagrangian calculation scheme and the mixed-discretization zoning technique used in *FLAC3D* ensure that plastic collapse and flow are modeled very accurately. Because no matrices are formed, large three-dimensional calculations can be made without excessive memory requirements. The drawbacks of the explicit formulation (i.e., small timestep limitation and the question of required damping) are overcome by automatic inertia scaling and automatic damping that minimizes effects on the path to failure. *FLAC3D* offers an ideal analysis tool for the solution of three-dimensional problems in geotechnical engineering.

FLAC3D is designed specifically to operate on Microsoft Windows systems and is currently supported on Windows 7, Windows 8, and Windows 10. Calculations on realistically sized three-dimensional models in geo-engineering can be made in a reasonable time period. For example, a model containing 125,000 zones of a Mohr-Coulomb material can be generated within 600 MB RAM. The runtime to perform 5000 calculation steps for a 10,000 zone model of Mohr-Coulomb material is roughly 30 seconds on a 3.2 GHz Intel 6-Core i7 CPU. The number of calculation steps required to reach a solution state with the explicit-calculation scheme can vary, but a solution typically can be reached within 3000 to 5000 steps for models containing up to 10,000 elements, regardless of material type. (The explicit-solution scheme is explained in the [Theoretical Background](#)

section.) With the advancements in floating-point operation speed and the ability to install additional RAM at low cost, it should be possible to solve increasingly larger three-dimensional problems with *FLAC3D*.

The *FLAC3D* program interface includes both command-line (CLI) and graphical (GUI) capabilities. All aspects of program operation are available via the CLI in the Console pane. Many *but not all* program operations can be performed in the program's GUI components. Some operations (plotting, working with extrusions or building blocks, etc.) will be comparatively easier in the GUI components, and others may be invoked more efficiently (or necessarily) at the command line. Many *FLAC3D* commands will be similar or identical to those found in the Itasca program *PFC*, and both programs share a common command syntax and structure.

FLAC3D features accelerated 3D graphics that allow for rapid model visualization. Extensive plotting capabilities—including easy mechanisms for constructing animations (movies)—are available to facilitate pre- and post-processing.

A comparison of *FLAC3D* to other numerical methods, a description of general features and updates in *FLAC3D* Version 6.0, and a discussion of the program's fields of application are provided in the topics that follow. If you wish to try *FLAC3D* right away, the best place to start is with the [Tutorials](#).

Why A Command-Driven Interface?

Whether generated interactively or manually, the basic description of any *FLAC3D* model is a data file. A data file is a standard text file containing commands that completely specify a *FLAC3D* model, from model creation to additional sequential operations required to undertake physical simulations.

At first exposure, the fact that *FLAC3D* uses data files to describe the model may seem antiquated, perhaps a relic of 1970s program design. To the contrary, this form of model description has proven to be quite valuable and is considered an integral part of what makes *FLAC3D*, and Itasca codes in general, powerful modeling tools. As our user interface design matures, our goal is not to remove the command-driven interface but, instead, to simplify its use, making simulations easier to undertake while retaining the flexibility this design embodies.

Below are a few reasons that using a data file description is advantageous.

- Compact representation

Even the most complex models can be created by a set of data files that are generally not more than a few hundred lines in length. In fact, the vast majority of models are far smaller. This fact means that the complete description of your *FLAC3D* model resides in a text file that is only a few kilobytes in size. As a result, it is trivial to share your model with others, email it to Itasca for support, archive your model for future reference, and use versioning software to track changes.

- Repeatability

Itasca works very hard to ensure that the same version of the code and the same data file will always produce the same result to machine precision. This means that when you send Itasca, your colleagues, or your clients a data file, you know that the result will be unchanging. Thus it is not necessary to archive the complete results of a modeling effort, which may be many gigabytes of save and result files. Instead, one can just retain the data file and the code version used to execute it.

- History and Path Dependence

Except for the most trivial models using the simplest of materials, the *path* used to reach the solution is a very important part of the model description. A data file allows the sequence of events to be described clearly and flexibly. Many programs may offer excavation sequence options, but the data file allows *any* sequence of events to be made if necessary. This includes changes to boundary conditions, changes to material properties, changes to fluid interactions, etc. as well as changes to the excavation sequencing. If one were to design a graphical user interface to include the entire list of options available via a data file, the result would be complex, requiring clumsy tools to edit and change.

- Flexibility

The data file allows the user maximum flexibility in how they choose to create and process a model, including the order in which things are specified. While there is a standard sequence of simulation steps recommended for simple models (e.g., geometry creation, naming of regions, material and property specification, boundary conditions, initial conditions, solving, excavating, solving, etc.), every model is different. Often complex models require modifications to the standard simulation progression. Itasca is committed to the idea of not constraining users to a small set of simulation options, providing users with the ability and tools to undertake physical simulations in the way they see fit.

- Scripting

The ability to combine model-creation commands with scripting in *FLAC3D* is tremendously powerful. For instance, an entire class of models can be investigated by trivially changing a set of initial parameters within a data file. Application of complex sequences, geometries, property distributions, etc. may be automated with a script in a way that would be very time consuming and difficult to replicate in a traditional graphical user interface. In addition, in-depth model querying and the inclusion of new physical phenomena, not included in *FLAC3D*, can be introduced via user-created scripts that execute during cycling.

The downside to such a command-driven interface is that it can be imposing for new users, who may have the impression that mastery of a large number of complex commands is necessary to undertake the simplest of modeling efforts. In truth, the learning curve is faster than being confronted by a complex graphical user interface that has numerous tools with a plethora of buttons in different panes—something we have all experienced. The commands have been purposefully structured using descriptive and consistent terminology, making it easy to read a data file and understand the operations it invokes. In addition, interactive command documentation is available as you create and edit a data file from within *FLAC3D*, simplifying the process substantially.

In order to make creating and editing data files as easy as possible, *FLAC3D* contains the following features:

- A powerful, built-in editor with syntax highlighting support.
- Updated command syntax, leading to increased clarity.
- CTRL-SPACE interactive inline help system.
- F1 context sensitive help.
- A comprehensive index of examples.

Why *FISH*?

Introduction

FISH is a built-in scripting language that gives the *FLAC3D* user powerful control over most every aspect of program operation. *FISH* is short for “FLAC-ISH” (or the language of *FLAC*), the code for which it was first developed. Now, in addition to *FLAC* and *FLAC3D*, *FISH* is also integrated into all Itasca commercial programs.

FISH is embedded deeply into *FLAC3D* at nearly every level. It can be used to parameterize data files so that a number of varying cases can be built into the same basic model. Every data type that makes up a *FLAC3D* model is also available for *FISH* to manipulate directly—before, after, and during the solution. This means that not only can *FISH* be used to create custom complex models and customized results, it can also be used to add custom physics to the solution process that is not part of the standard package.

FISH includes constructs to embed *FLAC3D* commands within *FISH* functions (see the command – end command block in the example shown in the [Run Control](#) section).

FISH is a semi-compiled language that uses dynamic typing for variables – syntax and use is similar to (but not exactly the same as) Python. It has been created to be very simple for small needs, but it provides structure and data types needed to support large and complex programs if necessary.

The following illustrations give just an idea of the power of *FISH*. More complete information is available in the sections [Tutorial: Working with FISH](#) and [FISH Scripting Reference](#). Examples below are given to represent five general areas of application:

- [Model Creation](#)
- [Parameterization](#)
- [Custom Visualization](#)
- [Physics Extension](#)
- [Run Control](#)

Model Creation: Stress Initialization

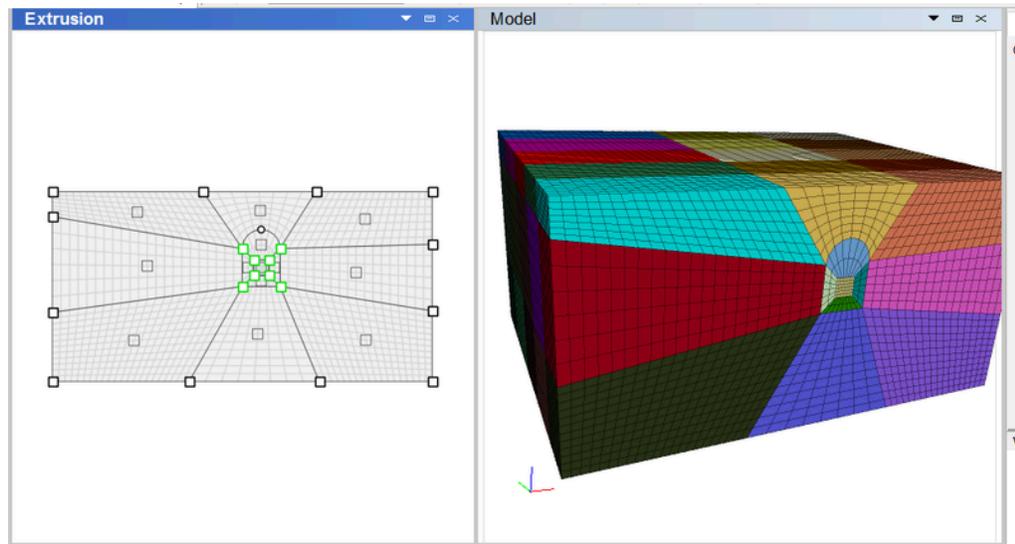
The following *FISH* routine specifies a lateral stress coefficient K_0 in a model that has reached initial equilibrium under gravity.

```
fish define ini_stress(k0)
  loop foreach zone zone.list
    vert = zone.stress.zz(zone) + zone.pp(zone) ; Vert. effective stress
    horz = vert * k0 ; Horiz. effective stress
    zone.stress.xx(zone) = horz - zone.pp(zone) ; Total Horizontal stress
    zone.stress.yy(zone) = horz - zone.pp(zone)
  end_loop
end
@ini_stress(0.3)
```

This is a particularly simple example for the purposes of illustration. The `zone initialize-stresses` command will determine an initial vertical stress due to gravity, and also initialize horizontal stresses. For complex geometries this initialization is inexact, and cycling to exact equilibrium will still be necessary. This function can be used to correct the horizontal stresses back to the K_0 desired.

Parameterization: Varying Model Geometry

For this example, we create a simple model of a tunnel interactively using the [2D extrusion tool](#). The tool's data and the resulting zones are shown below. Note that the points used for tunnel are marked in green as a group named `Tunnel`.



Using this as a template, we use the **State Record** pane to create a data file named `create_tunnel.dat` that reproduces this geometry. Then we can create a parameterized data file that allows us to test various tunnel locations while only changing one value (`tunnel_height` defined on line 3).

```

model new
; Specify model parameters
[tunnel_height = 20.0]
call 'create_tunnel' ; Load template of model geometry
; The template created has the tunnel base at 25,
; and tunnel points are marked with group 'Tunnel'
; Move points
extrude point transform translate 0 [tunnel_height-25.0] range group 'Tunnel'
; Generate zones
zone generate from-extruder

```

As seen on line 8, this example demonstrates the use of inline *FISH* (*FISH* fragments) that allow a single line of *FISH* to be embedded and evaluated anywhere within a data file.

Note that things like material properties, tunnel cross-section, and almost anything involved in the model can be parameterized in such a fashion—sometimes simply with parameter replacement in the data file as seen here, sometimes with the use of *FISH* functions (as seen above in the **Model Creation** section). This allows quick and easy exploration of the parameter space of a model that requires set up once and only once.

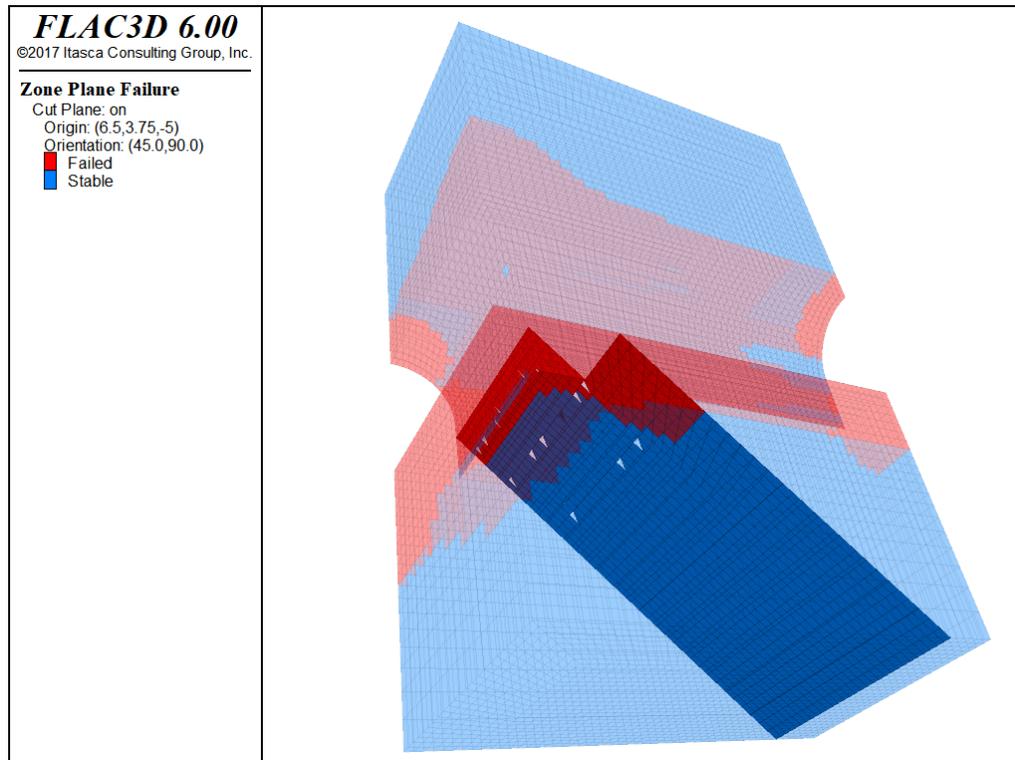
Custom Visualization: Positing a Plane of Weakness

Perhaps the most common use of *FISH* is to customize model results. Providing this ability was the original motivation for its introduction into Itasca software. *FISH* allows the user to plot any quantity of interest in the model without requiring the addition of a bewildering variety of rarely used options on a menu tree somewhere. The following is an example of a script that calculates, over the entire model, the normal and shear stress components on a particular plane orientation. It also creates a flag (*fail*, line 9) that indicates if failure would occur on that plane, given a specific friction angle (*tau_crit*, line 8).

```
fish automatic-create off
fish define shear_normal(dip,dd,fric_ang)
  local result = array.create(5)
  local norm = math.normal.from.dip.ddir(dip*math.degrad,dd*math.degrad)
  loop foreach local zone zone.list
    zone.plane.traction(zone,norm,result)
    local normal_stress_magnitude = result(1)
    local shear_stress_magnitude = result(2)
    local tau_crit = ...
      math.abs(normal_stress_magnitude)*math.tan(fric_ang*math.degrad)
    local fail = shear_stress_magnitude > tau_crit

    zone.extra(zone,1) = fail
    zone.extra(zone,2) = normal_stress_magnitude
    zone.extra(zone,3) = shear_stress_magnitude
  endloop
  array.delete(result)
end
@shear_normal(45,90,20)
```

Compressive stresses are negative in *FLAC3D* (note the definition of *tau_crit* on line 8). The following is a plot of the result, looking at a cut-away plane at the same orientation, from the [Pillar Loads At Intersecting Tunnels](#) example problem. Note that *FLAC3D* legend entries can be customized to make the content of the plot clearer—in this case *fail* evaluates to Boolean values (*true*, *false*), but on the plot legend these are re-labeled *Failed* and *Stable*.



Physics Extension: Ground Freezing

FISH functions can be called during cycling/solving to solution. *FISH* can be invoked at any point during each calculation cycle to override or to add additional physics to the model. Each cycle is one iteration of the steps in the explicit solution scheme; placement of *FISH* within that sequence can be specified by command.

The following is a simple illustration of a *FISH* function, `ground_freezing`, that could be used to model ground freezing during a coupled thermo-mechanical analysis.

```
call 'freeze_zone'
fish define ground_freezing
  loop foreach local zone zone.list
    if zone.group(zone,'state') == 'frozen' then
      continue ; Skip zones that are already frozen
    endif
    if zone.temp(zone) > 0.0 then
      continue ; Skip zones that are above freezing
    end_if
    freeze_zone(zone)
  endloop
end
```

This calls a *FISH* function `freeze_zone` (definition below) that actually changes the stress and stiffness of the zone and marks it as frozen.

```
fish define freeze_zone(zone)
  local porosity = zone.fluid.prop(zone, 'porosity')
  ; Note: Assuming fully saturated
  local expansion = porosity * 0.09 * 1.0
  ; Porosity * water expansion * saturation
  local bulk = zone.prop(zone, 'bulk')
  local stress_inc = bulk * expansion
  ; Amount to increment stress
  local bulk_inc = (8.96/2.16) * porosity
  ; Ratio of ice/water bulk * porosity
  zone.prop(zone, 'bulk') = bulk + bulk_inc
  zone.stress.xx(zone) = zone.stress.xx(zone) - stress_inc
  ; Compressive negative!
  zone.stress.yy(zone) = zone.stress.yy(zone) - stress_inc
  zone.stress.zz(zone) = zone.stress.zz(zone) - stress_inc
  zone.group(zone, 'state') = 'frozen'
end
```

Freezing could be added to the solution processing with a command like the following.

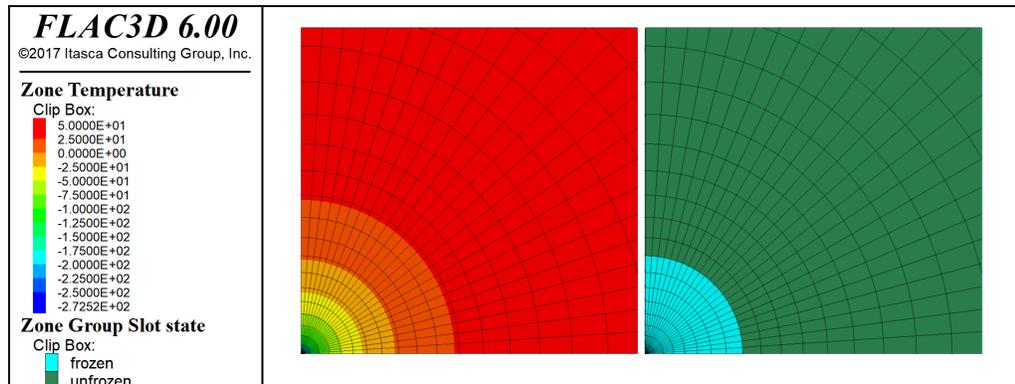
```
fish callback -100 @groundfreezing interval 10
```

This inserts a call to the `ground_freezing` *FISH* function at the start of every 10th solution step (to limit the effect on solution time). The same thing can be done as part of the `solve` command, as in the example (see lines 33–35 in the fragment below). A command such as the following would solve to 10 seconds of thermal time with a ground freezing check every 10 cycles.

```
model solve time-total 10.0 ...
  fish-call -100 @groundfreezing interval 10
```

The distinction between the two formulations shown above is that the former is global and will take effect with any subsequent `model solve` command; the latter specifies execution of *FISH* only for that particular issuance of model solve.

The following image displays the results when run on a modified version of the [Infinite Line Heat Source in an Infinite Medium](#) example problem.



```

model new
model configure thermal fluid
; --- model geometry
zone create cylinder point 1 (500,0,0) ...
                        point 2 (0,1,0) ...
                        point 3 (0,0,500) ...
                        size (48,1,24) rat 1.1 1 1
zone face skin slot 'uniform' ; Label model boundaries
zone group 'unfrozen' slot 'state' ; Initialize state marker
; --- mechanical model
zone cmodel assign elastic
zone property bulk 5e10 shear 3e10 density 2e3
zone gridpoint fix velocity-x 0 range group 'West' ; Boundary Conditions
zone gridpoint fix velocity-y 0 range group 'North' or 'South'
zone gridpoint fix velocity-z 0 range group 'Bottom'
; --- thermal model
zone thermal cmodel isotropic
zone thermal property conductivity 4 expansion 5e-6 specific-heat 1e3
zone gridpoint initialize temperature 50
zone gridpoint fix source -200 range pos-x 0 pos-z 0 ; Apply thermal sink
; --- fluid model
zone fluid cmodel isotropic
zone fluid property porosity 0.3
; --- settings
zone fluid active off ; No fluid coupling for this example
zone thermal implicit on
model thermal timestep fix 6.48e3
model save 'line-year0'
; --- Add ground freezing
call 'ground_freezing.f3fis'
; --- Coupled analysis
model mechanical slave on
model solve time-total [360*24*60*60] ... ; 1 year (360 days) of heating.
      mechanical ratio 1e-3 ... ; Relax solve requirements a bit for speed.
      fish-call -100 @ground_freezing interval 10 ; Call ground freezing
model save 'line-year1'

```

This example is very simple but still potentially useful. More accuracy can be added, depending on the needs of the model. For example, more properties can be modified, the constitutive model can be changed to one that supports creep, and the heat of fusion required to change water to ice can be taken into account.

Run Control: Yield at Multiple Joint Angles

The following is a simplification of **Uniaxial Compressive Strength of a Jointed Material Sample**. Eighteen model runs are made under *FISH* control to create a plot of failure stress versus joint angle (which is increased by five degrees with each run, starting from 0). The entire data file required is shown below:

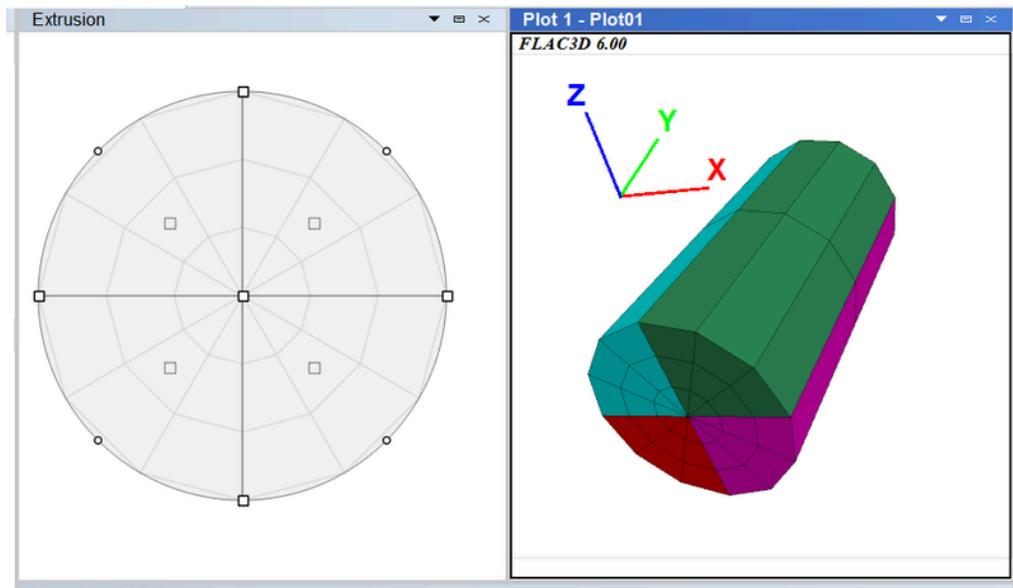
```

model new
call 'cylinder' ; Load extruder description of model geometry
call 'measure' ; Defines 'measure_stress', which measures surface stress
                  ; and stores in table

fish define triax_solver
  loop local k (0,18)
    local beta = k * 5.0
    command
      zone delete ; Destroy all zones
      zone generate from-extruder ; Create new zones
      zone cmodel ubiquitous-joint ; Assign constitutive model
      zone property bulk 1e8 shear 7e7 cohesion 2e3 ; Assign properties
      zone property friction 40 dilation 0 tension 2400
      zone property dip @beta dip-direction 0
      zone property joint-cohesion 1e3 joint-friction 30
      zone property joint-dilation 0 joint-tension 2000
      zone face apply velocity-y 1e-7 range position-y 0.0 ; Assign
      zone face apply velocity-y -1e-7 range position-y 4.0 ; boundary
                                                                ; conditions
      model cycle 6000 ; Cycle to get failure stress
    end_command
    measure_stress(beta)
  end_loop
end
@triax_solver
model save 'triax1'

```

Before the data file is run, the 2D extrusion tool is used to create a cylinder cross-section that is extruded in the *y*-direction. The extrusion set and the zones created from it are shown below.

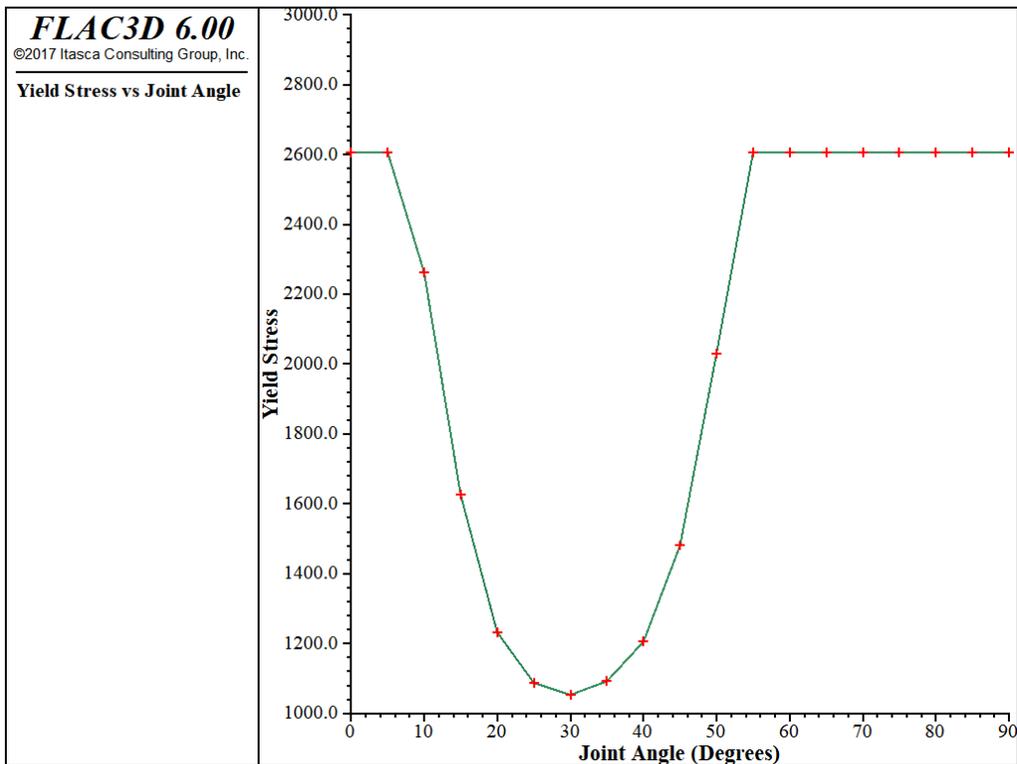


Next the **State Record Pane** is used to create a data file that recreates this result (the file is named `cylinder.f3dat` — called in the first line of the data file above). Next a *FISH* function that measures stress at the end of the cylinder and stores the result in a table is written and saved as `measure.f3fis` (this is called on the second line of the data file above and is shown at the end of this example, for reference).

With these two preliminaries in hand, the following data file can be run.

```
fish define measure_stress(angle)
  local force = 0.0
  loop foreach local gp gp.list
    if gp.pos.y(gp) # 0.0 then
      continue ; Skip all but bottom grid-points
    end_if
    force = force - gp.force.unbal.y(gp) ; Accumulate force
  end_loop
  local stress = force / (math.pi*math.pi) ; Area of unit circle
  table('result',angle) = stress
end
```

The figure below is produced from the result table.



Note that in the actual verification problem the applied boundary conditions use a *FISH* servo to optimize convergence accuracy and the result is compared against an analytical solution calculated via *FISH*.

The Lagrangian Finite Volume Grid

The Lagrangian finite volume grid spans the physical domain being analyzed. The smallest possible grid that can be analyzed with *FLAC3D* consists of only one zone. Most problems, however, are defined by grids that consist of hundreds, thousands, or millions of zones.

One *FLAC3D* zone is a hexahedron with eight vertices and six quadrilateral faces. Four other zone types are available (wedge, pyramid, d-brick, and tetrahedron), as degeneracies of this basic zone using fewer vertices and faces. See [Zone](#) for a discussion of zone data and conventions, and [Theoretical Background](#) for the theory, formulation, and implementation details.

The *FLAC3D* grid is specified in terms of global x -, y -, and z -coordinates. All gridpoints and zone centroids are located by their (x,y,z) position vector. A simple cubic grid is shown in the figure below. A set of ready-made (templated) grids can be accessed from the File ▸ Gridmenu option.

Every gridpoint and zone is identified by an identification number (ID), which can be used as a handle to refer to that specific object.

Grid generation with *FLAC3D* involves adjusting and shaping the mesh to fit the shape of the physical domain. The generation process may be performed in a variety of ways, such as using commands that build zones from **primitive shapes**, using the program's interactive **extrusion** or **building blocks** facilities, or using **advanced mesh generation** techniques with third-party tools.

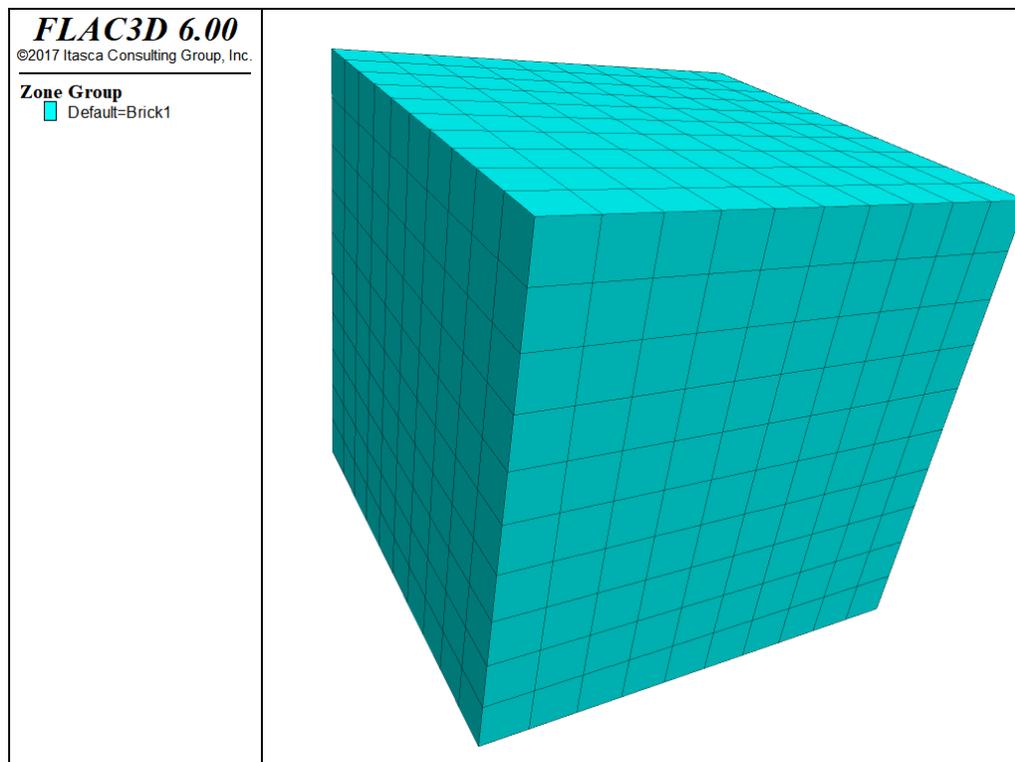


Figure 6: Finite difference grid with 1000 zones.

Nomenclature

FLAC3D uses nomenclature that is consistent, in general, with that used in conventional finite difference or finite element programs for stress analysis. The basic definitions of terms are reviewed here for clarification. The figure below illustrates the *FLAC3D* terminology.

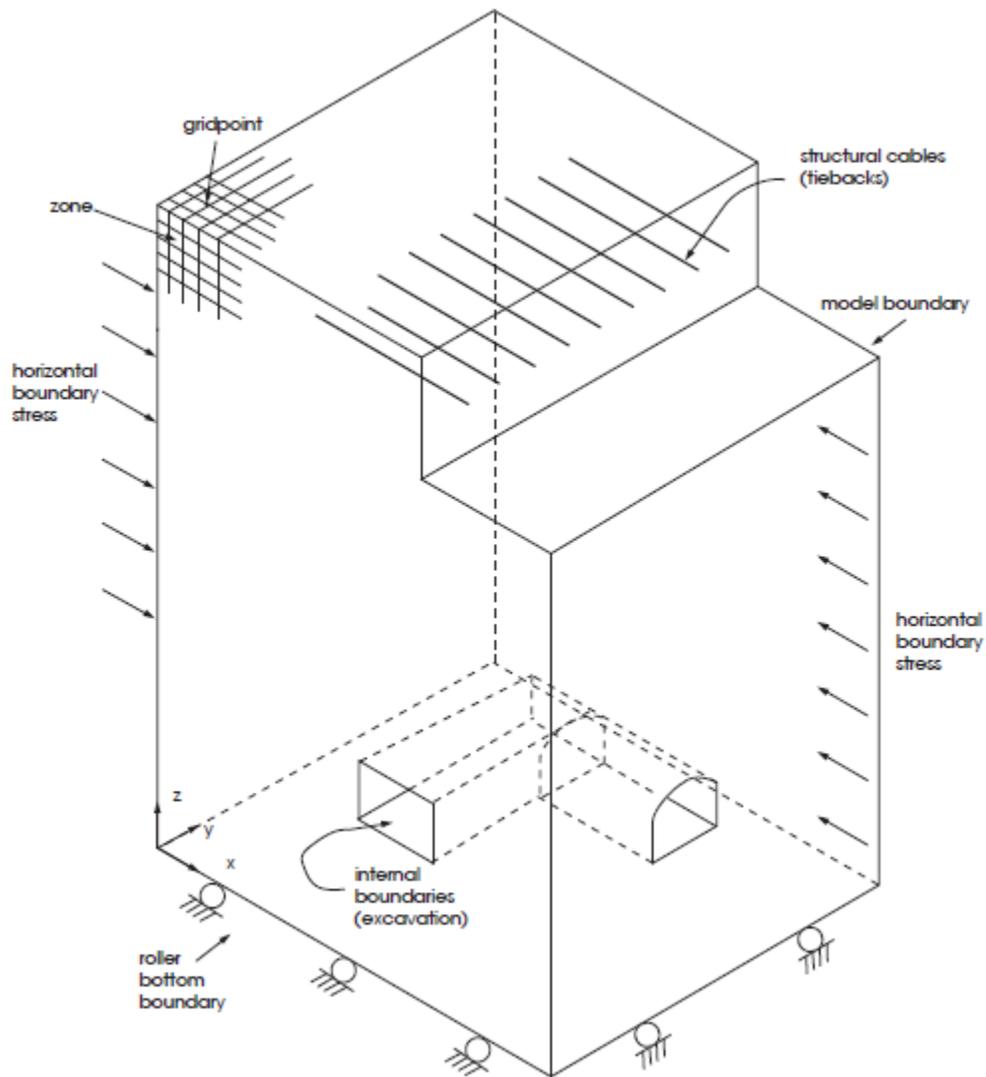


Figure 7: Example of a *FLAC3D* model.

model

The model (or *FLAC3D* model) is created by the user to simulate a physical problem. When referring to a *FLAC3D* model, the user implies a sequence of *FLAC3D* commands (see the [FLAC3D Elements](#) section) that define the problem conditions for numerical solution.

zone

The finite difference zone is the smallest geometric domain within which the change in a phenomenon (e.g., stress versus strain) is evaluated. Polyhedral zones of different shapes (e.g., brick, wedge, pyramid and tetrahedral-shaped zones) are used to create models and can be viewed upon plotting.

Each polyhedral zone contains two overlaid sets of five tetrahedral subzones, but the user is not normally aware of these. Another term for zone is element, but this is generally reserved for structural elements for clarity.

gridpoint

Gridpoints are associated with the corners of the finite difference zones. There are four, five, six, seven, or eight gridpoints associated with each polyhedral zone, depending on the zone shape. A set of x -, y -, z -coordinates is assigned to each gridpoint, thus specifying the exact location of the finite difference zones. Other terms for gridpoint are nodal point and node, but these are generally reserved for structural elements for clarity.

finite volume grid

The finite volume grid is an assemblage of one or more finite volume zones across the physical region that is being analyzed. Another term for grid is mesh. The finite difference grid also identifies the storage locations of all state variables in the model. The procedure followed in *FLAC3D* is that, in general, all the vector quantities (e.g., forces, velocities, and displacements) are stored at gridpoint locations, while all scalar and tensor quantities (e.g., stress and material properties) are stored at zone centroid locations. See [The Lagrangian Finite Volume Grid](#) topic for more information.

model boundary

The model boundary is the periphery of the finite difference grid. Internal boundaries (i.e., holes within the grid) are also model boundaries.

boundary condition

A boundary condition is the prescription of a constraint or controlled condition along a model boundary (e.g., a fixed displacement or force for mechanical problems, an impermeable boundary for groundwater flow problems, adiabatic boundary for heat transfer problems, etc.).

initial conditions

This is the state of all variables in the model (e.g., stresses or pore pressures) prior to any loading change or disturbance (e.g., excavation).

constitutive model

The constitutive (or material) model represents the deformation and strength behavior prescribed to the zones in a *FLAC3D* model. Several constitutive models to assimilate different types of behavior commonly

associated with geologic materials are available in *FLAC3D*. Constitutive models and material properties can be assigned individually to every zone in a *FLAC3D* model.

null zone

Null zones are zones that represent voids (i.e., no material present) within the finite difference grid.

subgrid

The finite difference grid can be composed of sub-grids. Sub-grids can be used to create regions of different shapes in the model (e.g., a dam sub-grid can be placed on a foundation sub-grid).

attached faces

Attached faces are grid faces of separated sub-grids that are attached or joined together. Attached faces must be coplanar and touching, but gridpoints on each face do not have to match. Sub-grids of different zone densities can be attached.

interface

An interface is a connection between sub-grids that can separate (e.g., slide or open) during the calculation process. An interface can represent a physical discontinuity such as a fault, contact plane, or interface between two different materials.

range

A range in a *FLAC3D* model is a filter that restricts the operations of a command to objects that fall within the definition of the filter. A range *may* or *may not* be associated with *specific* model objects (for example, a range specified in terms of ID numbers or groups will be associated with specific model objects, but a range specified in terms of x -, y -, and z -coordinates remains fixed in space, so the objects within it at one point of model processing may not be the same as the objects in it at a later point).

range element

A criterion in a range. Any range will have at least one range element, but may be composed of many range elements, may be designed to be a union or an intersection of the range elements supplied, and may invert any element provided by appending `not` at the end of the range element's specification.

group

A group in a *FLAC3D* model refers to a name that can be assigned to one or more objects. Group names are assigned in a category, called *slots*, with the constraint that in a given object, each slot can only have one group assigned at a time. An object may have many different group names, each in different slots. As compared with a range, groups will always refer to specific model objects. Groups provide a way to persistently “name” parts of the model, and subsequently operate with ease on those parts of the model by applying that name to the range keyword in commands (e.g., consider `zone cmodel assign mohr-coulomb range group "Layer1"`, which, as can be guessed, assigns the Mohr-Coulomb model to the zones in the group “Layer1”).

id number

Individual elements of a *FLAC3D* model are identified by identification (ID) numbers. Almost all model elements have ID numbers, including interfaces, gridpoints, zones, histories, tables, and structural element entities (i.e., beams, cables, piles, shells, liners, and geogrids). These are unique numbers that are assigned by the code allowing the user to identify specific elements in a model. Elements that have IDs supply a `list` keyword that is used to return ID-sorted data about the object (for example, try `zone list information`).

Component identification (`component-id`) numbers are also assigned to individual elements of a structural element entity. A unique component-id number is created for each node, element, and node/grid link that make up a beam, cable, pile, shell, liner, or geogrid entity.

names

For certain types of model element objects (like histories, tables, or interfaces) it is convenient to assign a clear short user-assigned name to an object so it can be referred to later. In general, this is done using a name, which is normally assigned when the object is created. A name is meant to be a short identifying string, although technically it can be of any length. As a special case, an integer can be used on the command line as a name, in which case it will be converted automatically into a string.

structural element

Two types of structural elements are available in *FLAC3D*. Two-noded, linear elements represent the behavior of beams, cables, and piles. Three-noded, flat triangular elements represent shells, liners, and geogrids. Structural

elements are used to simulate the interaction of structural support within a soil or rock mass. Nonlinear material behavior can be represented with some of the elements.

Each structural element entity (beam, cable, pile, shell, liner, or geogrid) is composed of three components: nodes; individual elements; and node/grid links. The characteristics of each of these components distinguish the behavior of the beam, cable, pile, shell, liner, and geogrid entities.

step

Because *FLAC3D* is an explicit code, the solution to a problem requires a number of computational steps. During computational stepping, the information associated with the phenomenon under investigation is propagated across the zones in the finite difference grid. A certain number of steps is required to arrive at an equilibrium (or steady-flow) state for a static solution. Typical problems are solved within 2000 to 4000 steps, although large complex problems can require tens of thousands of steps to reach a steady state. When using the dynamic analysis option, `model step` refers to the actual timestep for the dynamic problem. Other terms for step are timestep and cycle.

static solution

A static or steady-state solution is reached in *FLAC3D* when the rate of change of kinetic energy in a model approaches a negligible value. This is accomplished by damping the equations of motion. At the conclusion of the static solution stage, the model will either be at a state of equilibrium or at a state of steady flow of material if a portion (or all) of the model is unstable (i.e., fails) under the applied loading conditions. This is the default calculation in *FLAC3D*.^[2] Static mechanical solutions can be coupled to transient groundwater flow or heat transfer solutions. (As an option, fully dynamic analysis can also be performed by inhibiting the static solution damping.)

unbalanced force

The unbalanced force indicates when a mechanical equilibrium state (or the onset of plastic flow) is reached for a static analysis. A model is in exact equilibrium if the net nodal-force vector (the resultant force) at each gridpoint is zero. The maximum nodal-force vector is monitored in *FLAC3D* and printed to the screen when the `model step` or `model solve` command is invoked. The maximum nodal force vector is also called the *unbalanced* or *out-of-balance* force. The maximum unbalanced force will never exactly

reach zero for a numerical analysis, but the model is considered to be in equilibrium when the maximum unbalanced force is small compared to the total applied forces in the problem. If the unbalanced force approaches a constant nonzero value, this probably indicates that failure and plastic flow are occurring within the model. When a gridpoint is fixed in a given direction, the component of the unbalanced force in that direction is equivalent to the reaction force.

dynamic solution

For a dynamic solution, the full dynamic equations of motion (including inertial terms) are solved; the generation and dissipation of kinetic energy directly affect the solution. Dynamic solutions are required for problems involving high frequency and short duration loads (e.g., seismic or explosive loading). The dynamic calculation is an optional module for *FLAC3D* (see [Dynamic Analysis](#)).

large strain/small strain

By default, *FLAC3D* operates in small-strain mode; gridpoint coordinates are not changed even if computed displacements are large (compared to typical zone sizes). In large-strain mode, gridpoint coordinates are updated at each step according to computed displacements. In large-strain mode, geometric nonlinearity is possible.

project

A *project* refers to the collection of data files and other inputs, and save files and other outputs, that comprise a *FLAC3D* model. A project so-defined is stored in a project file. The terms “project” and “project file” are so nearly synonymous that they are likely to be used somewhat interchangeably in this documentation.

primitive

A templated shape that can be used with the `zone create` command to create a range of *FLAC3D* grids. The shapes are keywords of the command; refer to the reference information on the command for details on primitives.

element

See [zone](#).

mesh

See [finite volume grid](#).

console

A generic reference to the *Console* pane in *FLAC3D*.

Console pane

This pane is divided into two parts. The lower part of the pane is the command prompt, where users may enter commands one at a time. The upper part of the pane (variously referred to as the output, the output window, the console output, etc.) echoes commands and prints written output resulting from command processing. The *Options* dialog provides options for which output and how much of it will be displayed in the output.

command prompt

The lower portion of the *Console* pane, which can be used to enter commands line by line.

Endnotes

- [1] Itasca Consulting Group Inc. *FLAC (Fast Lagrangian Analysis of Continua)*, Version 8.0, 2014.
- [2] In some finite element (FE) literature, there is the mistaken notion that a dynamic solution method cannot produce a true equilibrium state, compared to an FE solution, which is believed to perfectly satisfy the set of governing equations at equilibrium. In fact, *both* methods only satisfy the equations approximately, but the level of residual errors can be made as small as desired. In *FLAC3D*, the level of error is objectively quantified as the ratio of unbalanced force at a gridpoint to the mean of the set of absolute forces acting at the gridpoint. This measure of error is very similar to the convergence criteria used in FE solutions. In both cases, the solution process is terminated when the error is below a desired value.

FLAC3D Features

Features

FLAC3D offers a wide range of capabilities to solve complex problems in mechanics, and especially in geomechanics. Like *FLAC*, *FLAC3D* embodies special numerical representations for the mechanical response of geologic materials. The program has fifteen basic built-in material models: the “null” model; three elasticity models (isotropic, transversely isotropic, and orthotropic elasticity); and eleven plasticity models (Drucker-Prager, Mohr-Coulomb, strain-hardening/softening, ubiquitous-joint, bilinear strain-hardening/softening ubiquitous-joint, double-yield, modified Cam-clay, cap-yield, elastic plastic (hyperbolic type) for soils, Hoek-Brown, and modified Hoek-Brown). These models are described in detail in the section [Constitutive Models](#). Each zone in a *FLAC3D* grid may have a different material model or property, and a continuous gradient or statistical distribution of any property may be specified.

Additionally, an interface (or slip-plane) model is available to represent distinct interfaces between two or more portions of the grid. The interfaces are planes upon which slip and/or separation are allowed, thereby simulating the presence of faults, joints or frictional boundaries. The interface model is described in the section [Interfaces](#).

Structures that interact with the surrounding rock or soil, such as tunnel liners, piles, sheet piles, cables, rock bolts or geotextiles, may be modeled with the structural element logic in *FLAC3D*. It is possible to either examine the stabilizing effects of supported excavations, or to study the effects of soil or rock instability on surface structures. The different types of structural elements are described in [sel](#).

A factor of safety can be calculated automatically for any *FLAC3D* model using a compatible material model. The calculation is based on a “strength reduction technique” that performs a series of simulations while changing the strength properties to determine the condition at which an unstable state exists. A factor of safety that corresponds to the point of instability is found, and the critical failure surface is located in the model. The factor-of-safety algorithm is described in the [Factor of Safety](#) section.

FLAC3D also contains a powerful built-in programming language, *FISH*, that enables the user to define new variables and functions. *FISH* offers a unique capability to users who wish to tailor analyses to suit their specific needs. For example, *FISH* permits the following:

- user-prescribed property variations in the grid (e.g., nonlinear increase in modulus with depth);
- plotting and printing of user-defined variables (i.e., custom-designed plots);
- implementation of special grid generators;
- servo control of numerical tests;
- specification of unusual boundary conditions;
- variations in time and space; and
- automation of parameter studies.

An introduction to *FISH* is given in [Tutorial: Working with FISH](#). See the section [FISH Scripting Reference](#) for a detailed reference to the *FISH* language.

FLAC3D contains extensive graphics facilities for generating plots of virtually any problem variable. Three-dimensional graphics rendering is provided in high-resolution video modes. Plotting features include hidden surface plots, surface contour plots, and vector plots. Plotted variables can be viewed in front of, behind, or on an arbitrary cross-section plane through the model. Interactive tools for model visualization are described in [Plotting \(the View Pane\)](#).

Optional Features

Four optional features (for dynamic analysis, thermal analysis, modeling creep-material behavior, and C++ plugins) are available as separate modules that can be included in *FLAC3D* at an additional cost per module.

Dynamic analysis can be performed with *FLAC* by using the optional dynamic-calculation module. User-specified acceleration, velocity, or stress waves can be input directly to the model, as either an exterior boundary condition or an interior excitation to the model. *FLAC3D* contains absorbing and free-field boundary conditions to simulate the effect of an infinite elastic medium surrounding the model. The dynamic calculation can be coupled to the groundwater flow model; the level of coupling, including dynamic pore-pressure generation (liquefaction), is discussed in [Modeling Physical Processes and Interactions](#). The dynamic analysis capability is described in [Dynamic Analysis](#).

There is a thermal analysis option available as a special module in *FLAC3D*. This model simulates the transient flux of heat in materials and the subsequent development of thermally induced stresses. The thermal model can be run independently or coupled to the mechanical–stress calculation or pore–pressure calculation, either static or dynamic. (The coupling interactions are described in [Modeling Physical Processes and Interactions](#).) The thermal analysis capability is described in [Thermal Analysis](#).

There are optional material models available that simulate time–dependent (creep) material behavior (all creep models are described in [Creep Constitutive Models](#)).

With the C++ plugin feature, user–defined constitutive models and *FISH* intrinsics can be written in C++ and compiled as DLL (dynamic link library) files that can be loaded whenever needed. Microsoft Visual C++ Version 10.0 (Microsoft Visual Studio 2010) is used to compile the DLL files. The procedure to write new constitutive models is described in [Writing New Constitutive Models](#).

Modeling Physical Processes and Interactions

The default calculation mode in *FLAC3D* is for static mechanical analysis. Alternatively, a groundwater flow analysis or a heat–transfer analysis can be performed independent of the mechanical calculation. Both the groundwater flow and thermal models may be coupled to the mechanical stress model and to each other. Because the full equations of motion are used in *FLAC*, the coupling mechanisms operate in dynamic analyses as well as static analyses.

The coupling mechanisms are divided into three types of interaction: mechanical and groundwater flow; mechanical and thermal; and thermal and groundwater flow. The level of interaction modeled in *FLAC3D* for each type is described below.

Mechanical–Groundwater Flow Coupling — Several types of fluid/solid interaction can be specified in *FLAC3D*. One type of interaction is consolidation, in which the slow dissipation of pore pressure causes displacements to occur in the solid (e.g., soil). Two mechanical effects are at work in this case: 1) the fluid in a zone reacts to mechanical volume changes by a change in the pore pressure; and 2) the pore–pressure change causes changes in the effective stress that affect the response of the solid (e.g., a reduction in effective stress may induce plastic yield). Coupling between fluid and solid due to deformable grains can also be specified.

FLAC3D can calculate pore–pressure effects, with or without pore–pressure dissipation, simply by setting the flow calculation on or off. Also, dynamic pore–pressure generation (e.g., related to liquefaction) can be modeled by accounting for irreversible volume strain in the constitutive model. This is done with two different built–in constitutive models: the “Finn” model, and the “Byrne” model. Both models are provided with the dynamic option.

By default, porosity and permeability are assumed constant. However, these properties can be made a function of volumetric strain via a *FISH* function. As a consequence, two–way coupling of mechanical stress and groundwater flow can be modeled with *FLAC3D*.

Other types of interaction, such as capillary, electrical, or chemical forces between particles of a partially saturated material, are not modeled directly by *FLAC3D*. But some of the effects may be included by providing suitable fish functions. Similarly, a *FISH* function may be used to vary the local fluid modulus as a function of other quantities such as pressure or time.

Thermal–Mechanical Coupling — The thermal–mechanical coupling in *FLAC3D* is one–way: temperature change may induce a mechanical stress change as a function of the thermal–expansion coefficient. Mechanical changes in the body, however, do not result in temperature change or changes to thermal properties. Additionally, mechanical properties can be made a function of temperature change, since *FISH* permits access to both temperatures and properties.

Thermal–Groundwater Flow Coupling — The thermal calculation may be coupled to the groundwater flow calculation by making pore pressure a function of temperature change. Volumetric strain can arise from thermal expansion of both the fluid and the grains within a saturated matrix. Pore pressure change results from this volumetric strain, as well as from mechanical volumetric strain. Groundwater flow can also influence heat transfer; an advection model that takes the transport of heat by convection into account is provided. The advection model can also simulate temperature–dependent fluid density and thermal advection in the fluid.

As with mechanical properties, groundwater properties can be made a function of temperature change by accessing temperature and property values via *FISH*.

Reference

Byrne, P. “A Cyclic Shear–Volume Coupling and Pore–Pressure Model for Sand,” in *Proceedings: Second International Conference on Recent Advances in Geotechnical Earthquake Engineering and Soil Dynamics (St. Louis, Missouri, March, 1991)*, Paper No. 1.24, 47–55 (1991).

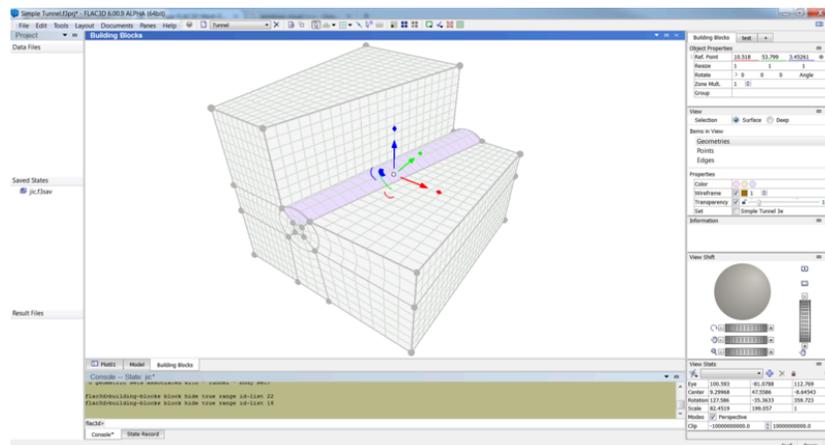
New Features in Version 6.0

Summary of Updates from Version 5.0

FLAC3D 6.0 contains many improvements; the new features are summarized in the following sections. Existing data files created for Version 5.0 will not operate as before. *FLAC3D* 6.0 will not be able to restore files saved by previous versions. For conversion of data files from version 5.0, see the topic [Data File Conversion](#) and use *FLAC3D* 5.0 to *FLAC3D* 6.0 Command Mapping and *FLAC3D* 5.0 to *FLAC3D* 6.0 FISH Mapping for reference.

Mesh Generation

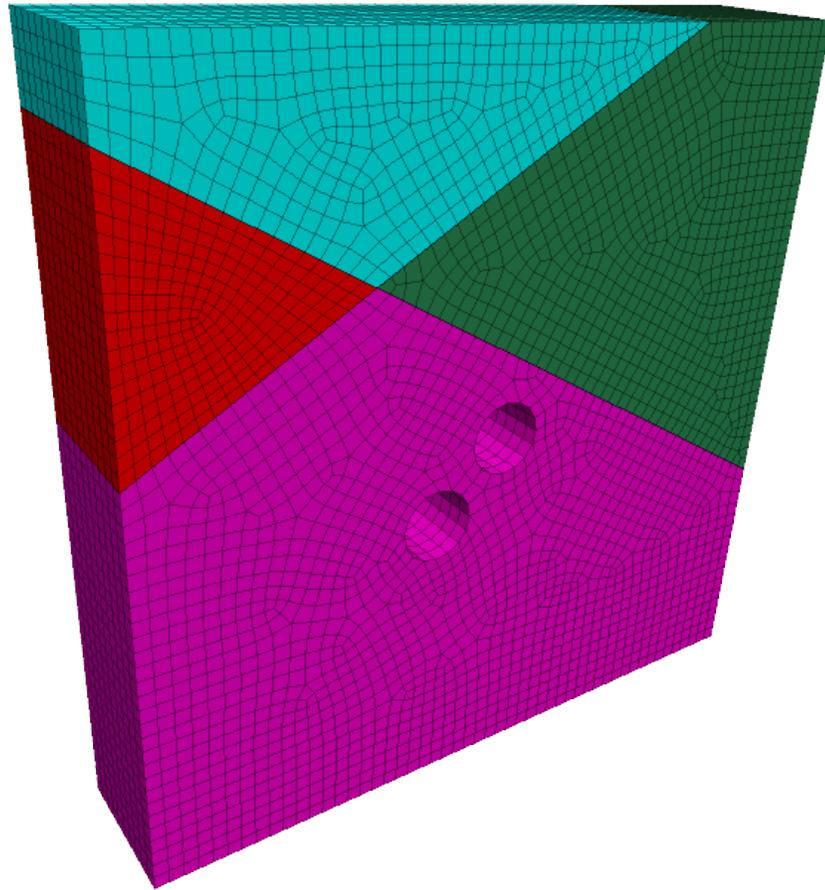
- Building Blocks



A new interactive structured mesh generation tool, the **Building Blocks Pane**, has been added to further increase the options available for mesh generation in *FLAC3D*. Building Blocks is akin to a 3D version of the **Extruder Pane**. The following are among the list of features supported:

- Use geometric surfaces (imported from CAD or otherwise) as background data.
 - Snapping points to geometric surfaces, edges, and nodes.
 - Draping selected faces and edge control points on curved geometric surfaces.
 - Easy selection of nodes, edge, faces, and blocks.
 - Arbitrary transformations of selected objects, including translation, rotation, and scaling.
 - Per edge control of zone discretization and distribution.
 - Automatic propagation of zone quantity and distribution to preserve connectivity.
 - Per-block zone multiplier, with automatic attach conditions generated across boundaries.
 - Blocks may be hidden temporarily for visualization.
 - Blocks may be split to generate more surfaces for manipulation.
 - Blocks may be copied and pasted, both within and between sets.
 - Library of predefined shapes and configurations, either to serve as a starting point or to cut and paste into your model.
 - Automatic zone size specification based on total number of zones or zone edge size.
 - The Extruder can send its description to Building Blocks for 3D modifications to a primarily 2D layout.
 - An initial Building Blocks representation can be automatically generated, given a starting closed geometric volume.
 - Building Blocks can be imported from a *FLAC3D* mesh, allowing a crude volume decomposition to be created automatically by a third-party program (or *Griddle*) and then customized and detailed interactively.
- Importing Meshes

In addition to importing *FLAC3D* grid files, *FLAC3D* can now directly import mesh files in the ANSYS and ABAQUS formats. This allows the use of any number of third party mesh generation programs that can export their results in one of those formats.



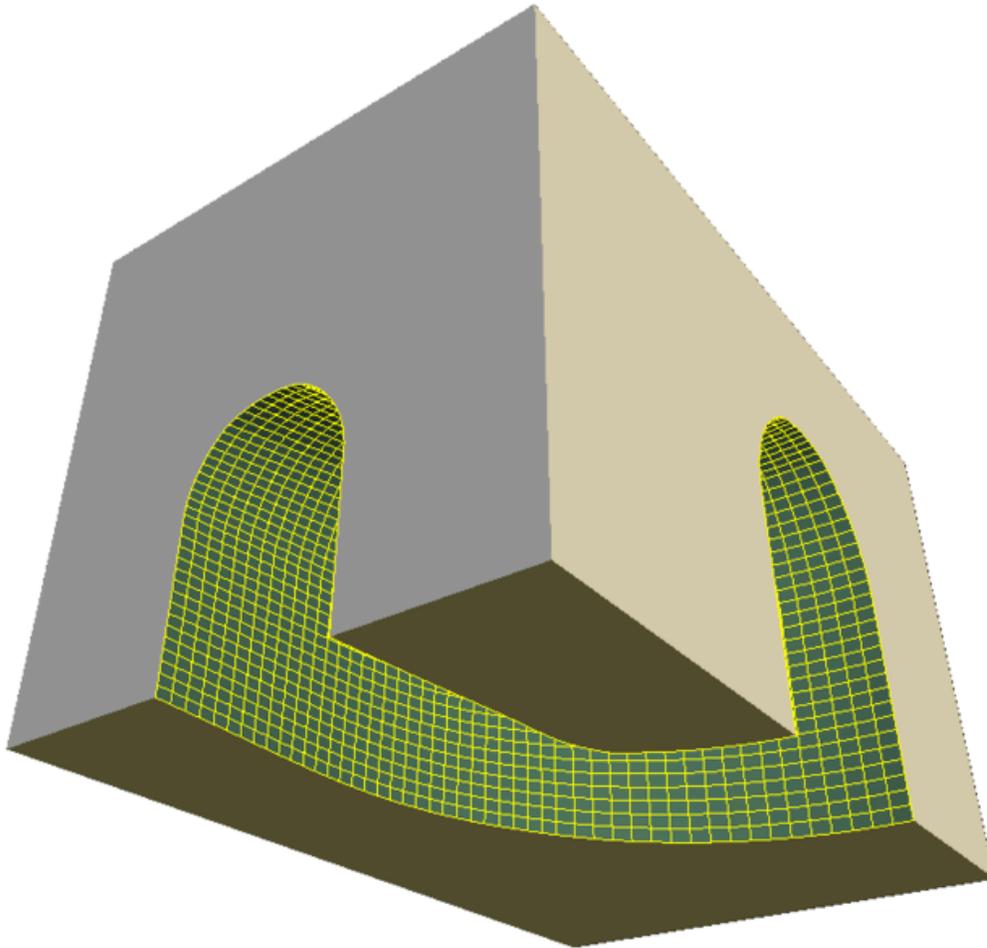
- Oct-tree and Densified Meshes

In the past, it has been difficult and time-consuming to create interfaces along faults in a mesh created using an oct-tree strategy and densification. The `zone separate by-face` command has been updated to allow clean separation of partially connected surfaces. In addition, the `zone attach by-face` command has been improved to more reliably connect the resulting grid. Finally, the `zone validate` command has been added to assist in detecting and visualizing areas that still may have not been correctly connected.

Interactive Modeling

A new interactive model manipulation pane has been created, called the **Model Pane**. This allows you to interactively view, select, name, and apply operations to zones and zone faces. As we continue to work on *FLAC3D*, more and more interactive tools will be created to support more stages of model creation.

Every change to the the model state done in this way is emitted as a command and is stored in the *State Record*, which can be used to create a data file for later recreation or parameterization of a base version created interactively.

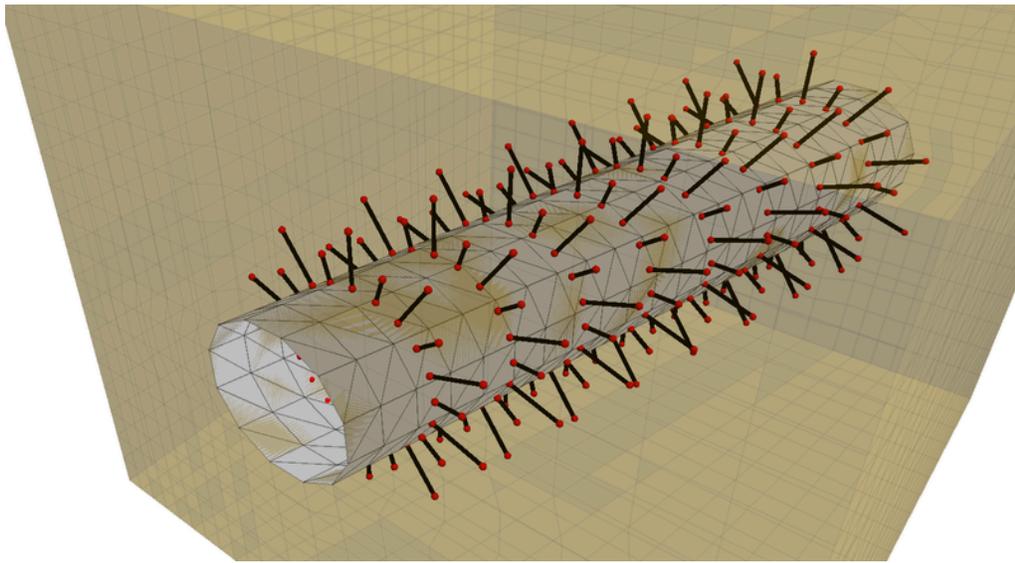


Structural Elements

The structural element logic in *FLAC3D* has particularly benefited from the improvements to the command syntax; it now uses the same conventions and keywords as the rest of the code. In addition, many options have been added to make the creation of structural supports easier. These include:

- Proper cable pre-tensioning can now be done with simple commands, rather than requiring *FISH*.

- The ability to create both 1D (cables, piles, beams) and 2D (shells, liners, geogrids) from geometric and CAD data. See, for example, the `structure cable import` command.
- The ability to separate an internal zone surface and install liners between them in one operation. See the `structure liner create by-face` command.
- The ability to automatically attach cables created to liner nodes. See the `structure cable create` command.
- **Model pane** features can be used to easily select complex curved surfaces on which to place structural reinforcement.



New Constitutive Models

In addition to ongoing incremental improvements to the behavior and response of existing constitutive models, the following new models are available in *FLAC3D* 6.0:

- **Plastic-Hardening** for modeling soils.
- **Swell** for wetting-induced deformations.
- **Ubiquitous-Anisotropic** that combines anisotropic elasticity with weak joint failure.
- **Mohr-Coulomb-Tension** that creates a “virtual” crack under tension, which can open and close for more realistic tensile behavior under dynamic loading.
- **Cap-Yield** for modeling soils.

- **Power-Mohr-Ubiquitous** that combines Mohr-Coulomb failure, ubiquitous joints, and power law creep.

Plus the ability to create your own constitutive behavior through **C++ User-Defined-Models** is easier than ever, with Visual Studio templates provided to get you started. As always, the source for all built in constitutive models is provided both for verification of implementation behavior and to serve as a basis for user modifications and extensions.

Commands and Scripting

The command syntax of *FLAC3D* has been updated for consistency and clarity. All commands attempt to conform to the standard pattern of **NOUN - VERB - OPTION - MODIFIERS - RANGE**. Commands have keywords using a new hyphenated syntax that allows the full command to be documented and listed for ease of understanding, but still allows abbreviation for those who do not want to type the entire word(s). Care has been taken to ensure that the same keyword is used consistently for the same concept throughout the code, making it much easier to transfer knowledge gained from one area of the code to another.

A new *FISH* intrinsic naming convention has also been implemented. This greatly increases the clarity of function names and allows the editor to automatically highlight both correct and incorrect intrinsic names as you type.

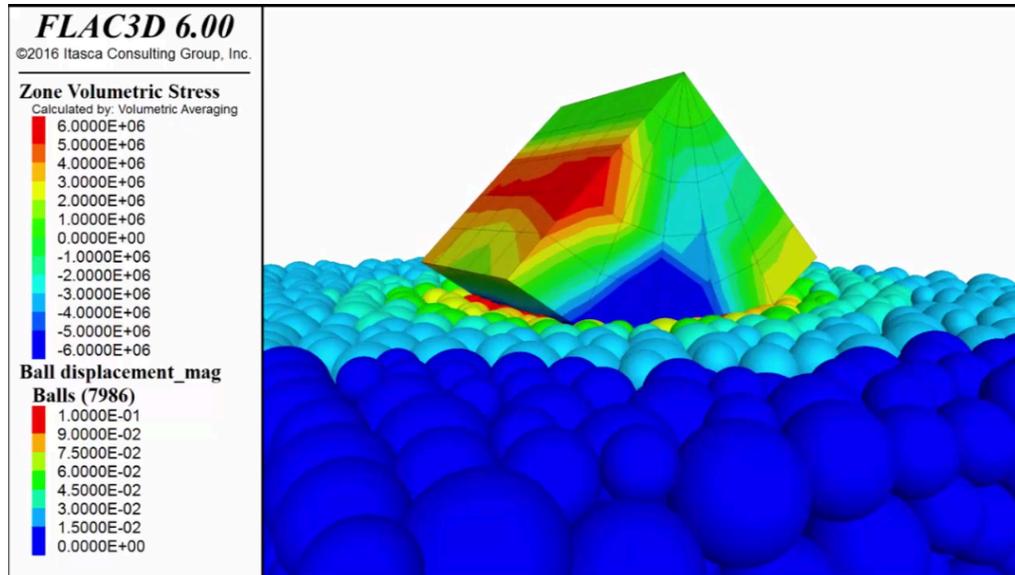
To assist users with data files and *FISH* they would like to continue to use, there is a conversion utility built into the *FLAC3D* 6.0 editor. This uses a heuristic to guess if a data file uses old 5.0 syntax and automatically prompts the user if they would like it to be converted to 6.0 syntax. This conversion can also be activated manually.

Many new types have also been added to *FISH*, including Boolean, Tensor, Matrix, and an Associative Array or Map. It is also much easier now to create custom C++ *FISH* intrinsics using the plugin option by creating a project from a Visual Studio template.

FLAC3D also features the addition of a new, much more powerful built-in editor. Features such as line numbers, code-folding, improved searching, and improved syntax highlighting are now available.

PFC Module Available

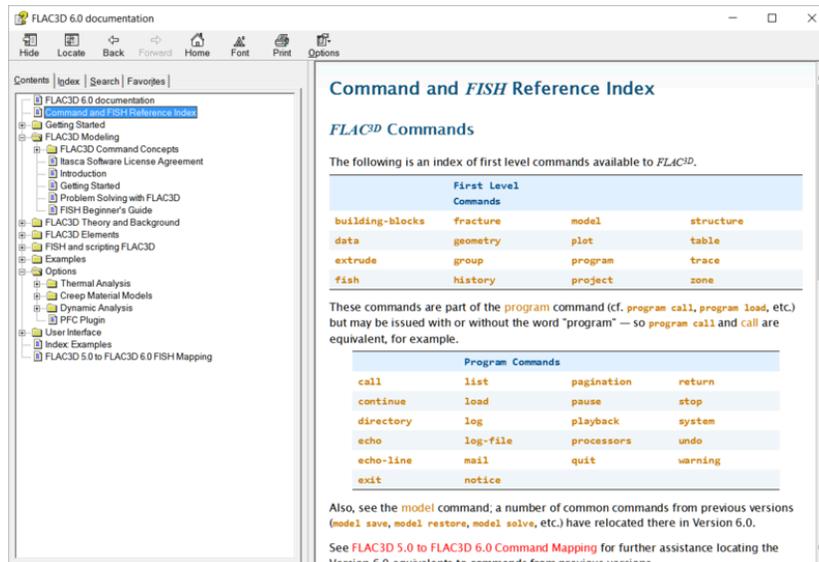
Itasca is moving toward a common platform where all of its software methods—currently sold as separate codes (*FLAC3D*, *PFC*, and so on)—can be loaded into a common system at run time. Starting with *FLAC3D* 6.0, a compatible version of *PFC* can be loaded at run time directly into *FLAC3D*, allowing direct coupling between balls and zones or structural elements. Preliminary direct coupling support has already been built in.



Interactive Help

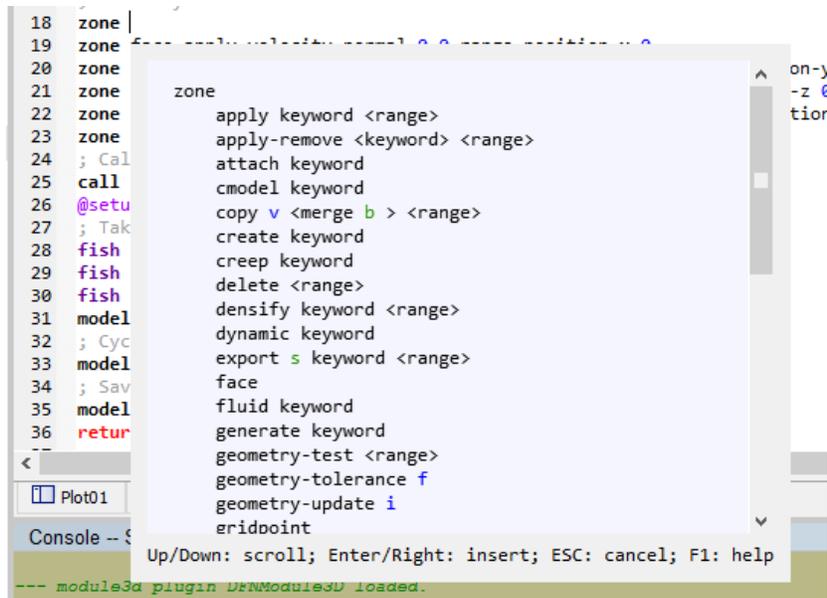
- On-screen Help

The entire documentation set is now available on-screen in a *Help* file. This permits users to rapidly search and explore content, review and copy examples, and makes additional integrated help facilities possible.



- Inline Help

By pressing **Ctrl+Spacebar**, a new inline help facility is activated that reminds you of the options available and helps you build a command interactively.



When working in a data file or at the command prompt, for instant help at any time, pressing **F1** will open *FLAC3D Help* to the full documentation for the command or keyword at the cursor position.

Further Additions and Updates

This is far from a complete list of additions and improvements. Some of what has not been described above includes:

- Faster initial conditions

The `zone initialize-stresses` command has been added, which automatically calculates initial approximations to the vertical stresses due to gravity even with irregular geometry and varying densities. The horizontal stresses may also be initialized to a ratio of the vertical. Other options are available.

- Update save/restore system

Save files (while incompatible with 5.0) are now using a new format that should allow for backward compatibility with future versions of *FLAC3D*.

- Automatic Zone Excavation Relaxation

The `zone relax` command has been added to automatically “zonk” excavations and keep spurious inertial effects from increasing the damage seen. The effects of the zones are gradually reduced over time, using a multiplier that reduces from 1.0 to 0.0. The precise shape of the relaxation curve is controllable by the user but defaults to a servo based on the current mechanical force ratio. When the reduction factor reaches 0.0, the zones are automatically set to NULL and the apply condition is removed.

- Faster Dynamic Simulations

When large size or stiffness differences are present in a model, the **dynamic multi-stepping** option has been much improved. Depending on the model, this can result in much faster dynamic simulations.

- Dynamic Input Wizard

This tool pre-processes input signals and outputs a table that can be imported for dynamic analyses. It can do frequency filtering and baseline correction using a variety of methods and export the resulting data.

- Updated APPLY logic

Apply boundary conditions can now use local (per gridpoint, face, or zone) *FISH* multipliers. Local systems are now resolved automatically when multiple constraints are placed on gridpoints. It is no longer necessary to specify the plane manually when applying normal velocity conditions on non-Cartesian aligned surfaces.

- Westergaard Hydrodynamic approximation

The Westergaard approximation for fluid boundaries in a dynamic simulation has been built directly into *FLAC3D* as an applied boundary condition.

- Results Files

FLAC3D can now export Results files; see the `model results` command. Much smaller than save files, these are intended for post-processing and analysis only. The user has full control over what aspects of the model are included. They (and save files) can also be used in combination with the **Generate Movie Frames** wizard to automatically produce movie frames of a model.

- Visualization

- The plot item interface has been improved, with drag/drop and cut/paste support added.

- Zone plot items have been optimized and multi-threaded, so they should generate and respond much faster.
 - Plot views (camera settings) can now be named and saved for later recovery and reference.
 - All of the attributes for plot items have been streamlined, so only valid options are displayed at any time.
 - The command syntax associated with plots has also been updated; plot item control switch keywords have been modified to be similar in layout and structure to the attributes seen interactively.
 - The export data file option from plot windows has been improved and made more reliable. The resulting data files are now formatted for readability and can be edited for custom scripting.
- User Interface improvements
 - Improved **Control Panel** with greater flexibility.
 - New Control Sets, including the **FISH browser** and the **File Browser**.
 - Better support for floating panes, including the option to embed local version of toolbars and the control panel.
- License verification enhancements
 - Ability to select a specific license from a server with multiple valid licenses.
 - The licensing system scans all available keys for one that matches both the code *and* the options in use.
 - Much better messaging and feedback when licensing errors occur.
 - Multiple separate licenses can now be included on one USB key.

Comparison with Other Methods

How does *FLAC3D* compare to the more common method of using finite elements for numerical modeling? Both methods translate a set of differential equations into matrix equations for each element, relating forces at nodes to displacements at nodes. Although *FLAC3D*'s equations are derived by the finite volume method, the resulting element matrices for an elastic material are identical to those of the finite element method (for constant-strain tetrahedra). However, *FLAC3D* differs in the following respects:

1. The “mixed discretization” scheme (Marti and Cundall 1982) is used for accurate modeling of plastic collapse loads and plastic flow. This scheme is believed to be physically more justifiable than the “reduced integration” scheme commonly used with finite elements.
2. The full dynamic equations of motion are used, even when modeling systems that are essentially static. This enables *FLAC3D* to follow physically unstable processes without numerical distress. The approach to provide a time-static solution is discussed in the definition for static solution.
3. An “explicit” solution scheme is used (in contrast to the more usual implicit methods). Explicit schemes can follow arbitrary nonlinearity in stress/strain laws in almost the same computer time as linear laws, whereas implicit solutions can take significantly longer to solve nonlinear problems. Furthermore, it is not necessary to store any matrices, which means: 1) a large number of elements may be modeled with a modest memory requirement; and 2) a large-strain simulation is hardly more time-consuming than a small-strain run, because there is no stiffness matrix to be updated.
4. *FLAC3D* is robust in the sense that it can handle any constitutive model with no adjustment to the solution algorithm; many finite element codes need different solution techniques for different constitutive models.

5. *FLAC3D* uses an **incremental formulation** in almost all areas. This means that displacements are not directly related to stresses and can be changed at any time, and that material properties can change without affecting the current stress state.

These differences are mainly in *FLAC3D*'s favor, but there are two disadvantages:

1. Linear simulations run more slowly with *FLAC3D* than with equivalent finite element programs. *FLAC3D* is most effective when applied to nonlinear or large-strain problems, or to situations in which physical instability may occur.
2. The solution time with *FLAC3D* is determined by the ratio of the longest natural period to the shortest natural period in the system being modeled. This point is discussed in more detail in the section on **Formulation of a 3D Explicit Finite Difference Model**, but certain problems are very inefficient to model (e.g., beams, represented by solid elements rather than structural elements, or problems that contain large disparities in elastic moduli or element sizes).

Reference

Marti, J., and P. A. Cundall. "Mixed Discretization Procedure for Accurate Solution of Plasticity Problems," *Int. J. Num. Methods and Anal. Methods in Geomech.*, 6, 129-139 (1982).

Fields of Application

FLAC3D was developed primarily for geotechnical engineering applications.

Some possible applications of *FLAC3D* are noted below. Because *FLAC3D* now has essentially the same capabilities as *FLAC*, many of the *FLAC* applications can now be extended into three dimensions with *FLAC3D*.

Table 1: Fields of Application

Area	Project Type	Problems Solved
Civil	Tunneling	Factor of safety
	Shafts	Probability of failure
	Caverns	Ground stability and improvement
	Rockfill and concrete dams	Tunnel support and design
	Excavations	Dynamic analysis
	Slopes	Evaluation of liquefaction potential
	Earth retaining structures	Groundwater flow and dewatering
	Harbor structures	Heat transfer
	Foundations	Back analysis and observational method
	Embankments	Ground freezing
	Dewatering	Settlements, consolidation, and creep
	Pavement and subgrade	Coupled thermal-mechanical-flow
	Waste disposal	Factor of safety
Mining	Open pit	Excavation stability
	Underground stope	Infrastructure design
	Room-and-pillar	Slope stability
	Longwall	Subsidence

Area	Project Type	Problems Solved	
	Caving	Dewatering	
	Solution mining	Blasting efficiency	
	Shafts and passes	Cavability	
		Recovery and dilution	
		Backfill	
		Tunneling and mine construction	
		Tailings stability	
		Tailings dams design and stability	
		Pillar sizing / spacing	
		Ground freezing	
		Excavation damage and disturbed zones	
		Ground control / remediation	
		Tunnel ground reaction curves / longitudinal profiles	
Oil & Gas		Conventional	Hydraulic fracturing and injection
		Unconventional	Well drilling and completions
	Well completions	Borehole breakout	
	Enhanced recovery	Sanding	
	Fluid injection	Induced seismicity and microseismics	
		Wellbore optimization and stability	
		Enhanced oil recovery	
		Casing failure analysis	
		Cap rock integrity	
		Coupled hydro-mechanical-thermal analysis	
		Fault stability	
		Compaction and subsidence	
		Reservoir scale modeling	
Fault movement and Integrity			

Area	Project Type	Problems Solved
		Salt cavern formation, stability, and gas storage
		Deep well injection of produced water
Power Generation	Engineered geothermal systems	Factor of safety
	Hydrothermal	Excavation damage and disturbed zones
	Nuclear reactor plants	Foundations
	Nuclear waste isolation	Engineered barrier evaluation
	Wind energy turbines	Dynamic response to earthquakes
	Hydroelectric dams	Groundwater infiltration
	Hydroelectric power houses	Deep well injection of blowdown waters
	Thermal plants	Rock characterization
	CO2 sequestration	Geophysical investigations Non-destructive examinations Microseismic and acoustic emission Cap rock integrity
		Site feasibility and suitability In-situ and laboratory testing Hydro-mechanical-thermal-chemical coupled effect
Manufacturing	Equipment design	High-deformation extrusions
	Process design	High-deformation punches
		Artificial diamond manufacturing

Modeling in Concept

The following topics present related sets of concerns when regarding the modeling process in conceptual terms.

The first topic, [Numerical Analysis, Geotechnical Engineering](#), discusses a recommended procedural approach for numerical analysis in geo-engineering.

The second topic, [Modeling Methodology](#), discusses some of the considerations that arise when pairing the problem being examined with a particular numerical methodology for solving it.

The third topic, [General Solution Procedure Illustrated](#), provides a generalized workflow of a numerical model as created and run in *FLAC3D*. That workflow is in effect and illustrated in the problems that appear in the [Tutorials](#) section, in the examples in the [Example Applications](#) section, and for the most part in almost any [example](#) that appears in this documentation.

Numerical Analysis, Geotechnical Engineering

The modeling of geo-engineering processes involves special considerations and a design philosophy different from that followed for design with fabricated materials. Analyses and designs for structures and excavations in or on rocks and soils must be achieved with relatively little site-specific data, and an awareness that deformability and strength properties may vary considerably. It is impossible to obtain complete field data at a rock or soil site. For example, information on stresses, properties and discontinuities can only be partially known, at best.

Because the input data necessary for design predictions are limited, a numerical model in geomechanics should be used primarily to understand the dominant mechanisms affecting the behavior of the system. Once the behavior of the system is understood, it is then appropriate to develop simple calculations for a design process.

This approach is oriented toward geotechnical engineering, in which there is invariably a lack of good data. But in other applications, it may be possible to use *FLAC3D* directly in design if sufficient data, as well as an understanding of

material behavior, are available. The results produced in a *FLAC3D* analysis will be accurate when the program is supplied with appropriate data. Modelers should recognize that there is a continuous spectrum of situations, as illustrated below.

Typical situation	Complicated geology; inaccessible; no testing budget	← →	Simple geology; \$\$\$ spent on site investigation
Data	NONE	← →	COMPLETE
Approach	Investigation of mechanisms	← Bracket field behavior by parameter studies →	Predictive (direct use in design)

Figure 1: Spectrum of modeling situations.

FLAC3D may be used either in a fully predictive mode (right-hand side of the figure) or as a “numerical laboratory” to test ideas (left-hand side). It is the field situation (and budget), rather than the program, that determines the types of use. If enough data of a high quality are available, *FLAC3D* can give good predictions.

Since most *FLAC3D* applications will be for situations in which little data are available, this section discusses the recommended approach for treating a numerical model as if it were a laboratory test. The model should never be considered to be a “black box” that accepts data input at one end and produces a prediction of behavior at the other. The numerical “sample” must be prepared carefully, and several samples tested, to gain an understanding of the problem. The table below lists the steps recommended to perform a successful numerical experiment; each step treated individually in the discussion that follows.

Table 1: Recommended Steps for Numerical Analysis in Geomechanics

Step 1	Define the objectives for the model analysis.
Step 2	Create a conceptual picture of the physical system.
Step 3	Construct and run simple idealized models.
Step 4	Assemble problem-specific data.
Step 5	Prepare a series of detailed model runs.
Step 6	Perform the model calculations.
Step 7	Present results for interpretation.

Step 1: Define the Objectives for the Model Analysis

The level of detail to be included in a model often depends on the purpose of the analysis. For example, if the objective is to decide between two conflicting mechanisms that are proposed to explain the behavior of a system, then a crude model may be constructed, provided that it allows the mechanisms to occur. It is tempting to include complexity in a model just because it exists in reality. However, complicating features should be omitted if they are likely to have little influence on the response of the model, or if they are irrelevant to the model's purpose. Start with a global view and add refinement as (and if) necessary.

Step 2: Create a Conceptual Picture of the Physical System

It is important to have a conceptual picture of the problem to provide an initial estimate of the expected behavior under the imposed conditions. Several questions should be asked when preparing this picture. For example: Is it anticipated that the system could become unstable? Is the predominant mechanical response linear or nonlinear? Are movements expected to be large or small in comparison with the sizes of objects within the problem region? Are there well-defined discontinuities that may affect the behavior, or does the material behave essentially as a continuum? Is there an influence from groundwater interaction? Is the system bounded by physical structures, or do its boundaries extend to infinity? Is there any geometric symmetry in the physical structure of the system?

These considerations will dictate the gross characteristics of the numerical model, such as the design of the model geometry, the types of material models, the boundary conditions and the initial equilibrium state for the analysis. They will determine whether a three-dimensional model is required, or a two-dimensional model can be used to take advantage of geometric conditions in the physical system.

Step 3: Construct and Run Simple Idealized Models

When idealizing a physical system for numerical analysis, it is more efficient to construct and run simple test models first, before building the detailed model. Simple models should be created at the earliest possible stage in a project, to generate both data and understanding. The results can provide further insight into the conceptual picture of the system; Step 2 may need to be repeated after simple models are run.

Simple models can reveal shortcomings that can be remedied before any significant effort is invested in the analysis. For example, do the selected material models sufficiently represent the expected behavior? Are the boundary conditions influencing the model response? The results from the simple models can also help guide the plan for data collection by identifying which parameters have the most influence on the analysis.

Step 4: Assemble Problem-Specific Data

The types of data required for a model analysis include the following:

- details of the geometry (e.g., profile of underground openings, surface topography, dam profile, rock/soil structure);
- locations of geologic structure (e.g., faults, bedding planes, joint sets);
- material behavior (e.g., elastic/plastic properties, post-failure behavior);
- initial conditions (e.g., in-situ state of stress, pore pressures, saturation);
and
- external loading (e.g., explosive loading, pressurized cavern).

Since typically there are large uncertainties associated with specific conditions (in particular, state of stress, deformability, and strength properties), a reasonable range of parameters must be selected for the investigation. The results from the simple model runs (in Step 3) can often prove helpful in determining this range, and in providing insight for the design of laboratory and field experiments to collect the needed data.

Step 5: Prepare a Series of Detailed Model Runs

Most often, the numerical analysis will involve a series of computer simulations that include the different mechanisms under investigation and span the range of parameters derived from the assembled database. When preparing a set of model runs for calculation, several aspects, such as the following, should be considered:

1. How much time is required to perform each model calculation? It can be difficult to obtain sufficient information to arrive at a useful conclusion if model runtimes are excessive. Consideration should be given to performing parameter variations on multiple computers to shorten the total computation time.
2. The state of the model should be saved at several intermediate stages so that the entire run does not have to be repeated for each parameter variation. For example, if the analysis involves several loading/unloading stages, the user should be able to return to any stage, change a parameter, and continue the analysis from that stage. The amount of disk space required for save files should be considered.
3. Are there a sufficient number of monitoring locations in the model to provide for a clear interpretation of model results and for comparison with physical data? It is helpful to locate several points in the model at which a record of the change of a parameter (such as displacement, velocity or stress) can be monitored during the calculation. Also, the maximum unbalanced force in the model should always be monitored to check the equilibrium or failure state at each stage of an analysis.

Step 6: Perform the Model Calculations

It is best to first make one or two model runs, split into separate sections, before launching a series of complete runs. The runs should be checked at each stage to ensure that the response is as expected. Once there is assurance that the model is performing correctly, several data files can be linked together to run a complete calculation sequence. At any time during a sequence of runs, it should be possible to interrupt the calculation, view the results, and then continue or modify the model as appropriate.

Step 7: Present Results for Interpretation

The final stage of problem solving is the presentation of the results for a clear interpretation of the analysis. This is best accomplished by displaying the results graphically, either directly on the computer screen or as output to a hardcopy plotting device. The graphical output should be presented in a format that can be directly compared to field measurements and observations. Plots should clearly identify regions of interest from the analysis, such as locations of calculated stress concentrations, or areas of stable movement versus unstable movement in the model. The numeric values of any variable in the model should also be readily available for more detailed interpretation by the modeler.

We recommend that these seven steps be followed to solve geo-engineering problems efficiently. The following sections describe the application of *FLAC3D* to meet the specific aspects of each of these steps in this modeling approach.

Modeling Methodology

In a field such as geomechanics, where data are not always available, the methodology used in numerical modeling should be different from that used in a field such as mechanical engineering. Starfield and Cundall (1988) provide suggestions for an approach to modeling that is appropriate for a data-limited system. This paper should be consulted before any serious modeling with *FLAC3D* is attempted. In essence, the approach recognizes that field data (such as in-situ stresses, material properties and geological features) will never be known completely. It is futile to expect the model to provide design data, such as expected displacements, when there is massive uncertainty in the input data. However, a numerical model is still useful in providing a picture of the *mechanisms* that may occur in particular physical systems. The model acts to educate the intuition of the design engineer by providing a series of cause-and-effect examples. The models may be simple, with assumed data that are consistent with known field data and engineering judgement. It is a waste of effort to construct a very large and complicated model that may be just as difficult to understand as the real case.

Of course, if extensive field data are available, then these data may be incorporated into a comprehensive model that can yield design information directly. More commonly, however, the data-limited model does not produce such information directly, but provides insight into mechanisms that may occur. The designer can then do simple calculations based on these mechanisms that estimate the parameters of interest or the stability conditions.

Modeling of Chaotic Systems

In some calculations, especially in those involving discontinuous materials, the results can be extremely sensitive to very small changes in initial conditions, or trivial changes in loading sequence. At first sight, this situation may seem unsatisfactory and may be taken as a reason to mistrust the computer simulations. However, the sensitivity exists in the physical system being modeled. There appear to be at least two sources for the seemingly erratic behavior:

1. There are certain geometric patterns of discontinuities that force the system to choose, apparently at random, between two alternative outcomes; the subsequent evolution depends on which choice is made. For example, the figure below illustrates a small portion of a jointed rock mass. If block A is forced to move down relative to B, it can either go to the left or to the right of B; the choice will depend on microscopic irregularities in geometry, properties or kinetic energy.

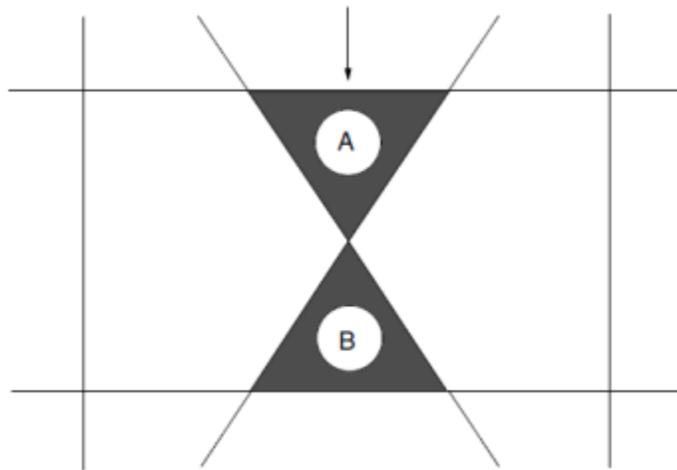


Figure 2: A small portion of a jointed rock mass.

2. There are processes in the system that can be described as “softening” or, more generally, as cases of positive feedback. In a fairly uniform stress field, small perturbations are magnified in the subsequent evolution, because a region that has more strain softens more and thereby attracts more strain, and so on, in a cycle of positive feedback.

Both phenomena give rise to behavior that is chaotic in its extreme form (Gleick 1987 and Thompson and Stewart 1986). The study of chaotic systems reveals that the detailed evolution of such a system is not predictable, *even in principle*. The observed sensitivity of the computer model to small changes in initial conditions or numerical factors is simply a reflection of a similar sensitivity in the real world to small irregularities. There is no point in pursuing increasingly more “accurate” calculations, because the resulting model is unrepresentative of the real world, where conditions are not perfect. What should our modeling strategy be in the face of a chaotic system? It appears that the best we can expect from such a model is a finite *spectrum* of expected behavior; the *statistics* of a chaotic system are well-defined. We need to construct models that contain distributions of initial irregularities (e.g., by using *FLAC3D*’s `gauss_dev` or `uniform_dev` parameter on the `property` command, or by specifying given distributions with a special *FISH* function). Each model should be run several times, with different distributions of irregularities. Under these conditions, we may expect the fluctuations in behavior to be triggered by the imposed irregularities, rather than by artifacts of the numerical solution scheme. We can express the results in a statistical form.

Localization, Physical Instability, and Path-Dependence

In many systems that can be modeled with *FLAC3D*, there may be several paths that the solution may take, depending on rather small changes in initial conditions. This phenomenon is termed *bifurcation*. For example, a shear test on an elastic/plastic material may either deform uniformly, or it may exhibit shear bands, in which the shear strain is localized rather than being uniformly distributed. It appears that if a numerical model has enough degrees-of-freedom (i.e., enough elements), then localization is to be expected. Indeed, theoretical work on the bifurcation process (e.g., Rudnicki and Rice 1975 and Vardoulakis 1980) shows that shear bands form even if the material does not strain-soften, provided that the dilation angle is lower than the friction angle. The “simple” Mohr-Coulomb material should always exhibit localization if enough elements to resolve one or more localized bands exist. A strain-softening material is more prone to produce bands.

Some computer programs appear incapable of reproducing band formation, although the phenomenon is to be expected physically. However, *FLAC3D* is able to allow bands to develop and evolve, partly because it models the dynamic equations of motion (i.e., the kinetic energy that accompanies band formation is released and dissipated in a physically realistic way). Several papers document

the use of two-dimensional *FLAC* in modeling shear band formation (Cundall 1989, 1990 and 1991). These should be consulted for details concerning the solution process. One aspect that is not treated well by *FLAC3D* is the *thickness* of a shear band. In reality, the thickness of a band is determined by internal features of the material, such as grain size.

These features are not built into *FLAC3D*'s constitutive models. Hence, the bands in *FLAC3D* collapse down to the smallest width that can be resolved by the grid, which is one grid-width if the band is parallel to the grid, or about three grid-widths if the band cuts across the grid at an arbitrary angle. Although the overall physics of band formation is modeled correctly by *FLAC3D*, band thickness and band spacing are grid-dependent. Furthermore, if the strain-softening model is used with a weakening material, the load/displacement relation generated by *FLAC3D* for a simulated test is strongly grid-dependent. This is because the strain concentrated in a band depends on the width of the band (in length units), which depends on zone size, as we have seen. Hence, smaller zones lead to more softening, since we move out more rapidly on the strain axis of the given softening curve. To correct this grid-dependence, some sort of length scale must be built into the constitutive model. There is controversy, at present, concerning the best way to do this. It is anticipated that future versions of *FLAC3D* will include a length scale in the constitutive models (probably involving the use of a Cosserat material, in which internal spins and moments are taken into account). In the meantime, the processes of softening and localization may be modeled, but it must be recognized that the grid size and angle affect the results: models must be calibrated for each grid used.

One topic that involves chaos, physical instability and bifurcation is *path-dependence*. In most nonlinear, inelastic systems, there are an *infinite* number of solutions that satisfy equilibrium, compatibility, and the constitutive relations. There is no "correct" solution to the physical problem unless the path is specified. If the path is not specified, all possible solutions are correct. This situation can cause endless debate among modelers and users, particularly if a seemingly irrelevant parameter in the solution process (e.g., damping) is seen to affect the final result. All of the solutions are valid numerically. For example, a simulation of a mining excavation with low damping may show a large overshoot and, hence, large final displacements, while high damping will eliminate the overshoot and give lower final displacements. Which one is more realistic? It depends on the path. If the excavation is done by explosion (i.e., suddenly), then the solution with overshoot may be the appropriate one. If the excavation is done

by pick and shovel (i.e., gradually), then the second case may be more appropriate. For cases in which path-dependence is a factor, modeling should be done in a way that mimics the way the system evolves physically.

General Solution Procedure Illustrated

The general solution procedure, illustrated in the figure below, is convenient because it represents the sequence of processes that occurs in the physical environment. The remainder of this tutorial illustrates the basic approaches in *FLAC3D* to performing a simple analysis that follows this solution procedure.

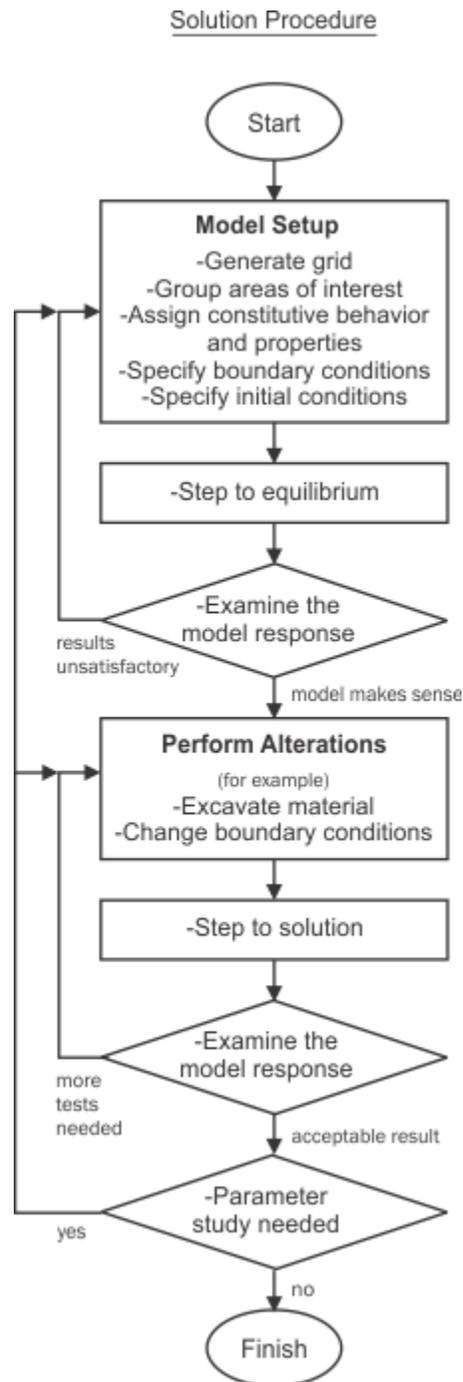


Figure 3: General *FLAC3D* solution procedure.

The Solution Procedure as a *FLAC3D* Project

The general solution procedure from *General Solution Procedure Illustrated* is reproduced here, with an adjacent overlay showing how the components of a *FLAC3D* project are organized as the solution procedure progresses.

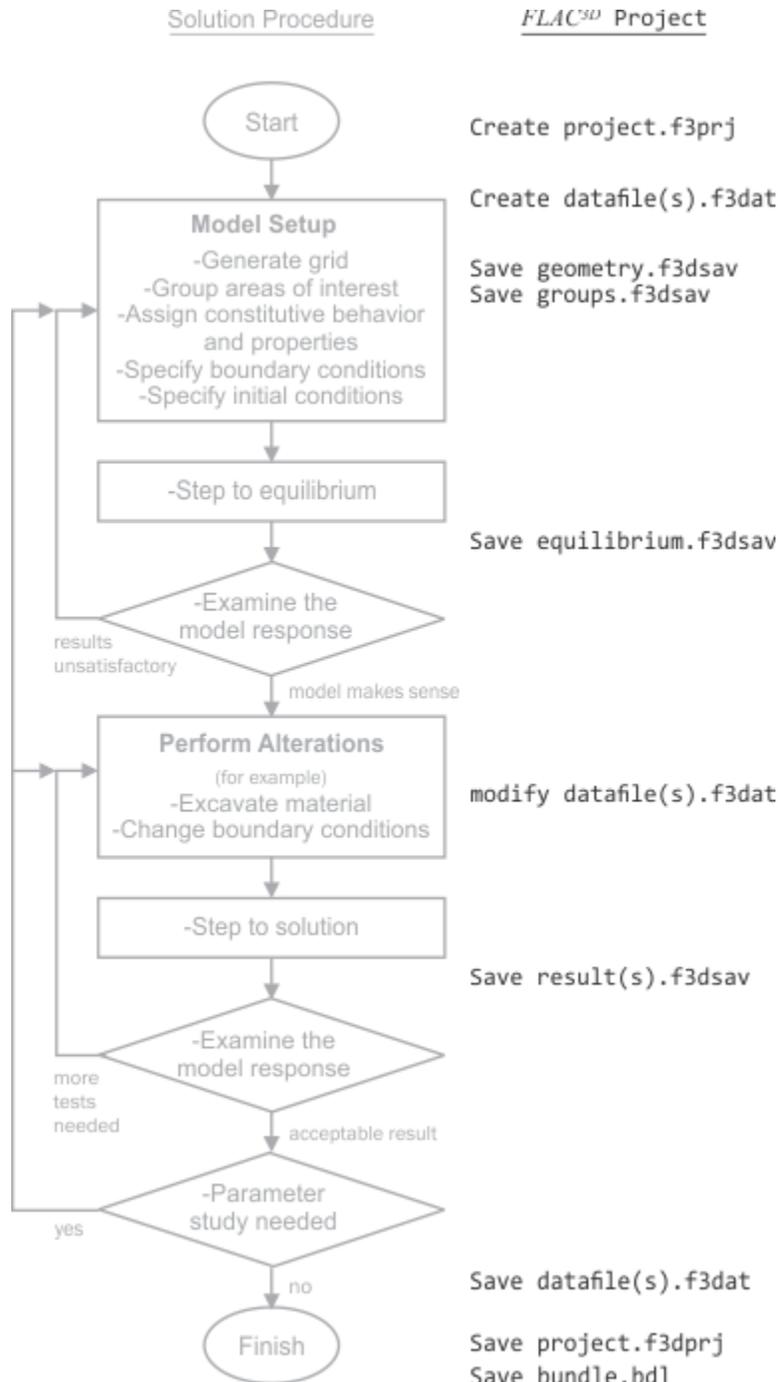


Figure 4: General *FLAC3D* solution procedure with project files overlay.

About *FLAC3D* Commands

All commands in *FLAC3D* are words that consist of a primary word (a noun/object) and a first keyword (a verb/action). Depending on the command, these two may be followed by: zero, one, or more options; zero, one, or more modifiers (which will vary the effect of the command); and a range (which will restrict the effect of the command). This pattern is expressed as

NOUN - VERB - OPTION(S) - MODIFIER(S) - RANGE

and it is observable in *all* *FLAC3D* commands.

Command Entry

Commands are typed literally at the command prompt or in a data file. Commands may be shortened to their first few letters. In this documentation, the underlined characters in a command signature (see *zone list*, for instance) indicate the minimum necessary characters to type for correct recognition of the command. Note that a hyphenated command will require minimum characters *and* the hyphen (e.g., *zone apply-remove* may be shortened to *z a-r*). *Ctrl + space* will access the inline help facility when typing commands either at the command prompt or in a data file.

Keywords, Value, Delimiters, and Command Lineation

Many keywords are followed by a series of values that provide the numeric input required by the keyword. The decimal point may be omitted from a real value, but may not appear in an integer value.

In this documentation, options in a command are denoted by *< >*. A keyword (optional or required) followed by ellipses (*...*) indicates that an arbitrary number of such parameters may follow.

Commands, keywords, and numeric values may be separated by any number of spaces, or by any of the following delimiters:

() , =

A semicolon (;) may be used to precede comments; anything that follows a semicolon in an input line is ignored. It is useful, and strongly recommended, to include comments in data files. Not only is the input documented in this way, but the comments are echoed to the output as well, providing the opportunity for quality assurance in your analysis.

A single input line, including comments, may contain any number of characters. Commands may contain up to 1024 characters. In data files, they may be written on multiple lines of text as long as a continuation character is used to link multiple lines together into one command line.

An ampersand (&) or an ellipsis (...) can be given at the end of an input line to denote that the next line will be a continuation of that line.

Syntax Highlighting

If not acting as a hyperlink, in this documentation, commands and *FISH* functions are color-highlighted to match the default syntax highlighting of *FLAC3D*. Note that this may be changed by the user in the program using the **Options dialog**, but those changes will not be reflected in the documentation. Refer to the tables below to explain the highlighting of the following two command presentations. The first is a command as it would appear in a data file in *FLAC3D*, the second is a command signature of the same command, as presented in this documentation.

```
zone face apply stress-xy 110.23 system dip-direction 47.4 ...
range position-x 22.7 1.19e-1
```

```
zone face apply keyword <range>
```

Table 2: (Default) Syntax Coloring/Highlighting for Commands and *FISH* in *FLAC3D*

Type style/ character	Used for
command	first-level <i>FLAC3D</i> command word
keyword	command keyword
map	<i>FISH</i> function name or variable
flt or int	a number of float or integer type

Type style/ character	Used for
str or bool	a string value or boolean value

Table 3: Typography for Commands and *FISH* in *FLAC3D* Help

Type style/ character	Used for
command	command word
keyword	keyword word
map	<i>FISH</i> function name or variable
flt or int	a number of float or integer type
str or bool	a string value or boolean value
< >	indicates item (keyword) enclosed is optional
...	indicates that an indeterminate number of keywords may follow
<range>	indicates an optional range phrase may appear; <i>always</i> appears as a hyperlink to the range phrase reference
<input type="button" value="Press Me"/>	button with the hot-key underlined
A	type the key between < > (here, < A >) on the keyboard
Shift-A	hold down the first key while pressing the second (here, < Shift > and the < A > key)

There are **additional typographic conventions** in use in this documentation that are presented in the section [Guide to the FLAC3D Help File](#).

Command Processing

Commands may be entered at the **command prompt** or they may be written in a **data file** and processed together (using the *Execute* button () on the toolbar) or in selected parts (using the *Execute selected* button () on the toolbar).

Command Reference

This topic and the ones that follow are designed only to provide an introduction to *FLAC3D* commands and some of the most fundamental ideas that go into working with them. All the objects that are used in a *FLAC3D* model, and the commands and *FISH* functions for working with them, are exhaustively referenced in the [FLAC3D Elements](#) section of this Help file. That section begins with [FLAC3D conventions](#), followed by a selection of [Command Constructs](#) that detail a number of capabilities that are frequently used within commands (ranges, groups, selection, hiding, etc.) and the keyword reference information necessary to use them.

A full index of commands is available in the [Command and FISH Reference Index](#).

Command Scope & Overwriting

Commands are sequential by nature, acting in the order that they are processed by *FLAC3D*. In particular:

1. By default, commands operate on all objects of the type indicated by the command. For example, `zone initialize` will operate on all zones. A few commands, such as `model new`, operate on nearly all objects.
2. Commands given later overwrite previously processed commands.
3. To make a command act on a subset of all possible target objects, use the `range` keyword. A `range` acts as a filter, specifying that a command will operate only on a subset of objects (as determined by the criteria that follow the `range` keyword).

These three ideas are illustrated with the following three commands.

```
zone cmodel assign mohr-coulomb
zone cmodel assign elastic
zone cmodel assign hoek-brown range position-x 1 500
```

The first command assigns the `mohr-coulomb` constitutive model to *all zones*. The second command assigns the `elastic` model to *all zones*, overwriting the first command. The third command, on the other hand, assigns the `Hoek-Brown` model to zones with centroid *x*-components lying within the range $x = 1$ and $x = 500$. This last command may *partially* overwrite the second command if not all

zones have centroids that fall within this range. Once all three commands are processed, any zones that fall outside the range x 1–500 are (still) assigned the elastic model, and there are no zones that are assigned the mohr-coulomb model.

The use of ranges and groups to refine the application of commands to targeted objects is further introduced in the [next](#) topic.

Targeting Commands with Ranges and Groups

As seen in the preceding topic, [Command Scope & Overwriting](#), the `range` keyword is used to select which objects a command will affect.

A `range` is phrase composed of the `range` keyword at the start, followed by one or more additional keywords that make up the range's definition. The range filters possible objects on which a command can operate. As the command is executed, each object is presented to the range and, if it is considered to fall within the range, then the command operates on the object. If the object falls outside the range it is skipped. A range is made up of any number of *range elements*, each of which is a specific filter composed of a keyword and a value or values. By default, an object is considered part of the range only if it is part of *every* range element. One might consider this behavior as an intersection of all range elements. This default behavior can be modified with the `union` keyword.

Range elements come in two broad categories: geometric and property-based. Geometric range elements specify a region of space. Every object being considered has a single representative point (generally the centroid, termed the object location) that is compared against the region defined by the range element. The most common geometric range elements are the `range position-x`, `range position-y`, and `range position-z` elements that check against the x , y , or z coordinate of objects' locations. Also the `position` range element, which checks if an object location falls within the specified axis-aligned bounding box, is quite useful.

Property-based range elements, on the other hand, filter based on a piece of object data. The most common property-based range element is the `group` element, that compares against `group` labels assigned to the object (see the [Group](#) section for more information on groups and their assignment).

To illustrate these differences, suppose a model consisting of zones is created and cycled without deletion of any zones. An `id 1 100` range element that gets the zones with IDs 1-100 will yield the same specific and unchanging set of zones every time. The `position-x 1 100` range element, that gets zones whose centroids lie between $x = 1$ and $x = 100$, may yield different zones at different times as cycling progresses. A `group Layer1` range element will always select the objects in the group `Layer1` regardless of model progress; only changing the objects comprising the group would cause the range element to return a differing set of objects.

The example below demonstrates the practical use of ranges.

```
zone create brick size 50 50 50
zone cmodel assign mohr-coulomb range position-x 0 10
```

By directly specifying the range when assigning the `mohr-coulomb` constitutive model, only a subset of zones have been operated upon. It may be the case, though, that this specific range is needed for further assignments. For instance, one might also want to assign different properties to the zones in this range. In that case, instead of retyping the range multiple times (which is prone to error, particularly as ranges get longer and more complex as seen next), it may be easiest to assign zones to a group first, and subsequently use this group via the `group` range element in later commands.

```
model new
zone create brick size 50 50 50
zone group create "Layer1" range union ...
                                position-x 0 10 ...
                                position-y 1 15 ...
                                position-z 30 50 not
zone cmodel assign mohr-coulomb range group "Layer1"
```

In this example, a group named `Layer1` is created. In the next line, the `mohr-coulomb` constitutive model is assigned by using the range element `group Layer1` for the range.

One can easily re-use this group to further assign properties to these zones and a constitutive model to the remaining zones.

```
zone property bulk 2.8e8 shear 1.e8 range group "Layer1"
zone cmodel assign elastic range group "Layer1" not
```

Continuing from the earlier lines, the first command assigns properties to those zones in *Layer1*. Then the *elastic* constitutive model is applied to all other zones by specifying the *not* keyword.

Refer to the *Range* section and the *Group* section in *Command Constructs* for complete information on working with ranges and groups.

Command Tools

When working with commands, there are multiple tools/utilities in the program to assist the user in building and understanding commands. These are described briefly here to provide an overview of the various assistive facilities provided for working with commands. Full usage information for each is presented in the *FLAC3D Interface* section of this Help file.

Inline Help

The *Inline Help* facility, available at the command line and in any file (*data*, *FISH*, etc.) loaded in the *Edit* pane, provides an interactive tool for building commands *and* directly accessing help for commands and *FISH* functions. By providing a stepwise path for command construction, the facility is also an excellent learning tool, as it gives the user the ability to see and learn how each structural juncture of a command or *FISH* function fits with the others and to develop familiarity with the keyword set that is available at that juncture.

Context-Sensitive Help

In the editor or in the command prompt, pressing the F1 key will activate context-sensitive help. If the line represents a command, the documentation for that command should appear in the help. If the line represents *FISH* and the cursor was over a valid intrinsic, the documentation for that intrinsic should appear in the help.

Question Mark

While working on the command line, typing *?* and pressing *Enter* will show a list of keywords available at the current cursor position. This is similar to the *Inline Help* facility, though simpler and less persistent—the user must invoke the *?* command each time the list of keywords is needed.

Conversion Tool

When a data file is loaded in *FLAC3D*, the program will automatically detect if it contains elements of pre-version-6.0 syntax, and, if so, offer to convert the file to 6.0 syntax. The conversion facility may also be explicitly invoked by the user from the main menu (Tools › Conversion...). After conversion, the facility will automatically archive the “old” data file and will explicitly mark and cue within the data file any command that cannot be automatically converted.

Additional Resources in Help

At the start of this file, an [index](#) of all the top-level commands and a categorical presentation of all *FISH* functions is available. Near the end of the help, there is a [table that maps](#) (where possible) old *FLAC3D* 5.0 commands to their *FLAC3D* 6.0 equivalents, followed by a table that [maps the same](#) for *FISH* functions. These are a useful reference for users needing to supply adjustments, corrections, or clarification to the output coming from files converted with the conversion tool.

Syntax Changes from Version 5.0 to 6.0

Users of *FLAC3D* Version 5.0 or earlier will note that the command syntax described in [About FLAC3D Commands](#) is a significant change from past *FLAC3D* command sets. Indeed, data files and commands used in earlier versions are not compatible with this version. The fundamental change is structural. The old syntax, familiar to users of past versions, could be described in previous manuals in the following general construction

```
COMMAND keyword value ... <keyword value ...> ...
```

the new syntax follows a pattern that is likelier to appear (and be rendered, in this documentation) as

```
command keyword <range>
```

Structurally, the old commands followed a syntactical pattern that could be generalized as:

```
VERB - OPTIONS - MODIFIER - RANGE
```

In this formulation, the object being acted on by the command was implicit in the VERB. It was necessary to know, from memory, what that object would be by knowing the command. The new syntax is explicit—the object of the command

appears at the start of the command. An action involving zones, for instance, will always involve a command that starts with the word `zone`. This explicit, straightforward approach yields a much easier command set, in terms of learning acquisition and in terms of readability.

However, greater clarity is only one benefit of the new syntax. A second is that the commands are hierarchically traversable, which is the underlying mechanism that allows *FLAC3D* 6.0 to provide an `inline help` utility. This new feature both speeds the process of command construction and gives instant access to this help file at *whichever part of the command* is of interest.

In addition, this version of *FLAC3D* provides a `conversion` tool for bringing old data files into the current syntax. This tool, and the inline help facility, are briefly introduced in the next topic.

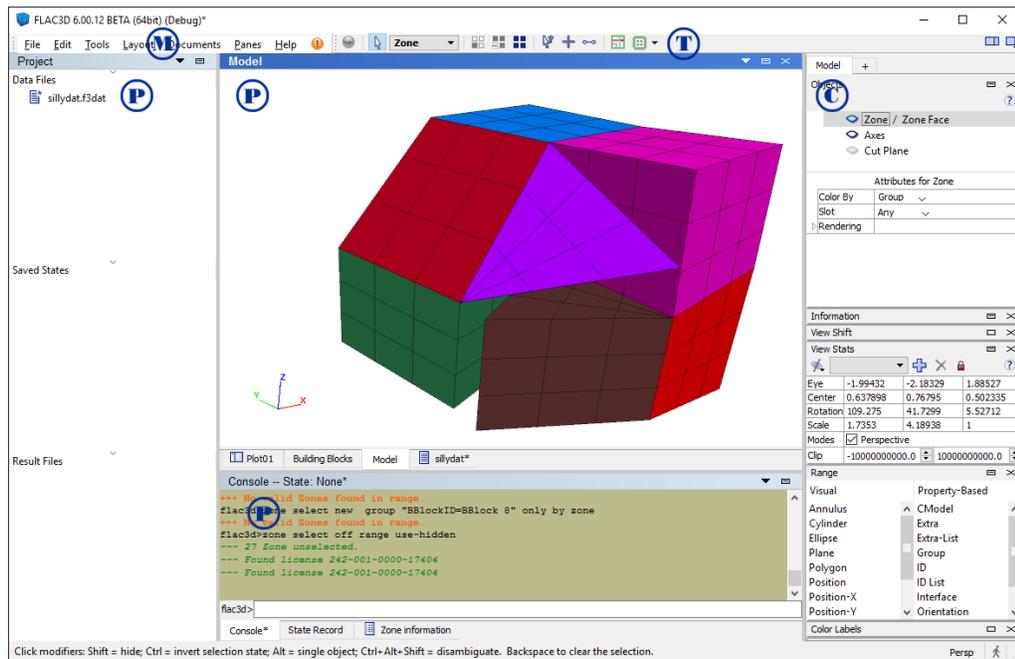
Additional Points on the New Syntax

Some other aspects of the new syntax are worth noting for users familiar with the old syntax.

1. The main command may be—in fact, is likely to be—made up of more than one word, since there is an “object” at the start of each command (cf. the `zone` `commands`).
2. A command or keyword may be hyphenated. Use of underscores in commands and keywords is eliminated. If a command has a hyphen in it, a valid shortened version of that command will **require** the hyphen and parts on either side of it.

FLAC3D Program Layout

By way of introduction, refer to the following screen image. The fundamental design aspects of the *FLAC3D* interface are enumerated. A good grasp of these basic elements will facilitate understanding all the material presented in *FLAC3D* Interface.



The interface of *FLAC3D* is fundamentally centered on the interaction of workspaces, called “panes” (P in the above image), and the tools provided by the *Control Panel* (C in the above) and toolbar (T in the above) for working in them.

P – panes. *FLAC3D* has a pane-based design. A pane is a window with a title bar and controls for handling it. There are multiple pane *types* in *FLAC3D*. Each pane type provides a distinct functionality (hence the different types). In the image above, the *Project* pane (at left, for project/file management), the *Model* pane (center top, for model object selection and grouping)—note the darker blue title bar indicates it is the active pane, and the *Console* pane (center bottom, for issuing commands) are visible.

Panes of any and of mixed type may be stacked on top of one another, in which case they appear as tabbed sets. They may be floated and docked, which means the arrangement of panes in the program is completely flexible.

C and T - *Control Panel* (C) and toolbar (T). The *Control Panel* provides boxed tools sets (referred to as “control sets”) of functionality that are specific to the currently active pane. It may be hidden, but when shown/visible it is *always* rendered at the right side of the program window. The item marked “T” is the toolbar. The toolbar, like the *Control Panel*, is contextual to the currently active pane. The tools appearing on it are for use in that pane and will change when a different pane type is selected.

M - main menu. Unlike the *Control Panel* and the toolbar, the main menu provides fixed options, regardless of the currently active pane type. These focus mostly on file management operations, editing, and pane management, as well as providing a *Help* menu for user support.

One Program, Two Sides

One necessary aspect of learning how to work with *FLAC3D* is to understand the elements that make up a *FLAC3D* model, the commands needed for working with them, and the *FISH* functions associated with them. All of this is detailed in reference form in the *FLAC3D Elements* section of this Help file.

The section that follows that one, *FLAC3D Interface*, provides the other, visual half of the picture. Though not every command has a user interface equivalent, there is significant overlap between the written command set and the actions that may be performed in the user interface. And bear in mind that any program action of the user interface that modifies the current model state is translated into a command and stored in the state record.

In a number of cases (constructing meshes in either the *Extrusion* or *Building Blocks* facilities, for instance), the user interface approach will be easier and faster than explicitly issuing commands at the command prompt or via data file. However, to obtain the advantage of efficiency that the user interface provides in those areas, it is necessary to understand the underlying commands as well as understand how to emit them through the tools in the user interface.

Files

There are several types of files that are either used or created by *FLAC3D*. The files are typically distinguished by their extensions. The more common ones are described below. Relationships between the core files of a *FLAC3D* project are described in the next topic, *FLAC3D Projects Introduced*.

data file

FLAC3D commands may be issued “interactively” at the command prompt, or via a *data file*. The data file is a formatted ASCII file created by the user, which contains the set of *FLAC3D* commands that represents the problem being analyzed. In general, creating data files is the most efficient way to use *FLAC3D*. Data files are opened and the commands within them are executed by the program using the `program call` command, or via the File ▸ Open Item... menu command. Though data files can have any file name and any extension, it is recommended that a common extension (e.g., “.F3DAT” for *FLAC3D* input commands and “.F3FIS” for *FISH* function statements) be used to distinguish these files from other types of files.

save file

A *save file* is a binary file containing the values of all state variables and user-defined conditions up to the current moment of program execution. The primary reason for creating save files is to allow one to investigate the effect of parameter variations without having to rerun a problem completely. A save file can be restored and the analysis continued at a subsequent time (see the `model restore` command). Normally, it is good practice to create several save files during a *FLAC3D* run. Save files are created with the `model save` command; if this command is given without a `filename` keyword, the default name “FLAC3D.F3DSAV” will be supplied. Other terms for save file are **SAV file**, **save state**, or **saved state**.

log file

The *log file* is a formatted ASCII file that captures all output text from the *Console* pane. The log file is useful in providing a record of the *FLAC3D* work session; it also provides a document for quality-assurance purposes. The file is created/maintained if the command `program log` is set on. The user may supply a file name using the `program log-file` command; if the command is not given but logging is on, the default file name “FLAC3D.LOG” will be used.

history file

A *history file* is a formatted ASCII file created at the user's request when issuing the command `history export` command. The user may specify a name for the file using the `file` keyword of the command; if not, the default name "FLAC3D.HIS" is used. Commands for creating history files may be issued interactively or within data files. A record of the history values is written to the file, which can be examined using any text editor that can access formatted ASCII files. Alternatively, the file may be processed by a commercial graph-plotting or spreadsheet package.

table file

A *table file* is a formatted ASCII file created by exporting a table using the `table export` command. This table may be modified by the user and imported into another model state using the `table import` command. The table file format is simple and easy to create from spreadsheets or other ASCII data. It specifies a single sequence of (x,y) value pairs, either at regular or irregular x -intervals.

fish i/o file

A user-written *FISH* function may create, write, and read files (see the [FISH Functions](#) section for file input/output) of two forms. If the binary form is selected, the exact binary representation of a *FISH* variable is stored on file. If the ASCII form is selected, numeric or character data may be written, or read from, a file. The latter form is a useful way to process ASCII data in a nonstandard way on file.

project file

The *project file* represents all the files and user interface elements involved in a given *FLAC3D* project, which may involve many different specific models and model states. The data in a project file is separate from the model state and is not affected by a `model new` or `model restore`. The project file stores the user-interface layout, the plot files, the data files being edited, and the list of data and save files that have been used or created for the project.

result file

A *result file* is a smaller version of a save file (see above) containing only specific user-specific pieces of information about the model. Since they are much smaller (generally around 5% of the size of a save file) many more of them may be created without undue use of storage. A result file may be

imported, overwriting the current model state, so that the contents can be visualized. Result files may be automatically output at user-specified intervals during the calculation.

geometry file

A *geometry file* is a set of nodes, edges, or polygons in space. Sometimes this is referred to as CAD data. This data could have originated from other sources, or it could have been generated by *FLAC3D*. *FLAC3D* currently recognizes three geometry file formations: DXF, STL, or GEOM. The first two are industry standards, the last is an Itasca-specific format that preserves object meta-data (*FISH* extra variable assignment and/or group assignments).

mesh file

A *mesh file* is a file format that specifies a discretization of space into gridpoints and zones (or nodes and elements if using typical finite-element parlance). *FLAC3D* defines its own mesh formation, F3GRID. *FLAC3D* can also directly import ANSYS (.LIS) and ABAQUS (.INP) formats.

user data file

FLAC3D currently supports three types of user-defined data types. Scalar fields, vector fields, and tensor fields. These data types support both export and import into custom ASCII file formation, via the `data scalar import` or `data scalar export` commands (for example). These are used primarily to record possible model data or to compare against data generated outside *FLAC3D* (like seismic events imported as scalar data).

bundle file

A *bundle file* is generated from a project and represents all the files necessary to recreate the project and model states combined into one file. The project file, all data files, and the record stripped from all model save states is included. This is a simple way to archive your *FLAC3D* project for the future and is also a very convenient thing to give to Itasca when asking support questions—sending a bundle file ensures we have everything we need to duplicate your issue.

FLAC3D Projects Introduced

The *FLAC3D* project is a file (*.f3dprj) that sits at the heart of any project. It provides the main mechanism for keeping a project—which may be composed of many data files, *FISH* files, and save files—in good working order.

Projects, Files, and the Call Stack

A typical *FLAC3D* project can and will reach a state where the “call stack” is complicated, with many branchings. Remember that the program, at any given moment in time, is in a state that can be described with a *linear* history of commands. However, from a file/project management perspective, the path to arrive at that state may be forked multiple times along the way, since one data file may call another (or many others) and continue doing so as long as those calls do not become recursive—to say nothing of the further complications that *FISH* looping and conditional statements may introduce.

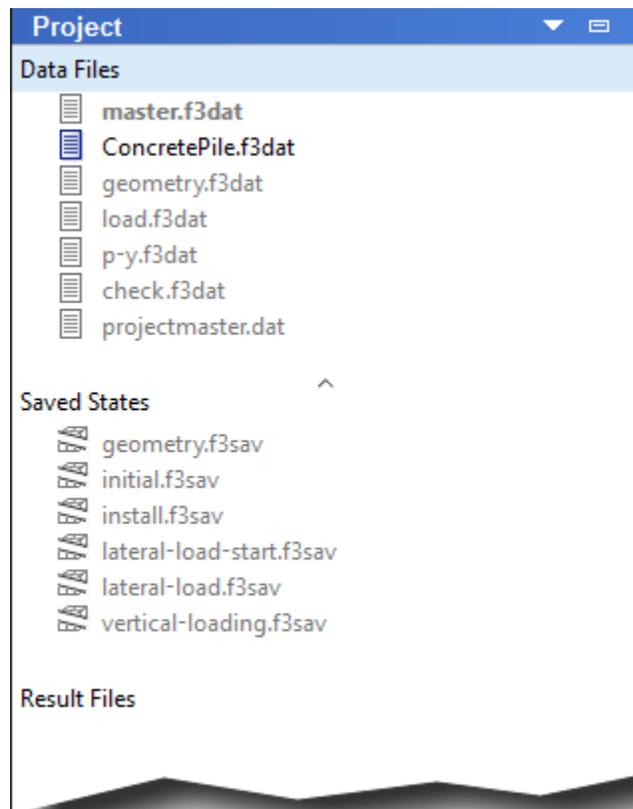
Consider the following, which is a simplified schematic rendering of the relationship between the files that make up the project detailed in [Axial and Lateral Loading of a Concrete Pile](#) (you can load this in *FLAC3D* via [Help](#) ▸ [Examples...](#) ▸ [Example Applications](#) ▸ [ConcretePile](#)).

```

call 'project_master'
|
call 'master'
|
call 'concretepile'
|
model new
|
call 'geometry'
|
model save 'geometry'
|
model solve ratio 1e-4
|
model save 'initial'
|
model restore 'initial'
|
model solve ratio 1e-4
|
model save 'install'
|
call 'load'
|
model step 58000
|
model save 'vertical-loading'
|
model restore 'install'
|
model solve ratio 1e-4
|
model save 'lateral-load-start'
|
call 'p-y'
|
model step 4165000
|
model save 'lateral load'

```

When loaded in *FLAC3D*, this project is managed in the Project Pane pane as below.



As can be seen, called (or more generally “opened”) files (data files — inputs) or saved files (SAV or save files — outputs), are tracked by the project file. The “Data Files” group will also track any grid files opened into the project. However, the *Project* pane simply lists these files in groups, without reference to their hierarchical relationships.

Lastly, observe that all the files shown in the *Project* pane are *separate* from the project file; they are not embedded within the project file.

Additional Project Settings Handled by the Project File

In addition, the project file records and saves certain state information that is restored when the project is loaded, including every aspect of the program layout, any plot or model view that is created and left open at the time the project is saved (and the exact state of that view), and the state of the various program options (as set in *Options Dialog*).

Guide to the *FLAC3D* Help File

This *FLAC3D* Version 6.0 Help contains the complete documentation for the program. It replaces the printed *FLAC3D Manual*. It contains a number of advantages to the printed manual set that are worth noting.

- The Help file is accessible instantly and directly from the program (use the Help ▸ Help menu command).
- It is integrated with the commands and *FISH* functions via the [Inline Help](#) utility; access to reference documentation is nearly instantaneous and fully targeted.
- It contains a built-in *Search* tab that allows for documentation-set-spanning searches for material.
- Each page may be scanned for content using the *Find* dialog (use `Ctrl + F` to access).
- Material in the documentation can be easily copied and pasted if needed.
- It provides a *single* source for all program documentation.
- It contains a feedback mechanism to allow users to provide comments on the material presented (see the link for “Was this topic helpful?” at the bottom of every page).

The organization of this Help file along with brief summaries of the contents of each section follows.

FLAC3D 6.0 Documentation

The homepage of the Help file.

Command and FISH Reference Index

A single-page presentation of all the first-level command words, and a table that presents *FISH* functions, by categorical group.

FLAC3D Modeling

This section is a user’s guide. It is designed to introduce the software and the major design ideas behind it, but briefly. Much of the information presented in the [Introduction](#) portion of this section is given a more exhaustive presentation in later sections as well. It gives a first look at commands, the user interface, and project management. It also offers discussions on

modeling methodology and the recommended problem-solving procedure in *FLAC3D*. The **Tutorials** section that follows is the best place to begin getting hands-on experience with the program. The **Problem Solving with FLAC3D** section examines each of the major steps in the *FLAC3D* modeling process with detailed consideration.

FLAC3D Elements

This section is the complete reference for all commands (except those that are part of program options) and *FISH* functions. The material here corresponds to the material formerly presented in the print volumes *Command Reference* and *Structural Elements*, and the *FLAC3D*-specific functions from *FISH in FLAC3D*.

FLAC3D Interface

The integrated model development environment for *FLAC3D* is described and documented here, providing the *how-to* for the program's multiple methods of issuing commands and developing *FISH* scripts.

FLAC3D Theory and Background

The theoretical formulation for *FLAC3D* is described in detail in this section. This includes both the description of the mathematical model that describes the mechanics of a system and the numerical implementation. Reference information on the interface logic and factor of safety calculations is included here. The material here closely corresponds to the content formerly presented in the print manual volumes *Theory and Background*, *Fluid-Mechanical Interaction*, and *Constitutive Models*.

FISH Scripting Reference

A complete introduction and general reference information to *FISH* is provided here (note *FLAC3D*-specific *FISH* functions will be referenced in the **FLAC3D Elements** section). All statements, variables, and functions built into *FISH* are referenced. A guide on how to compile and input custom *FISH* intrinsics is included. This section corresponds to the descriptive material formerly presented in the *FISH Reference* volume.

Examples

An extensive set of examples is provided here. The projects for all of them are included with the program and are accessible from the `Help · Examples...` command. The material here closely corresponds to and expands on what was formerly presented in the print manual volumes *Example Applications* and *Verification Problems*.

Options

All material pertaining to the options (Thermal Analysis, Creep, Dynamic Analysis) or modular additions (Coupling with PFC) for *FLAC3D* are documented here.

FLAC3D Install & Resources

This section provides reference information on getting *FLAC3D* installed and running, information on where to obtain technical support, and revision notes for the current version of the program.

Index: Examples

This page categorically lists all examples that appear in the documentation.

FLAC3D 5.0 to FLAC3D 6.0 Command Mapping

This page gives tables that indicate how old *FLAC3D* commands map to the commands in the new version 6.0 syntax.

FLAC3D 5.0 to FLAC3D 6.0 FISH Mapping

This page lists, where possible, the version 6.0 counterparts to *FISH* functions that were available in version 5.0. Note: There will not necessarily be a direct mapping of old to new function arguments nor of return types. *FLAC3D* 6.0 takes advantage, on occasion, of the new types available in *FISH* (tensors, matrices, etc.) to streamline *FISH* intrinsics.

User Support

We believe that the support Itasca provides to code users is a major reason for the popularity of our software. We encourage you to contact us when you have a modeling question. We will provide a timely response via telephone, email or fax. General assistance in the installation of *FLAC3D* on your computer, plus answers to questions concerning capabilities of the various features of the code, are provided free of charge. Technical assistance for specific user-defined problems can be purchased on an as-needed basis.

The first place to look for support is through the Technical Support Dialog built into the user interface, available in the program menu at “Help/Support...”

If that does not work for some reason, or if you wish to communicate via phone or email, please contact us.

Itasca Consulting Group
111 Third Avenue South, Suite 450
Minneapolis, Minnesota 55401 USA
Phone: (+1) 612-371-4711
Fax: (+1) 612-371-4717
Email: software@itascacg.com
Web: www.itascacg.com

We also have a worldwide network of code agents who provide local technical support. Details may be obtained from Itasca. Note that, in general, we prefer to provide support via email, and we will also generally forward your question to the software agent responsible for your region.

More information about support and related tasks (updating the software, identifying version and license numbers, etc.) is available in the [Documentation & Support](#) sub-section of the [FLAC3D Install & Resources](#) section.

About Itasca Consulting Group Inc.

Itasca Consulting Group, Inc., is more than a developer and distributor of engineering software. Itasca is a consulting and research firm consisting of a specialized team of civil, geotechnical, and mining engineers with an established record in solving problems in many areas:

- Mining Engineering and Energy Resource Recovery
- Nuclear Waste Isolation and Underground Space
- Defense Research
- Software Engineering
- Seismic Engineering
- Groundwater Analysis and Dewatering
- Petroleum Engineering

Itasca was established in 1981 to provide advanced rock mechanics services to the mining industry. Today, Itasca is a multidisciplinary geotechnical firm with over 100 professionals in offices worldwide. The corporate headquarters for Itasca are located in Minneapolis, Minnesota. Worldwide offices of Itasca are: Itasca Denver Inc. (Denver, Colorado); Itasca Consultants AB (Luleaa, Sweden); Itasca Consultants S.A.S. (Ecully, France); Itasca Consultants GmbH (Gelsenkirchen, Germany); Itasca Consultores S.L. (Llanera, Spain); Itasca S.A. (Santiago, Chile); Itasca Consulting Canada Inc. (Sudbury, Canada); Itasca Consulting China Ltd. (Wuhan, China); HydroChina — Itasca R & D Center (Hangzhou, China); Itasca Houston Inc. (Houston, Texas); Itasca Australia Pty. Ltd. (Melbourne, Australia); ASC (Shrewsbury, United Kingdom); and Itasca India Consulting Pvt. Ltd. (Nagpur, India).

Itasca's staff members are internationally recognized for their accomplishments in geological, mining, petroleum, seismology, and civil engineering projects. Itasca staff consists of geological, mining, hydrological, petroleum and civil engineers who provide a range of comprehensive services, such as: 1) computational analysis in support of geo-engineering designs; 2) design and performance of field experiments and demonstrations; 3) laboratory characterization of rock properties; 4) data acquisition, analysis, and system

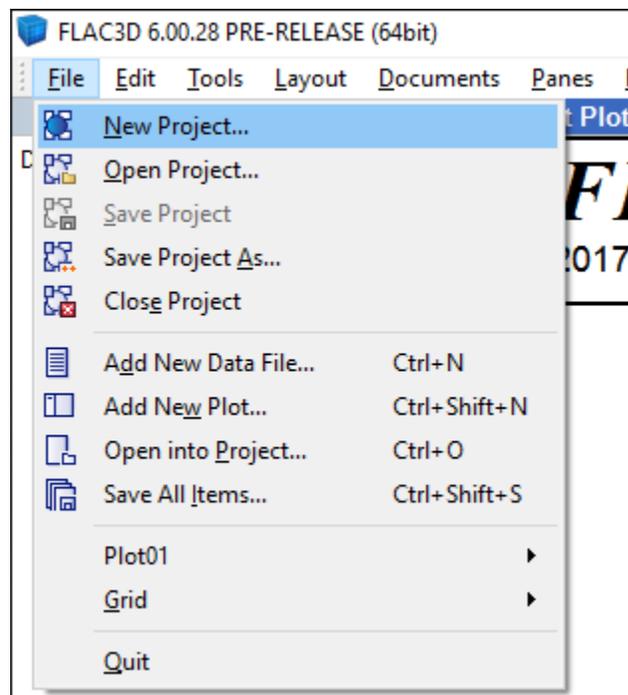
identification; 5) groundwater modeling; and 6) short courses and instruction in the geomechanics application of computational methods. If you should need assistance in any of these areas, we would be glad to offer our services.

Tutorial: Quick Start

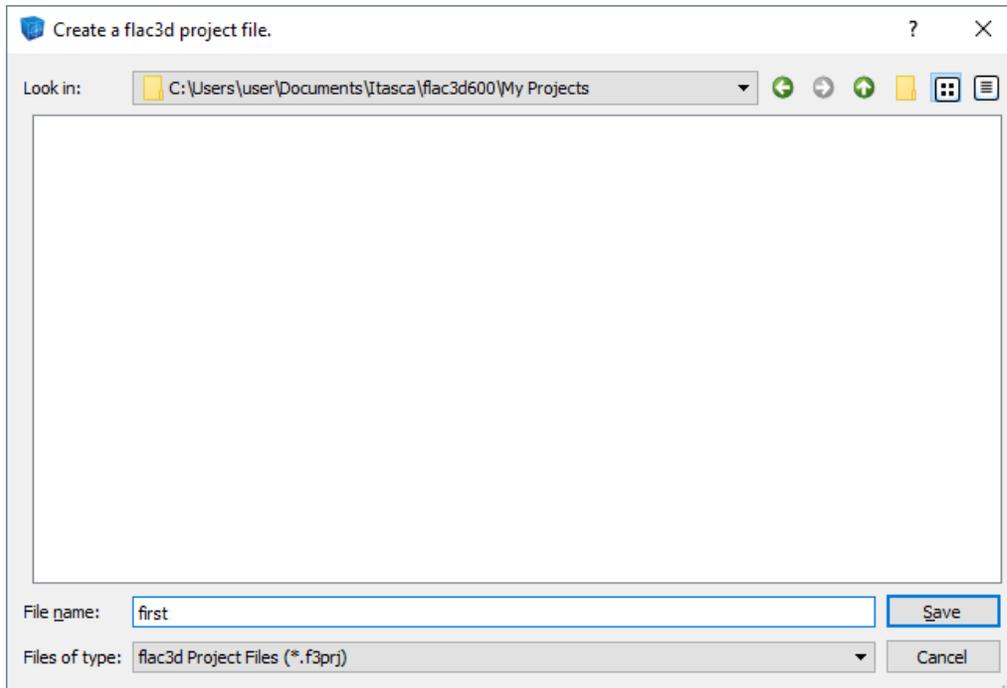
This tutorial steps through the actions necessary to quickly get a first *FLAC3D* model to solution. Very little time is spent explaining the various user interface elements being used. Rather, the focus is on getting something together and providing basic familiarity with the user interface. A more detailed tutorial is presented next in [Tutorial: Illustrative Model — Mechanics of Using FLAC3D](#), which devotes more explanation to the recommended modeling sequence and the user interface elements involved.

It is assumed, at this point, that *FLAC3D* is running and the initial setup dialogs have been processed.

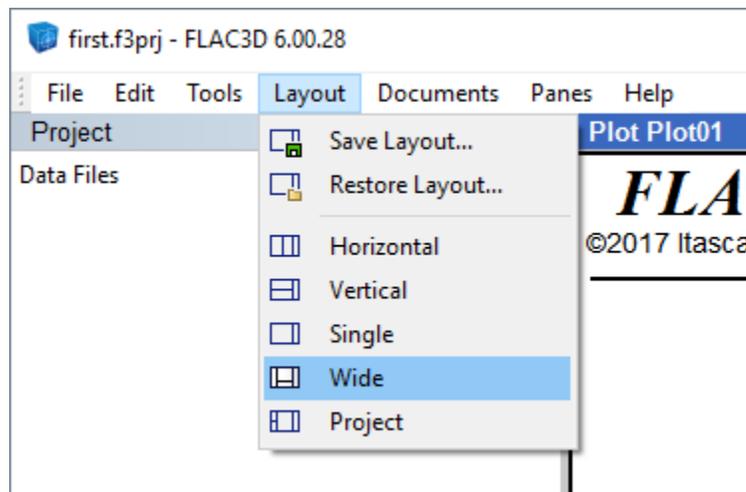
Choose File · New Project from the main menu.



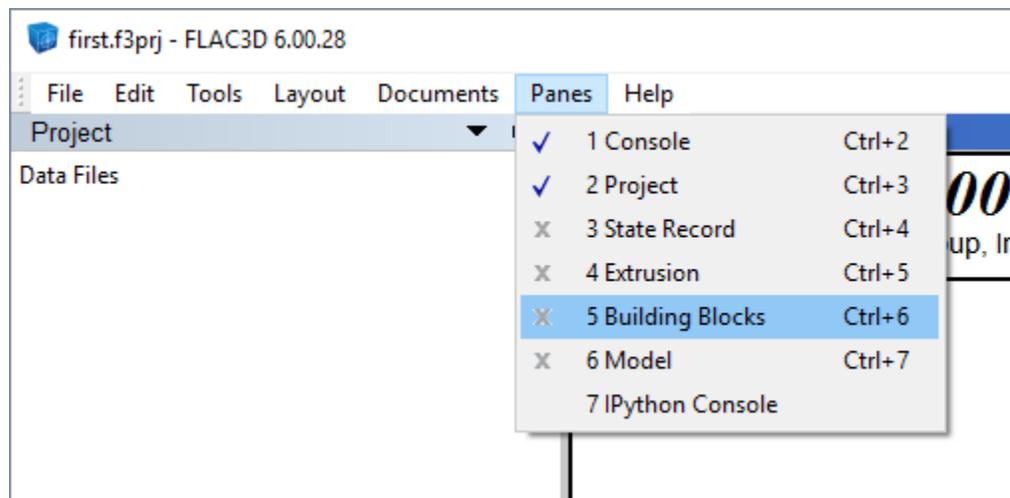
Then save a new project file as “first”.



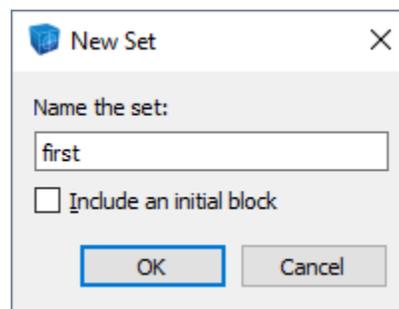
The *FLAC3D* user interface is very flexible and customizable. First configure the program to use the same basic layout as presented in this tutorial. Choose **Layout** ▸ **Wide** from the main menu.



Next, create the model geometry using **Building Blocks**. Choose **Pane** ▸ **Building Blocks** from the main menu.



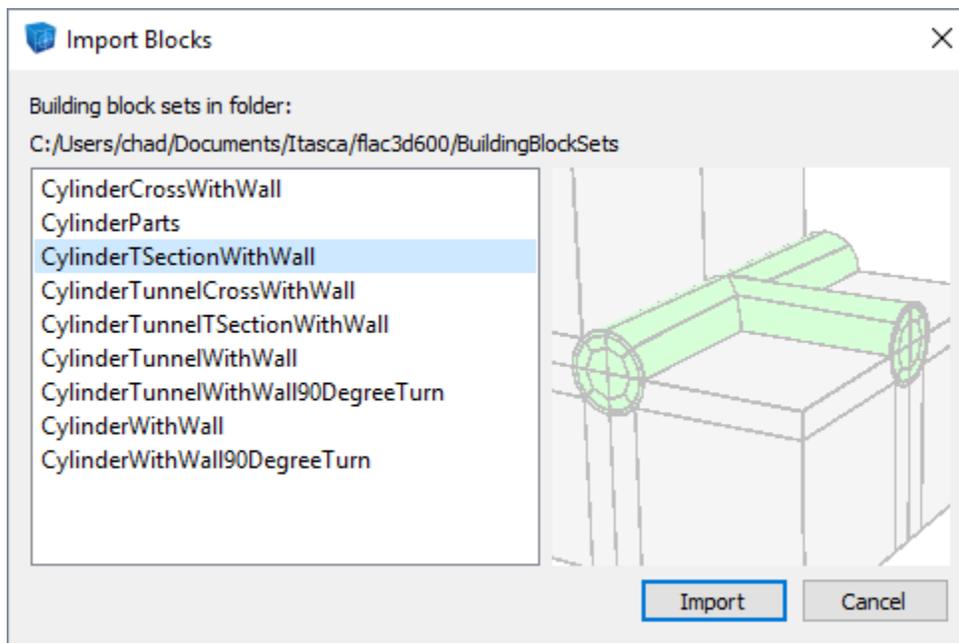
Then left-click anywhere in the blank gray area presented. *FLAC3D* will ask you if you want to create a Building Blocks set. Type “first” as the set name and **uncheck** the *Include an initial block* option. Then press to create the Building Blocks set.



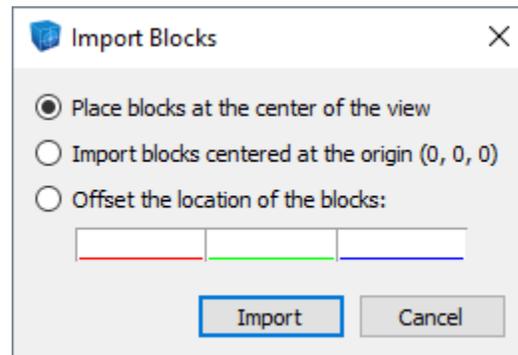
At this point, the set is empty. Press the *Import Blocks* button () on the toolbar.



In the *Import Blocks* dialog, select the “CylinderTSectionWithWall” entry, and press .



Import to the default location by pressing with the *Place blocks at the center of the view* option selected.



Blocks will be imported and be selected on completion of the import. Left-click in a white space area to clear the selection. Right-click and drag to spin the blocks around for viewing. At this point, the model should appear somewhat like this.



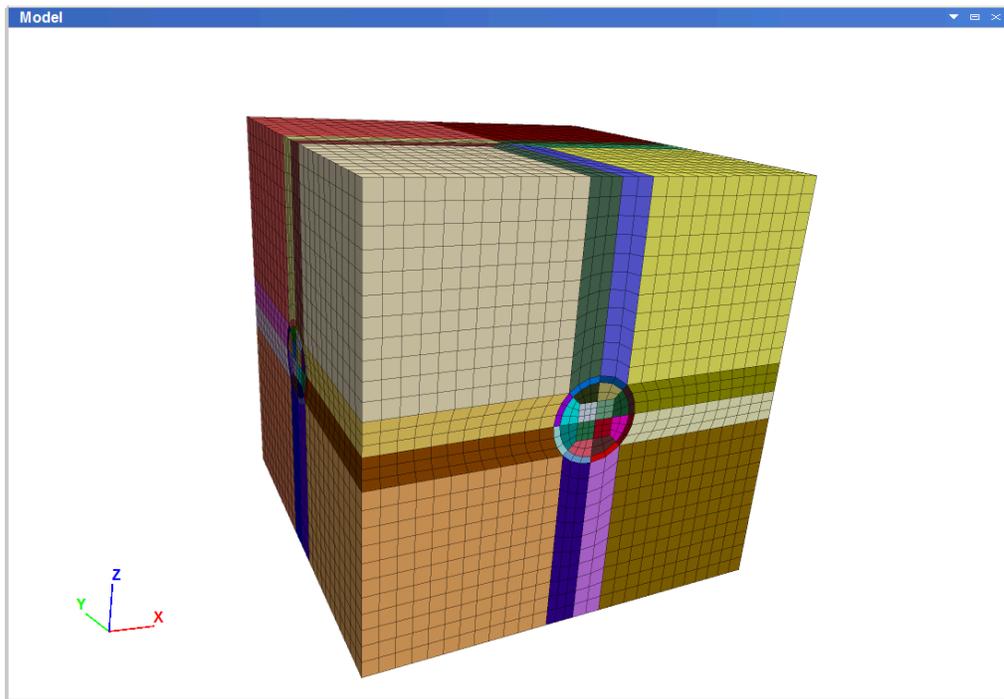
At this point it would be normal to make customizations to match the specific case as needed; however, for this tutorial, we will proceed directly to zone creation from the blocks.

Select the *Generate zones* tool button () at the far right of the toolbar.



This will generate actual *FLAC3D* zones representing the model. These will be shown in the *Model Pane* so they can be seen, named, and manipulated.

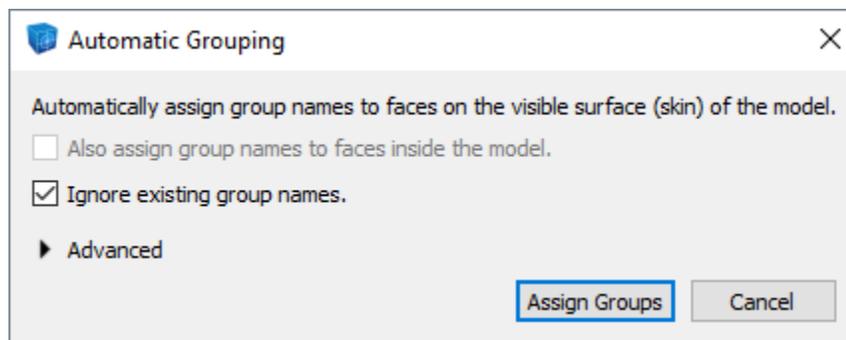
The model should appear something like the image below.



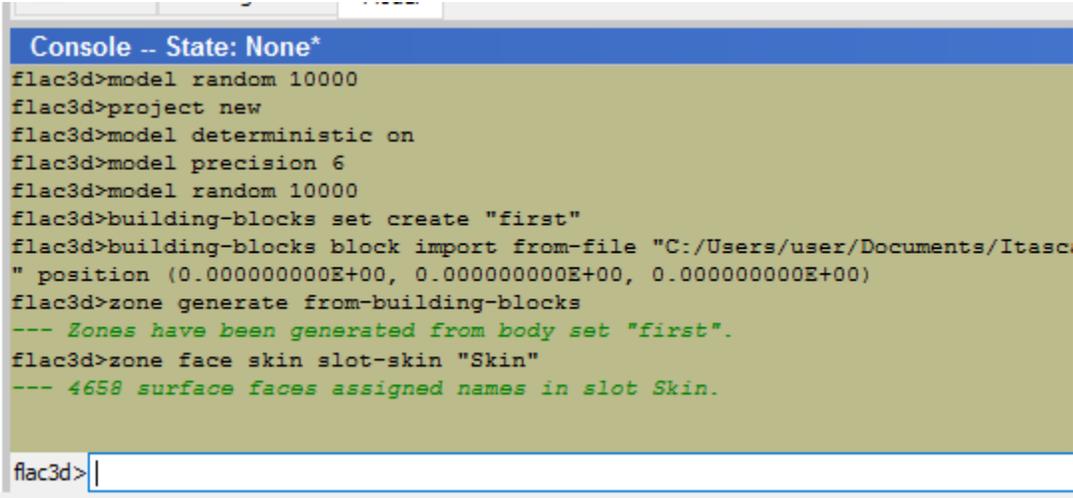
For this simple model, names for the model boundaries are all that will be added in the *Model* pane. Click on the *Assign group names to faces...* tool button () near the right-hand side of the toolbar.



In the *Automatic Grouping* dialog that appears, check *Ignore existing group names* to get the simplest boundary naming. Then press the button.



The initial geometry description and the region naming for the model are complete. It is a good idea to save the model at this point. This will be useful for later reference. Activate (click in) the **Console Pane** at the bottom of the interface.

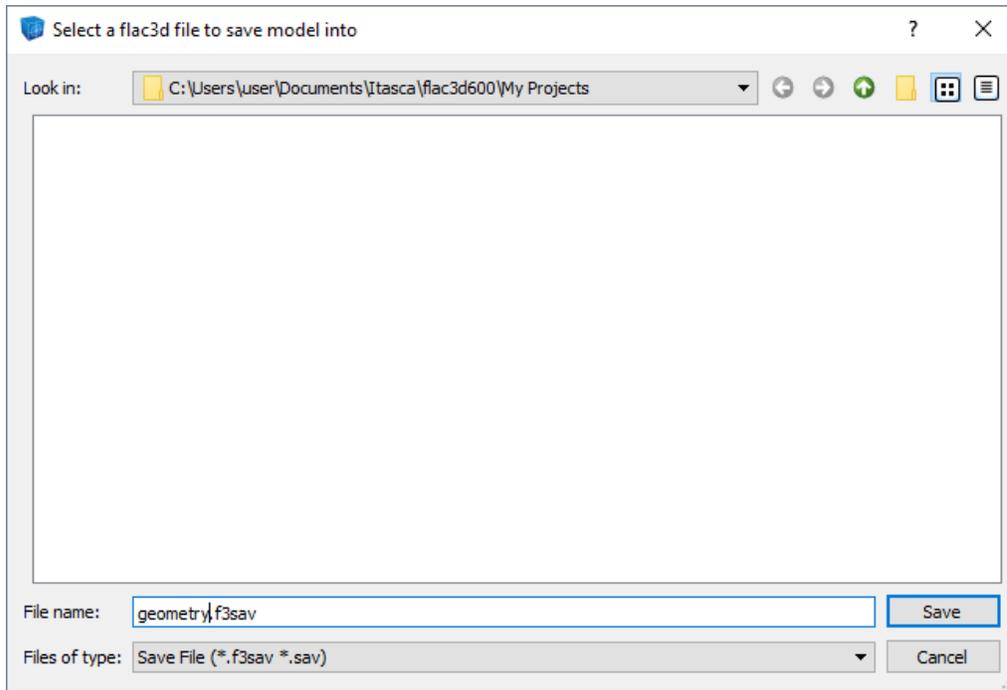


```
Console -- State: None*
flac3d>model random 10000
flac3d>project new
flac3d>model deterministic on
flac3d>model precision 6
flac3d>model random 10000
flac3d>building-blocks set create "first"
flac3d>building-blocks block import from-file "C:/Users/user/Documents/Itasc
" position (0.000000000E+00, 0.000000000E+00, 0.000000000E+00)
flac3d>zone generate from-building-blocks
--- Zones have been generated from body set "first".
flac3d>zone face skin slot-skin "Skin"
--- 4658 surface faces assigned names in slot Skin.
flac3d>|
```

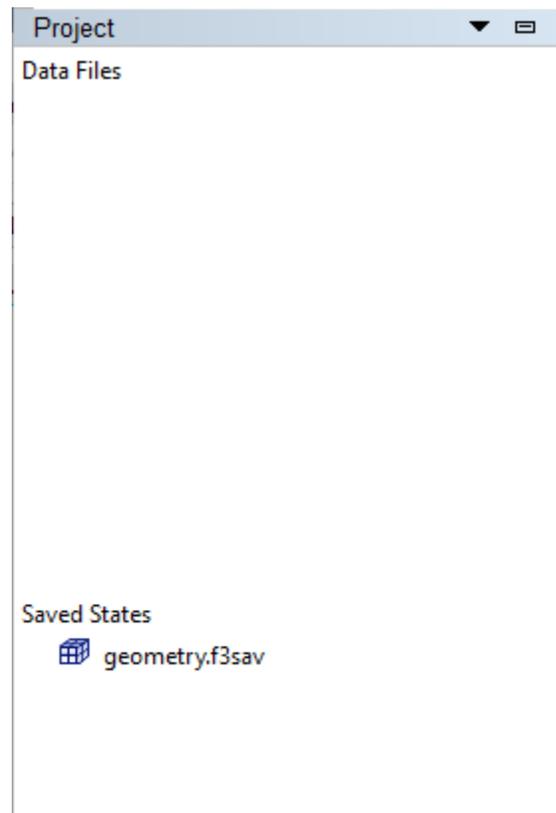
Then in the main toolbar, select the **Save As** tool button ().



Save the current model state as “geometry”.

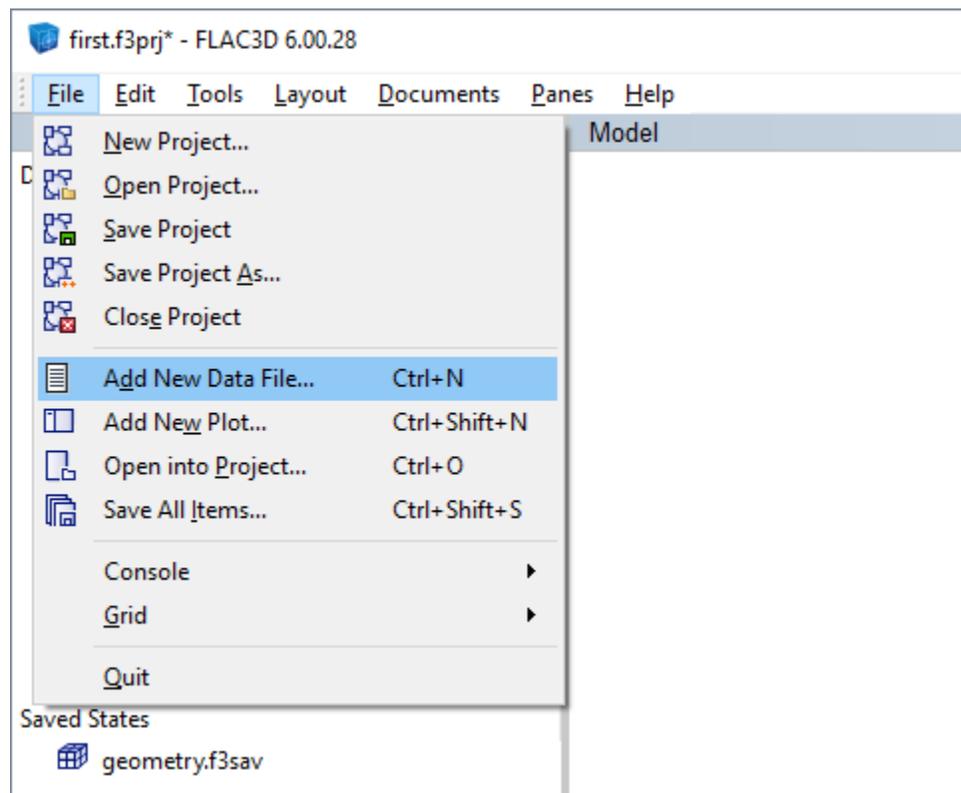


And note that this save state appears in the **Project Pane** at left.

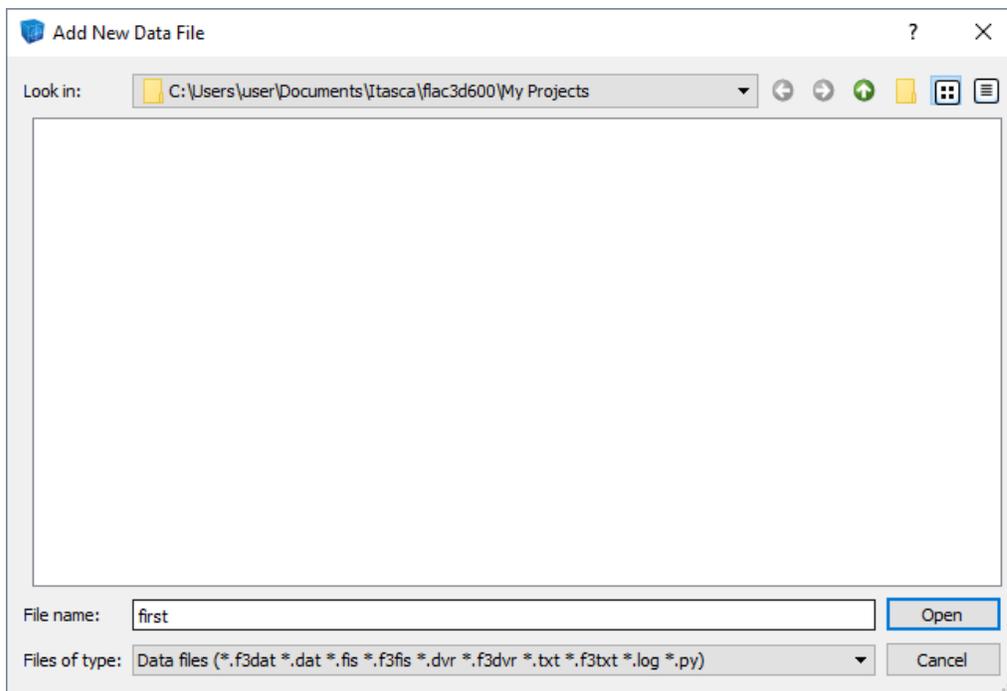


At this point, the *State Record Pane* could be used to create a data file, which in turn could be used to recreate this state whenever necessary. For many purposes, however, it is sufficient to use the save file directly, which is how this simple example will proceed.

Further specification of the model will require directly entering commands in a data file. *FLAC3D* models are entirely *command-driven*, although, as is seen above, some commands may be emitted automatically by interactive user interface elements. To create a new data file in the *editor*, go to the File › Add New Data File... menu entry.



In the ensuing dialog, create a new data file called “first”.



The first command in this data file will restore the geometry model state just created as a starting point, using the `model restore` command. Type (or copy and paste from this document) the following line at the top of the editor. (From this point forward, continue typing or copy-pasting the command listings in the gray boxes, as they appear, into the data file.)

```
model restore "geometry"
```

The next step is to assign the constitutive model and properties for the model. Constitutive models are assigned using the `zone cmodel assign` command, and properties are assigned using the `zone property` command. Note that constitutive models must be assigned first, before properties can be specified for the model(s) assigned.

```
; Constitutive Model and Properties
zone cmodel assign mohr-coulomb
zone property bulk 65000 shear 30000 density 2.0
zone property cohesion 10 friction 34 tension 1.0
```

Next, assign the boundary conditions for the model. In this case the model will have roller boundaries on all four sides and the bottom. The top will be left free. The boundaries were named automatically in the **Model Pane** previously using the *Assign group names to faces...* tool (). The names given were “North”,

“South”, “East”, “West”, “Top”, and “Bottom”. Boundary conditions are typically assigned using the `zone face apply` command, and roller boundaries are created using the `velocity-normal` boundary condition.

```
; Boundary Conditions
zone face apply velocity-normal 0 range group "West" or "East"
zone face apply velocity-normal 0 range group "North" or "South"
zone face apply velocity-normal 0 range group "Bottom"
```

See [Displacement Boundary](#) for a discussion of why this was done as three separate commands.

Next, it is necessary to assign initial conditions. For this simple case, this is only stresses due to gravity, so gravity is assigned using the `model gravity` command. Stresses due to gravity are initialized with the `zone initialize-stresses` command.

```
; Initial Conditions
model gravity 9.81
zone initialize-stresses
```

Next, the model is solved to reach initial equilibrium. Because the internal zoning is somewhat irregular, the `zone initialize-stresses` command comes *close* to equilibrium but is not perfect. Solving the model is necessary to reach complete equilibrium. Since the model is already close, this takes little time. After reaching equilibrium, the state is saved with the name `initial` to serve as a starting point for future changes.

```
; Solve to initial equilibrium
model solve
model save "initial"
```

In a larger *FLAC3D* project, it would be practical to start a new data file with `model restore "initial"`, so the investigation could continue without having to repeatedly create an initial equilibrium state. This example, since it is relatively small and fast, simply continues construction of the model from a single data file.

The next step is to excavate the tunnels. While this could be done by immediately removing the zones (either by changing to the null model using `zone cmodel assign null` or by deleting the zones using `zone delete`), it is better practice to

excavate the zones gradually so quasi-inertial effects do not exaggerate the failure as a result of excavation. This is done using the `zone relax excavate` command as follows.

```
; Excavate Tunnel
zone relax excavate range group "Space"
```

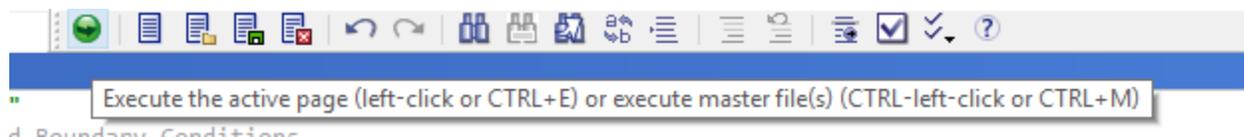
Following that, the model is solved to equilibrium again and the final model state is saved.

```
; Solve to equilibrium after excavation
model solve
model save "first"
```

The complete resulting data file should look like this:

```
model restore "geometry"
; Constitutive Model and Properties
zone cmodel assign mohr-coulomb
zone property bulk 65000 shear 30000 density 2.0
zone property cohesion 10 friction 34 tension 1.0
; Boundary Conditions
zone face apply velocity-normal 0 range group "West" or "East"
zone face apply velocity-normal 0 range group "North" or "South"
zone face apply velocity-normal 0 range group "Bottom"
; Initial Conditions
model gravity 9.81
zone initialize-stresses
; Solve to initial equilibrium
model solve
model save "initial"
; Excavate Tunnel
zone relax excavate range group "Space"
; Solve to equilibrium after excavation
model solve
model save "first"
```

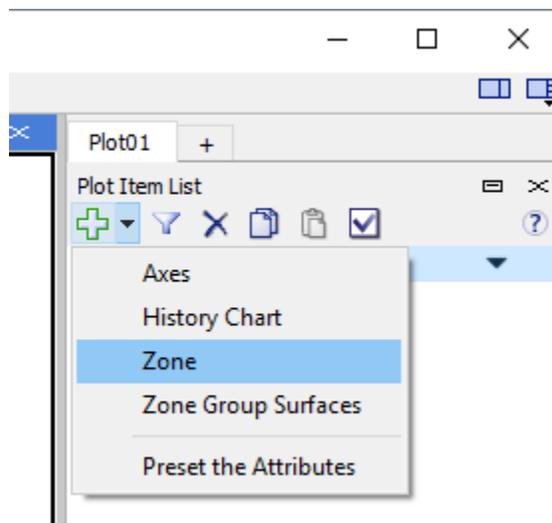
Press the *Execute* button () on the left side of the toolbar to run the data file (see below image).



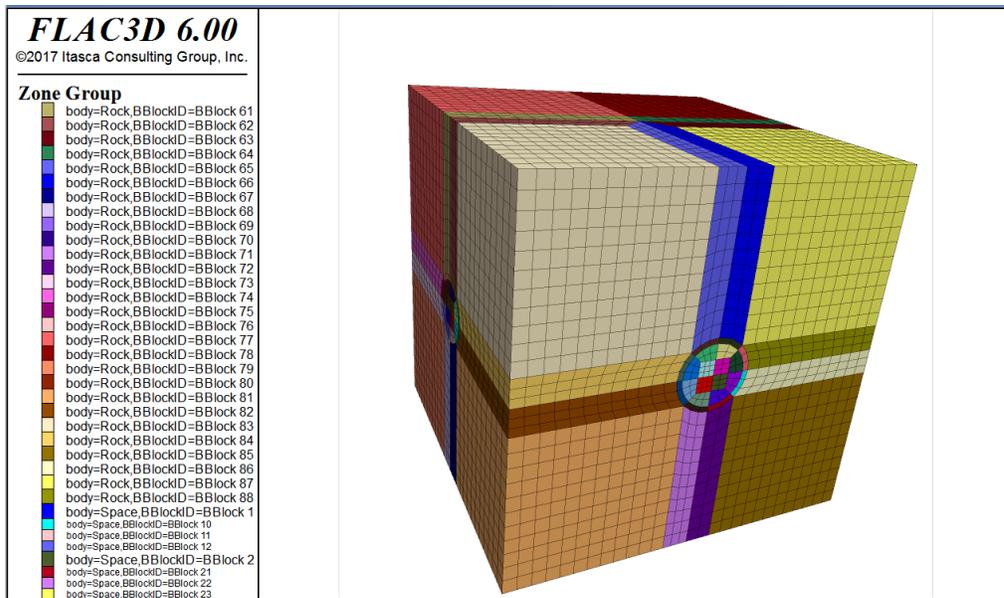
If there are any errors, an error message will appear in a dialog and the line will be highlighted in the editor. Just repair the problem (using this tutorial as a reference if necessary) and press the *Execute* button again.

On the computer used to generate this tutorial, solving to equilibrium after excavation took a little more than a minute. During this time, we can bring up a plot to watch the model evolve and examine the final result. If the cycling information dialog is bothersome, press the button to embed cycling information into the *Console Pane*.

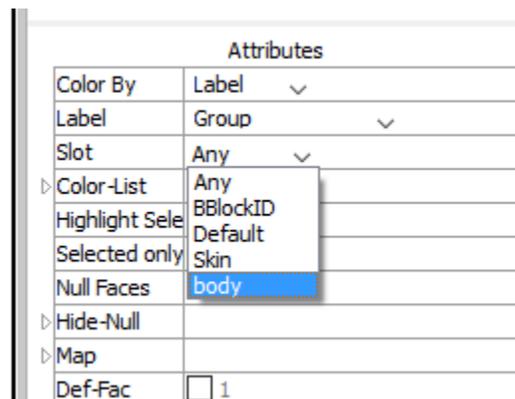
Click on the *Plot01* tab in the large central area of the user interface to bring up the default *Plot Pane*. In the *Control Panel* on the right-hand side, click on the down arrow next to the *Build Plot* tool button (), and select the *Zone* entry.



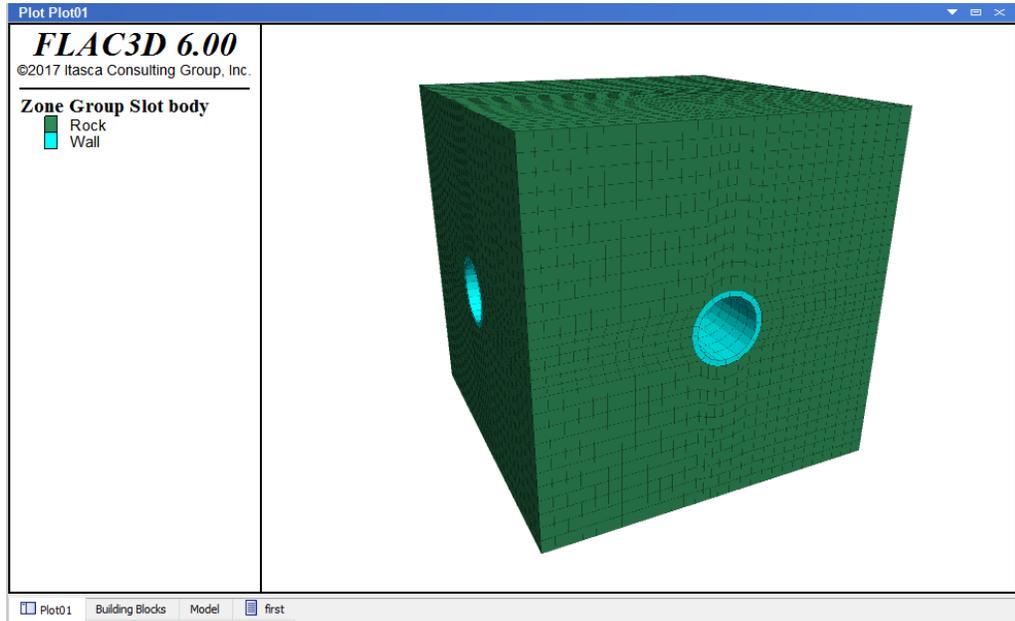
After right-clicking and dragging to look around, the initial plot should appear something like the following.



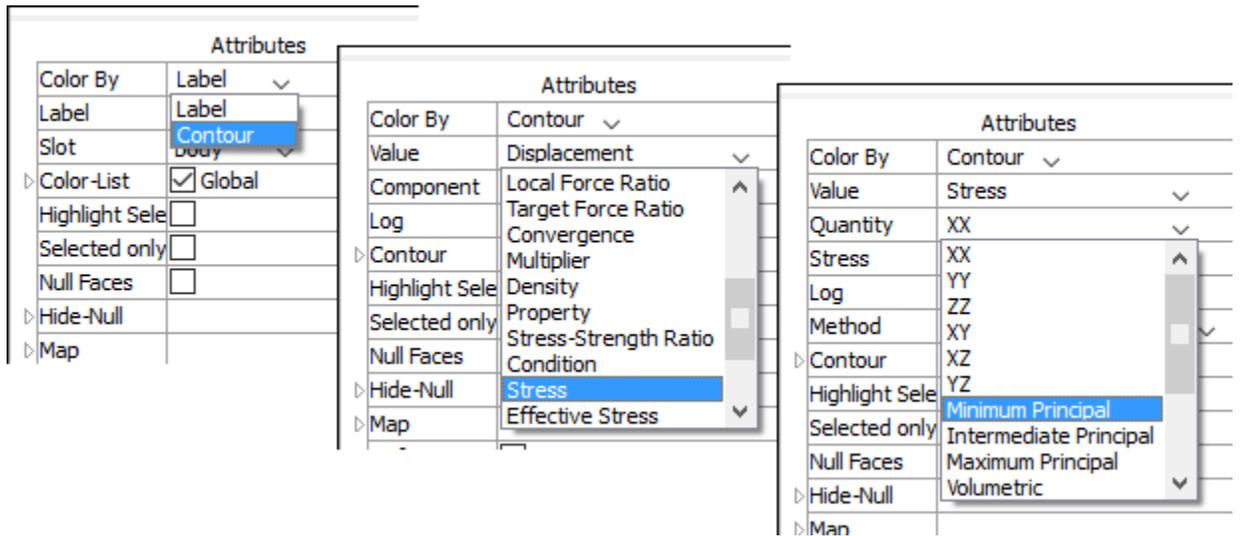
In the *Attributes* section of the *Control Panel*, click in the drop down labeled *Slot* (which is currently set to *Any*) and select *body*.



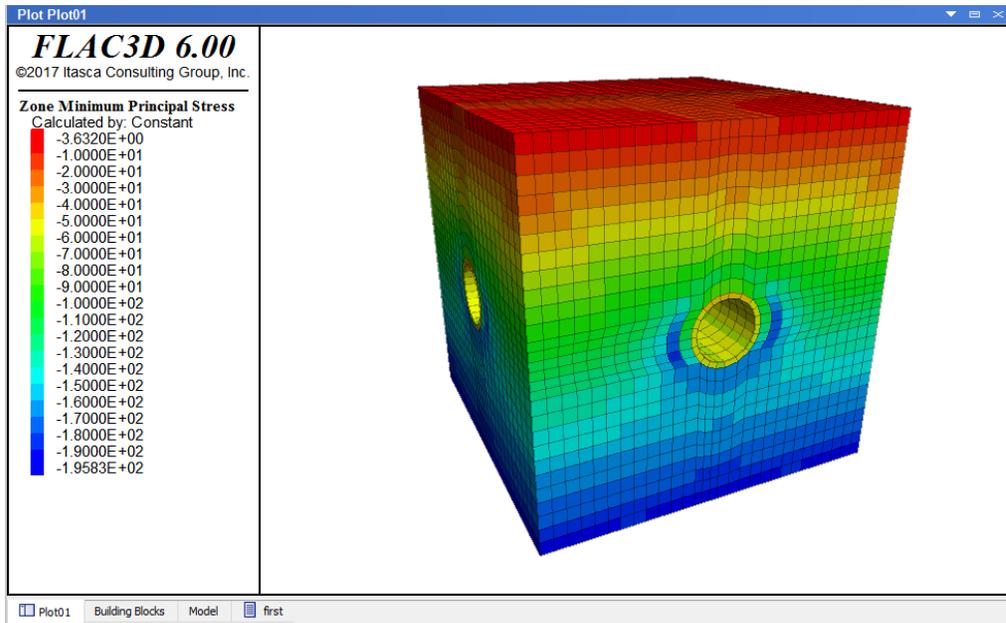
This will show the group assignments made by the Building Blocks template. If the model run has fully completed, the tunnel zones are already excavated in the plot; if it hasn't, the zones may appear in the plot as a third group named "Space".



To see model results, choose *Contour* in the *Color By* attribute, then *Stress* in the *Value* attribute, then *Minimum Principal* in the *Quantity* attribute.

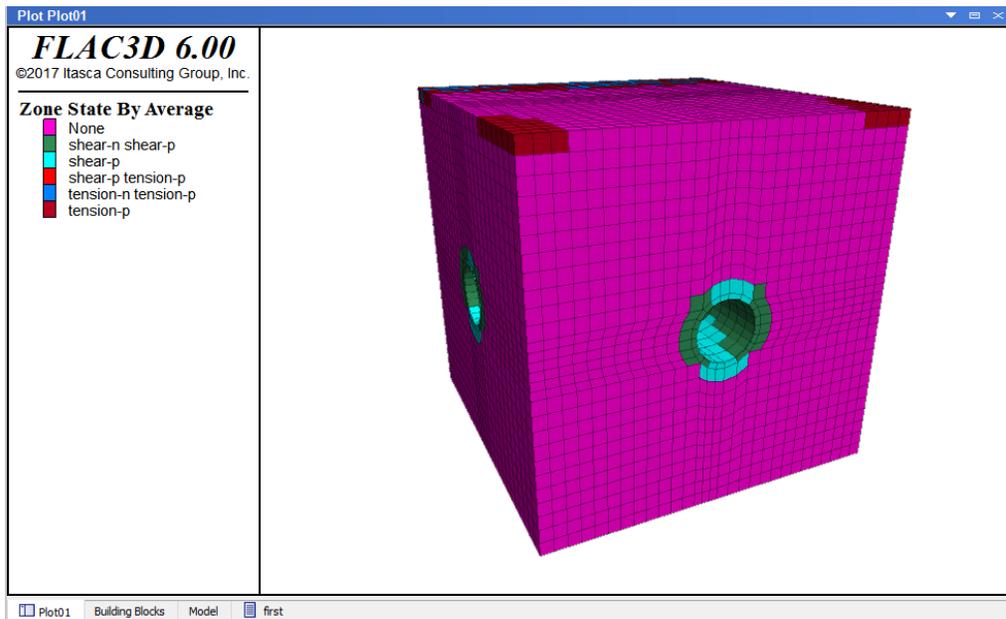


This should give a result similar to the one shown below.



If the solve is still in early stages, it is possible to see the stresses reduce in the zones that make up the tunnel, and then those zones will be removed entirely. If not, the “first.f3dat” data file can be run again to watch it in action.

The extent of failure is seen by selecting *Label* again from the *Color By* attribute, followed by *State* from the *Label* attribute. This should result in a plot that looks like the one below.

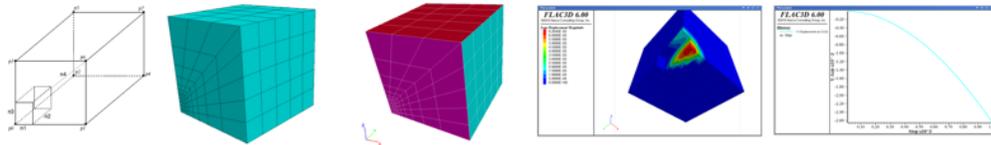


At this point, save the project by selecting File › Save Project on the main menu. Note that the project file stores the plot(s) that have been created. These are *not* part of the model states (save files).

Tutorial: Illustrative Model — Mechanics of Using *FLAC3D*

This tutorial provides the new user an initial look at the basics of modeling with *FLAC3D*. The structure of this tutorial, delineated below, corresponds to the general structure for *FLAC3D* modeling that is presented at length in the sections of the [Problem Solving with *FLAC3D*](#) section and is charted in [General Solution Procedure Illustrated](#).

This tutorial problem presents a trench excavated in a soil with low cohesion. After failure is observed during excavation, support is added to stabilize the trench.



For this tutorial, the sequence charted in [General Solution Procedure Illustrated](#) may be broken down as follows.

First, a model must be set up in *FLAC3D* with the following four steps.

1. Create a project for the model.

The project acts as a central collection of all resources and outputs that will be part of the *FLAC3D* model.

2. Create a mesh that discretizes the model volume.

The grid defines the geometry of the problem.

3. Assign names to regions of the model by using Groups.

Groups are used to label areas of specific interest in the model.

4. Assign constitutive behavior and material properties.

The constitutive behavior and associated material properties dictate the type of response the model will display upon disturbance (e.g., deformational response due to excavation).

5. Apply boundary conditions.
6. Assign initial conditions.

Initial conditions define the in-situ state (i.e., the condition before a change or disturbance in the problem state is introduced).

7. Step to initial equilibrium state.

After the model setup is complete, the initial equilibrium state is calculated for the model. It is usually at this point where the intended analysis can really begin.

8. Alteration of the model as required by the problem.

Alteration(s) are then made (excavate material, change boundary conditions, install support, etc.), and the resulting response of the model is calculated. At this point, the nature of the problem will determine what steps will follow and there is no specific normative approach. In this tutorial, we continue and conclude with the following steps.

9. Examine model response (plotting).
10. Further alteration (install support).
11. Project completion (storage).

Solutions

The actual solution of the problem is different for an explicit program like *FLAC3D* than it is for conventional implicit-solution programs. (This is a large topic; see the [FLAC3D Theory and Background](#) section.) *FLAC3D* uses an explicit time-marching method to solve the discretized equations. The solution is reached after a series of computational steps. In *FLAC3D*, the number of steps required to reach a solution can be controlled automatically by the code or manually by the user. However, the user ultimately must determine if the

number of steps is sufficient to reach convergence to equilibrium. In this tutorial, the way this is done is described in the [t1_06equilibrium](#) and [Further Discussion: Step to Equilibrium](#) topics.

Building a Data File

FLAC3D is a command-driven program. Those commands can be issued from the command prompt, or, in a number of cases, they are generated interactively by the user interface. There is no difference between the commands as issued from one place or the other, merely the matter of efficiency or speed in the issuing, and user preference.

However, there is a third place where commands are saved/recorded: the data file. The data file is essentially a batch file, that is, a list of commands that can be issued and processed in unbroken sequence. This is both more efficient than either the command prompt or mouse/keyboard commands—either of which is limited to issuance of one command at a time—and has the added advantage of providing a beginning-to-end, readable problem representation.

Model construction in *FLAC3D* is thus a matter of data file construction. Though typing a data file in its entirety prior to any command processing is possible, in practice this is rarely done. Instead, a *final* data file is arrived at in a stepwise fashion that is usually a blend of commands from the file itself and/or other data files, the command prompt, and the user interface, not to mention the number of “drafts” that occur in refining the modeling approach taken to best represent the problem. The art of data file construction in *FLAC3D* is a matter of efficiently using all three paths to issue commands, as well as the save/restore system, to work through the problem.

The State Record Pane

In *FLAC3D*, all operations that modify the current *model* state do so by issuing a command. Operations that only affect the *program* or *project* state do not have this requirement and generally do not issue a command in the interface, although commands exist that only change those states. All commands that occur in a given model state are recorded in the [State Record](#). This record is included in save files that represent the model state created using `model save`. Commands are recorded regardless of whether they are generated directly by the user or interactively by the user interface. These commands are available for

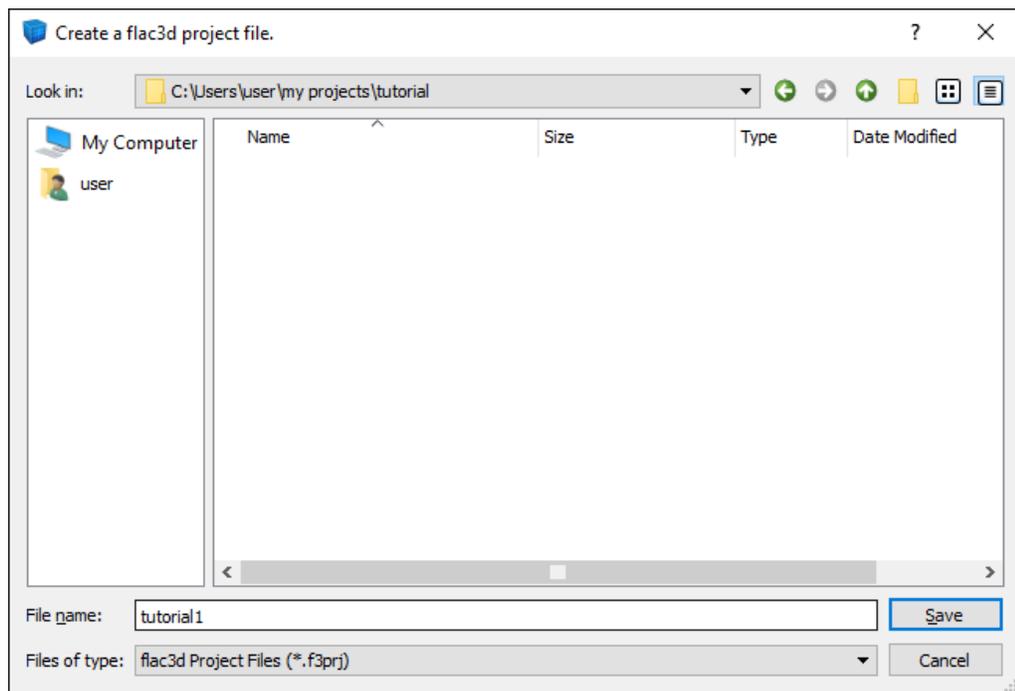
review in the **State Record Pane**. This state record can be played back directly using the `program playback` command, or they can be converted into an equivalent data file for later repetition or parameterization.

The state record does not record the contents of a data file that is executed. These commands are considered to be already recorded in the data file itself. Rather, the record tracks the data file name, size, and last modified stamp for reference.

1. Create a Project

Create a project for this tutorial.

1. Choose File › New Project... from the main menu.
2. Navigate to the folder where the project file will be saved.
3. Name the project “tutorial1” in the *File Name* field.
4. Press or press Enter .



Set Up the Workspace

Setup and save the workspace.

1. Select Layout › Wide .
2. Select Panes › State Record .

3. Select File ▸ Save Project .

When the project is saved in step 3, the state of the workspace (including the current layout settings) is saved with the project. This tutorial will make use of tools available in the *State Record* and *Model* panes, so we will need them both at hand.

Create a Data File

For this tutorial, a single complete data file for the problem will be created.[1]

1. Select File ▸ Add New Data File... or press `Ctrl + N`.
2. At the “File Name” field, enter “tutorial”.
3. Press or press `Enter` .

Note the new data file is made the active pane.

Discussion

When a project file is created and saved, *FLAC3D* uses the folder containing that project file as the current working directory. This location will be the default for most file-managing dialogs created by the user-interface (although some may remember the location of the last file accessed in that session). In addition, any command that saves a file may be issued without a path, as the working directory is assumed if no path is supplied.[2]

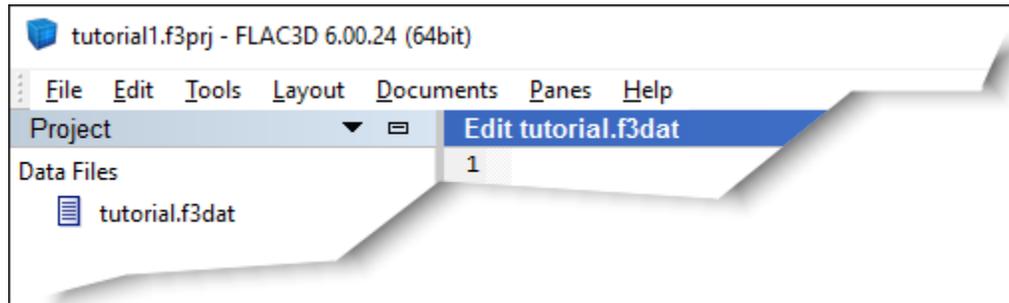
Creating a project at the outset of any modeling process is the *strongly* recommended approach to using *FLAC3D*. The project **handles a range of tasks** for managing the modeling project, in addition to the ones described above. The existence of a project makes retaining settings, re-opening or restoring files, and project support and archiving significantly easier.

Refer to the topic [FLAC3D Projects Introduced](#) and the section [Projects & Files \(the Project Pane\)](#) for complete information on working with projects.

Projects: In the Interface

After creating the project and data file, some additional aspects of the program’s behavior may be noted.

- The *FLAC3D* program title bar shows the project name. It also reports the full version of the program as well.



- The data file has been added to the *Project* pane (be sure you are using the “Wide” layout; select Layout · Wide if your *FLAC3D* does not match the image shown).
- Also note that when the data file is created, a blank data file is opened in an *Edit* pane, which is stacked in a tab set on top of the *Plot01* pane and automatically made the active pane (the title bar of the active pane in the program is colored a darker blue than those of the inactive panes). In general, creating a new file (or plot, or other program resource) will make the pane containing that resource the active pane.
- The name of the data file appears at the top of the *Edit* pane containing the file.
- The *Control Panel* and the program toolbar contain tools for working in the *Edit* pane. To further illustrate, select the *Plot01* tab adjacent to the *tutorial* tab and see how the *Control Panel* and toolbar change. Next click the title bar of the *Console* pane, then click the title of the *Project* pane. Note the changes, and that in each case the clicked item becomes the active pane. At all times, the *Control Panel* and the program toolbar will contextually supply tools/facilities based on the pane type that is active.

Endnotes

- [1] The file created in this tutorial is supplied with the program, so if the user prefers, the file can simply be loaded and the tutorial can be followed without building the file. The file is:

```
[currentuser]\Documents\Itasca\flac3d600\datafiles\UsersGuide\Tutorial\  
Illustrative\tutorial.f3dat
```

- [2] It is possible to *override* the use of the project file's folder as the current working directory using the `program directory custom` command. It is also possible to save a file from the command prompt to a folder other than the project folder by supplying an explicit path.

2. Generate the Grid

The grid for this problem is created with one command.

1. Type the lines below into the “tutorial.f3dat” data file (or you can cut and paste them directly from the listing below).

```

; create geometry/zones
model new
zone create radial-brick      ...
    point 0 (0,0,0) point 1 (10,0,0) ...
    point 2 (0,10,0) point 3 (0,0,10) ...
    size 3,5,5,7      ...
    ratio 1,1,1,1.5  ...
    dim 1 4 2 fill group "exc"
model save "geom"

```

2. Press the *Execute* button (). The commands are processed and echoed in the *Console* pane output.
3. Activate the *Model* pane (click the tab that says “Model”).
4. Right-click and drag on the model to 3D rotate. See [View Manipulation Quick Reference](#) for more ways of manipulating the view if you’d like to go further in examining the model.

Discussion

The first line clears *FLAC3D* of model-state information (see `model new` for reference). To keep things simple, we will re-run the entire data file using  each time we add lines as this tutorial progresses. Putting `model new` at the start ensures the model will run without error each time we do so.

The second line creates the geometry. In that line, a `radial-brick` primitive is used to create zones. The keywords that follow the name of the primitive specify size, grid spacing (ratio), and brick-size (dimensions). This command occupies six typed lines of text, but through use of the continuation operator (`...`) *FLAC3D* treats these as one line, making it a single command.

Creating a problem's grid with primitives using the `zone create` command is a fast and efficient method of constructing zones. However, as the complexity of model geometry increases, the use of primitives also gets increasingly difficult, and *FLAC3D* offers other methods of constructing grids that may be preferable. An introduction to zone construction methods and the multiple considerations that go into grid construction are described in [Grid Generation](#).

It is a recommended practice to save the model state at the point where the zones for the problem geometry are created. Since one `zone create` command is sufficient to obtain the geometry needed for this problem, the final line above saves the model state in the file "geom.f3sav" (note the file extension is added automatically).

Further Discussion: Meshing With Primitives

One basic method of grid generation is performed via the `zone create` command and its associated keywords, which defines the number of zones in a model and shapes the grid to fit a specified problem region.

Several primitive shapes are built into the generator to expedite mesh generation for simple problem shapes. These include brick, wedge, pyramid, and cylinder shapes. The example listing below is the command used in [Tutorial: Use of Common Commands](#) to create a brick-shaped grid containing six zones in the x -direction, eight zones in the y -direction, and eight zones in the z -direction.[1]

```
model new
zone create brick size 6,8,8
```

The number of zones is specified by the `size` keyword.

Be careful when selecting the number of zones for a model, because a balance must be struck between the accuracy required and the solution speed.

Although there are many aspects of a *FLAC3D* model that affect the calculation speed, once the basic parameters are defined, the speed varies directly as a function of the number of elements. As a rule-of-thumb, models containing up to roughly 5000 elements will typically reach a solution state for a given alteration in approximately 2000 to 4000 steps. On a 2.67 GHz Intel i7 computer, the runtime for a 5000 element model to perform 4000 steps is roughly 30 seconds. Check the speed of calculation on your computer for the specific model to estimate the runtime required.

When building a *FLAC3D* model, it is a good practice to start with a grid that has fewer zones to perform simple test runs and make refinements to the grid. Then, increase the number of zones to improve the accuracy.

In its simplest form, the `zone create` command can supply new coordinates to a grid. For example, assume that the actual coordinates of the trench model run from -10.0 to 10.0 in the x -direction, -10.0 to 10.0 in the y -direction, and -20.0 to 0.0 in the (vertical) z -direction. The next listing shows how this would be specified.

```
model new
zone create brick size 6,8,8 ...
    point 0 -10, -10, -20 ...
    point 1 10, -10, -20 ...
    point 2 -10, 10, -20 ...
    point 3 -10, -10, 0
```

In the preceding example, `point 0`, `point 1`, `point 2`, and `point 3` are keywords that denote four of the corners of a standard *FLAC3D* mesh (see diagram below). Notice that an ellipsis (...) is given at the end of the first, second, third, and fourth lines. The ellipsis denotes that the next line is a continuation of the same command, consequently the first command, `zone create`, appears on five typed lines in the text but is processed by *FLAC3D* as a single line.

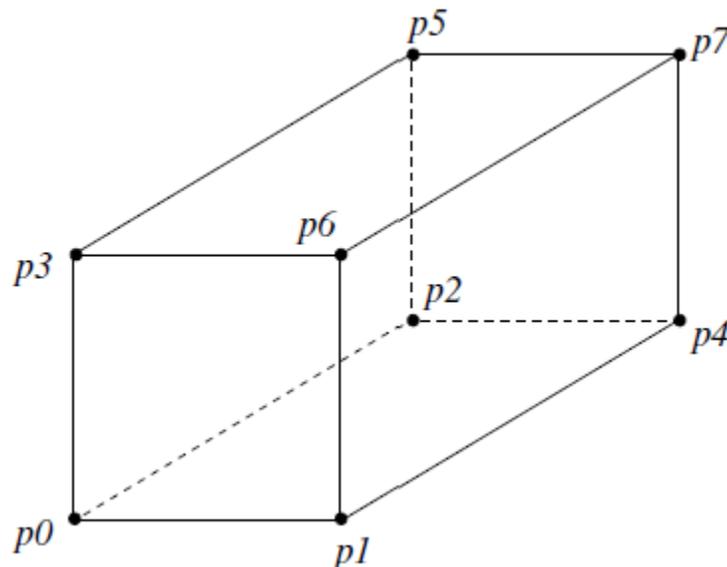


Figure 1: Corner coordinate keywords for a standard *FLAC3D* brick mesh shape.

Only four corners are required to define a parallelepiped-shaped mesh. More corners can be specified to define an irregular surface. The next example shows how to make a sloping surface at the top of the mesh.

```

model new
zone create brick size 6,8,8 ...
  point 0 -10, -10, -20 ...
  point 1 10, -10, -20 point 2 -10, 10, -20 ...
  point 3 -10, -10, 0 point 4 10, 10, -20 ...
  point 5 -10, 10, 10 point 6 10, -10, 0 ...
  point 7 10, 10, 10

```

The coordinates corresponding to the keywords `point 0`, `point 1`, `point 2`, `point 3`, `point 4`, `point 5`, `point 6`, and `point 7` can be located at arbitrary points in space. However, the eight corners must have the topological sense shown in the diagram above (i.e., `point 0`, `point 1`, `point 2`, `point 3` must form a right-handed system) to create positive zone volumes. Non-coplanar faces are permitted, but they should be avoided if possible because a small volume error may be introduced.

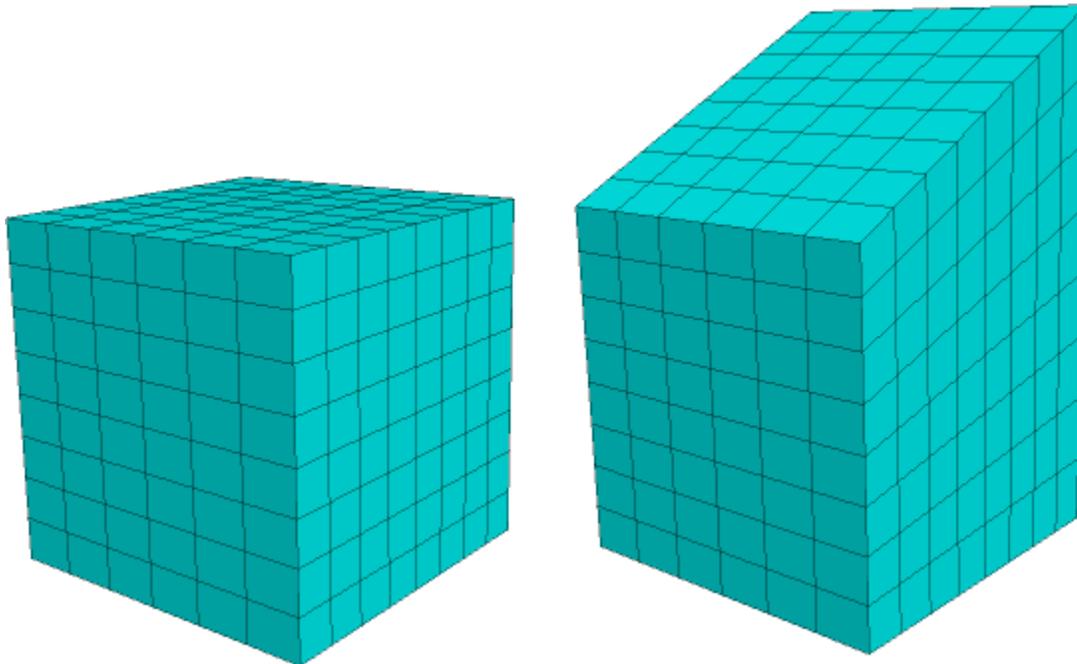


Figure 2: The simple *brick* and the sloped-surface *brick* from the preceding two examples.

Note the boundaries of the model influence the results. The boundary must be placed far enough away from the excavation to reduce these effects. A gradually graded mesh can be created in *FLAC3D* to move the model boundaries farther out without significantly increasing the number of zones. For example, the command `zone create radial-brick` creates a radially graded mesh around a brick-shaped mesh. The command in the next example creates a $3 \times 5 \times 5$ zone brick-shaped mesh surrounded by a seven-zone radially graded mesh.

```
model new
zone create rad-brick ...
point 0 (0,0,0) point 1 (10,0,0) point 2 (0,10,0) point 3 (0,0,10) ...
size 3,5,5,7 ...
ratio 1,1,1,1.5 ...
dim 1,4,2 fill
```

The x, y, z coordinate system in *FLAC3D* is always right-handed and, as a default, the z -axis is drawn in the vertical direction on the screen. In the simple `brick` examples above, we assumed the z -axis was pointing in the vertical direction. However, we do not have to interpret the z -axis to mean the up direction. For this one, we will assume that the y -axis is the vertical direction. As long as we define the model in a right-handed system, we can create the grid in any direction we desire.

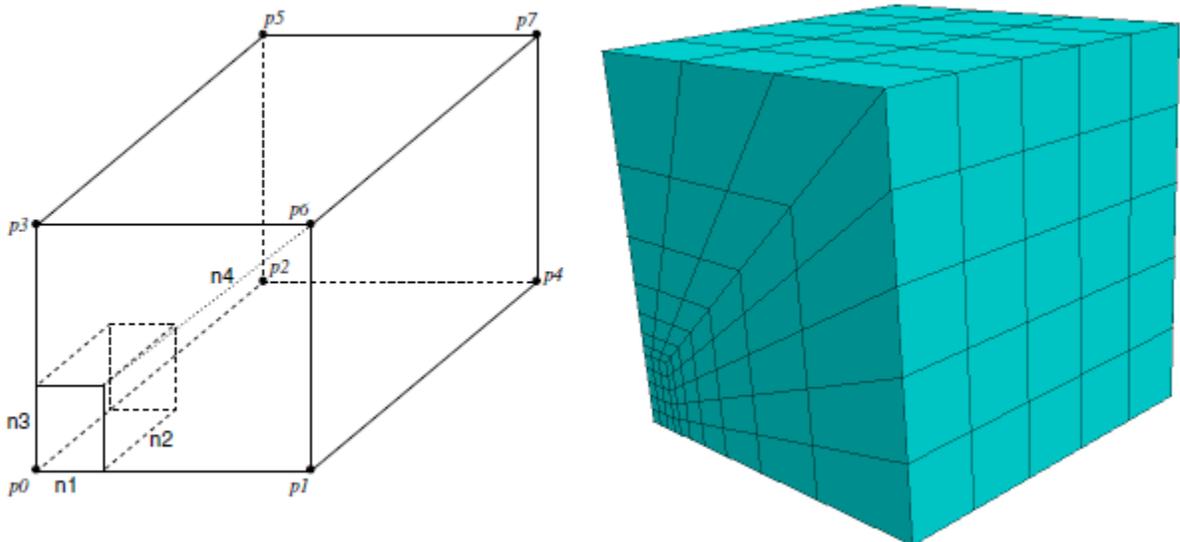


Figure 3: (Left) Element numbering for a radially graded mesh around brick (size $n1\ n2\ n3\ n4$); (Right) the grid resulting from the previous example.

The `size` keyword specifies the number of zones in the brick, and the number radially surrounding the brick (as seen in the resulting grid on the right). The keyword `ratio` is given to manipulate the grid spacing. The four values that follow `ratio` are geometric ratios between successive zone sizes. The first three values are ratios for the zones in the brick, and the fourth value is the ratio for the zones surrounding the brick. In this example, the ratio is 1.0 for the brick zones and 1.5 for the zones surrounding the brick. This causes each successive ring of zones surrounding the brick to be 1.5 times larger in the radial direction. The `dimension` keyword defines the dimensions of the brick region (i.e., 1 m × 4 m × 2 m). The `fill` keyword fills the brick region with zones. If `fill` is omitted, no zones will be generated within the brick region.

Several `zone create radial-brick` commands can be given to create different regions of a model. For example, the preceding example creates one-quarter of the grid for the trench model in this tutorial. The following example shows how four `zone create radial-brick` commands that create quarter symmetries may be combined to create an entire grid for the problem.

```

model new
zone create radial-brick ...
    point 0 (0,0,0) point 1 (10,0,0) ...
    point 2 (0,10,0) point 3 (0,0,10) ...
    size 3 5 5 7 ...
    rat 1 1 1 1.5 ...
    dim 1 4 2 fill
zone create radial-brick ...
    point 0 (0,0,0) point 1 (0,0,10) ...
    point 2 (0,10,0) point 3 (-10,0,0) ...
    size 5 5 3 7 ...
    rat 1 1 1 1.5 ...
    dim 2 4 1 fill
zone create radial-brick ...
    point 0 (0,0,0) point 1 (-10,0,0) ...
    point 2 (0,10,0) point 3 (0,0,-10) ...
    size 3 5 5 7 ...
    rat 1 1 1 1.5 ...
    dim 1 4 2 fill
zone create radial-brick ...
    point 0 (0,0,0) point 1 (0,0,-10) ...
    point 2 (0,10,0) point 3 (10,0,0) ...
    size 5 5 3 7 ...
    rat 1 1 1 1.5 ...
    dim 2 4 1 fill

```

Notice that *size*, *ratio*, and *dimension* all refer to local axes as defined by point 0, point 1, point 2, point 3 in the radial-brick diagram above, *not* to the global *x*, *y*, *z*-axes. The values will change, depending on the specification of point 0, point 1, point 2, and point 3. The radially graded trench model is shown in the next figure. Be sure that *all* gridpoints match at boundaries between shapes. Errors will occur in your analysis if gridpoints of connected shapes are not at the same locations along boundaries. Refer to [Grid Generation](#) for further discussion on the creation of meshes and the proximity tolerances of adjacent gridpoints.

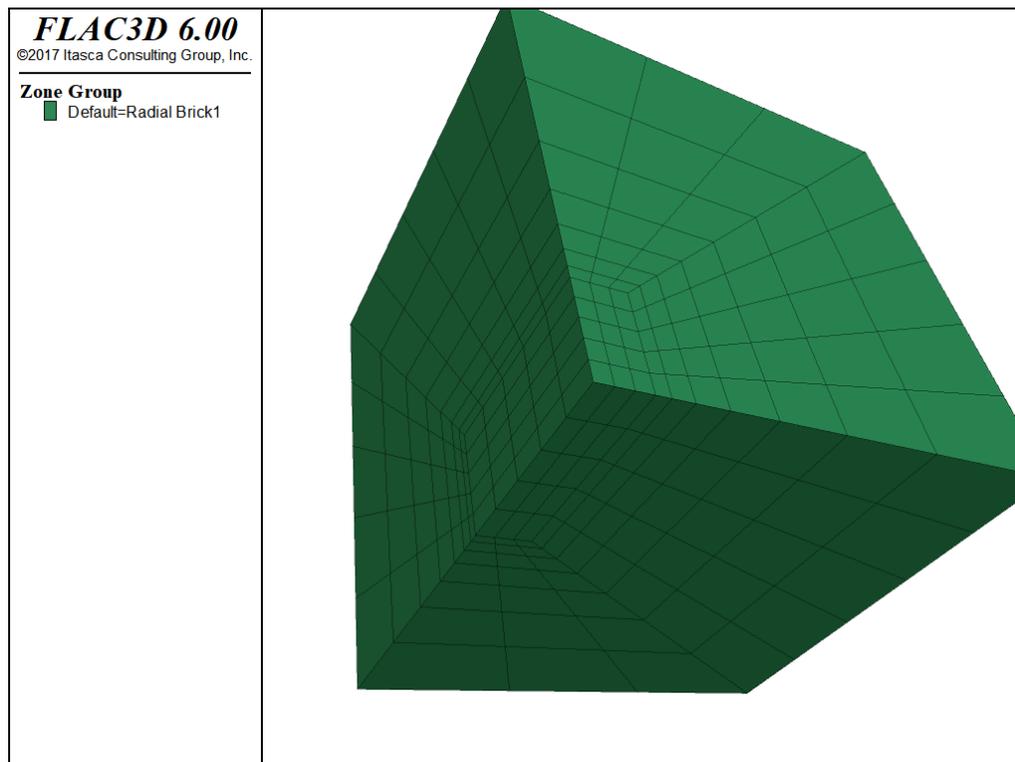


Figure 4: Radially graded trench model.

The preceding example illustrates the requirement of accurate mapping of adjacent elements in *FLAC3D*. There are other ways to achieve the same result shown above, because we can take advantage of planes of symmetry. Parts of the model can be reflected about specified planes to produce a larger model. The following example shows how this can be achieved.

```
model new
zone create radial-brick ...
      point 0 (0,0,0) point 1 (10,0,0) ...
```

```

point 2 (0,10,0) point 3 (0,0,10) ...
size 3,5,5,7 ...
ratio 1,1,1,1.5 ...
dim 1 4 2 fill
zone reflect dip 0 dip-direction 90
zone reflect dip 90 dip-direction 90

```

The meshes resulting from the preceding two examples are the same. However, looking at “Plot01”, one difference may be observed: in the first example, four zone groups are created (as seen in the figure above), while in the second, only one zone group is created. This is because *FLAC3D* will automatically group all the zones that are generated from the operation of a `zone create` command.

About the Use of `ratio`

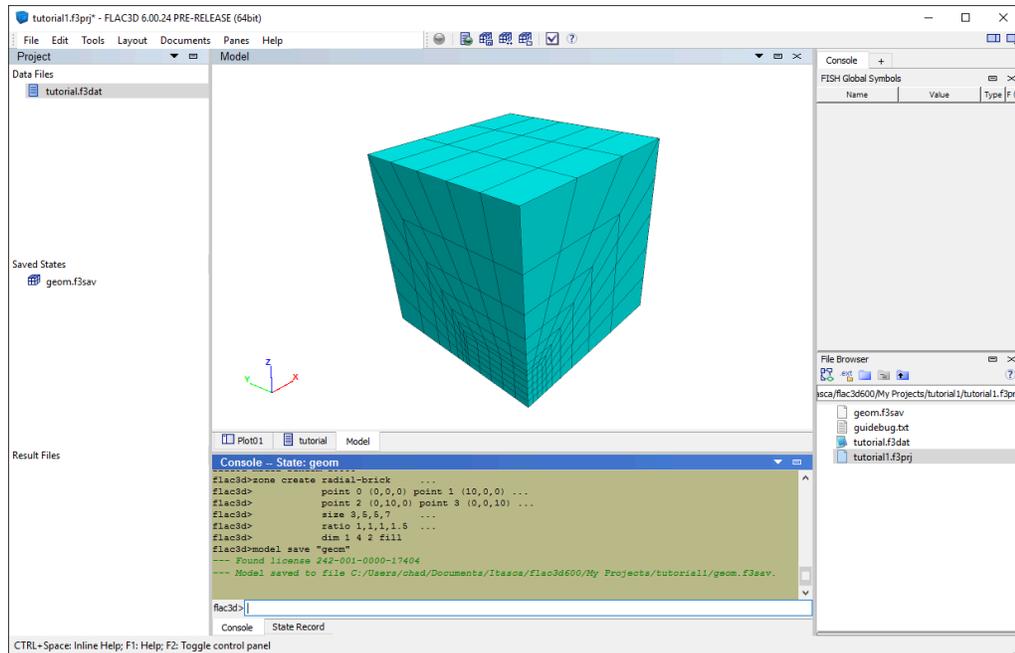
When selecting a value for the geometric ratio, there are a number of criteria that should be considered.

- Finer meshes lead to more accurate results in that they provide a better representation of high stress gradients.
- Accuracy increases as zone aspect ratios tend to unity.
- If different zone sizes are needed, a more gradual change from the smallest to the largest will improve the results; abrupt changes in zone size should be avoided.

However, as the mesh is made finer and the number of zones increases, more RAM is required and the computation time lengthens. Improving the mesh geometry by adjusting the zone size will certainly improve RAM efficiency, and will reduce computation time. It will also take some time to plan and set up. Moving from a “brute force” approach (i.e., single-sized elements in a large uniform mesh) to a more elegant solution will improve RAM use and calculation speed when dealing with a more complicated geometry. [Grid Generation](#) presents an overview of the methods of grid generation available in *FLAC3D*.

Meshing: In the Interface

After processing the commands that generate the problem mesh, some aspects of the interface can be observed.



- An asterisk (*) appears next to the project name in the title bar to indicate the project is not saved. In *FLAC3D*, when an asterisk appears adjacent to a name in a list or on a title bar, it indicates that the item has been modified since it was last saved.
- The “tutorial.f3dat” file *does not* have an asterisk next to it. This is because the file was *automatically* saved when the *Execute* button () was pressed—which is the normal program response when that button is used.
- The *Project* pane now lists “geom.f3sav” in the “Saved States” list.
- The *Model* pane displays the model zones that are created.
- Click on the title bar of the *Project* pane, then click the title bar or the tab for the *Model* pane, then the *Console* pane. When clicked, the title bar of the clicked pane turns darker blue to indicate it is now the active pane. Observe the effect of changing the active pane on the toolbar and the *Control Panel* —tracking what pane in the program is active is important, since the toolbar and the *Control Panel* supply facilities that are specific to that pane type.

- The commands that were processed from the data file when the *Execute* button () was pressed are echoed in the console output. The observant will note that the effect of pressing the button is really to cause a `program call` command to be issued for the currently active data file.

Endnote

[1] If you want to run the examples shown in the “Further Discussion: Meshing With Primitives” section, there are two approaches you can take.

- Load the files:* Each file shown on this page is supplied with the program. You can open the file into the tutorial project by using the menu command `File > Open Into Project...` (or press `Ctrl + O`). The files are located in the current user’s application data area in the folder:

```
[appdata location]\datafiles\UsersGuide\Tutorial\Illustrative
```

The files shown in this section, in the order they appear, are:

```
simple-brick.f3dat
using-actual-coordinates.f3dat
sloping-surface.f3dat
regular-radial.f3dat
combining-gen.f3dat
reflecting.f3dat
```

- Build the files:* You can copy and paste the examples shown on this page into a new data file(s). Open a data file for each example, or paste each into a single data file. If you choose the latter approach, to run a specific example you should select the lines of the example and press the “Execute Selected” button () on the toolbar.

All the examples begin with a `model new` command, which will clear the tutorial example model (or any other preceding example) before the current example is run.

3. Create Model Groups

Next we obtain zone face groups that will be used later when setting up boundary and initial conditions. In addition, we create the five excavation stages by grouping the zones that will be removed from the model at each stage.

1. Make the *Model* pane active, if it isn't already (click "Model" tab).
2. Press the *Auto Assign Face Groups* button ().
3. In the ensuing dialog, check *Ignore Existing Group Names* and press .
3. In the lower center tab group, make the *State Record* pane active by clicking anywhere in the pane (if currently on top of the tab group) or on its tab (if currently "beneath" another pane).

4. Select the line

```
>>zone face skin slot-skin "Skin"
```

Right-click and choose *Copy as Data File*.

5. Make the tutorial data file active, set the cursor at the end of the file, and paste (`Ctrl + V` or `Edit ▸ Paste`).
6. Create groups for each of the five excavation stages. Type (or copy-paste from below) the lines that create the five groups ("exc1" through "exc5") in the data file.

7. At the end of the data file, type (or copy-paste from below) `model save "groups"`. The commands from this step and step (6) that have been added to “tutorial.f3dat” should appear as below:

```
zone group "exc1" range position (0, 0,0) (1,0.8,2)
zone group "exc2" range position (0,0.8,0) (1,1.6,2)
zone group "exc3" range position (0,1.6,0) (1,2.4,2)
zone group "exc4" range position (0,2.4,0) (1,3.2,2)
zone group "exc5" range position (0,3.2,0) (1,4.0,2)
```

Note that these commands could also have been generated interactively using range selection tools in the *Model* pane. This was not done here for brevity.

8. Rerun the model to this point by pressing solve ().

Discussion

The *Auto Assign Face Groups* in the *Model* pane is a tool that identifies groups of contiguous zone faces based on a break angle (45° by default) and creates automatically named zone face groups for them. [1] In this case, using the tool results in six face groups being created, one for each “side” of the problem’s cubic geometry.

Note something specific happens here: the model state is changed to include the newly created face groups. However, the command that caused this change is *not* a part of the “tutorial” data file. In order to ensure our data file is complete, we need to capture the command that created the groups. We use the *State Record* pane to do this. As its name implies, the *State Record* pane records *any* command (from a data file, from the command prompt, from the user interface) that changes the model state, as it is issued, and stores it in a record. The record therefore contains all commands necessary to create the current model state. As the object in this tutorial is to arrive at a single complete data file, we use the *State Record* pane to copy the command that was used to create the face groups with the *Auto Assign Face Groups* tool and paste it into our data file.

The zones in each of the five excavation stages are grouped as well using the `zone group` command. Here one of the principal advantages of putting objects into named groups is clear: the names assigned to groups can have a logical

relationship to one another and to the rest of the model. When the commands that will be used to perform the excavation sequence appear later, the names of the groups will help clarify what is going on in the commands.

Once the logical parts of the model have been identified and put into groups, it is a good idea to save the model state (review `solutionasflac3dproject`). The final line that is added at step (7) does this.

When to Group

Designating groups is not *strictly* necessary at this stage, but it is highly advisable, as construction of commands that use groups as `range` targets is usually faster, easier, and perhaps more innately sensible than constructing a range with the other range elements. Groups are also quite useful later on for plotting. Lastly, using groups helps conceptualize the model in terms of its meaningful parts. That said, the *Model* pane can be used for model object selection and grouping at any time in a modeling project.

Further Discussion: Model Groups

In a sense, all objects in a model are not equal. Some represent important model constructs (e.g., boundaries, interfaces), some represent natural features (joints, seams, or beds), and some represent human constructions (tunnels, structural support). And others don't. Groups are a way of identifying the model components that are a part of the former group, as they will usually become the focal point of the majority of commands that are used to construct the model.

Ranges and Groups, Briefly

As this tutorial provides a first look at the *FLAC3D* `range` logic and groups, a word to distinguish them may be of use. A range uses one or more criteria to identify the objects on which a command should operate—if a range is not supplied on a command, *all* possible target objects of the command will be affected (see [Command Scope & Overwriting](#)). A group is what its name implies: a collection of specific objects going by a given group name.

The initial definition of a group is made by specifying a range that determines the member objects.

```
zone group "lower" range position-z 1 100
```

This creates a group of zones named “lower”. It is possible that after cycling, the zones that fall in that *range z 1 100* will be different. The zones in the *group* “lower”, however, will be the same.

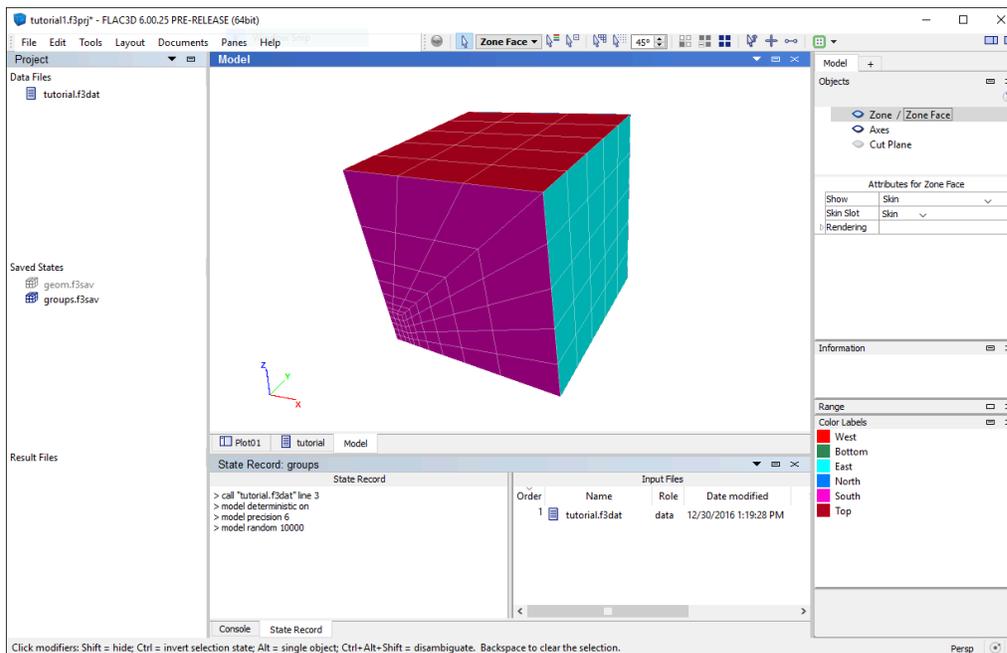
Once created, a group can be the target of a command by specifying it as the defining criterion of a range. Consider:

```
zone cmodel assign elastic range group "lower"
```

This assigns the Elastic model to the zones in the group “lower”.

Grouping: In the Interface

Click the *Model* pane to make it active (if it isn’t already). The model should appear similar to the image below.



Zones that are created by any method—with the `zone create` command, or through the *Extrusion* or *Building Block* panes—are rendered in the *Model* pane.

This area is capable of allowing the user to interact with model elements—zones or zone faces at this stage of the process—via mouse-driven selection tools. Any area of the model that may be of specific interest can be selected and defined to be a group.

- The “Saved States” section of the *Project* pane now lists two `.f3sav` files. The active model state is “groups.f3sav” (the inactive model state listed here, “geom.f3sav”, is grayed out).
- The “Objects” control set in the *Control Panel* indicates what is rendered in the view. At any time, the zones in a model may be rendered as either zones or zone faces, but not both. For this reason, the item is shown as “Zone / Zone Faces”, with a box around the currently rendered item. Clicking the word “Zone” or “Zone Faces” will set the clicked item to be the one rendered in the pane. As an immediate visual feedback, when zones are being rendered, the faces are outlined in black. When Zone Faces are being rendered, the faces are outlined in white.
- Double-click the item “geom.f3sav” in the “Saved States” section of the *Project* pane and observe the result. Double-clicking restores the “geom” model state, and it activates the *Project* pane. The model disappears from view! Now reactivate the *Model* pane by clicking on it. Observe that currently “Zone Faces” are to be rendered, but looking at the “Attributes for Zone Face” area, it can be seen that the “Show” attribute for this item is set to “Skin”. There was nothing in the “Skin” slot when `geom.f3sav` was saved; therefore, *there is nothing to show*. Change the “Show” item to “Surface of Visible Zones”; this will make the model visible again.
- As “geom.f3sav” is the active state, all zone faces are rendered in a single color, because there are no face groups in this model state. Double-click “groups.f3sav” to restore that model state, then click in the *Model* pane to reactivate it. Now the model is displayed with its excavation groups visible. Check the “Color Labels” control set to identify the color assignments in the current rendering.
- To experiment, open the “Rendering” item in “Attributes for Zone Face” and observe the effect of changing the settings it contains.

Rendering in either the *Model* pane or in a *View* pane (the pane type for plots) follows the same basic principle, which is also explained in [Plotting: In the Interface](#) following the [9. Examine Model: Plotting](#) topic.

- The items to be rendered in the view are listed in the upper section of the “Objects” (*Model* pane) or “Plot Item List” (*View* pane) control set in the *Control Panel*.
- With an object selected in the upper section, the lower section provides an “Attributes” set that supplies controls for setting how the selected object should be rendered.

Endnote

- [1] The *Ignore Existing Names* checkbox is used to ensure that the groups created are named *without* reference to existing groups. If not checked, then new group names will be formed as a concatenation of any pre-existing group names and the ones automatically generated during the processing of the `zone face skin` command.

4, 5, 6. Specifying Models, Boundaries, and Initial Conditions

Next a series of commands is added to complete the initial model definition.

1. Type or copy-paste the lines below into the “tutorial” data file.

```

; assign constitutive model, initial, and boundary conditions
zone cmodel assign mohr-coulomb
zone property density 1000 ...
      bulk 1e8 ...
      shear 3e8 ...
      friction 35 ...
      cohesion 1e3 ...
      tension 1e3
zone face apply velocity-normal 0 range group "West"
zone face apply velocity-normal 0 range group "Bottom"
zone face apply velocity-normal 0 range group "North"
model gravity (0,10,0)
zone initialize-stresses ratio 0.5
zone face apply stress-normal 0 gradient (0,[-10*1000*0.5],0) ...
      range group "East" or "Top"
; use of inline FISH in preceding line illustrates
; the derivation of the value needed (-5000)

```

2. Rerun the model to this point by pressing solve ().

Discussion

Assigning a Constitutive Model

Once the grid generation and model partitioning (grouping) is complete, one or more material models and associated properties must be assigned to all zones in the model. These are done using the commands `zone cmodel assign` and `zone property`. Here, the `zone cmodel assign` command that assigns a `mohr-coulomb` model is supplied without a `range`. Consequently, it will be applied to *all* zones in the model.

Properties are set with `zone property` commands in a way that spans model assignments. Setting the bulk property *without* a range will set it for any zone whose assigned constitutive model uses a `bulk` property (and this is *most* of them). Limiting the specification of a property to zones with a particular constitutive model assignment is done by using a range that invokes the constitutive model assignment as a criterion in a range. For example `zone property bulk 1e4 range model mohr-coulomb` (contrast this to what is given above) would set the `bulk` property *only* for zones that have been assigned the `mohr-coulomb` model.

Lastly, note the introduction of inline *FISH*. The second term of the `gradient` vector is provided as a mathematical expression enclosed in brackets ([]). The brackets invoke the *FISH* language, which takes care of performing the calculation and returning the result.

See [Further Discussion: Assigning a Constitutive Model](#) for additional discussion of constitutive model assignment.

Applying Boundary Conditions

The three `zone face apply` commands are used to set roller boundary conditions on the symmetry planes “West”, “North”, and “Bottom” sides of the model (the named groups from the [preceding step](#) come in handy here).

See [Further Discussion: Applying Boundary Conditions](#) for additional discussion of applying boundary conditions.

Discussion: Specify Initial Conditions

In this problem, the initial stress state is subjected to gravitational loading. This is added with the `model gravity` command.

By default, boundaries in *FLAC3D* are free of stress and are unconstrained. For this problem, all stresses (σ_{xx} , σ_{yy} , and σ_{zz}) are initialized with the single command `zone initialize-stresses`; it is possible to supply a separate stress for each component as well (cf. `zone initialize stress xx`).

Finally, we choose to apply a stress boundary to the far field using the `zone face apply` command; the boundary is set to match the changes due to gravity through use of the `gradient` keyword.

See [Further Discussion: Specifying Initial Conditions](#) for additional discussion of specifying initial conditions.

Further Discussion: Assigning a Constitutive Model

FLAC3D has nineteen built-in mechanical material models. Three models are sufficient for most analyses the new user will make: `null`, `elastic`, and `mohr-coulomb`.

The `null` model represents material that is removed or excavated from a model. The `elastic` model assigns isotropic elastic material behavior. The `mohr-coulomb` model assigns Mohr-Coulomb plasticity behavior.

Models `elastic` and `mohr-coulomb` require that material properties be assigned via the `zone property` command. For the `elastic` model, the required properties are bulk and shear modulus, or Young's modulus and Poisson's ratio.

NOTE: Bulk modulus, K , and shear modulus, G , are related to Young's modulus, E , and Poisson's ratio, ν , by the following equations:

$$K = \frac{E}{3(1 - 2\nu)}$$

$$G = \frac{E}{2(1 + \nu)}$$

or

$$E = \frac{9KG}{3K + G}$$

$$\nu = \frac{3K - 2G}{2(3K + G)}$$

The following are required properties for the Mohr-Coulomb plasticity model:

- bulk and shear moduli, or Young's modulus and Poisson's ratio;
- friction and dilation angles;
- cohesion; and
- tensile strength.

If any of these properties is not assigned, its value is set to zero by default.

Zones may be assigned different material models. For example, an elastic model may be prescribed for the upper half of a 10 m × 10 m × 10 m grid, and a Mohr-Coulomb model for the lower half of the grid. The example below^[1] shows how this is done using a `range` based on *z*-coordinates.

```

model new
zone create brick size 10,10,10
; elastic in upper half of grid
zone cmodel assign elastic range position-z=5,10
zone property bulk 1.5e8 ...
           shear 0.3e8 ...
           range position-z=5,10
; mohr-coulomb in lower half
zone cmodel assign mohr-coulomb range position-z=0,5
zone property bulk 1.5e8 ...
           shear 0.6e8 ...
           friction 30 ...
           cohesion 5e6 ...
           tension 8.66e6 ...
           range position-z=0,5

```

Further Discussion: Applying Boundary Conditions

Boundary conditions are normally specified with the `zone apply` or `zone face apply` commands; they are removed with corresponding `zone apply-remove` and `zone face apply-remove` commands.

The boundary conditions in a numerical model consist of the values of field variables that are prescribed at the boundary of the numerical grid. Note that by using a boundary condition command, while *FLAC3D* is calculating a solution, a condition or constraint is imposed that will not change unless specifically changed by the user. Boundaries can be either real or artificial: real boundaries exist in the physical object being modeled; artificial boundaries are introduced to reduce the size of the model.

Artificial boundaries fall into two categories: planes of symmetry and planes of truncation. A symmetry plane takes advantage of the fact that the geometry and loading in a system are symmetrical about one or more planes. A truncation plane is a boundary sufficiently far from the area of interest that the behavior in that area is not greatly affected. It is important to know how far away to place these boundaries and what errors they might precipitate in the stresses and displacements computed for the area of interest.

In this tutorial, we identify two planes of symmetry for the problem: one is the xy -plane; and the other is the yz -plane through the center of the trench. The commands that create this much of the model are as seen here.[2]

```

model new
; Create zones
zone create radial-brick ...
    point 0 (0,0,0) point 1 (10,0,0) ...
    point 2 (0,10,0) point 3 (0,0,10) ...
    size 3,5,5,7 ...
    ratio 1,1,1,1.5 ...
    dim 1 4 2 fill group 'exc'
zone face skin
; Constitutive model and properties
zone cmodel assign mohr-coulomb
zone property bulk 1e8 shear 3e8 friction 35 ...
    cohesion 1e3 tension 1e3 density 1000
; Boundary conditions
zone face apply velocity-normal 0 range group 'West'
zone face apply velocity-normal 0 range group 'Bottom'
zone face apply velocity-normal 0 range group 'North'
zone face apply stress-normal 0 gradient (0,-5000,0) range group 'East'
zone face apply stress-normal 0 gradient (0,-5000,0) range group 'Top'
model gravity (0,10,0)
zone initialize-stresses ratio 0.5

```

This yields a quarter-symmetry model as shown below (compare this to the full geometry as shown in the figure in the topic [Further Discussion: Meshing With Primitives](#)).

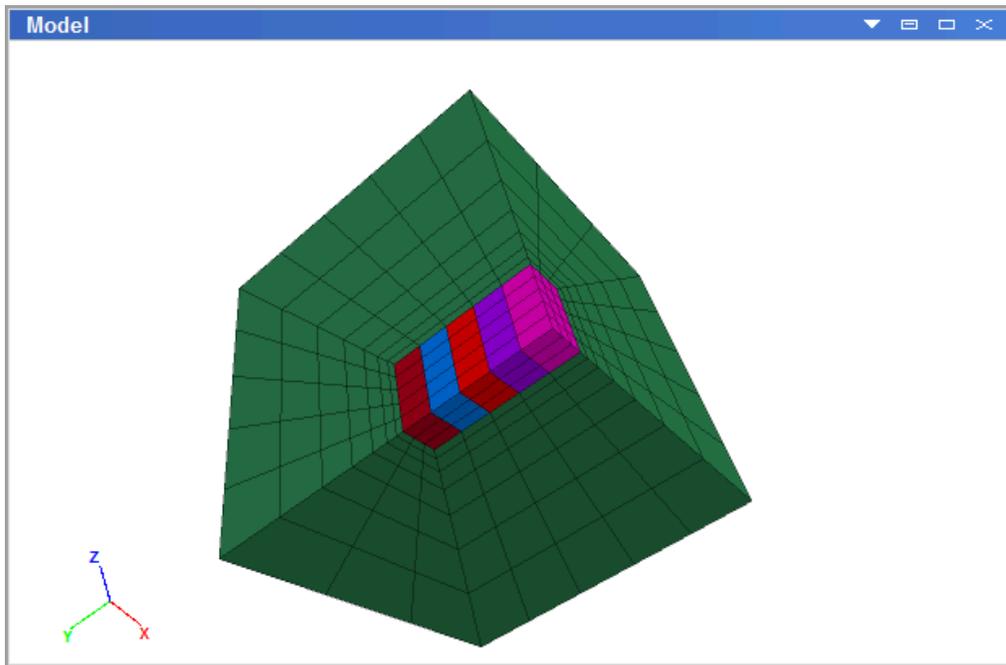


Figure 1: Quarter-symmetry model of trench excavation.

By default, the boundaries in *FLAC3D* are free of stress and are unconstrained. Forces or stresses may be applied to any boundary, or portion of a boundary, by means of the `zone apply` or `zone face apply` command. Individual components of the stress tensor may be specified. For example, a constant, compressive xx -stress component of 10 MPa can be applied to a boundary located at $x = 10$ with the command

```
zone face apply stress-xx -10e6 range position-x 10
```

Remember that compressive stresses are negative. The `range` keyword is used at the end of the command line to specify the range of action for the `zone face apply` command shown here. In this case, the boundary condition applies to any boundary faces that fall within the default tolerance ($1e-7$) of $x = 10$. A linearly varying yy -stress component can be applied with the `gradient` keyword. For example,

```
zone face apply stress-yy -20e6 gradient (0,0,20e5) ...
range position-y 0 position-z 0 10
```

With this command, a yy -stress component is applied to boundary faces located at $y = 0$. The yy -stress varies linearly with z from $\sigma_{yy} = 0$ at $z = 10$, to $\sigma_{yy} = -20 \times 10^6$ at $z = 0$.

When `gradient` is used, a value varies according to the relation

$$S = S^{(o)} + g_x x + g_y y + g_z z$$

in which $S^{(o)}$ is the value at the global coordinate origin at $(x = 0, y = 0, z = 0)$ and is the value immediately following the `stress-syy` keyword, and g_x , g_y , and g_z specify the variation of the value in the x -, y -, and z -directions and are the three numbers following the `gradient` keyword.

Displacements cannot be controlled directly in *FLAC3D*. In fact, they play no part in the calculation process. In order to apply a given displacement to a boundary, it is necessary to prescribe the boundary's velocity for a given number of steps. If the desired displacement is D , a velocity V over N steps (where $N = D/V$) may be applied.[3] In practice, V should be kept small and N large in order to minimize shocks to the system being modeled. The `zone face apply` command can be used to specify the velocities; gradients may also be specified.

In addition to `gradient`, the keywords `add`, `multiply`, and `vary` are available for use to modify the values of `zone apply` and `zone face apply` commands (and others).

The following command shows how to fix the x -displacement to zero along a boundary located at $x = 0$:

```
zone face apply velocity-x 0 range position-x 0
```

This, in effect, creates a roller boundary at which the boundary plane is fixed in the x -direction, but free to move in the y - and z -directions.

A pinned boundary condition (i.e., constrained in the x -, y -, and z -directions) can be specified with the command

```
zone face apply velocity (0,0,0) range position-x 0
```

For real boundaries, the choice of stress or displacement boundary conditions is usually clear. However, for artificial boundaries, such as planes of truncation, either type may be selected. Experience has shown that, generally:

- a fixed boundary causes both stresses and displacements to be underestimated;
- a stress boundary causes both stresses and displacements to be overestimated; and
- the two types of boundary conditions bracket the true solution, so that it is possible to do tests with both boundaries and get a reasonable estimate of the true solution by averaging the two results.

Further Discussion: Specifying Initial Conditions

Unlike the boundary condition commands, the initial condition commands assign initial values to selected variables; these can change while the computation proceeds. For example, in geological materials, there is an in-situ state of stress in the ground before any excavation or construction is started. By setting the initial conditions in a *FLAC3D* grid, an attempt is made to reproduce this in-situ state because it will influence the subsequent behavior of the model. (This influence is discussed in more detail in [Boundary Conditions](#) topic.)

To prescribe an initial stress state, we use the `zone initialize` command. For example,

```
zone initialize stress xx -50e6 yy -40e6 zz -10e6
```

This assigns initial compressive stress components of

$$\begin{aligned}\sigma_{xx} &= -50\text{e}6 \\ \sigma_{yy} &= -40\text{e}6 \\ \sigma_{zz} &= -10\text{e}6\end{aligned}$$

to all zones in the model. All six stress components can be initialized (either individually or all together in one command), and a stress gradient can be specified for boundary stresses in the same manner as previously discussed in [Further Discussion: Applying Boundary Conditions](#) (see also [Value Modifiers \(Add, Multiply, Gradient, Vary\)](#) for usage).

If the initial stress state is subjected to gravitational loading, this may be added via the `model gravity` command. For example,

```
model gravity (0,-9.81,0)
```

in which a gravitational acceleration vector of 9.81 m/sec^2 is applied in the negative y -direction. If gravitational loading is specified, the material mass density must also be initialized with the command `zone initialize density`.

In the example in this tutorial, σ_{xx} and σ_{zz} stresses and gradients are equal to half the σ_{yy} stresses and gradients. Stresses may be initialized to any value, but you must check the stress state to ensure that it does not violate the yield criterion (Mohr-Coulomb, in this case). The initial equilibrium state is discussed further in the next section.

Endnotes

[1] If you want to run the example shown in the data file:

a. *Load the file:* This file is “different-model.f3dat” in the folder

`[appdata location]\datafiles\UsersGuide\Tutorial\Illustrative`

Use the menu command **File** > **Open Into Project...** (or press **Ctrl + O**) to open it.

b. *Build the file:* Copy and paste the examples shown on this page into a new data file(s). Use **Add New Data File...** (or press **Ctrl + N**) to open a blank new data file.

[2] If you want to run the example shown in the data file:

a. *Load the file:* This file is “quarter-symmetry-trench.f3dat” in the folder

`[appdata location]\datafiles\UsersGuide\Tutorial\Illustrative`

Use the menu command **File** > **Open Into Project...** (or press **Ctrl + O**) to open it.

b. *Build the file:* Copy and paste the examples shown on this page into a new data file(s). Use **Add New Data File...** (or press **Ctrl + N**) to open a blank new data file.

[3] Note that velocity has units of *displacement per timestep* for the static solution mode, and *displacement per unit time* for the dynamic mode.

7. Step to Equilibrium

Next a series of commands is added to complete the initial model definition.

1. Type or copy-paste the lines below into the “tutorial” data file.

```
; step to equilibrium
model largestrain true
model history mechanical ratio
model solve
model save "tr_eq"
```

2. Rerun the model to this point by pressing solve ().
3. Make “Plot01” the active tab.
4. Press the *Build Plot* button () in the *Control Panel*, and double-click “History Chart”.
5. In the “Attributes” *control set*, press the plus button () adjacent to the label “mechanical ratio limit”.
6. In the attributes for the History item, open the “Y-Axis” group (press the arrow adjacent to the label) and check the “Log” box to set the *y*-axis of the plot to a logarithmic scale.

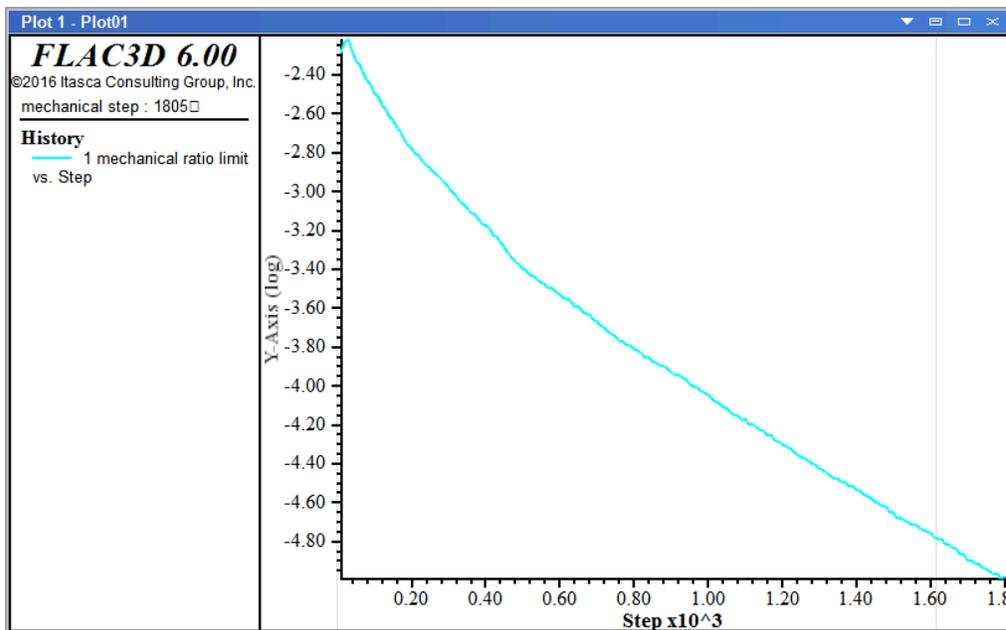
Discussion

The *FLAC3D* model must be at an initial force-equilibrium state before alterations can be performed. For simple model geometries, the boundary conditions and initial conditions may be assigned such that the model is exactly at equilibrium initially. However, in most cases, it is necessary to step the model to the initial equilibrium state under the given boundary and initial conditions, particularly for problems with complex geometries or multiple materials. This is done using either the `model step` (or its synonym `model cycle`) or `model solve` command.

In this example, before the `model solve` command is issued, the program is set to use *large-strain* calculation, which will move the position of the zones as displacements develop. Also, a history of average convergence ratio is set to be recorded when the model cycles. The history will be used as a means of visualizing and verifying that the model reaches equilibrium.

The model will cycle for 1805 steps when the `model solve` command is given. When complete, this is another juncture where saving the model state is advisable; the last command in the listing above does this.

To verify the model has reached equilibrium, we create a plot of the convergence ratio (steps (3)–(6) above do this). The resulting image should match the one below. The plot indicates that a ratio of $1e-5$ is reached after 1805 steps.



Further Discussion: Step to Equilibrium

A model is in perfect equilibrium when the net nodal-force vector at each gridpoint is zero (see [Theoretical Background](#) and following). The convergence criteria used is monitored in *FLAC3D* and printed to the screen when the `model solve` command is executing. In this way, the user can assess when equilibrium has been reached.

For a numerical analysis, the average force ratio will never reach exactly zero. It is sufficient, though, to say the model is in equilibrium when it reaches a small value. The default limit of $1e-5$ is considered adequate in the majority of cases.

This is an important aspect of numerical problem-solving with *FLAC3D*. *The user must decide when the model has reached equilibrium.*

There are several features built into *FLAC3D* to assist in making this decision. The history of the maximum unbalanced force may be recorded by adding a history with the command

```
model history mechanical ratio
```

Additionally, the history of selected variables (e.g., velocity or displacement at a gridpoint) may be recorded. The following commands are examples:

```
zone history velocity-x position (3,4,4)
zone history velocity-z position (0,0,8)
```

The first history command records x -velocity at location ($x = 3, y = 4, z = 4$), while the second records z -displacement at location ($x = 0, y = 0, z = 8$).

After running several hundred (or thousand) calculation steps, a history of these records may be plotted to indicate the equilibrium condition. The example below illustrates this process.[1]

```
model new
; Create zones
zone create brick size 6 8 8
zone face skin
; Assign constitutive model and properties
zone cmodel assign elastic
zone property bulk 1e8 shear 0.3e8
; Boundary Conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
zone face apply stress-normal -1e6 range group 'Top'
; Histories
model history mechanical ratio
zone history velocity-x position (3,4,4)
zone history displacement-z position (0,0,8)
; Step
model step 1500
;model solve
```

The initial average force ratio maximum unbalanced force is about 0.19. After 1500 steps, this force has dropped to approximately .00002. By plotting the first history, it can be seen that the average force ratio is approaching zero. This can be done by creating a plot window, adding a “History” plot, and selecting the *mechanical ratio limit* history in the *Attribute* panel.

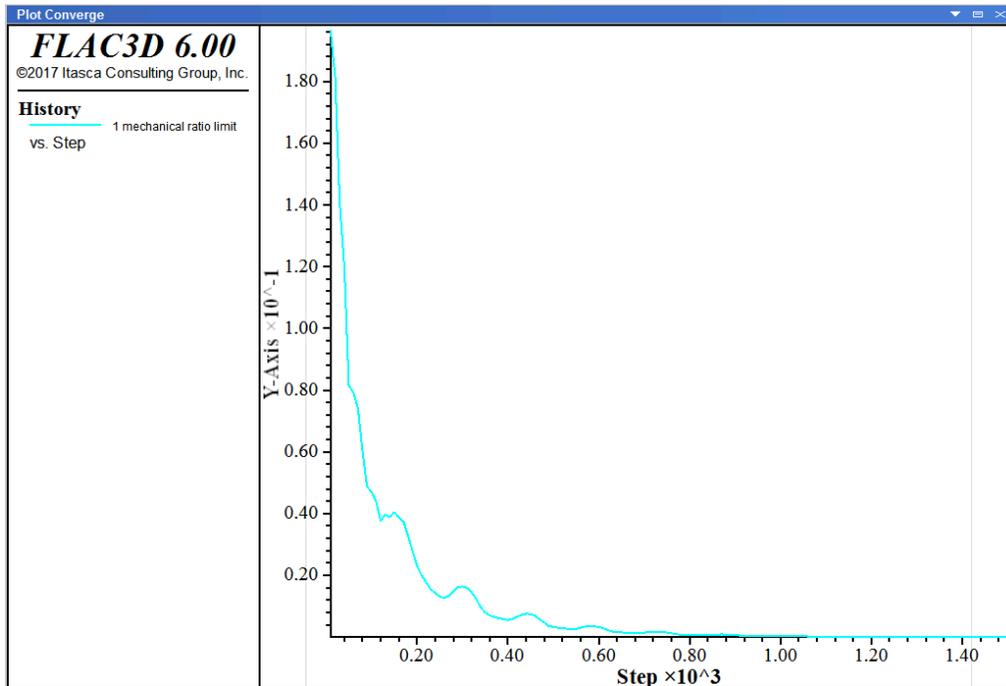


Figure 1: Maximum unbalanced force history.

The velocity also approaches zero, and the displacement history becomes constant; these are indicators that an equilibrium state is reached.

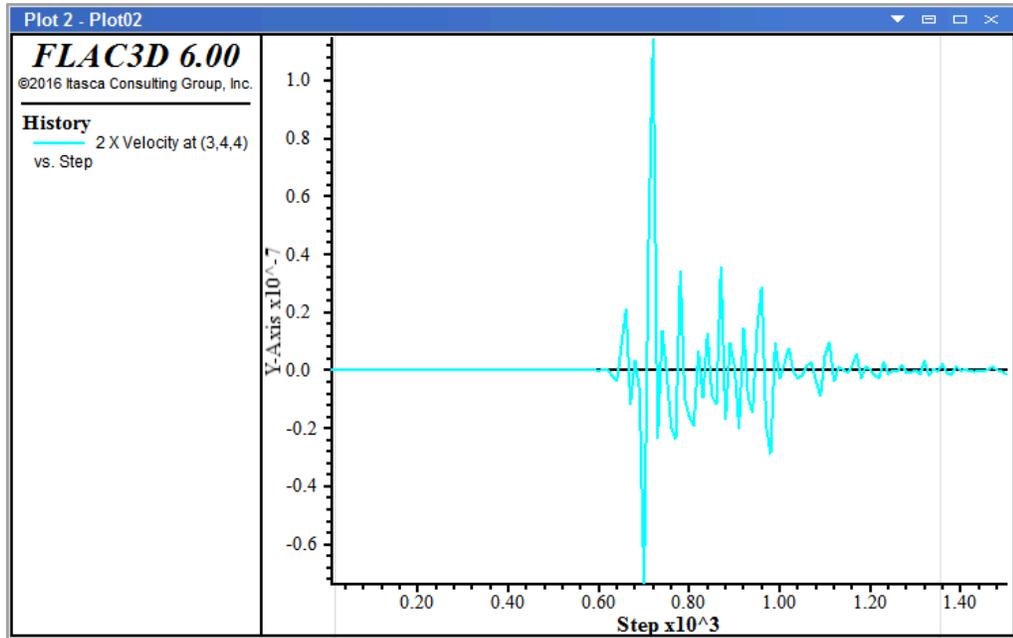


Figure 2: x-velocity history at (x = 3, y = 4, z = 4).

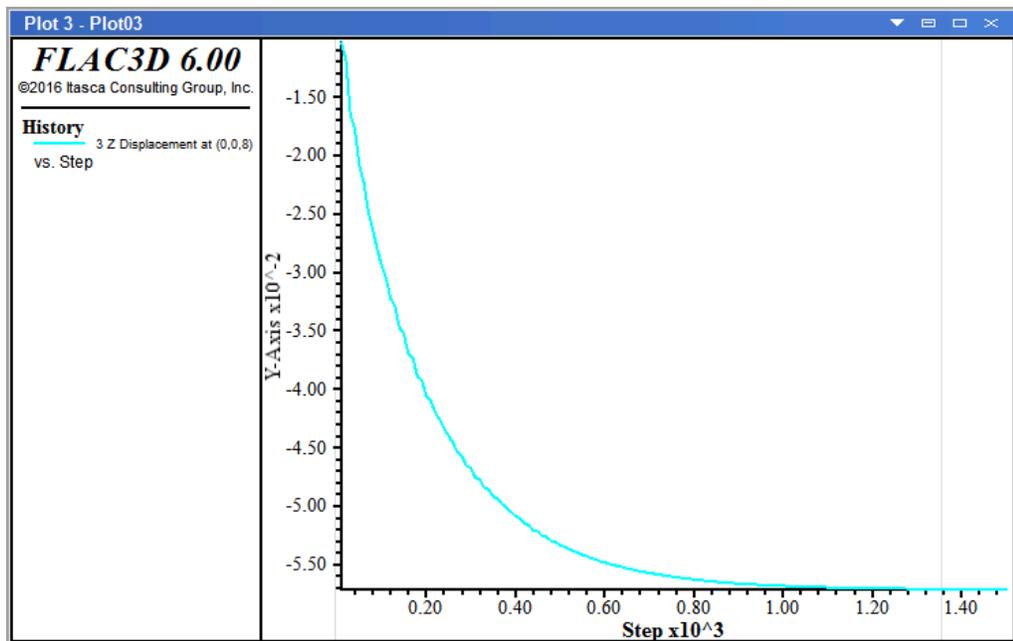


Figure 3: z-displacement history at (x = 0, y = 0, z = 8).

The `model solve` command can be used in place of `model step` if the user wishes *FLAC3D* to stop automatically when the maximum unbalanced force falls below a specified limit. Uncomment (delete “;”) the command `;model solve` and comment (add “;” in front of) `model step 1500` and re-run the preceding

problem. This time, *FLAC3D* should stop the calculation at step 1657. The plots (if created as previously shown) are automatically updated, though the results are essentially the same as shown above.

By default, an average force ratio limit of 1.0×10^{-5} is used if no other limits are specified in a `model solve`. The ratio can be changed by typing

```
model solve ratio 1e-6
```

before giving the `model solve` command. Here `1e-6` is a user-supplied value that sets the limit of the current convergence ratio.

In the example above, the initial equilibrium stage can be achieved without stepping by inserting a `zone initialize stress` command before the `model solve` command:

```
zone initialize stress xx -1e6 yy -1e6 zz -1e6
```

Now, the average force ratio is nearly zero. After inserting the line above, press the  button to re-run the problem. Note that, in this case, the initial displacements in the model are automatically zero. Looking back at the example shown in the topic [Further Discussion: Applying Boundary Conditions](#), the model described by the data file should be at a stress equilibrium state. If you run that data file and then type in

```
model solve
```

at the command prompt, you will find that a small number of cycles is still required to reach the limit of 1×10^{-5} . This is because of the graded mesh. If the zones were all of equal size, the unbalanced force would be nearly zero. However, with graded meshes, it is usually necessary to perform some stepping to bring the model to an initial equilibrium state.

In an analysis, it is very important that the model be at equilibrium before alterations are made. Several histories should be recorded throughout a model to ensure that a large force imbalance does not exist. It does not adversely affect the analysis if more steps than needed are taken to reach equilibrium. However, it will affect the analysis if too few steps are taken.

A *FLAC3D* calculation can be interrupted at any time during stepping by pressing `<Shift+Esc>`. It can be convenient to use the `model step` command with a high step-number to periodically interrupt the stepping, check the histories, and resume stepping until the equilibrium condition is reached.

Stepping: In the Interface

When cycling, there are some aspects of program operation that are worth noting.

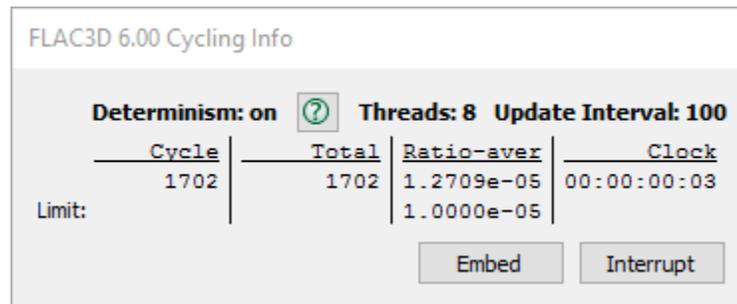


Figure 4: The “Cycling Info” dialog.

- The *Cycling Info* dialog displays a running total of the number of steps in the current step/cycle/solve command, the total number of cycles in the model, the average stress ratio, and the clock time for the current step/cycle/solve command.
- The current cycling may be interrupted by pressing the `Interrupt` button or by pressing the `Shift + Esc` key.
- With regard to the tutorial, note the most recent saved state, “`tr_eq.f3sav`”, is added to the “Saved States” list in the *Project* pane (and it is the active state).
- Here and anywhere else it is encountered in *FLAC3D*, a *Help* button () will open a help file topic that provides contextual information for the current item.

Endnote

[1] If you want to run the example shown in the data file:

- a. *Load the file:* This file is “stepping-to-equilibrium.f3dat” in the folder [appdata location]\datafiles\UsersGuide\Tutorial\Illustrative
Use the menu command File • Open Into Project... (or press `Ctrl + O`) to open it.
- b. *Build the file:* Copy and paste the examples shown on this page into a new data file(s). Use Add New Data File... (or press `Ctrl + N`) to open a blank new data file.

8. Alter Model: Excavation

Now the model excavations are performed.

1. Type or copy-paste the lines below into the “tutorial” data file.

```

; monitor trench excavation
zone gridpoint initialize displacement (0,0,0)
zone history displacement-x position (1,.01,.01)
zone history displacement-z position (.01,.01,2)

; excavation step 1
zone cmodel assign null range group "exc1"
model step 1000
model save "exc1"
; excavation step 2
zone cmodel assign null range group "exc2"
model step 1000
model save "exc2"
; excavation step 3
zone cmodel assign null range group "exc3"
model step 1000
model save "exc3"
; excavation step 4
zone cmodel assign null range group "exc4"
model step 1000
model save "exc4"
; excavation step 5
zone cmodel assign null range group "exc5"
model step 1000
model save "exc5"

```

2. Rerun the model to this point by pressing solve ().
3. Make the “Plot01” tab active.

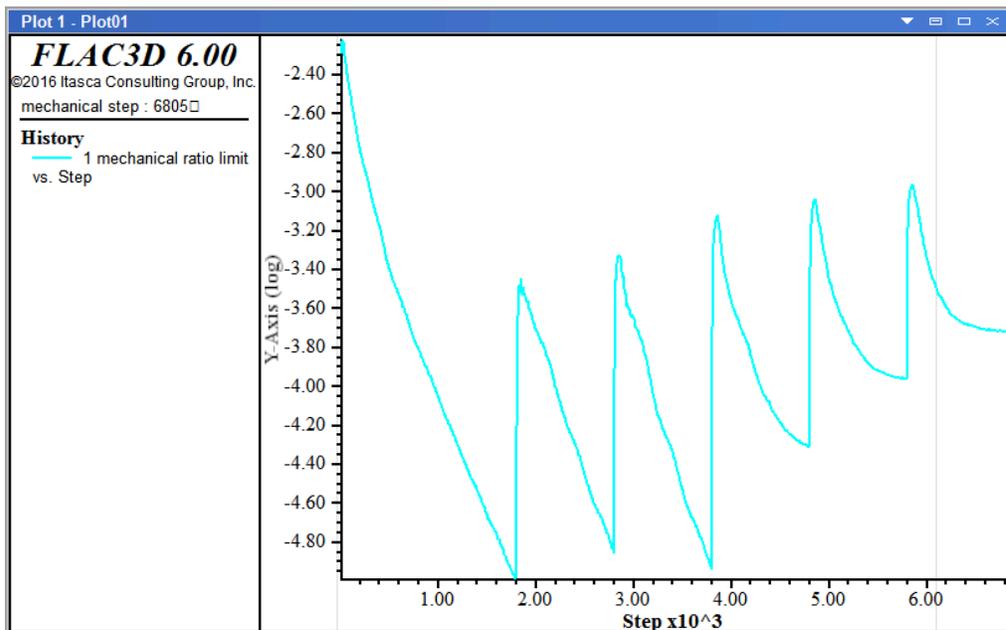
Discussion

Before performing the excavations, a couple of additional histories are set up—one for x -displacement at the gridpoint at (1,0,0), and one for z -displacement at the gridpoint at (0,0,2).

Excavation is a matter of first assigning the zones of the current excavation stage to the `null` model. The model is then cycled for 1000 steps, and then the model state is saved before moving on to the next excavation stage. Note the zone groups for the excavation stages that were created in 3. **Create Model Groups** are used for the `zone cmodel assign` commands that perform the excavation.

Looking at “Plot01”, we see that the five excavation stages have been added to the history seen in the previous step (we see one + five minimum points in the chart). Significantly, we see that the model fails at the third stage—it *does not* reach/approach a convergence ratio of $1e-5$ before the next excavation step begins.

At this point, it should be observed that the use of the `model step` command in this example is a bit unrealistic. It is used here to help illustrate the modeling process efficiently. However, in a more rigorous model, the command `model solve` would be used, and the failure in the third step would be evident—the model would simply continue to cycle. You can test this by substituting `model solve` for `model step 1000` at the third excavation stage. The model will eventually stop cycling, but with an “Illegal geometry” error; it never reaches equilibrium. This is a common terminating point on models exhibiting continuous deformation. It indicates a zone in the model has deformed to a state where calculations can no longer be performed on it.



The next modeling phase of this tutorial will involve installing cables as support after the second stage and observing the effect. However, before doing so, we will explore plotting to get a further look at the failure that occurs in the third stage.

Further Discussion: Performing Alterations

FLAC3D allows model conditions to be changed at any point in the solution process. These changes may take the following forms:

- excavation of material;
- addition or deletion of gridpoint loads or stresses;
- change of material model or properties for any zone; and
- fix or free velocities for any gridpoint.

In this tutorial, excavation is performed using the `zone cmodel assign null` command. Gridpoint loads can be applied at any gridpoint with the `zone apply (force, force-x, etc.)` command. Stress alterations can be made at model boundaries with the `zone face apply` command, as shown previously. Material models and properties* are changed with the `zone cmodel assign` and `zone property` commands. Gridpoint velocities are fixed or freed using the `zone face apply` command to set `velocity` to the desired value(s). It should be evident that several commands can be repeated to perform various model alterations.

If model zones contain a plasticity model (e.g., `zone cmodel assign mohr-coulomb`), it is possible that an alteration may be such that force equilibrium cannot be achieved. In other words, the unbalanced forces in part or all of the model cannot approach zero, in which case, the average force ratio will approach a constant nonzero value, indicating that steady-state flow of material is occurring (i.e., a portion/all of the model is failing).

See `t109_installingsupport` after the topic [10. Further Alterations: Support](#) in this tutorial for a simple example model that exhibits clearly identifiable failure.

9. Examine Model: Plotting

The following steps illustrate the basic mechanics of plotting.

1. Select File ▸ Add New Plot... and name it “xdisp” in the ensuing dialog. Note the newly created empty plot is now the active pane.
2. Press the *Build Plot* button (), select *History Chart* and press in the ensuing dialog.
3. In the “Attributes” section on the *Control Panel*:
 - add History 2 “X Displacement at ...” by pressing the add button adjacent to its label ();
 - open the “X-axis” item, uncheck “Auto” on “Minimum” and “Maximum”, and type “0” in for the minimum and “6000” in for the maximum; and
 - open the “Y-axis” item, uncheck “Auto” on “Minimum”, and type in the “-0.008” for the minimum value.
4. Repeat steps 1-2 to create a History plot of *z*-displacement named “zdisp”.
5. In the “zdisp” plot, go to the “Attributes” section:
 - add History 3 “Z Displacement at ...” by pressing the add button adjacent to its label ();
 - open the “X-axis” item, uncheck “Auto” on “Minimum” and “Maximum”, and type “0” in for the minimum and “6000” in for the maximum; and
 - open the “Y-axis” item, uncheck “Auto” on “Minimum”, and type in the “-0.0038” for the minimum value.
6. Press `Ctrl + Shift + N`, and name the new plot “xdisp2”.

7. Press the *Build Plot* button (), select *Zone* and press in the ensuing dialog.
8. In the “Attributes” control set:
 - select “Contour” on the “Color By” switch;
 - select “Displacement” on the “Value” switch;
 - select “Z” on the “Component” switch;
 - open the “Contour” item, and check the “Reversed” box; and
 - uncheck “Auto” on “Minimum,” and set the value of the minimum to “-0.0003”.
9. Now, with any of these three plots active, step through the saved model states in the *Project* pane by double-clicking each saved state. Start with the “tr_eq” state (initial equilibrium), then progress through the excavation stages (“exc1” through “exc5”).

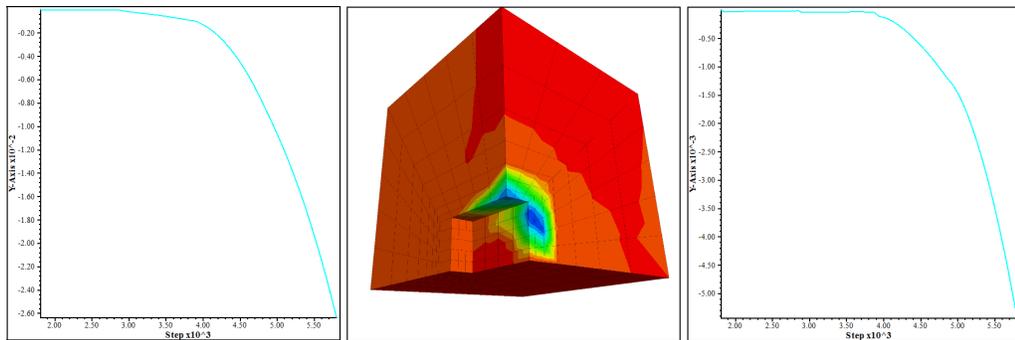
Discussion

To observe the failure as the excavation sequence progresses, we plot the history items that were defined prior to running the model (immediately after initial equilibrium had been established). History plots will, by default, automatically compute the maximum and minimum values for the scales on the x - and y -axis of the chart. This is efficient and time-saving when looking at that history at a single model state. However, when looking at the same thing across a series of model states, it makes comparison of one view to the next difficult, because the scale on either axis is likely to change from one state to the next. For this reason, when setting up the plots here to help visualize the failure that occurs starting in the third excavation stage, we “fix” the scales on the plots by turning the “Auto” attribute off. As a result, when we step through the saved states (step (9) above), the fixed scales allow us to clearly see the failure develop without any “jumping around” in the chart.

Plots As Project Entities

Unlike command input or model states, plots are not stored separately as files. Once created, they will be saved with the project. Closing the pane containing a plot (\times) eliminates it completely; it would need to be rebuilt to be returned. Plots can be hidden (press (\square) on its pane) as needed; a hidden plot will still be listed on—and is re-accessed from—the “Documents” menu.

Four reference plots are named. When creating a new plot, *FLAC3D* will automatically suggest a numbered name (e.g., “Plot04”), but as can be seen here, providing a more descriptive name is helpful, especially when many plots have been created.



Plotting: In the Interface

The basic mechanics of model visualization in the *Model* pane or in a plot in a *View* pane are as follows.

1. Put objects into the view. In the *Model* pane, this occurs automatically as model objects (zones, structural elements) are created. In a *View* pane, this is done by picking items from the *Build Plot* dialog using the *Build Plot* tool (\oplus).
2. The objects that are currently rendered in the view are listed above in the *Control Panel*.
3. To control *how* an object is rendered, select it on the list above. This will activate the controls for visualization that appear below—for that object—in the “Attributes” section.

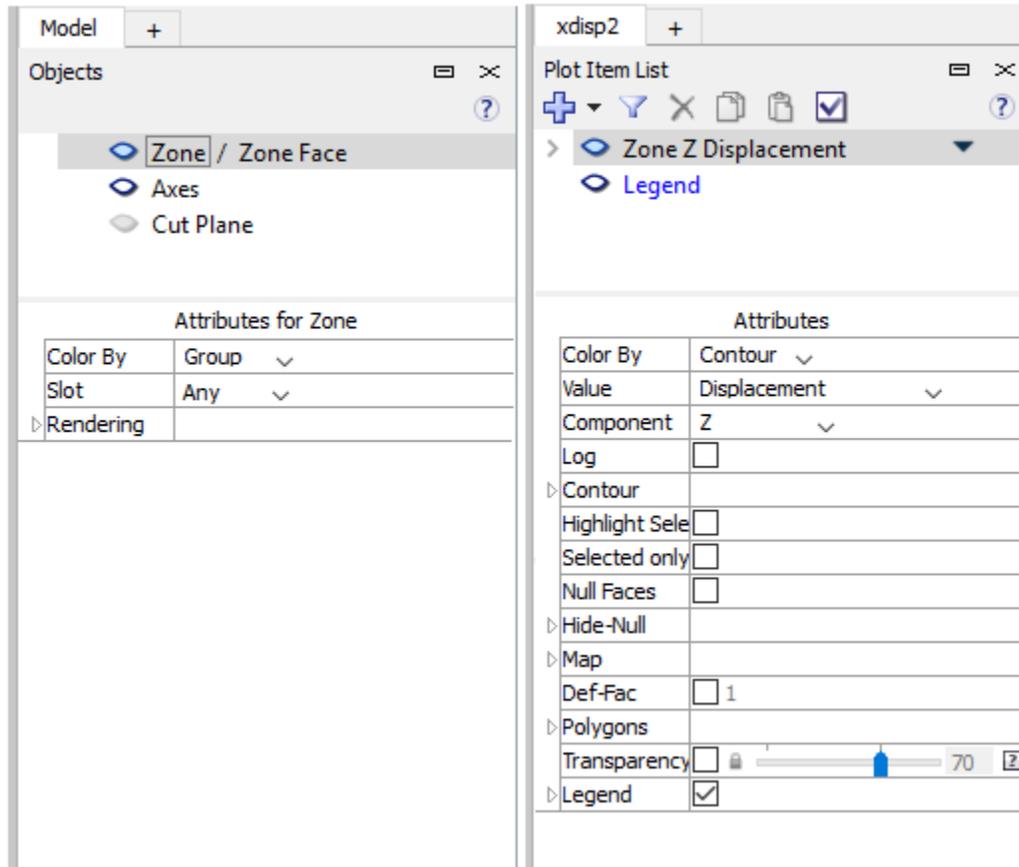


Figure 1: Two views of the Control Panel: the “Objects” control set in the Model pane (left), and the “Plot Item List” in a View pane.

- In the *Model* pane, Zones and Zone Faces are listed on the same line of the “Objects” control set. Only one may be rendered at a time. A box indicates which is currently being rendered. Clicking the other item’s label will cause it to be rendered instead.
- Selecting a plot item in a *View* pane causes a menu selector icon (▾) to appear to the right of the item name. The menu provides options to set attributes, add a plot item, or to cut, copy, delete, or paste the current item.

10. Further Alterations: Support

Support is installed after the second excavation stage to stabilize the trench and prevent the failure seen in the third excavation stage.

1. Type or copy-paste the lines below into the “tutorial” data file.

```

; install cable support
model restore "exc2"
structure cable create ...
    by-ray (1.0,0.4,1.5) (1,0,0) 4 segments 4
structure cable create ...
    by-ray (1.0,0.4,0.5) (1,0,0) 4 segments 4
structure cable create ...
    by-ray (1.0,1.2,1.5) (1,0,0) 4 segments 4
structure cable create ...
    by-ray (1.0,1.2,0.5) (1,0,0) 4 segments 4
structure cable property ...
    young 2e9 ...
    yield-tension 1e8 ...
    cross-sectional-area 1.0 ...
    grout-cohesion 1e10 ...
    grout-stiffness 2e9 ...
    grout-perimeter 1.0
zone cmodel assign null range group "exc3"
model step 1000
model save "cab3"
structure cable create ...
    by-ray (1.0,2.0,1.5) (1,0,0) 4 segments 4
structure cable create ...
    by-ray (1.0,2.0,0.5) (1,0,0) 4 segments 4
structure cable property ...
    young 2e9 ...
    yield-tension 1e8 ...
    cross-sectional-area 1.0 ...
    grout-cohesion 1e10 ...
    grout-stiffness 2e9 ...
    grout-perimeter 1.0
zone cmodel assign null range group "exc4"
model step 1000
model save "cab4"
structure cable create ...
    by-ray (1.0,2.8,1.5) (1,0,0) 4 segments 4
structure cable create ...
    by-ray (1.0,2.8,0.5) (1,0,0) 4 segments 4
structure cable property ...
    young 2e9 ...

```

```

        yield-tension 1e8 ...
        cross-sectional-area 1.0 ...
        grout-cohesion 1e10 ...
        grout-stiffness 2e9 ...
        grout-perimeter 1.0
zone cmodel assign null range group "exc5"
model step 1000
model save "cab5"

```

2. Rerun the model to this point by pressing solve ().

Discussion

We return to the model state after two excavation stages, while it is still stable, by using the `model restore` command to put the model at that point. Now we add support *in addition* to nulling the zones for each excavation stage. This is done by spatially defining the cable, and then defining its properties. Then (as before) the excavation stage zones are nulled, the model is cycled 1000 steps, and the model state is saved. This sequence is repeated to create a new third, fourth, and fifth excavation stage.

Further Discussion: Using Save/Restore and Staged Modeling

At this point, the function of the commands `model save` and `model restore` in a staged analysis should be apparent. At the end of a modeling stage (e.g., initial equilibrium), the model state can and should be saved using `model save`. This file can be restored at a later time using `model restore`, which will put the model back to the point at which the model was saved. Consequently, it is not necessary to build a model from scratch every time a change is made; simply save the model before the change and restore it to that point whenever a new change is to be analyzed.

For example, in this tutorial trench problem, the trench is excavated in 2 m deep stages. The model state is saved at the end of each stage, beginning with the initial equilibrium stage (“tr eq”). Looking at each stage in sequence, we can then determine the stage at which the trench fails, and investigate the influence of support (e.g., tiebacks or soil nails) on stabilizing the trench. For comparison, below we look at a model that excavates the trench all at once.

When we determine, upon running the model, that the trench fails at the third excavation stage, we simply restart the save file at the second excavation stage (`exc2.f3sav`), install cables, and then proceed with the excavation. This time, the trench walls are stable.

By using save files this way, we can perform a series of different calculations, starting from whatever stage we desire, and investigate various problem conditions, such as spacing and properties of cable elements.

Simpler Example

To illustrate the value of breaking the modeling process into stages, we can take the tutorial problem and simplify it. If we were to set up the model such that the entire trench is excavated at once, we might have a file that appears as below.

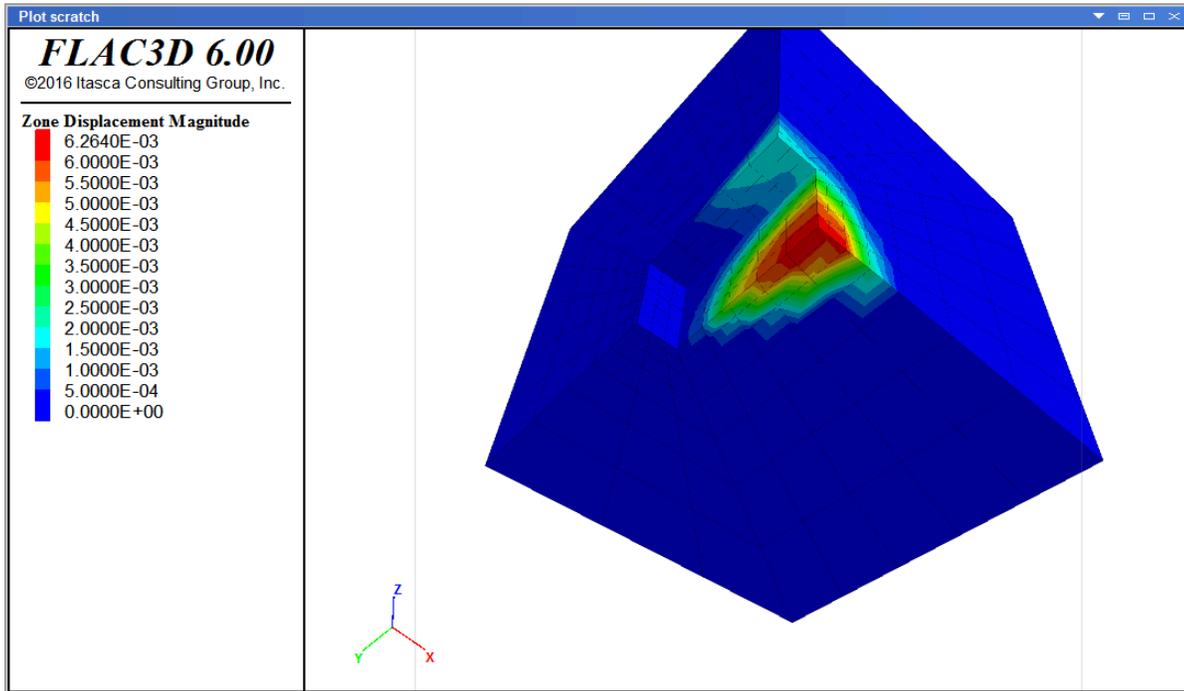
```

model new
; Create zones
zone create radial-brick ...
    point 0 (0,0,0) point 1 (10,0,0) ...
    point 2 (0,10,0) point 3 (0,0,10) ...
    size 3,5,5,7 ...
    ratio 1,1,1,1.5 ...
    dim 1 4 2 fill group 'exc'

zone face skin
; Assign constitutive model and properties
zone cmodel assign mohr-coulomb
zone property bulk 1e8 shear 3e8 friction 35 ...
    cohesion 1e3 tension 1e3 density 1000
; Boundary condition
zone face apply velocity-normal 0 range group 'West'
zone face apply velocity-normal 0 range group 'North'
zone face apply velocity-normal 0 range group 'Bottom'
zone face apply stress-normal 0 gradient (0,-5000,0) ...
    range group 'East' or 'Top'
; Initial conditions
model gravity (0,10,0)
zone initialize-stresses ratio 0.5
; Histories
zone history displacement-x position 1,0,0
; Excavation and setup
model largestrain on
zone relax excavate range group 'exc'
; ... and then calculate the response:
model step 1000

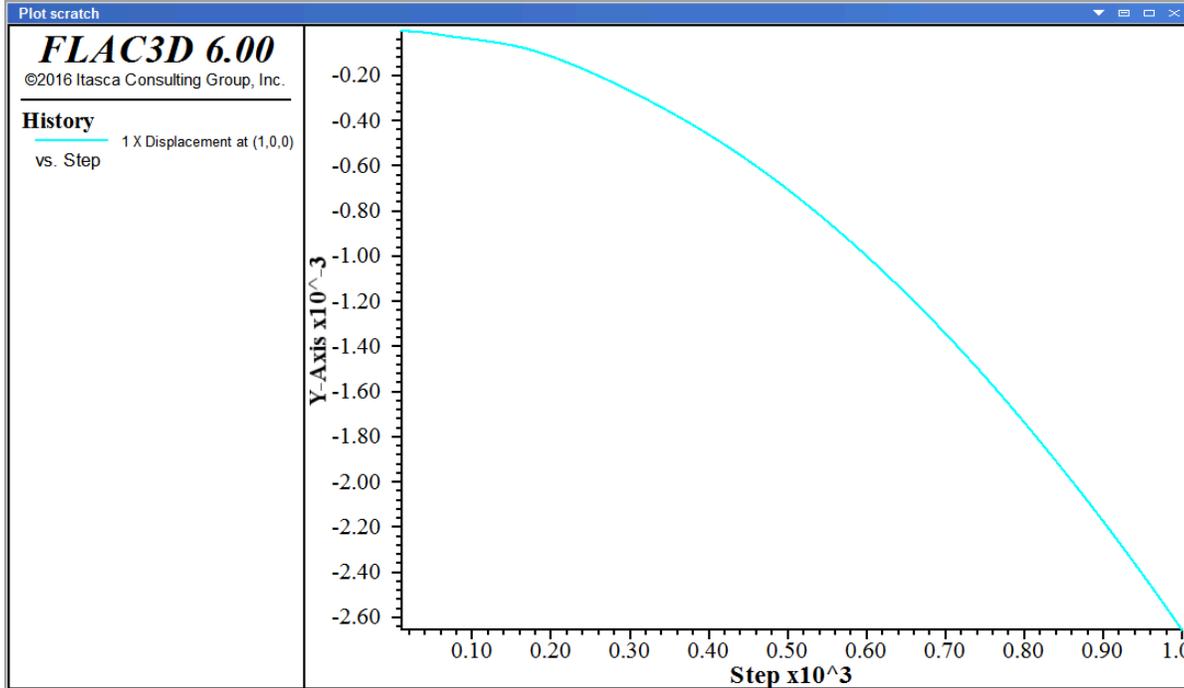
```

The displacement contours indicate the trench fails.



The failure can also be indicated by plotting the displacement history at the trench wall.

The continuously increasing displacement indicates failure.



You can see how this might be an early “draft” of the tutorial problem. Given this as a starting point, and given that failure is observed, a natural next step would be to look at model behavior when excavation is split up into a sequence, as is seen in the tutorial model. More precisely locating the point of failure in the model, in turn, provides an indicator of where it will be necessary to introduce support for the excavation.

11. Project Finishing

The project is complete at this point. For the purposes of good management, a few additional steps are advisable.

1. Use File › Save Project to save the current project.
2. Bundle the project by selecting the menu command Tools › Bundle › Pack...
3. In the ensuing dialog, review the contents to be packed, press , and provide a name for the bundle file.

Discussion

With the model runs completed, the project is saved and, for good measure, a bundle file is created for the project.

At this point, the modeling process, as shown in the [General Solution Procedure Illustrated](#) topic, and how to accomplish that process in the *FLAC3D* program, as shown in [The Solution Procedure as a FLAC3D Project](#) topic, should be clearer. This tutorial illustrates that model setup, and initial definitions tend to follow a singular path; however, once past the equilibrium stage, the potential for branching and bifurcation due to exploration, revision, and alteration is nearly limitless. In that context, a well-designed project with distinct stages is critically important.

About Bundles

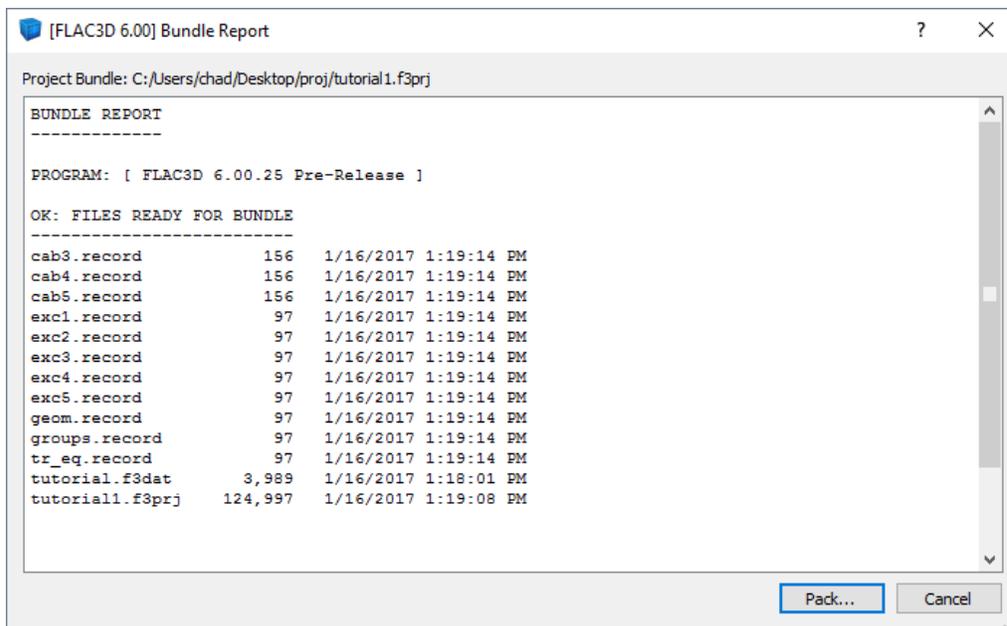
A bundle is a compressed archive file that gathers up the following:

- any files input in the model (the files listed in the “Data Files” section of the *Project* pane);
- state record files (a record file corresponding to each save file listed in the “Saved States” section of the *Project* pane); and
- the project file

With a bundle file, all materials necessary to re-create the project are stored in a single location. Because the bundle file does not store save files, which can be quite sizable when a model is large, it is a useful, efficient way to archive a project. The trade-off for that efficiency, however, is that using a bundle to re-create a project will require re-running the component data files or replaying the state record files in order to regenerate the project saved state(s).

Bundles are also an excellent way of attaching the project to a technical support request. This can be done automatically when requesting technical support via the technical support dialog (Help ▸ Support...).

Finishing: In the Interface



- The bundle file report “previews” the content that will be included in the bundle.
- There should be a `.record` file corresponding to each `.f3sav` file in the project.
- The project file and all input files (data files, etc.) are included in the bundle.

- The file report categorizes the bundle contents in different sections: “OK”, “Warning: Files Modified But Not Saved”, “Error: Files Cannot Be Located (Link Broken)”, etc. For this tutorial, which is comparatively simple, all content is “OK”. Refer to the [Bundle Report](#) topic for reference.

Tutorial: Working with *FISH*

This section is intended for people who have run *FLAC3D* (at least for simple problems) but have not used the *FISH* language—no programming experience is assumed. To get the maximum benefit from the examples given here, you should try them out with *FLAC3D* running interactively. The short programs may be typed directly into the command line, accessed via the **console** pane. After running an example, give the *FLAC3D* command `model new`, to “wipe the slate clean,” ready for the next example. Alternatively, the more lengthy *FISH* functions may be typed into data files using the built-in **editor**; these can be executed directly in the editor or `program call` in the console.

Type the lines in the example below at the *FLAC3D*'s `command prompt`, pressing `<Enter>` at the end of each line.

Defining a *FISH* Function

```
fish define fname
  abc = 22 * 3 + 5
end
```

Note that the command prompt changes to `Def>` after the first line has been typed in; it changes back to the usual prompt when the `end` statement is entered. This change in prompt lets you know whether you are using commands or if you are defining a *FISH* function. All lines following the `fish define` statement are taken as part of the definition of a *FISH* function (until the `end` statement is entered). However, if you type in a line that contains an error (e.g., you type the `=` sign on an empty line), then you will get the `Flac3d>` prompt again with an error message. In this case, one must redefine the *FISH* function again from scratch, starting with the `fish define` statement. Because it is very easy to make mistakes, *FISH* functions are often typed into data files using the built-in editor. These can either be executed directly from the editor or can be executed via the `program call` command just like a *FLAC3D* data file not containing *FISH*. We will describe this process later; for now, we'll continue to work from the command

line. Assuming that you typed in the preceding lines without error, and that you now see the `Flac3d>` prompt, you can “execute” the function named `fname` defined in the example above by typing the line

```
@fname
```

This line causes the arithmetic operations defined in the function to be executed. This is an “assignment statement.” If an equal sign is present, the expression on the right-hand side of the equal sign is evaluated and given to the variable on the left-hand side. Note that arithmetic operations follow the usual conventions: addition, subtraction, multiplication and division are done with the signs `+`, `-`, `*`, and `/`, respectively. The sign `^` denotes “raised to the power of.”

The actions above create two objects: a function object that can be called at the user’s discretion, and a variable called `abc`, where `abc = 71` after the function has been executed. One can see both the `fname FISH` function and the variable `abc` in the `global symbols` control set. In addition, one can print the value of `abc` to the command line by invoking an inline `FISH` expression via square brackets:

```
[abc]
```

The message

```
71 (integer)
```

will appear on the command prompt. The function `fname` can also be executed with the inline `FISH` syntax as outlined below. Once the symbol `abc` is defined, we can now refer to it in many ways using `FLAC3D` commands. A `FISH` function can have any name, but in order for a value to be returned by the function, the function name should be identical to a value name updated inside the function, as outlined below.

Variables

We now type a slightly different program (using the `model new` command to erase the old function and reset the model state).

Using a variable

```
model new
```

```
fish define abc
  hh = 22
  abc = hh * 3 + 5
end
```

Here we introduce a “variable,” `hh`, which is given the value of 22 and then used in the next line. Notice that the *FISH* function name has been changed to `abc`, identical to the *FISH* variable defined in the function. If we execute this function via inline *FISH* (i.e., `[abc]`), then exactly the same output as in the previous case appears. However, we now have two *FISH* symbols. The two symbols both have values, but one (`abc`) is known as a “function,” and the other (`hh`) as a variable. The distinction is as follows:

When a FISH symbol name is mentioned, the associated function is executed if the symbol corresponds to a function. However, if the symbol is not a function name, then the current value of the symbol is used.

The mechanism of naming a *FISH* function to be identical to a variable set inside the function allows one to provide a return value from the execution of the *FISH* function. This mechanism is frequently used, especially when *FISH* histories (using the `fish history` command) are needed or when *FISH* callbacks (using the `fish callback` command) are used.

The following experiment may help to clarify the distinction between variables and functions. Before doing the experiment, note that inline *FISH* can be used to set the value of any user-defined *FISH* symbol, independent of the *FISH* function in which the symbol was introduced. Now type the following lines without giving the `model new` command, since we want to keep our previously entered program in memory:

```
[abc = 0]
[hh = 0]
[hh]
[abc]
[hh]
```

The values of `abc` and `hh` are set to 0 in the first two lines. The function `abc` is not executed, as the first line is an assignment operation. Note that `abc` still refers to the same *FISH* function even though its value has been modified. `abc` will always refer to the defined *FISH* function unless it is redefined with the `fish define` command. Following the two assignment operations, the value of `hh` is printed.

The line `[abc]` executes the *FISH* function and prints the updated value corresponding to that execution on the command line. The function is executed, as `abc` refers to a function and the value of `abc` is not being assigned. Thus the value of `abc` has been changed along with the value of `hh`, as the last line demonstrates.

Though we highly recommend using the inline *FISH* syntax to manipulate *FISH* variables, an alternate syntax to achieve the same effect is given below:

```
fish set @abc = 0 @hh = 0
list @hh
list @abc
list @hh
```

In this case, the second `fish list` command causes `abc` to be executed, as `abc` is the name of a function, causing the values of both `hh` and `abc` to be updated.

As a test of your understanding, type in the slightly modified sequence shown in the next example and figure out why the displayed answers are different.

Test your understanding of function and variable names

```
model new
fish define abc
  abc = hh * 3 + 5
end
[hh = 22]
[abc]
[abc = 0]
[hh = 0]
[hh]
[abc]
[hh]
```

Using *FISH* for Custom Calculations

The following example shows how the total load on the top platen of a triaxial test sample can be stored as a history for later inspection.

Capturing the history of a *FISH* variable

```
model new
zone create brick size 1 2 1
zone cmodel assign mohr-coulomb
zone property shear=1e8 bulk=2e8 cohes=1e5 tens=1e10
```

```

zone gridpoint fix velocity range position-y 0.0
zone face apply velocity-y -1e-5 range position-y 2.0
fish define get_ad
  ad1 = gp.near(0,2,0)
  ad2 = gp.near(1,2,0)
  ad3 = gp.near(0,2,1)
  ad4 = gp.near(1,2,1)
end
@get_ad
fish define load
  load = gp.force.unbal.y(ad1) + gp.force.unbal.y(ad2) ...
        + gp.force.unbal.y(ad3) + gp.force.unbal.y(ad4)
end
fish history load
zone history displacement-y position (0,2,0)
model step 1000
plot item create chart-history history 1 vs 2 reverse on

```

Note that the *FISH* variable `load` is equal to the sum of four other variables, `gp.force.unbal.y(index)`. The *FISH* grid variable `gp.force.unbal` (which may be accessed by component or as a vector) is an example of a grid quantity that is available within a *FISH* program and is termed a *FISH* intrinsic; see [Zone Gridpoint Functions](#) for a complete list of the grid-related intrinsics. Often times, *FISH* intrinsics are predefined model quantities but, in other cases, they define more complex model operations.

In the example, the *FISH* intrinsic `gp.force.unbal.y` provides the *y*-component of the unbalanced gridpoint force; each instance of `gp.force.unbal` must be followed by a gridpoint memory address (or pointer). The address is found with the function `get_ad`, which uses the *FISH* gridpoint intrinsic `gp.near` to find the address of the gridpoints closest to the (x,y,z) global coordinates $(0,2,0)$, $(1,2,0)$, $(0,2,1)$, and $(1,2,1)$. By creating a history, the *FISH* function `load` is executed when histories are updated during cycling (see the `model update-interval` command to change the history update frequency). Since the value of `load` is updated in the function (i.e., the function name and update variable name are the same), the history contains updated values each time it is called. At the end of the run, one can simply plot the history of `load` (history 1) just like a predefined *FLAC3D* history. Similarly, we may use *FISH* functions to post-process or output the computed history for further analysis.

In addition to the above-mentioned *FISH* intrinsics for gridpoints, there are a plethora of intrinsics available for use in *FISH*. Zone-specific intrinsics are given [here](#) and structural element intrinsics are given [here](#). A catalog of general

intrinsic is given [here](#). Using `math`, for instance, enables things like sines and cosines to be calculated from within a *FISH* function. The list below samples some of the functions available from the set of `math` intrinsic.

<code>math.abs(n)</code>	absolute value of <code>n</code>
<code>math.cos(n)</code>	cosine of <code>n</code> (<code>n</code> is in radians)
<code>math.log(n)</code>	base-ten logarithm of <code>n</code>
<code>math.max(n1, n2)</code>	returns maximum of <code>n1</code> , <code>n2</code>
<code>math.sqrt(n)</code>	square root of <code>n</code>

A more complex example using a number of intrinsic will be presented later, but now we must discuss one more way a *FLAC3D* data file can make use of user-defined *FISH* names:

*Wherever a value is expected in a *FLAC3D* command, you may substitute the name of a *FISH* variable or function using inline *FISH* notation (`[]`) or prefixed by `@`.*

This simple statement is the key to a very powerful feature of *FISH* that allows such things as ranges, applied stresses, properties, etc., to be computed in a *FISH* function and used in *FLAC3D* commands in symbolic form. Hence, parameter changes can be made very easily, without the need to change many numbers in an input file.

As an example, let us assume that we know the Young's modulus and Poisson's ratio of a material. If the user wishes to use the bulk and shear moduli, these may be derived with a *FISH* function, using the equations below:

$$G = \frac{E}{2(1 + \nu)}$$

$$K = \frac{E}{3(1 - 2\nu)}$$

Coding these equations into a *FISH* function (called `derive`) can then be done as shown in this example:

FISH functions to calculate bulk and shear moduli

```

model new
fish define derive(y_mod,p_ratio)
  s_mod = y_mod / (2.0 * (1.0 + p_ratio))
  b_mod = y_mod / (3.0 * (1.0 - 2.0 * p_ratio))
end
[derive(5e8,0.25)]
[b_mod]
[s_mod]

```

Note that we execute the function `derive` by giving its name with arguments inside parenthesis. In this case the arguments, the Young's modulus and Poisson's ratio, respectively, are used as temporary variables to compute the bulk and shear moduli; as a result, they do not appear as *FISH* variables after `derive` has been executed. The computed values of the bulk and shear moduli (`b_mod` and `s_mod`, respectively) can subsequently be used, in symbolic form, in *FLAC3D* commands as shown in the following example:

Using symbolic variables in *FLAC3D* input

```

zone create brick size (2,2,2)
zone cmodel assign elastic
zone property bulk [b_mod] shear [s_mod]
zone list prop bulk
zone list prop shear

```

The result of these operations can be checked by printing `bulk` and `shear` in the usual way (e.g., using the `zone list property` command).

FISH Rules, Syntax, and Statements, Illustrated

There is great flexibility in choosing names for *FISH* variables and functions. The underscore character (`_`) may be included in a name. Names must begin with an alphabetical letter (i.e., cannot be a number or a special character) and must not contain any of the arithmetic operators (`+`, `-`, `/`, `*`, or `^`). A chosen name cannot be the same as the name of one of the built-in *FISH* intrinsics. *FISH* variable names are not case sensitive, so that `aVariable` and `AVARiable` are the same variable.

In addition to inspecting *FISH* variable/function names in the [global symbols](#) control set, one can also obtain a list of all current variables and functions using the `fish list` command. A printout of all current values, sorted alphabetically by name, is produced as a result of this command.

```
fish list
```

We now examine ways decisions can be made, and repeated operations done, via *FISH* programming. The following *FISH* statements allow specified sections of a program to be repeated many times:

```
loop var (a1, a2)
```

```
endloop
```

The words `loop` and `endloop` are *FISH* statements, the symbol `var` stands for the loop variable, and `a1`, `a2` stand for expressions (or single variables) that bound the loop. The next example shows the use of a loop (or repeated sequence) to produce the sum and product of the first ten integers:

Controlled loop in *FISH*

```
model new
fish define xxx
  sum = 0
  prod = 1
  loop n (1,10)
    sum = sum + n
    prod = prod * n
  end_loop
  io.out('The sum is ' + string(sum) + ' ...
        ' and the product is ' + string(prod))
end
@xxx
```

In this case, the loop variable `n` is given successive values from 1 to 10, and the statements inside the loop (between the `loop` and `endloop` statements) are executed for each value. As mentioned, variable names or an arithmetic expression could be substituted for the numbers 1 or 10. Note that the `exit` statement can be used to break out of a *FISH* loop and the `continue` statement can be used to skip the remaining instructions in the loop, moving to the next sequence of the loop.

It is important to note that this formulation of looping is different from a for loop in most high-level programming languages. For instance, one cannot easily control the ending condition (i.e., loop from 1 to 10 excluding 10) or the incrementing mechanism (i.e., loop from 1 to 10 by twos or loop backward). A standard for loop is also available in *FISH* to provide for additional loop control.

Traditional for loop in *FISH*

```

model new
fish define xxx
  sum = 0
  prod = 1
  loop for (n = 1, n <= 10, n = n + 1)
    sum = sum + n
    prod = prod * n
  end_loop
  io.out('The sum is ' + string(sum) + ...
        ' and the product is ' + string(prod))
end
@xxx

```

Besides standard looping as depicted above, one can easily loop over sets of model objects (i.e., zones, gridpoints, structural element nodes, etc.) using the **loop foreach** construct. In this case, a container of objects must be given by a *FISH* intrinsic such as `zone.list`. A practical use of the **loop foreach** construct is to install a nonlinear initial distribution of elastic moduli in a *FLAC3D* grid. Suppose that the Young's modulus at a site is given by this equation:

$$E = E_0 + c\sqrt{z}$$

where z is the depth below surface, and c and E_0 are constants. We write a *FISH* function to install appropriate values of bulk and shear modulus in the grid, as in this example:

Applying a nonlinear initial distribution of moduli

```

model new
zone create brick point 0 (0,0,0) point 1 (-10,0,0) ...
                point 2 (0,10,0) point 3 (0,0,-10)
zone cmodel assign elastic
fish define install(y_zero,cc)
  loop foreach pnt zone.list
    z_depth = -zone.pos.z(pnt)
    y_mod = y_zero + cc * math.sqrt(z_depth)
    zone.prop(pnt,'young') = y_mod
  end_loop
end

```

```
@install(1e7,1e8)
zone property poisson 0.25
plot item create zone contour property name 'young'
```

Again, you can verify correct operation of the function by printing or plotting shear and bulk moduli.

In the function `install`, the loop takes place over all zones in the global list of zones. The *FISH* statement `loop foreach` is a variation of the `loop` statement that sets `pnt` to each zone in `zone.list`. Inside the loop, the z -coordinate of each zone centroid is used to calculate the Young's modulus, given in the equation above. We assume that the datum (or ground surface reference point) is at $z = 0$. The variables `zone.pos.z(pnt)` and `zone.prop(pnt, 'young')` are *zone* intrinsic. (Recall that we talked about the *gridpoint* intrinsic `gp.force.unbal` earlier.) Here, we set properties directly from within a *FISH* function, rather than with a `zone property` command as in an earlier example.

Having seen several examples of *FISH* functions, let's briefly examine the question of *FISH* syntax and style. A complete *FISH* statement occupies one line. However, a line may be typed across two or more lines as long as each line but the ultimate is terminated with the continuation character (`...`). Use of temporary variables as hinge points to concatenate lengthy formulas can also be handy. The following example shows how this can be done:

Splitting lines

```
model new
fish define long_sum ;example of a sum of many things
  local temp = v1 + v2 + v3 + v4 + v5 + v6 + v7 + v8 + v9 + v10
  long_sum = temp + v11 + v12 + v13 + v14 + v15
end
```

In this case, the sum of fifteen variables is split into two parts. The `local` designation for `temp` means that it is used just during the execution of `long_sum` and discarded afterward. By default, *FISH* variables are `global`, meaning that they stick around in memory space until a `model new` command is given. One can specify a variable as global with the `global` statement. It is good practice, and may be more efficient, to designate all *FISH* variables that will not be used by other functions or commands as `local`. For instance, one can make a loop using a local variable as follows:

```
loop foreach local pnt zone.list
```

of

```
loop local n(1,10)
```

Also note the use of the semicolon after the definition of `long_sum`; this indicates a comment. Any characters that follow a semicolon are ignored by the *FISH* compiler, but they *are* echoed to the log file. It is good programming practice to annotate programs with informative comments. Some of the programs have been shown with *indentation* (i.e., space inserted at the beginning of some lines to denote a related group of statements). Any number of space characters may be inserted (optionally) between variable names and arithmetic operations to make the program more readable. Again, it is good programming practice to include indentation to indicate things like loops, conditional clauses, and so on. Spaces in *FISH* are “significant” in the sense that space characters may not be inserted into a variable or function name.

One other topic that should be addressed now is that of variable *type*. You may have noticed, when printing out variables from the various program examples, that numbers are either printed without decimal points or in “E-format” (i.e., as a number with an exponent denoted by “E”). At any instant in time, a *FISH* variable or function name is classified as one of various (and growing) types: *array*, *Boolean*, *integer*, *matrix*, *pointer*, *real*, *string*, *structure*, *tensor*, *2D vector*, or *3D vector*. These types may change dynamically, depending on context, but the casual user should not normally have to worry about the type of a variable, since it is set automatically. Consider the following example:

Variable types

```

model new
fish define types
  v1 = 2
  v2 = 3.4
  v3 = 'Have a nice day'
  v4 = v1 * v2
  v5 = v3 + ', old chap'
  v6 = vector(1,2,3)
  v7 = matrix(vector(1,1,1))
  v8 = true
end
@types
fish list

```

The resulting screen display looks like this:

```

      Name      Value
-----
(function) types 0 (integer)
v1      2 (integer)
v2      3.400000000000000e+00 (real)
v3      'Have a nice day' (string)
v4      6.800000000000000e+00 (real)
v5      'Have a nice day, old chap' (string)
v6      (1.000000000000000e+00,2.000000000000000e+00,
        3.000000000000000e+00) (vector3)      [**]
v7      3 x 1 (matrix)
v8      true (boolean)

```

[**]This line is printed in *FLAC3D* output as one line but has been split here to fit within the space available.

The variables `v1`, `v2`, and `v3` are converted to *integer*, *float* (or *real*), and *string*, respectively, corresponding to the numbers (or strings) that were assigned to them. Integers are exact numbers (without decimal points) but are of limited range; floating-point numbers have limited precision (about 15 decimal places), but are of much greater range; string variables are arbitrary sequences of characters; and *pointers* are used to address *FLAC3D* model components or other internal *FLAC3D* objects. There are various rules for conversion between the types. For example, `v4` becomes a floating-point number, because it is set to the product of a floating-point number and an integer; the variable `v5` becomes a string because it is the sum (concatenation) of two strings.

There is another language element in *FISH* that is commonly used: the **if then else** statement. The following three statements allow decisions to be made within a *FISH* program:

```
if a1 test a2 then
else
endif
```

These statements allow conditional execution of *FISH* function segments; **else** and **then** are optional. The item `test` consists of one of the following symbols or symbol pairs:

= # > < >= <=

The meanings are standard except for #, which means “not equal.” The items `expr1` and `expr2` are any valid expressions or single variables. If the test is true, then the statements immediately following **if** are executed until **else** or **endif** is encountered. If the test is false, the statements between **else** and **endif** are executed if the **else** statement exists; otherwise, the program jumps to the first line after **endif**. The action of these statements is illustrated in the next example:

Action of the **if else endif** construct

```
model new
fish define abc(xx)
  if xx > 0 then
    abc = 33
  else
    abc = 11
  end_if
end
[abc(1)]
[abc(-1)]
```

The displayed value of `abc` in this example depends on the argument provided to `abc` when it is executed. You should experiment with different test symbols (e.g., replace `>` with `<`).

Until now, our *FISH* functions have been invoked from *FLAC3D*, either by using the square brackets [] of inline *FISH*, by giving the function name prepended with the @ character, or by using the `fish list` command. It is also possible to do the reverse, to give *FLAC3D* commands from within *FISH* functions. Most valid *FLAC3D* commands can be embedded between the following *FISH* statements:

`command`

`endcommand`

There are two main reasons for eliciting *FLAC3D* commands from a *FISH* program. First, it is possible to use a *FISH* function to perform operations that are not possible using the predefined *FISH* intrinsics mentioned above. Second, we can control a complete *FLAC3D* run with *FISH*. As an illustration of the first use of the `command endcommand` statement, we can write a *FISH* function to install a number of cable elements at different depths in a material. The number of cables and number of segments are provided as arguments to the function. When many cable elements are required, it becomes tedious to type many separate `structure cable create` commands, each with different grid values. However, with *FISH*, we can elicit *FLAC3D* commands from within a loop and assign the location of the cable ends automatically during the loop, as illustrated in this example:

Automated placing of cable elements

```

model new
zone create brick size 10 3 5
fish define place_cables(num,segs)
  loop local n (1,num)
    local z_d = float(n) - 0.5
    command
      structure cable create ...
                        by-line 0.0 1.5 [z_d] 7.0 1.5 [z_d] ...
                        segments [segs]
    end_command
  end_loop
end
@place_cables(5,7)

```

After entering these statements, you should list and plot the cable data to verify that five cables have been created, and that they are located at the appropriate positions in the grid. In the example, we use variable `z_d` as a parameter in the function `place_cables`; `z_d` is the *z*-coordinate of the endpoints of each cable. Neither the loop index `n` or `z_d` are needed in future computations and are, therefore, designated as `local`.

We can modify this example to perform a construction sequence of excavation and installation of cables. This illustrates the second use of `command endcommand`. We use the `zone gridpoint free` and `model solve` commands to “excavate” the boundary plane at $x = 0$ in five steps. At the end of each step, we install a row of three cables and then excavate the next section of the boundary.

Sequence of excavation and cable placement

```

model new
zone create brick size 10 3 5
zone cmodel assign mohr-coulomb
zone property bulk 1e8 shear 0.3e8 friction 35
zone property cohesion 1e3 tension 1e3
zone initialize density 1000
model gravity 0 0 -10
zone gridpoint fix velocity-x range position-z 0.0
zone gridpoint fix velocity-y range position-z 0.0
zone gridpoint fix velocity-z range position-z 0.0
zone gridpoint fix velocity-y range position-y 0.0
zone gridpoint fix velocity-y range position-y 3.0
zone gridpoint fix velocity-x range position-x 0.0
zone gridpoint fix velocity-x range position-x 10.0
model largestrain on
model history mechanical unbalanced-maximum
model solve
model save 'cab_str'
zone gridpoint initialize displacement-x 0
zone gridpoint initialize displacement-y 0
zone gridpoint initialize displacement-z 0
zone history displacement-x position 0 1 5
fish define place_cables(num,segs)
  loop local n (1,num)
    local z_d = 5.5 - float(n)
    local z_t = z_d + 0.5
    local z_b = z_d - 0.5
    command
      zone gridpoint free velocity-x ...
                          range position-x 0.0 position-z [z_b] [z_t]
    model solve
    structure cable create by-line 0.0 0.5 [z_d] 7.0 0.5 [z_d] ...
                          segments [segs]
    structure cable create by-line 0.0 1.5 [z_d] 7.0 1.5 [z_d] ...
                          segments [segs]
    structure cable create by-line 0.0 2.5 [z_d] 7.0 2.5 [z_d] ...
                          segments [segs]
    structure cable property young 2e10 yield-tension 1e8 ...
                          cross-sectional-area 1.0 ...
                          grout-stiffness 2e10 ...
                          grout-cohesion 1e10 grout-perimeter 1.0
  end_command
end_loop

```

```

end
@place_cables(5,7)
model save 'cab_end'

```

Arrays and Maps

It is often the case that one would like to store a list of objects that they will loop over in the future. These may be computed values from zones, for instance, or specific gridpoint pointers themselves. *FISH* has two containers to use in these circumstances, termed arrays and maps.

An array holds a list of *FISH* variables of any type that can be looped over or accessed by the integer index of the element of the array. Arrays can be multidimensional and do not resize dynamically. The simple example below shows how one can create an array of integers and then sum the values.

Array example

```

model new
fish define array_operation
  ;create and populate an array with products of 2
  arr = array.create(10)
  loop local n(1,10)
    arr[n] = 2*n
  end_loop

  ;compute the sum and product of elements in the array
  sum = 0
  prod = 1
  local i = 1
  loop while (i <= array.size(arr,1))
    sum = sum + arr[i]
    prod = prod * arr[i]
    i = i + 1
  end_loop
  io.out('The sum is ' + string(sum) + ...
        ' and the product is ' + string(prod))
end
@array_operation

```

In this example, an array is created and filled with numbers. The **loop while** construct is used to loop over the array entries and the sum and product are computed and output.

A map, on the other hand, is an associative container, meaning that one can access the members of a map by an integer or string used to insert a value in the map. Maps can dynamically be resized and added to one another (appending maps together), and are the preferred constructs for storing lists of *FISH* variables for later access.

Map example

```

model new
fish define map_operation
  ;create and populate a map with products of 2
  my_map = map(1,2)
  loop local n(2,10)
    map.add(my_map,n,2*n)
  end_loop

  ;compute the sum and product of elements in the map
  sum = 0
  prod = 1
  loop foreach n my_map
    sum = sum + n
    prod = prod * n
  end_loop
  io.out('The sum is ' + string(sum) + ...
        ' and the product is ' + string(prod))
end
@map_operation

```

Unlike with arrays, maps can be looped through using the `loop foreach` construct. In this case, `n` is the value held in each map entry, not the integer name of the object in the map. Likewise, instead of using integers to insert objects into the map, one could use strings such as `first`, `second`, etc. This allows one to easily and efficiently store and access *FISH* variables by a user-defined name.

Further Information

We have now covered some aspects of the *FISH* language and how it interacts with *FLAC3D*. A complete guide to the language, including language rules, statements, intrinsic functions, and examples, is provided in the [FISH Scripting Reference](#) section of this *Help* file.

Problem Solving with *FLAC3D*

This section provides guidance in the use of *FLAC3D* in problem solving for static mechanical analysis.[1] It does so by breaking the modeling process down to a sequence from project start to project completion, as follows.

1. Project Planning and Setup
2. Grid Generation
3. Identifying Regions of the Model
4. Working with Geometric Data
5. Choice of Constitutive Model
6. Material Properties
7. Boundary Conditions
8. Initial Conditions
9. Stepping To Equilibrium
10. Loading And Sequential Modeling
11. Structural Support
12. Interfaces
13. Tips and Advice
14. Interpretation Of Results
15. Project Completion

Each of these modeling aspects is discussed in detail. The user who is familiar with the two-dimensional program *FLAC* will find that the modeling approach is very similar in *FLAC3D*. The major difference is the procedure for grid generation. We recommend that **Grid Generation** be studied carefully, and that the user grasps the techniques for grid generation presented there before creating their own model grids.

You will note that *FISH* programs are used in this section to assist with model generation and problem solving. If you have not used the *FISH* language before, we recommend that you first read **Tutorial: Working with FISH**.

The philosophy of modeling in the field of geomechanics was presented earlier in the topic **Modeling in Concept**. The novice modeler may wish to review that section first. The methodology of modeling in geomechanics can be significantly different from that of other engineering fields, such as structural engineering. It is important to keep this in mind when performing any geomechanics analysis.

Section Outline

- Approach and Project Setup
 - Modeling on a Spectrum
 - Recommended Steps
 - Start a Project
- Grid Generation
 - Primitive-Based Grids
 - Overview of the Grid Primitives
 - Connecting Adjoining Primitive Shapes
 - Fitting the Grid to Simple Shapes
 - Grid Generation with *FISH*
 - Extrusion-Based Grids
 - Create a Set
 - Define the 2D Geometry
 - Zoning and Other Operations
 - Define the Extrusion
 - Building Blocks-Based Grids
 - Create a Building Blocks Set
 - Snap Blocks Together
 - Refinement Operations and Utilities
 - Zoning
 - Grids from Outside *FLAC3D*

- Grids from *Rhino/Griddle*
 - Grid Generation: Additional Facilities
 - Densifying Grids
 - Geometry-Based Densification: Octree Meshing
 - Surface Topography and Layering
- Identifying Regions of the Model
 - Groups and Slots
 - Using the Group Range Element
 - Select and Hide
 - Using the Model Pane
 - Objects in the *Model* Pane
 - Visualizing Objects
 - Selection
 - Groups
 - Additional Commands
- Working with Geometric Data
 - Geometric Data
 - Geometry Visualization
 - Geometry Painting
 - Geometric Filtering – Geometry Range Elements
 - Geometry Data and Group Assignment
- Choice of Constitutive Model
 - Overview of Constitutive Models
 - The Constitutive Models in *FLAC3D*
 - Selection of an Appropriate Model
 - The Effect of Water
 - Ways to Implement Constitutive Models
- Material Properties
 - Intrinsic Deformability Properties
 - Intrinsic Strength Properties
 - Post-Failure Properties
 - Extrapolation to Field-Scale Properties
 - Spatial Variation and Randomness of Property Distribution
- Boundary Conditions
 - Stress Boundary
 - Applied Stress Gradients
 - Changing Boundary Stress
 - Cautions and Advice
 - Displacement Boundary

- Local System and Applied Velocities
 - Surface Corners and Velocity Boundaries
 - Fix vs Apply
 - Reaction Forces
 - Nonuniform Velocities
- Real Boundaries — Choosing the Right Type
- Artificial Boundaries
 - Symmetry Planes
 - Boundary Truncation
- Initial Conditions
 - Uniform Stresses — No Gravity
 - Stresses with Gradients — Uniform Material
 - Stresses with Gradients — Nonuniform Material
 - Stress Initialization in a Nonuniform Material
 - Compaction within a Nonuniform Grid
 - Initial Stresses following a Model Change
 - Stress and Pore-Pressure Initialization with a Phreatic Surface
 - Initialization of Velocities
- Reaching Equilibrium
 - Convergence Criteria
 - Maximum Out-of-Balance Force
 - Local Maximum Force Ratio
 - Average Force Ratio
 - Maximum Force Ratio
 - Ratio
 - Convergence
 - Choosing Convergence Criteria
 - Evaluating Equilibrium
 - Effects of Large Stiffness Differences
- Loading and Sequential Modeling
 - Example: Loading on Three Tunnels
- Structural Support
- Interfaces
- Tips and Advice
 - 1. Check Model Runtime
 - 2. Effects on Runtime
 - 3. Considerations for Zoning Density
 - 4. Automatic Detection of an Equilibrium State
 - 5. Considerations for Selecting Damping

- 6. Check Model Response
- 7. Initializing Variables
- 8. Minimizing Transient Effects on Static Analysis
- 9. Changing Material Models
- 10. Running Problems with In-Situ Field Stresses and Gravity
- 11. Determining Collapse Loads
- 12. Determining Factor of Safety
- 13. Use Bulk and Shear Moduli
- *FLAC3D* Runtime Benchmark
- Interpretation
 - Unbalanced Force and Convergence
 - Gridpoint Velocities
 - Plastic Indicators
 - Histories
 - Endnote
- Project Completion
- References

Endnote

- [1] Problem solving for coupled mechanical-groundwater analysis is discussed in *Fluid-Mechanical Interaction*, and for coupled mechanical-thermal analysis in *Thermal Analysis*. Problem solving for dynamic analysis is discussed in *Dynamic Analysis*. The fluid formulation is a standard feature of *FLAC3D*; thermal and dynamic analysis are available separately as *FLAC3D* Options.

Approach and Project Setup

The modeling of geo-engineering processes involves special considerations, and a design philosophy different from that followed for design with fabricated materials. Analyses and designs for structures and excavations in or on rocks and soils must be achieved with relatively little site-specific data, and an awareness that deformability and strength properties may vary considerably. It is impossible to obtain complete field data at a rock or soil site. For example, information on stresses, properties, and discontinuities can only be partially known, at best.

Since the input data necessary for design predictions are limited, a numerical model in geomechanics should be used primarily to understand the dominant mechanisms affecting the behavior of the system. Once the behavior of the system is understood, it is then appropriate to develop simple calculations for a design process.

This approach is oriented toward geotechnical engineering, in which there is invariably a lack of good data. But in other applications, it may be possible to use *FLAC3D* directly in design if sufficient data, as well as an understanding of material behavior, are available. The results produced in a *FLAC3D* analysis will be accurate when the program is supplied with appropriate data.

Modeling on a Spectrum

Modelers should recognize that there is a continuous spectrum of situations, as illustrated below.

Typical situation	Complicated geology; inaccessible; no testing budget	← →	Simple geology; \$\$\$ spent on site investigation
Data	NONE	← →	COMPLETE
Approach	Investigation of mechanisms	← · Bracket field behavior by parameter studies · →	Predictive (direct use in design)

Figure 1: Spectrum of modeling situations.

FLAC3D may be used either in a fully predictive mode (right-hand side of the image above) or as a “numerical laboratory” to test ideas (left-hand side). It is the field situation (and budget), rather than the program, that determines the types of use. If enough data of a high quality are available, *FLAC3D* can give good predictions.

Since most *FLAC3D* applications will be for situations in which little data are available, the next topic presents a recommended approach for treating a numerical model as if it were a laboratory test. The model should never be considered to be a “black box” that accepts data input at one end and produces a prediction of behavior at the other. The numerical “sample” must be prepared carefully, and several samples tested, to gain an understanding of the problem.

Recommended Steps

The table below lists the steps recommended to perform a successful numerical experiment; each step is discussed separately below.

Table 1: Recommended Steps for Numerical Analysis in Geomechanics

Step 1	Define the objectives for the model analysis.
Step 2	Create a conceptual picture of the physical system.
Step 3	Construct and run simple idealized models.
Step 4	Assemble problem-specific data.
Step 5	Prepare a series of detailed model runs.
Step 6	Perform the model calculations.
Step 7	Present results for interpretation.

Step 1: Define the Objectives for the Model Analysis

The level of detail to be included in a model often depends on the purpose of the analysis. For example, if the objective is to decide between two conflicting mechanisms that are proposed to explain the behavior of a system, then a crude model may be constructed, provided that it allows the mechanisms to occur. It is tempting to include complexity in a model just because it exists in reality. However, complicating features should be omitted if they are likely to have little influence on the response of the model, or if they are irrelevant to the model's purpose. Start with a global view and add refinement as (and if) necessary.

Step 2: Create a Conceptual Picture of the Physical System

It is important to have a conceptual picture of the problem to provide an initial estimate of the expected behavior under the imposed conditions. Several questions should be asked when preparing this picture. For example: Is it anticipated that the system could become unstable? Is the predominant mechanical response linear or nonlinear? Are movements expected to be large or small in comparison with the sizes of objects within the problem region? Are there well-defined discontinuities that may affect the behavior, or does the material behave essentially as a continuum? Is there an influence from

groundwater interaction? Is the system bounded by physical structures, or do its boundaries extend to infinity? Is there any geometric symmetry in the physical structure of the system?

These considerations will dictate the gross characteristics of the numerical model, such as the design of the model geometry, the types of material models, the boundary conditions, and the initial equilibrium state for the analysis. They will determine whether a three-dimensional model is required, or a two-dimensional model can be used to take advantage of geometric conditions in the physical system.

Step 3: Construct and Run Simple Idealized Models

When idealizing a physical system for numerical analysis, it is more efficient to construct and run simple test models first, before building the detailed model. Simple models should be created at the earliest possible stage in a project, to generate both data and understanding. The results can provide further insight into the conceptual picture of the system; step 2 may need to be repeated after simple models are run.

Simple models can reveal shortcomings that can be remedied before any significant effort is invested in the analysis. For example, do the selected material models sufficiently represent the expected behavior? Are the boundary conditions influencing the model response? The results from the simple models can also help guide the plan for data collection by identifying which parameters have the most influence on the analysis.

Step 4: Assemble Problem-Specific Data

The types of data required for a model analysis include the following:

- details of the geometry (e.g., profile of underground openings, surface topography, dam profile, rock/soil structure);
- locations of geologic structure (e.g., faults, bedding planes, joint sets);
- material behavior (e.g., elastic/plastic properties, post-failure behavior);
- initial conditions (e.g., in-situ state of stress, pore pressures, saturation);
- and
- external loading (e.g., explosive loading, pressurized cavern).

Because there are typically large uncertainties associated with specific conditions (in particular, state of stress, deformability, and strength properties), a reasonable range of parameters must be selected for the investigation. The results from the simple model runs (in step 3) can often prove helpful in determining this range, and in providing insight for the design of laboratory and field experiments to collect the needed data.

Step 5: Prepare a Series of Detailed Model Runs

Most often, the numerical analysis will involve a series of computer simulations that include the different mechanisms under investigation, and span the range of parameters derived from the assembled database. When preparing a set of model runs for calculation, several aspects, such as the following, should be considered:

1. How much time is required to perform each model calculation? It can be difficult to obtain sufficient information to arrive at a useful conclusion if model runtimes are excessive. Consideration should be given to performing parameter variations on multiple computers to shorten the total computation time.
2. The state of the model should be saved at several intermediate stages so that the entire run does not have to be repeated for each parameter variation. For example, if the analysis involves several loading/unloading stages, the user should be able to return to any stage, change a parameter, and continue the analysis from that stage. The amount of disk space required for save files should be considered.
3. Are there a sufficient number of monitoring locations in the model to provide for a clear interpretation of model results and for comparison with physical data? It is helpful to locate several points in the model at which a record of the change of a parameter (such as displacement, velocity or stress) can be monitored during the calculation. Also, the maximum unbalanced force in the model should always be monitored to check the equilibrium or failure state at each stage of an analysis.

Step 6: Perform the Model Calculations

It is best to first make one or two model runs, split into separate sections, before launching a series of complete runs. The runs should be checked at each stage to ensure that the response is as expected. Once there is assurance that the model is performing correctly, several data files can be linked together to run a complete calculation sequence. At any time during a sequence of runs, it should be possible to interrupt the calculation, view the results, and then continue or modify the model as appropriate.

Step 7: Present Results for Interpretation

The final stage of problem solving is the presentation of the results for a clear interpretation of the analysis. This is best accomplished by displaying the results graphically, either directly on the computer screen or as output to a hard-copy plotting device. The graphical output should be presented in a format that can be directly compared to field measurements and observations. Plots should clearly identify regions of interest from the analysis, such as locations of calculated stress concentrations, or areas of stable movement versus unstable movement in the model. The numeric values of any variable in the model should also be readily available for more detailed interpretation by the modeler.

We recommend that these seven steps be followed to solve geo-engineering problems efficiently. The following sections describe the application of *FLAC3D* to meet the specific aspects of each of these steps in this modeling approach.

Start a Project

When the initial preparations are complete, the modeling process in *FLAC3D* starts with the creation of a project. This is a simple but important step. The project “houses” the model and the resources that go into creating it, as well as the outputs that are generated by it.

Create a Project File

To create a new project file, select `File --> New Project...` from the main menu. This will open a standard dialog that will allow the user to specify the name and folder location of the project file.

Once the file is created, the folder containing the project file will serve as the working directory for the project. All model state (SAV) files will be stored there unless expressly specified by command to be stored elsewhere. In either case, the project will track their existence as part of the project. This allows for a range of facilities, including rapid movement between saved states within the interface, and collection and storage of the commands that comprise each saved state within the *State Record* pane, which in turn allows for a program undo capability and for creating a bundle file.

The project also saves user interface settings (the layout), model options that have been specified for this project, the state of all program options that are in effect for the project, calculation modes used in the project, and all plots of the model that are created.

The project does not contain saved states, imported files, or data files. These exist as files outside the project. The project *does* track the use of these files and the locations from which they are drawn.

Project Tools

FLAC3D provides the *Project* pane for project management. In addition, it provides the *Pack* facility on the toolbar (Tools ▸ Pack...) for gathering the project file, project source files, and saved state records into a single, easily transmitted/stored file (similar to a compressed archive file). Project files (as part of a bundle) are also the preferred inclusion when sending a technical support request through the *Technical Support* dialog (Help ▸ Technical Support...).

Grid Generation

FLAC3D has three main built-in methods for creating zones for grids: **primitives**, **extrusions**, and **building blocks**. Each is different. Each offers distinct advantages and disadvantages. For initial generation of grids, they are mutually exclusive (i.e., you cannot create one set of zones using primitives *and* an extrusion at the same time). However, the program offers commands that allow zones from any method to be further modified, combined, and so forth, as needed. The right path to the geometry of a model might involve an adept combination of these methods of zone generation.

In addition, *FLAC3D* can create zones from imported files. **Third-party tools** (volume-based CAD programs such as *Rhino*) can be combined with Itasca's advanced automatic mesher *Griddle* to create zones as well. This last is the most advanced approach to zone creation, suitable for the most complex meshes for which *FLAC3D*'s built-in tools are either insufficient or excessively labor intensive.

FLAC3D also has **additional facilities** for generating and modifying zones. For circumstances where geometry is complex but exact conformance of zone faces to surfaces is not physically significant, it is common to create **octree** meshes using geometric descriptions and mesh densification, as shown briefly in **Geometry-Based Densification: Octree Meshing**. Where there are complex unintersecting layers, or uneven surface topography, the `zone generate from-topography` command can be used as illustrated in **Surface Topography and Layering**.

This section describes the use of the built-in methods of grid generation and briefly describes the additional capabilities provided by third-party tools and imports. The considerations involved with grid generation are extensive. Selection of the right mesh generation method and the efficient deployment of that method is a critical part of the *FLAC3D* modeling process.

As can be intuited from the number of tools *FLAC3D* makes available, there is no single mesh generation method that is the best answer for every case. The optimal approach is very dependent on the model geometry and the goals of the analysis. The following is a brief initial guide for which *built-in* methods should be chosen:

- If the problem has very simple, regular geometry, or one that happens to conform to one of the shapes available to the `zone create` command, then the `primitive` approach is the fastest and easiest.
- If the geometry of the problem can be described in a two-dimensional diagram, or with only minor variations in the third dimension, then the `extrusion` should be considered first. Note that 2D starting geometry can be exported to Building Blocks for 3D modifications.
- If the problem is more complex but still consists of relatively regular shapes, then `Building Blocks` should be considered. This is fairly common for civil engineering problems involving tunnels, structures, or foundations. Irregular shapes can be accommodated with this tool with either extensive manual adjustment or by strategic use of the draping tool, but this is recommended only for limited areas in an otherwise regular model.
- If the problem is very irregular and/or involves complex irregular intersecting surfaces, then one needs to decide if exact conformation of the zone faces to the surfaces describing the model is important to the result. Often, for irregular orebodies or other material boundaries, this is not important to the overall physical response of the model. In this case, using an octree approach (perhaps after first using Building Blocks to create the area of most interest) is often used. This type of model is not uncommon in mining.
- If the problem is very irregular and exact conformance of the mesh is important, then Itasca's *Griddle* or another third-party meshing tool should be considered.

By any means of construction, grid generation involves tradeoffs between computational efficiency, realism of model geometry, and accuracy of results. The following factors govern those tradeoffs.

1. Hexahedral zones are preferable because:
 - the mixed-discretization procedure makes them respond more accurately to situations involving constant-volume plastic deformation (e.g., as seen during Mohr-Coulomb shear failure); and

- the mixed-discretization procedure makes their response more accurate in general (slightly higher-order response).

However, they are more difficult to use than tetrahedral zones when building geometrically complex models.

2. Computational speed decreases with the number of zones.
3. Finer meshes lead to more accurate results in that they provide a better representation of high-stress gradients.
4. Accuracy increases as zone aspect ratios tend to unity.
5. If different zone sizes are needed, then the more gradual the change from the smallest to the largest, the better the results.

Primitive-Based Grids

At first pass, it may seem that the primitive-based grid generation of *FLAC3D* is limited to rather simple, regular-shaped regions. In the introductory sections, the examples provided are often uniform, polyhedral grids. *FLAC3D* primitive shapes, however, can be distorted to fit arbitrary and complicated volumetric regions. *FLAC3D* provides powerful grid generation commands to manipulate primitives to fit various shapes of three-dimensional problem domains.

The procedure for implementing primitive-based grids by command is described in this topic. An overview of the operation is given first, in [Overview of the Grid Generator](#). This is followed, in [Fitting the Grid to Simple Shapes](#) and [Grid Generation with FISH](#), by presentations on various aspects of grid generation, along with guidelines to follow in designing the grid for accurate solutions. Examples are given to illustrate each aspect.

One important aspect in grid generation is that all physical boundaries to be represented in the model simulation (including regions that will be added, or excavations created at a later stage in the simulation) *must* be defined before the solution stepping begins. Shapes of structures that will be added later in a sequential analysis must be defined and then removed (via either `zone cmodel assign null` or `zone delete`) until they are to be activated. The purpose of the grid generator is to facilitate the creation of all required physical shapes in the model.

All examples listed in the section are included in the project file “grid-generation.f3prj”.

Overview of the Grid Primitives

Grid generation using *primitives* in *FLAC3D* involves patching together grid shapes of specific connectivity to form a complete model with the desired geometry. Several types of primitives are available, and these can be connected and conformed to create complex three-dimensional geometries.

The generation of zones for each primitive type is performed with the `zone create` command. Single reference points can be defined using the `zone gridpoint create` command to put gridpoints at specific locations and subsequently refer to them in the `zone create` command. The `zone gridpoint merge` command can be used to ensure that separate primitives are connected properly. All gridpoints along matching faces of zone primitives must fall within a specified tolerance for two primitives to be merged. Alternatively, the `zone attach` command is available to connect primitive meshes of different zone sizes. *FISH* can be used to adjust the final mesh, if necessary, to conform to the surfaces of the model region. The following sections describe the use of each of these facilities to create a *FLAC3D* mesh.

Zone Generation

The primitive-based *FLAC3D* grid is generated with the `zone create` command. This command actually accesses a library of primitive shapes; each shape has a specific type of grid connectivity. The primitive shapes available in *FLAC3D*, listed in order of increasing complexity, are as follows.

Table 1: Primitive Mesh Shapes Available with the `zone create` Command

Keyword	Definition
brick	brick-shaped mesh
wedge	wedge-shaped mesh
uniform-wedge	uniform wedge-shaped mesh
tetrahedron	tetrahedral-shaped mesh
pyramid	pyramid-shaped mesh

Keyword	Definition
cylinder	cylindrical-shaped mesh
degenerate-brick	degenerate brick-shaped mesh
radial-brick	radially graded mesh around brick
radial-tunnel	radially graded mesh around parallelepiped-shaped tunnel
radial-cylinder	radially graded mesh around cylindrical-shaped tunnel
cylindrical-shell	cylindrical shell mesh
cylindrical-intersection	intersecting cylindrical-shaped tunnels
tunnel-intersection	intersecting parallelepiped-shaped tunnels

As was seen in [Tutorial: Illustrative Model — Mechanics of Using FLAC3D](#), `zone create` commands can be used alone to create a zoned model. If the 3D domain consists of simple shapes, the primitives can be applied individually or connected together to create the *FLAC3D* grid.

As an example, a quarter-symmetry model can be created for a cylindrical tunnel with the command

```
zone create radial-cylinder size 5 10 6 12 fill
```

The `size` keyword defines the number of zones in the grid. For the cylindrical tunnel, each entry following the `size` keyword corresponds to the number of zones in a specific direction. In this case, there are five zones along the inner radius of the cylindrical tunnel, ten zones along the axis of the tunnel, six zones along the circumference of the tunnel, and twelve zones between the periphery of the tunnel and the outer boundary of the model. The image below shows the model grid. The `fill` keyword is given to fill the tunnel with zones.

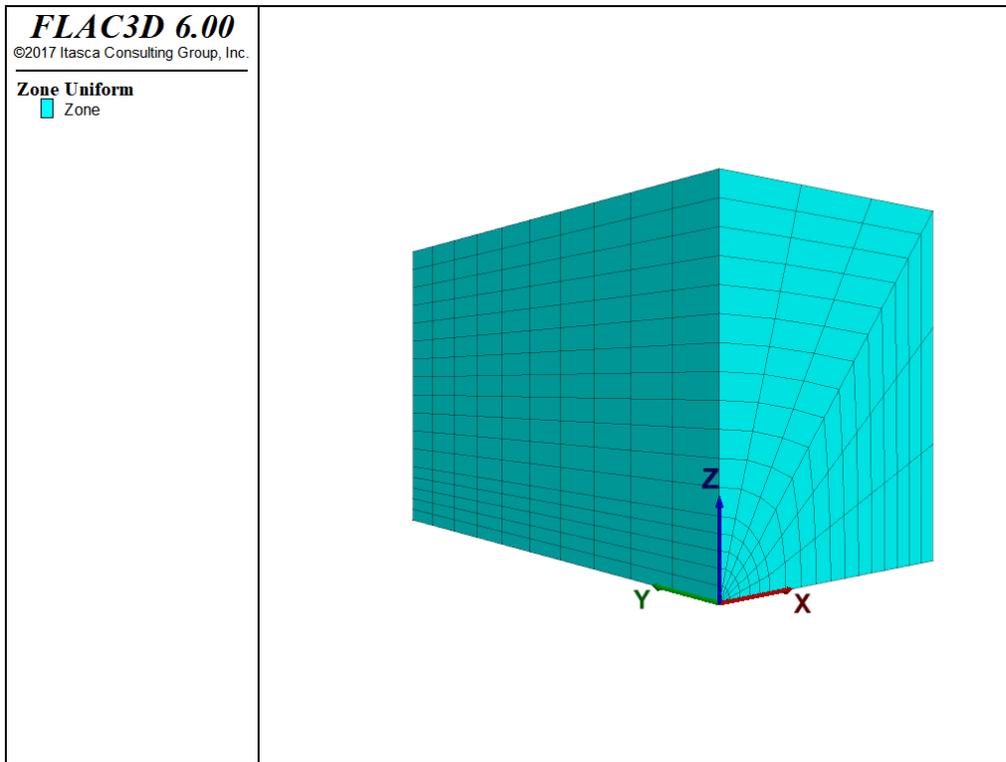


Figure 1: Grid of cylindrical tunnel model.

In addition to `size` and `fill`, there are several other keywords available to define the characteristics of the primitive shapes. The available characteristic keywords for primitive shapes are summarized in the table below. Refer to the `zone create` command and the reference diagrams it contains to identify which keywords and numerical entries are applicable for each primitive shape. For example, click the thumbnail on `zone create radial-cylinder` to see the order in which the `size` entries (`i1`, `i2`, `i3`, `i4`) should be entered for the cylindrical tunnel.

Table 2: Characteristics Keywords for `zone create` Primitive Shapes

Keyword	Definition
<code>dimension</code>	dimensions (length) of interior regions
<code>edge</code>	edge length for the sides of the mesh
<code>fill</code>	fill the interior region with zones
<code>point</code>	reference (corner) points for the shape. First should follow a point index (0 through 16) and then an <code>x,y,z</code> coordinate location.

Keyword	Definition
ratio	geometric ratio used to space zones
size	number of zones for each shape

The `ratio` keyword is of particular significance when designing a grid to provide an accurate solution without requiring an excessive number of zones. For example, if fine zoning is required immediately around the periphery of the cylindrical tunnel in order to provide a more accurate representation of high-stress gradients, `ratio` can be used to adjust the zone size to be small close to the tunnel, and gradually increase in size away from the tunnel.

To see the effects of using the `ratio` keyword, type the command

```
zone create radial-cylinder size 5 10 6 12 ratio 1 1 1 1.2
```

Each `size` entry is controlled by a `ratio`. In this example, the fourth `size` entry has a geometric ratio of 1.2 (i.e., each successive zone is 1.2 times larger than the preceding zone, moving from the tunnel periphery to the outer boundary—see the next figure). A ratio smaller than 1.0 can be given to change from an increasing to a decreasing geometric ratio.

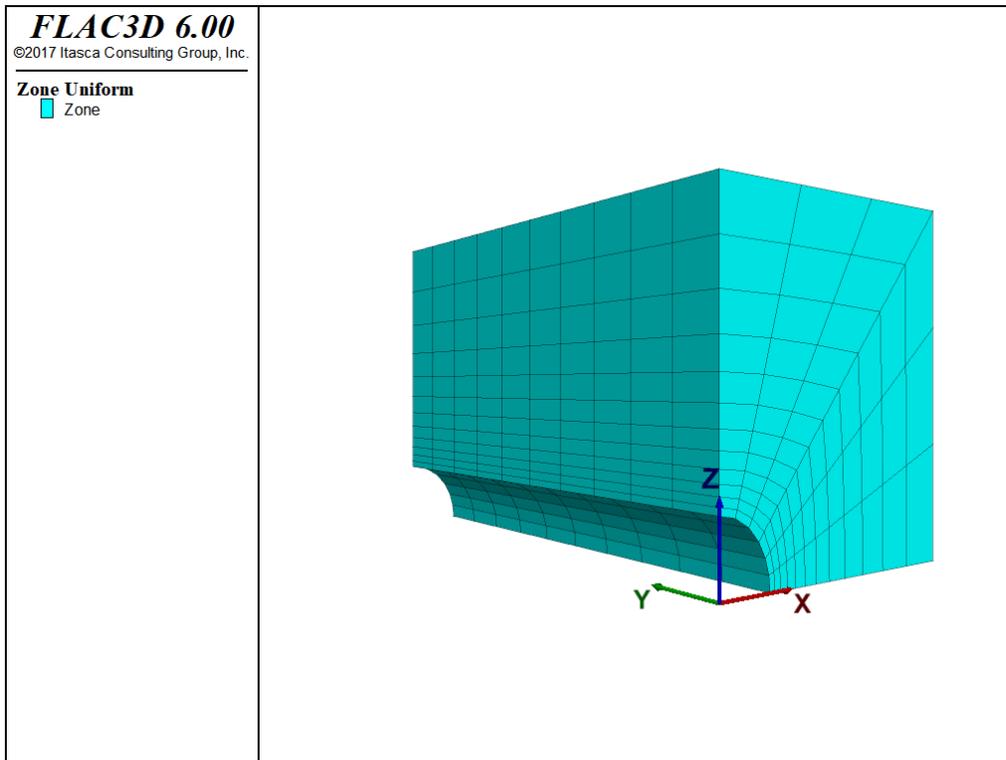


Figure 2: Radially graded grid around cylindrical tunnel.

Sizing the grid for accurate results, but with a reasonable number of zones, can be complicated. Three factors should be remembered:

1. Finer meshes lead to more accurate results in that they provide a better representation of high-stress gradients.
2. Accuracy increases as zone aspect ratios tend to unity. A rule of thumb is try to avoid aspect ratios greater than 10 or so.
3. If different zone sizes are needed, then the more gradual the change from the smallest to the largest, the better the results.

The examples in the following sections illustrate some applications of these factors.

Several `zone create` commands can be given to connect two or more primitive shapes together to build a grid. For example, to build a horseshoe-shaped tunnel, the `radial-cylinder` and `radial-tunnel` shapes can be used as demonstrated in this example:

Building a horseshoe-shaped tunnel — half model

```

zone create radial-cylinder size 5 10 6 12 rat 1 1 1 1.2 ...
    point 0 (0,0,0) point 1 (100,0,0) ...
    point 2 (0,200,0) point 3 (0,0,100)
zone create radial-tunnel size 5 10 5 12 rat 1 1 1 1.2 ...
    point 0 (0,0,0) point 1 (0,0,-100) ...
    point 2 (0,200,0) point 3 (100,0,0)

```

Refer to the reference figures for `zone create radial-cylinder` and `zone create radial-tunnel` when building these shapes. The model boundary dimensions are $100 \times 200 \times 100$; the boundary coordinates are defined with `point` keywords. The grid is shown below. Note that the `radial-tunnel` shape is turned 90° to fit beneath the `radial-cylinder` shape. This is accomplished by specifying different `point` coordinate entries for the `radial-tunnel` shape.

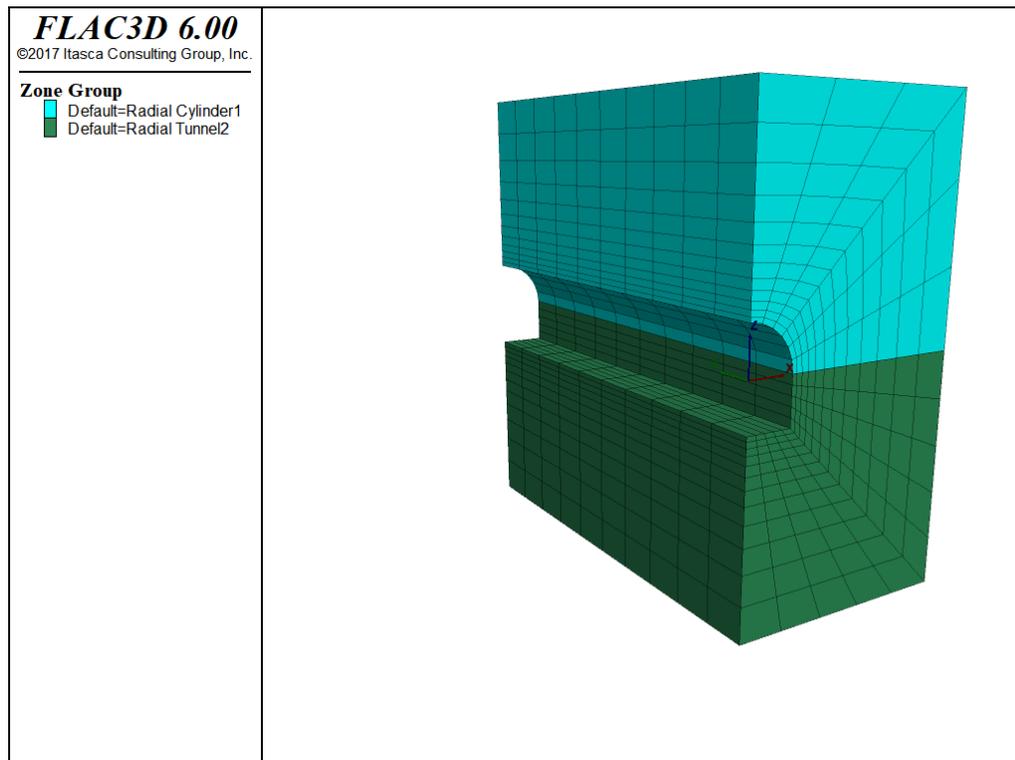


Figure 3: Horseshoe-shaped tunnel made from `radcylinder` and `radtunnel` primitives.

With `zone create`, two additional options are available to assist with the creation of a grid composed of multiple shapes: `zone copy` and `zone reflect`. The former is used to copy a shape or shapes to a new position by adding an offset vector to all

the gridpoints. The latter is used to reflect the shape or shapes across a symmetry plane. The next example shows the additional command needed to reflect the geometry created by the earlier commands.

Building a horseshoe-shaped tunnel — full model

```
zone create radial-cylinder size 5 10 6 12 rat 1 1 1 1.2 ...
    point 0 (0,0,0) point 1 (100,0,0) ...
    point 2 (0,200,0) point 3 (0,0,100)
zone create radial-tunnel size 5 10 5 12 rat 1 1 1 1.2 ...
    point 0 (0,0,0) point 1 (0,0,-100) ...
    point 2 (0,200,0) point 3 (100,0,0)
zone reflect dip 90 dip-direction 270 origin (0,0,0)
```

The resulting grid is shown below. The symmetry plane is a vertical plane (located by the `dip`, `dip-direction`, and `origin` keywords) coincident with the $x = 0$ plane. Note that dip angle (`dip`) and dip direction (`dip-direction`) assume that x corresponds to “East,” y to “North”, and z to “Up.”

A third option, the `zone gridpoint create` command, is available to position single points in the model region. This is useful for positioning reference points of primitives. The section [Fitting the Grid to Simple Shapes](#) below presents an example use of `zone gridpoint create` to position the invert of two tunnels of different sizes at the same elevation.

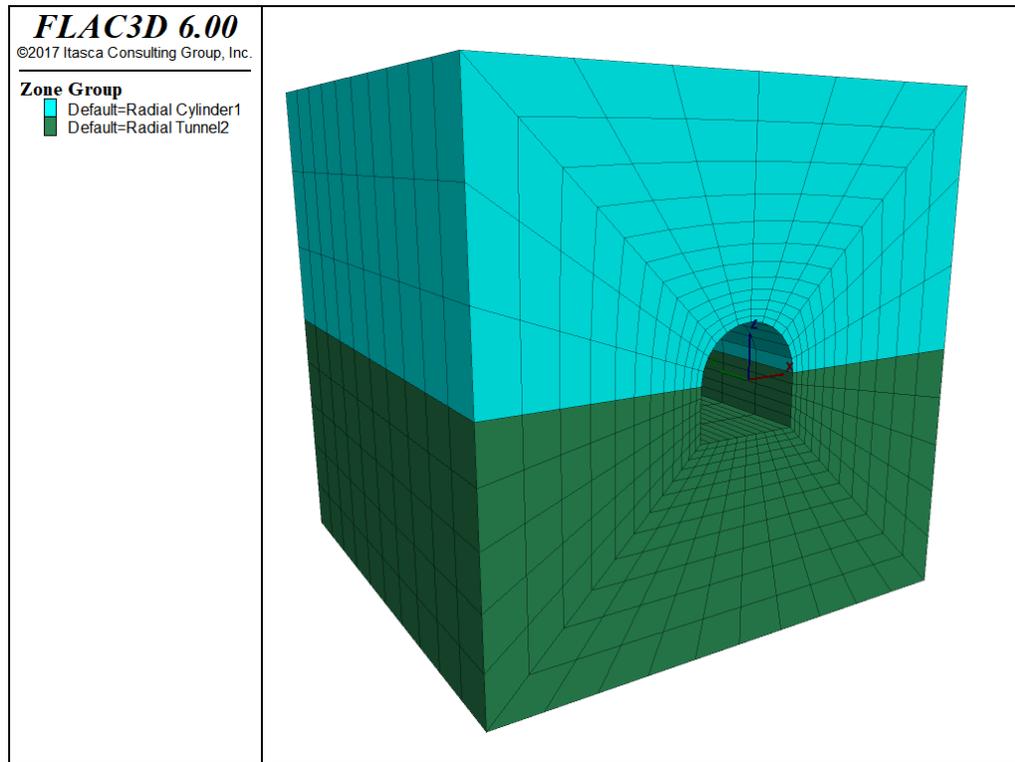


Figure 4: Complete horseshoe-shaped tunnel made from the *reflect* keyword.

Connecting Adjoining Primitive Shapes

When building a geometry out of primitives, the sides of the primitives *must* connect to form an unbroken continuum. During execution of a `zone create` command, a check is made for each boundary gridpoint against the boundary gridpoints of zones that already exist. Internal gridpoints are not checked. If two boundary gridpoints fall within a tolerance of 1×10^{-7} (relative to the magnitude of the gridpoints position vector) of each other, they are assumed to be the same point, and the first gridpoint is used instead of creating a new one for all subsequent calculations.

The user is responsible for ensuring that all gridpoints along adjoining primitives correspond to one another. The use of reference points with the command `zone gridpoint create` during model creation can be useful to make sure that the bounding brick is specified correctly for both primitives. Make sure that the number of zones is correct and that the `ratios` used for the zone distribution are consistent. Note that if the ratio for one primitive is going the opposite direction of the other, the inverse ratio should be used for one of the primitives to ensure that boundary gridpoints match.

This version of *FLAC3D* does not issue a warning message if gridpoints at boundaries do not match. Localized velocity anomalies will be observed at nonmatching gridpoints in the model when the calculation is started. If it is discovered that some gridpoints don't match, the `zone gridpoint merge` command can be used to merge these gridpoints after the `zone create` command has been applied.

The `zone attach` command can be used to connect primitives with different zone sizes. There are some restrictions, though, to the range in zone size that may be specified with this approach. For the most accurate calculations, the ratio of zone sizes should be a multiple integer ratio (e.g., 2 to 1, 3 to 1, 4 to 1). It is recommended that the ratio be tested first by running the model under elastic conditions. If a discontinuity is observed in the displacement, or stress distribution across the attached grids, then the ratio of zone sizes may need to be adjusted. However, if the discontinuity is small and far from the region of interest, it may not have a significant influence on the calculation.

The next example illustrates the use of the `zone attach` command and the effect of different zone sizes. A brick primitive with a zone dimension of 0.5 is connected to a brick primitive with a zone dimension of 1. The resulting *z*-displacement contours are shown in the next figure below.

In the next example, alternatively, the first two command lines can be changed to

```
zone create brick size 4 4 4
zone densify segments 2 range position-x 2 4
```

where `zone densify segments 2` refines the upper zones (between the *z*-coordinate of 2 and 4) with the segment number of 2 on each edge.

Two unequal sub-grids

```
zone create brick size 4 4 2 point 0 (0,0,0) point 1 (4,0,0) ...
                                point 2 (0,4,0) point 3 (0,0,2)
zone create brick size 8 8 4 point 0 (0,0,2) point 1 (4,0,2) ...
                                point 2 (0,4,2) point 3 (0,0,4)
zone attach by-face range position-z 2
zone cmodel assign elastic
zone property bulk 8e9 shear 5e9
zone gridpoint fix velocity-z range position-z 0
zone gridpoint fix velocity-x range union position-x 0 position-x 4
zone gridpoint fix velocity-y range union position-y 0 position-y 4
zone face apply stress-zz -1e6 ...
```

```

range position-z 4 position-x 0,2 position-y 0,2
model history mechanical unbalanced-maximum
model solve
model save 'att'

```

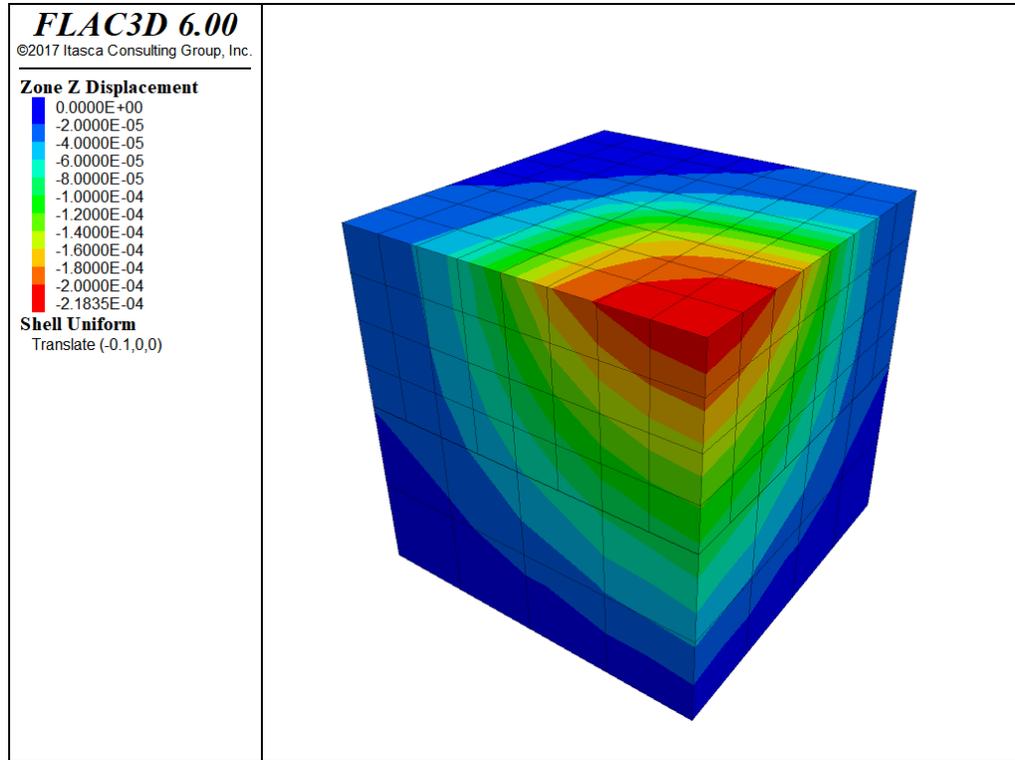


Figure 5: z-displacement contours in two attached grids with zone ratio of 2 to

1.

To test the accuracy, we do a similar run, but for a single grid with a constant zone dimension of 0.5. The data file is given next. The results are shown in the figure that follows. This plot is almost identical to the one above.

A single grid for comparison to two sub-grids

```

zone create brick size 8 8 8 point 0 0,0,0 point 1 4,0,0 ...
point 2 0,4,0 point 3 0,0,4
zone cmodel assign elastic
zone property bulk 8e9 shear 5e9
zone gridpoint fix velocity-z range position-z 0
zone gridpoint fix velocity-x range union position-x 0 position-x 4
zone gridpoint fix velocity-y range union position-y 0 position-y 4
zone face apply stress-zz -1e6 ...
range position-z 4 position-x 0,2 position-y 0,2
model history mechanical unbalanced-maximum

```

```
model solve
model save 'noatt'
```

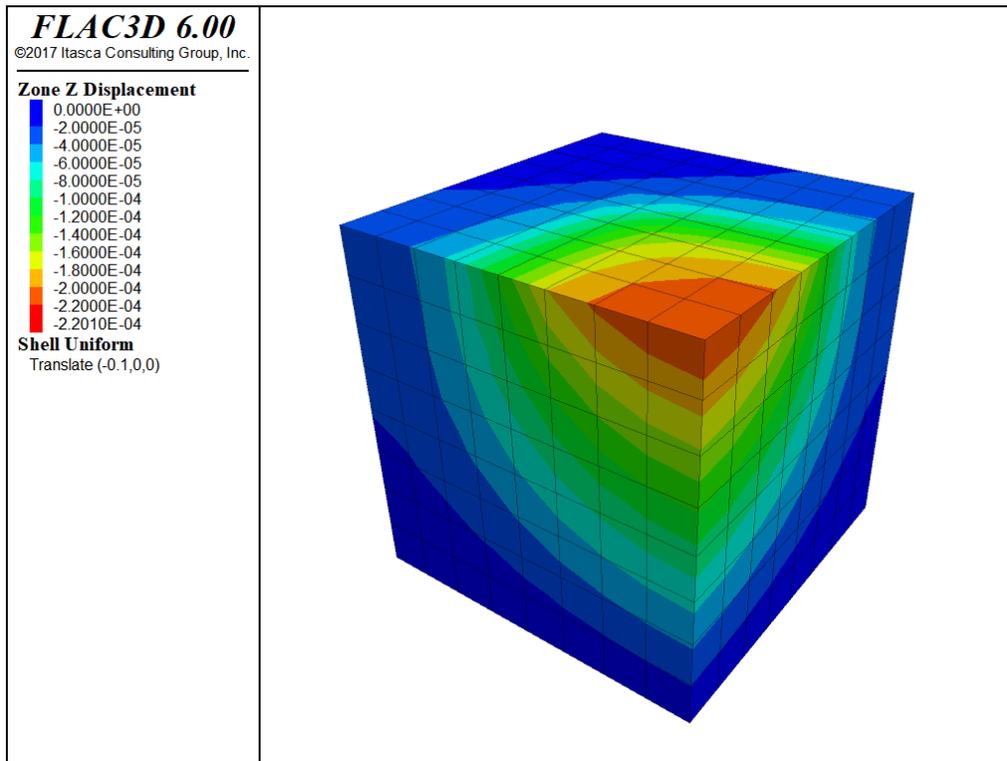


Figure 6: z-displacement contours in single grid.

Fitting the Grid to Simple Shapes

The intention of grid generation is to fit the model grid to the physical region under study. For simple geometries, the `zone create` command is all that is required to generate a model grid to fit the problem domain. To determine whether `zone create` is sufficient, try defining the problem domain by one or more of the primitive shapes shown in the command's documentation.

For example, consider a problem geometry involving three parallel tunnels (a service tunnel located midway between two larger main tunnels). The three tunnels are all cylindrical in shape, so the `radial-cylinder` primitive shape is the logical choice to build the tunnel grids. A vertical symmetry plane may be assumed to exist along the centerline of the service tunnel. Thus, it is only necessary to create a mesh for one main tunnel and half of the service tunnel.

We have two important concerns when building any model: 1) the density of zoning required for an accurate solution in the region of interest; and 2) how the location of the grid boundaries influence model results.

It is important to have a high density of zoning in regions of high stress- or strain- gradients. Often, it is possible to perform two-dimensional analyses to define these regions. For this problem, a 2D *FLAC* calculation can easily be run to determine an acceptable density of zones around the tunnels. For demonstration purposes, we select a zone size of roughly one-half the service tunnel radius for the zones surrounding the tunnels.

The first step in grid generation for this problem is to use `radial-cylinder` primitives to create the grids for the tunnels. The complicating factor is that the tunnels are of different sizes and have the same invert elevation. The service tunnel has a radius of 3 m, and the main tunnels have a radius of 4 m. The length of the model corresponds to a 50 m length of the tunnels. The following example shows the commands to create the grid surrounding the tunnels.

Creating a grid for two tunnels with the same invert elevation

```

; main tunnel
zone create radial-cylinder point 0 (15,0,0) point 1 (23,0,0) ...
                             point 2 (15,50,0) point 3 (15,0,8) ...
                             size 4 10 6 4 dim 4 4 4 4 rat 1 1 1 1 ...
                             fill group 'service'
zone reflect dip 90 dip-direction 90 origin (15,0,0)
zone reflect dip 0 origin (0,0,0)
; service tunnel
zone gridpoint create (2.969848, 0.0, -0.575736) name 'p2124'
zone gridpoint create (2.969848, 50.0, -0.575736) name 'p2125'
zone create radial-cylinder point 0 (0, 0, -1) point 1 (7,0,0) ...
                             point 2 (0,50,-1) point 3 (0,0,8) ...
                             point 4 (7,50,0) point 5 (0,50, 8) ...
                             point 6 (7,0,8) point 7 (7,50, 8) ...
                             point 8 gridpoint 'p2124' ...
                             point 10 gridpoint 'p2125' ...
                             size 3 10 6 4 dim 3 3 3 3 rat 1 1 1 1
zone create radial-cylinder point 0 (0, 0, -1) point 1 (0,0,-8) ...
                             point 2 (0,50,-1) point 3 (7,0,0) ...
                             point 4 (0,50,-8) point 5 (7,50, 0) ...
                             point 6 (7,0,-8) point 7 (7,50,-8) ...
                             point 9 gridpoint 'p2124' ...
                             point 11 gridpoint 'p2125' ...
                             size 3 10 6 4 dim 3 3 3 3 rat 1 1 1 1

```

The main tunnel grid is created first: one-quarter of the grid is generated; then the grid is reflected across horizontal and vertical planes to create the grid for the entire tunnel. The `reflect` option cannot be used for the service tunnel, because the invert of the service tunnel is required to be at the same elevation as the main tunnel. The vertices locating the service tunnel radius must be adjusted in the `radial-cylinder` primitive. This is done by first defining these locations using the `zone gridpoint create` command. The corner points `point 8` and `point 10` in one `radial-cylinder` primitive, and `point 9` and `point 11` in the other primitive, are located at the reference points. This ensures that the two primitives will match at boundary gridpoints when the grid is generated. The figure below shows the grid for the tunnels. The vertical plane at $x = 0$ is a symmetry plane.

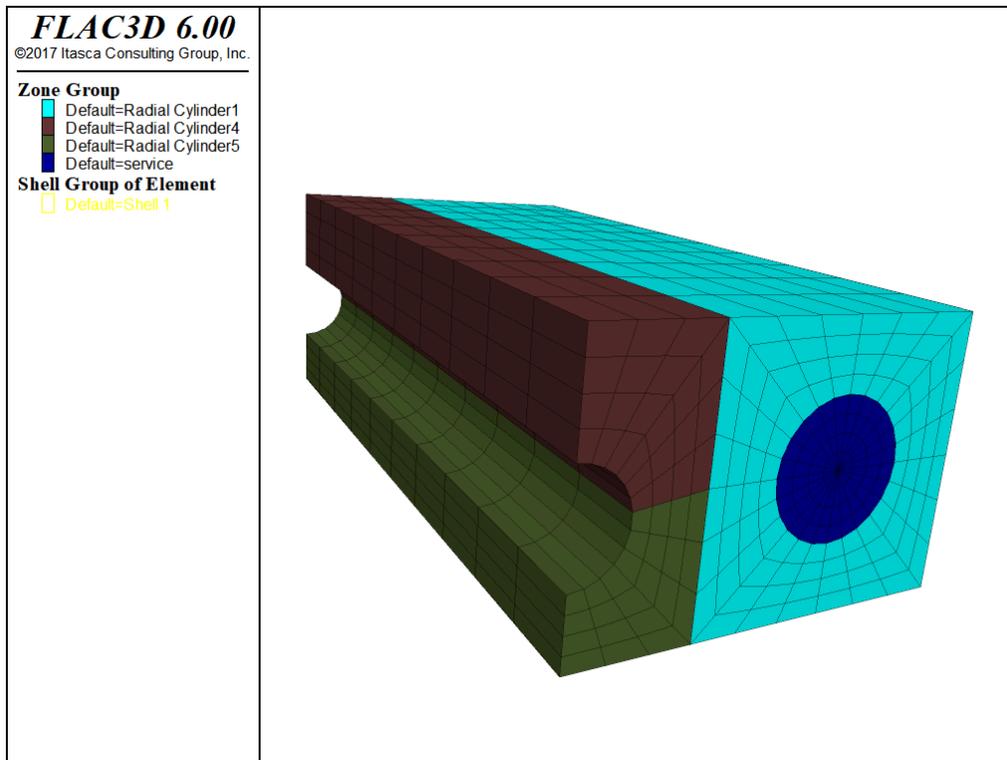


Figure 7: Inner grid for the service and main tunnels.

For this model, we begin with the main tunnel filled with zones and the service tunnel not filled. The excavation and construction stages analyzed with this model are described later in this topic. Before the main tunnel is excavated, we define a liner for the service tunnel; this is accomplished with the `structure shell create` command, which creates a tunnel lining composed of structural

shell elements.* Structural elements should generally be used to represent thin tunnel liners because they provide a more accurate representation of liner bending than a liner composed of finite-difference zones. See the section [Structural Element Operations](#) for detailed information on the structural element logic in *FLAC3D*. The next example gives the command to create the liner.

Creating a liner in the service tunnel

```

; liner
structure shell create by-face range cylinder ...
                                end-1 (0,0,-1) end-2 (0,50,-1) ...
                                radius 3

```

The liner contains 240 structural shell elements and is connected to the *FLAC3D* grid at 143 structural-node links. The grid with the liner is shown below.

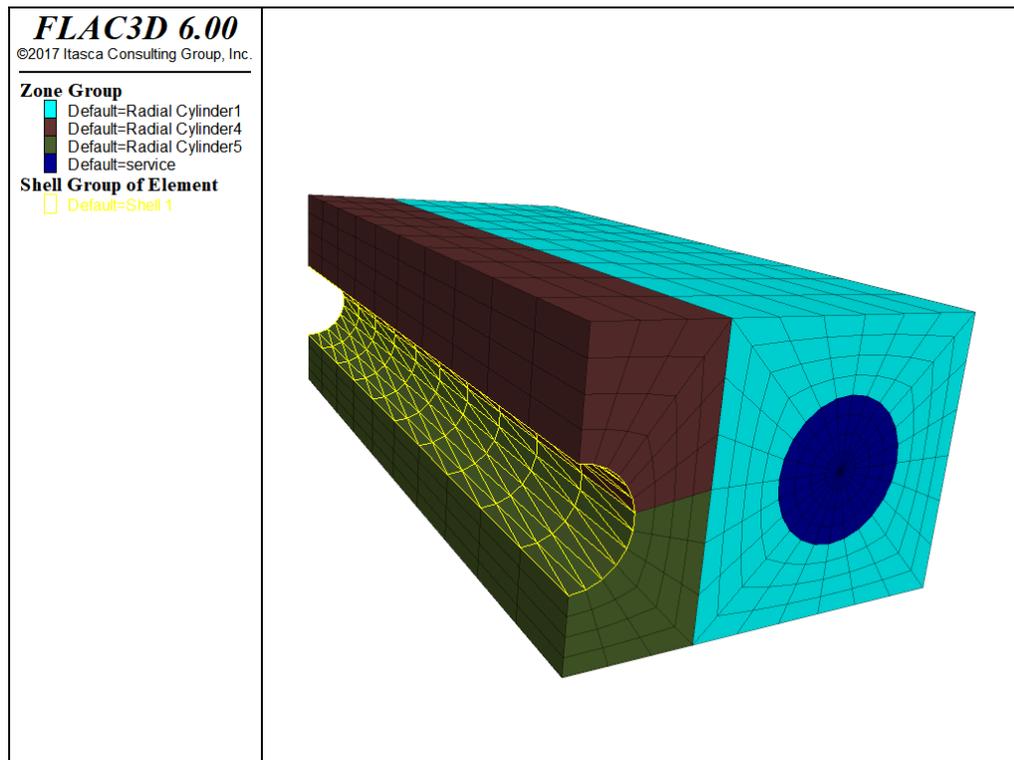


Figure 8: Liner elements in service tunnel.

Finally, the outer-boundary grid is created around the tunnel grid. For analyses of underground excavations, boundaries should be located roughly ten excavation diameters from the excavation periphery. The distance, however, can vary

depending on the purpose of the analysis. If failure is of primary concern, then the model boundaries may be closer; if displacements are important, then the distance to the boundaries may need to be increased.

It is important to experiment with the model to assess boundary effects. Begin with a coarse two-dimensional grid, using *FLAC*, and bracket the boundary effect using fixed and free boundary conditions while changing the distance to the boundary. The resulting effect of changing the boundary can then be evaluated in terms of differences in stress or displacement calculated in the region of interest. The boundary location should then be tested with a coarse grid in *FLAC3D*.

We create the boundary grid with the `radial-tunnel` and `brick` primitives for this model. The `zone reflect` command is used to reflect the grid across the plane at $z = 0$. We use the `range` keyword to restrict the action of `zone reflect`. See the following example.

The final grid for this problem is shown below.

Creating the boundary grid

```
; outer boundary
zone create radial-tunnel point 0 ( 7, 0, 0) point 1 (50, 0, 0) ...
                        point 2 ( 7,50, 0) point 3 (15, 0,50) ...
                        point 4 (50,50, 0) point 5 (15,50,50) ...
                        point 6 (50, 0,50) point 7 (50,50,50) ...
                        point 8 (23, 0, 0) point 9 ( 7, 0, 8) ...
                        point 10 (23,50, 0) point 11 ( 7,50, 8) ...
                        size 6 10 3 10 ratio 1 1 1 1.1
zone create brick point 0 ( 0, 0, 8) point 1 ( 7, 0, 8) ...
                  point 2 ( 0,50, 8) point 3 ( 0, 0,50) ...
                  point 4 ( 7,50, 8) point 5 ( 0,50,50) ...
                  point 6 (15, 0,50) point 7 (15,50,50) ...
                  size 3 10 10 ratio 1 1 1.1
zone reflect dip 0 origin (0,0,0) ...
                range position-x 0 23 position-y 0 50 position-z 8 50
zone reflect dip 0 origin (0,0,0) ...
                range position-x 23 50 position-y 0 50 position-z 0 50
```

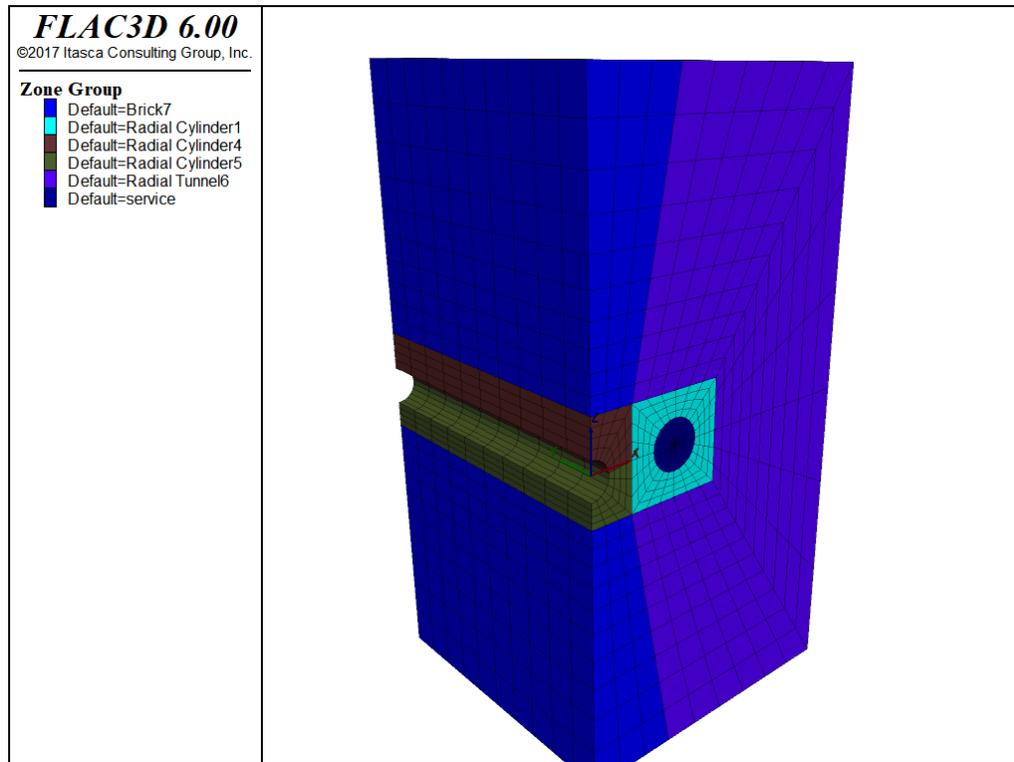


Figure 9: Complete grid for service and main tunnels.

Grid Generation with *FISH*

FISH can be used to specify a geometric shape that is not readily available by using the built-in primitives in *FLAC3D*. It is also possible to create your own primitive shapes using *FISH*. In the following example, we create a radially graded mesh around a spherical cavity. Only one-eighth of the grid is generated. The grid can be reflected to create the complete spherical cavity.

A `radial-brick` primitive is the basis for creating our “radsphere” shape. We first define parameters that describe the sphere within a cube: the radius of the spherical cavity; the length of the outer cube edge; the number of zones along the outer cube edge; and the number of zones in the radial direction from the inner cube to the outer cube. Then we define a `radial-brick` such that the spherical cavity will be inscribed in the inner cube. The next example lists the commands to create the initial grid for a geometric ratio of 1.2. The grid is shown in the figure that follows.

Parameters to create a radially graded mesh around a spherical cavity

```
[global rad=4.0 ] ; radius of spherical cavity
```

```
[global len=10.0 ] ; length of outer box edge
zone create radial-brick edge @len size 6,6,6,10 ...
      rat 1.0 1.0 1.0 1.2 dim @rad @rad @rad
```

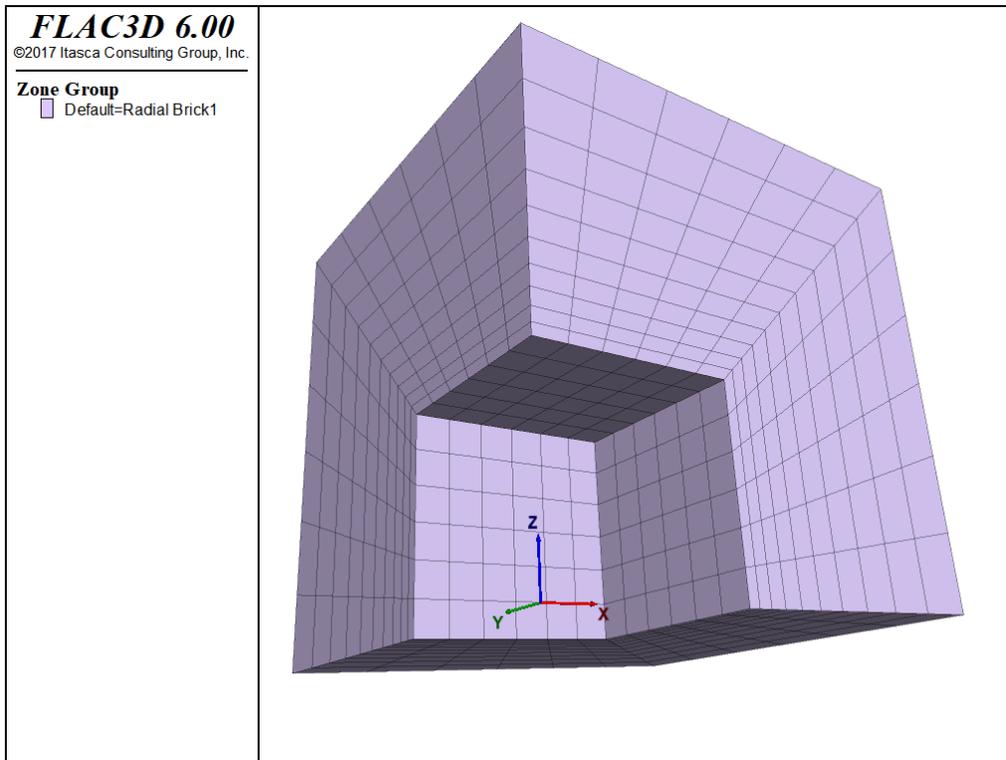


Figure 10: Initial radial-brick primitive to create a radially graded mesh around a spherical cavity.

The gridpoints within the radial brick are now relocated to form the mesh around a spherical cavity. The *FISH* function `make_sphere` loops through all the gridpoints and remaps their coordinates using a linear interpolation along radial lines from the sphere origin to the gridpoints along the outer box sides. The next example shows the `make_sphere` *FISH* function, and the figure following shows the final grid.

FISH function to position gridpoints for a mesh around a spherical shape

```
fish define make_sphere
; Loop over all GPs and remap their coordinates:
; assume len>rad
loop foreach local gp gp.list
  local p = gp.pos(gp) ; Get gp coordinate: p
  local dist = math.mag(p)
  if dist>0 then
```

```

local k = rad/dist
local a = p * k ; Compute a=point on sphere radially "below" p.
local maxp = math.max(p->x,p->y,p->z)
k = len/maxp
local b = p * k ; Compute b=point on outer box boundary
                  ; radially "above" p.
local u = (maxp-rad)/(len-rad) ; Linear interpolation: P=A+u*(B-A)
gp.pos(gp) = a + (b-a)*u
end_if
end_loop
end
return

```

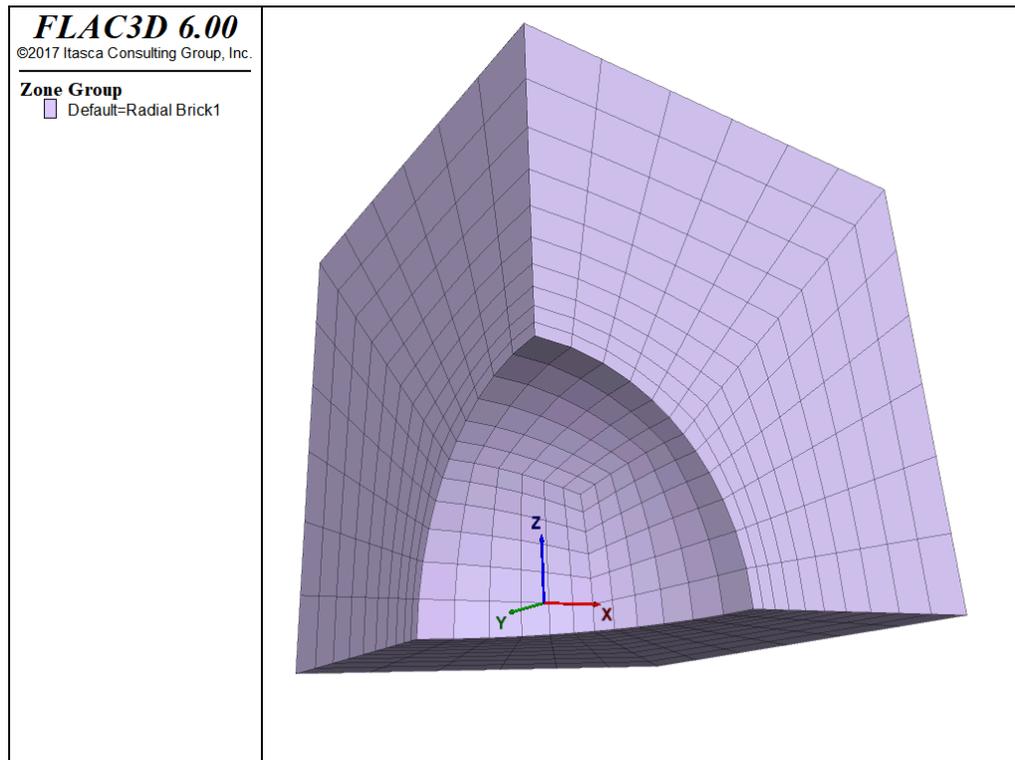


Figure 11: Radially graded mesh around a spherical cavity.

Extrusion-Based Grids

The **Extrusion Pane** provides a facility for creating grids based on two-dimensional geometries that are extruded (linearly extended) in the third dimension.

When working with an extrusion in the **Extrusion Pane**, all actions that change the state of the model are emitted as commands. These commands are echoed in the **Console Pane**, and they are recorded in the **State Record** pane. It is possible

and sometimes preferable to create zones entirely from the **Extrusion Pane** and start the next stage of modeling with a save file instead of creating a data file. The commands used to create the geometric model state are always stored with the save file. However, it is often advantageous to work with a data file instead, as this is a much smaller and more convenient way to recreate the geometry if necessary. The **State Record** pane may be used to export all the commands that have been used to create an extrusion to a data file. These commands are documented in the **Extrude Operations** section.

Use of the tools provided in the **Extrusion Pane** are described in detail in the **Extrusion Pane** topics found in the **FLAC3D Interface** section of this Help. The following material provides a narrative overview of the **Extrusion Pane** and how it is used to create grids.

Create a Set

To start an extrusion, a new set must be created in the **Extrusion Pane** (the pane, if not visible, is accessed via the main menu: Panes • Extrude). Multiple extrusion sets may be defined. The “Extrusions” selector on the toolbar is used to switch between defined sets. Only one set may be active at a time.

Define the 2D Geometry

There are two view modes in the *Extrusion* pane: the “Construction” view (), and the “Extrusion” view (). The former is used to define the 2D geometry. The latter is used to define the linear extrusion and is described in the next section below.

The 2D geometry is defined by drawing points and edges between points. Points may be defined by coordinates or may be defined “freehand.” Edges may be linear, curved, or arced. Curves may be arbitrarily shaped using control points. A curve may contain as many control points as needed. Further refinement is available along edges using a splitting tool.

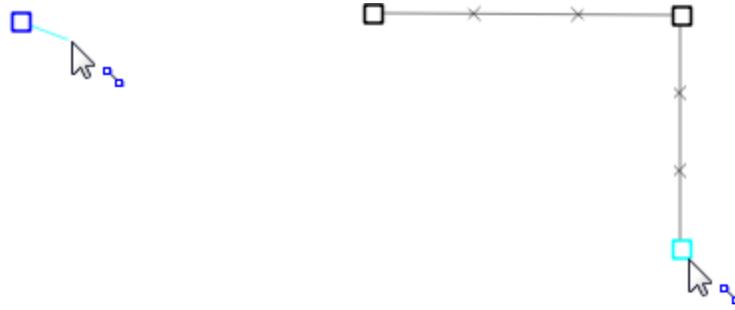


Figure 12: Points (left) and edges (right) are defined. “Exes” along the edges preview zoning. When an edge is drawn, it is initially assigned three zones by default.

When sets of edges are assembled into triangular or quadrilateral shapes, they may be defined as blocks. Once “blocked”, a given shape may be (re)zoned or grouped. When the extrusion is built into a grid in *FLAC3D*, only those parts of the extrusion set that have been blocked will create zones.

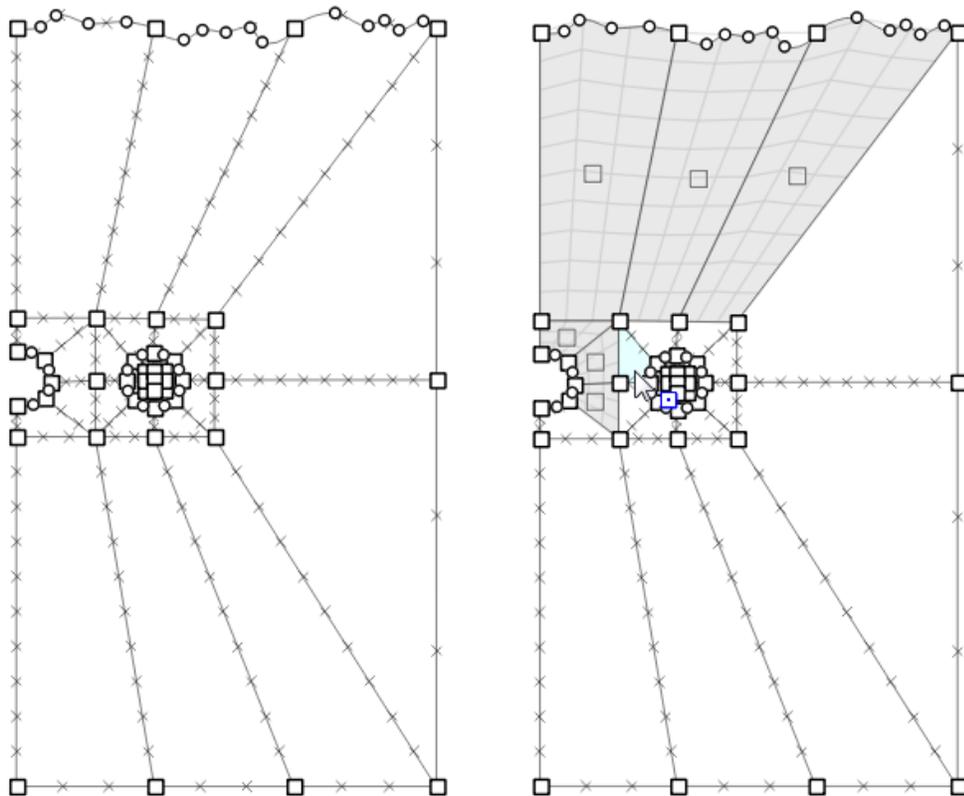


Figure 13: A set of points and edges are assembled into a quadrilateral or triangular shapes (left); they are then interactively turned into blocks (right).

Zoning and Other Operations

Zoning is set along edges by specifying the number of desired zones on that edge. Zone settings will propagate across the model as needed. A multiplier can be applied to blocks to increase zoning by an integer-factor.

In addition, there is an autozone facility available that can zone all blocks in the 2D geometry at once by defining a uniform zone length, a number of zones desired across the largest extent of the geometry, or a total number of zones within the geometry.

Groups: It is possible to select one or more points, edges, or blocks in the “Construction” view and assign them to a group and assign groups to segments and nodes in the “Extrusion” view. When this is done, any zone that is generated from the grouped points, edges, or blocks will be assigned to a zone group with the given name. Groups assigned in the “Construction” view will be in slot “Construction”, and groups assigned in the “Extrusion” view will be in slot “Extrusion”. In addition, the zones from every block will be automatically assigned a unique name in the “Block” slot, and every segment in the “Extrusion” view will be assigned a unique name in the “Segment” slot.

Splitting: The split tool can be used to split edges or blocks. When a split operation occurs, the split will propagate across all existing edges and blocks as needed.

Tracing: Bitmap or vector (DXF) images may be imported, sized, and oriented and used as “trace” guides for construction of the 2D geometry. In the latter case, a DXF will retain its “object-ness,” so it can be snapped-to when constructing points and edges for the extrusion.

Define the Extrusion

Once the 2D geometry is drawn and blocked, the extrusion is finished by switching to the “Extrusion” view to define the extent and the partitioning (blocking) of the extrusion. This is a 1D operation; the user assigns points to a horizontal line to indicate blocks, and then the segments of the line are used to define the zoning that will occur within the block represented by that segment.

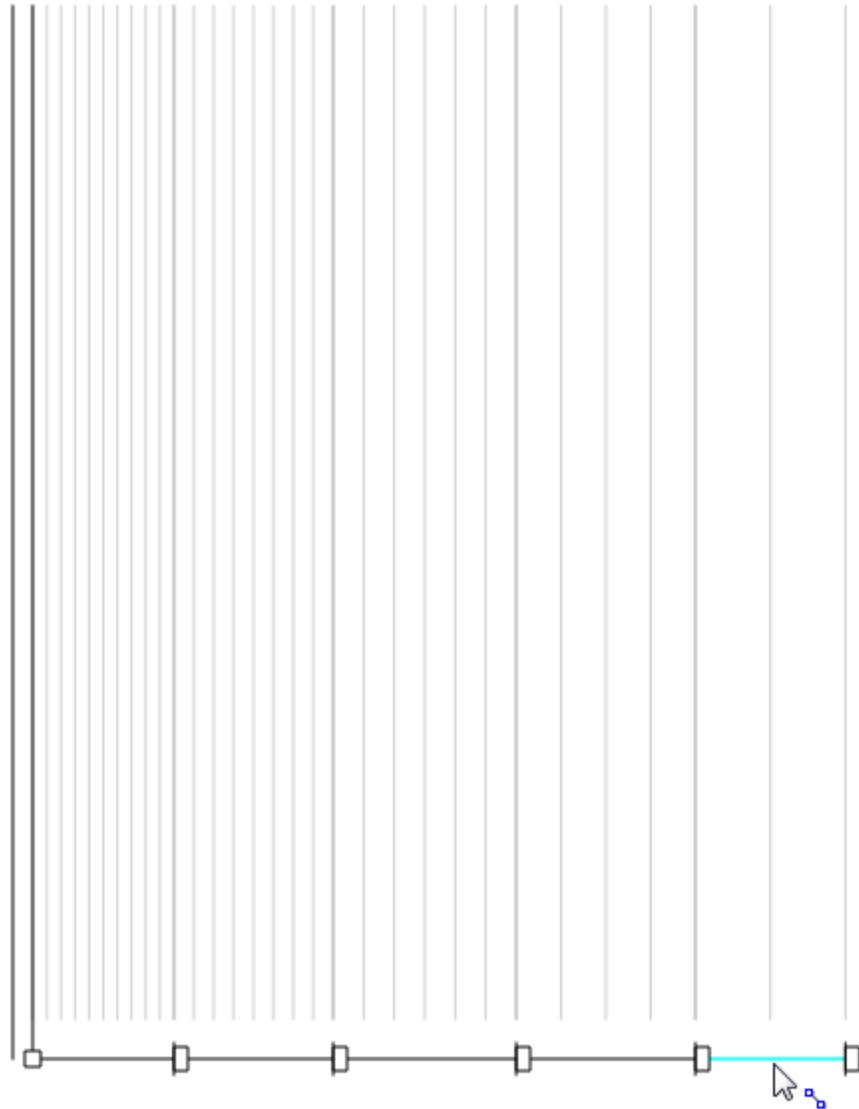


Figure 14: The extent of the extrusion and partitioning it into zoned blocks is performed in the “Extrusion” view.

When the third dimension is fully specified, zones are created from the extrusion set. Alternatively, the user may choose to export the extrusion set’s collection of 3D blocks to a new *Building Blocks* set in the **Building Blocks Pane** (see the **Building Blocks-Based Grids** topic next), where, presumably, further editing and refinement may occur prior to generating zones from the set.

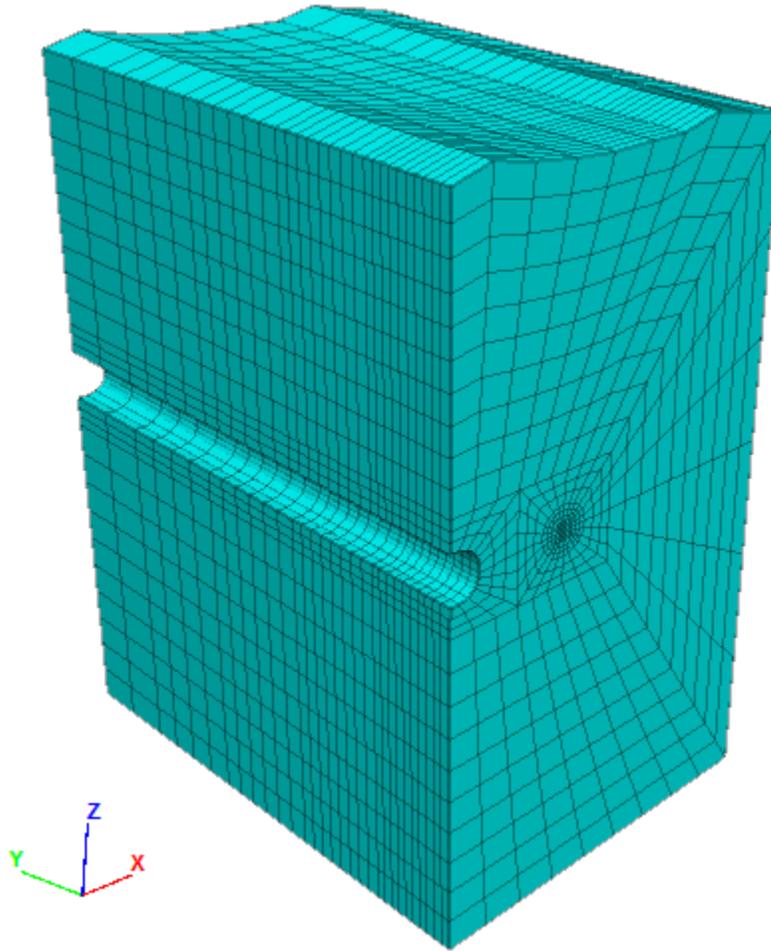


Figure 15: Zones created from the extrusion set.

Compare the image above to the final grid shown in the example of creating a grid for two tunnels with the same invert elevation. Observe that the resulting grids are comparable, but here a topography has been added to the top of the extrusion and the zoning has been graded in the direction of extrusion.

Building Blocks-Based Grids

The **Building Blocks Pane** provides a facility for creating grids based on three-dimensional “building blocks” that are fitted together and then zoned.

When working with an extrusion in the **Building Blocks Pane**, all actions that change the state of the model are emitted as commands. These commands are echoed in the **Console Pane**, and they are recorded in the **State Record pane**. It is possible and sometimes preferable to create zones entirely from the **Building**

Blocks Pane and start the next stage of modeling with a save file instead of creating a data file. The commands used to create the geometric model state are always stored with the save file. However, it is often advantageous to work with a data file instead, as this is a much smaller and more convenient way to recreate the geometry if necessary. The **State Record** pane may be used to export all the commands that have been used to create an extrusion to a data file. These commands are documented in the **Extrude Operations** section.

Use of the tools provided in the **Building Blocks Pane** are described in detail in the **Building Blocks Pane** topics found in the **FLAC3D Interface** section of this Help. The following material provides a narrative overview of the **Building Blocks Pane** and how it is used to create grids.

Create a Building Blocks Set

To start with building blocks, a new set must be created in the **Building Blocks Pane** (the pane, if not visible, is accessed via the main menu: Panes ▸ Building Blocks). Multiple sets may be defined. You can click anywhere in the blank area of the pane to create the initial set. The *Sets of Blocks* selector on the toolbar is used to switch between defined sets. Only one set may be active at a time.

Snap Blocks Together

On initiation of a new set, a single block is provided (assuming the box was checked in the previous dialog). The set may be developed by adding blocks using the *Add Blocks* tool (), which is done by selecting the face on an existing block to which a block should be added, then choosing the block-type to be added by rotating the mouse wheel. A quadrilateral face may have a hexahedral “brick” block type added to it, or a “wedge” type in one of two orientations. A triangular face may have a “wedge” or a “tetrahedron” block added to it.

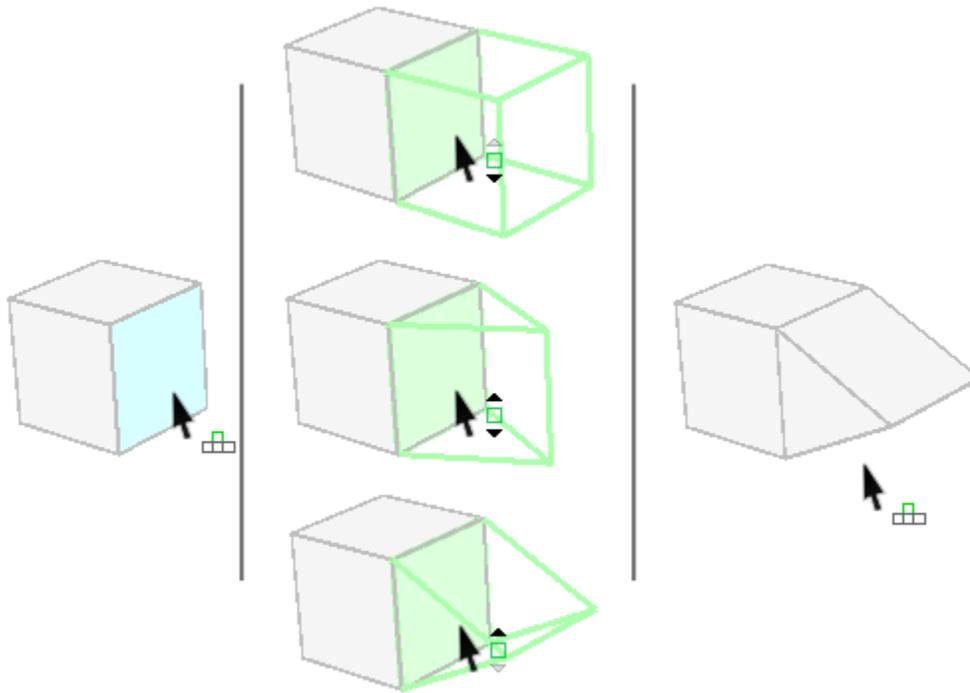


Figure 16: Left: use the “Add Blocks” tool to select a face on an existing block;
center: use the mouse wheel to scroll through the options of block type to add;
right: click on the desired “proto-block” to set that block in place.

Blocks are composed of points (vertices), edges, and faces. Manipulations on blocks may be performed on the whole block, or on its constituent parts. On creation, a block is given a default zoning of three zones per edge.

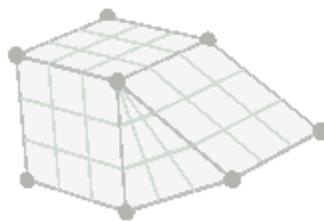


Figure 17: In base “selection” mode, the components of the block (faces, edges, points) are shown. Zoning of the block is indicated by the gray-filled grid shown on faces.

Blocks or their parts, individually or in like groups, may be moved, resized, rotated as needed and as sensible (for instance, a group of points cannot be resized as points do not have a size, though they may be moved or rotated; a single point, similarly, cannot be rotated). A screen control allows these manipulations to be performed by hand; alternatively, individual properties (position, rotation, etc.) may be specified on selected object(s) in the **Control Panel**.

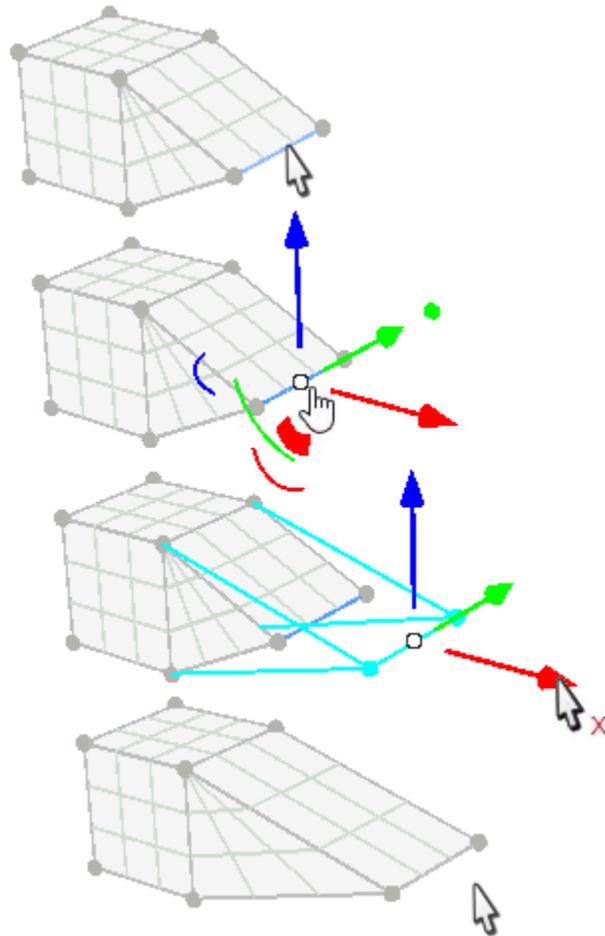


Figure 18: Selecting and moving an edge of a block.

Though there are many more facilities available in the **Building Blocks Pane** (described in the sections that follow below), at its simplest, an entire grid can be constructed following just the steps above. When the geometry shape is complete

and zoning is set as needed, pressing the “Generate Zones” button (`|zoneit|`) in the *Building Blocks* pane will cause the zones to be generated and rendered in the *Model* pane.

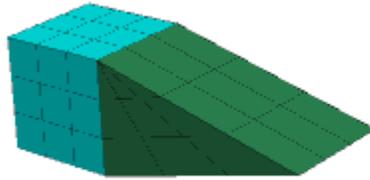


Figure 19: Zones generated from a building blocks set. Note that zones in each block from the building blocks set are automatically grouped together by block (as indicated by the coloring of the zones here).

Refinement Operations and Utilities

The simplistic illustrations above demonstrate the underlying concepts behind the *Building Blocks* facility. Additional tools and operations allow for significantly greater complexity to be achieved when constructing building-blocks-based grids, some of which include the following:

- Import of CAD data (DXF, STL, etc.) allows construction (tracing, as it were) of a building blocks set that is fitted to the existing 3D geometry.
 - CAD data are objects that support point-snapping, which facilitates shaping blocks to fit the imported geometry.
- Edges may be split; splits will propagate automatically across existing blocks.
- Edges may be curved to fit any shape with the addition of control points.
- Blocks may be hidden/shown as needed.
- Blocks may be copied and pasted.
- Unattached blocks may be snapped to each other by matching like faces.
- A group of selected blocks comprising a layer may be duplicated and assigned a varying thickness.
- Properties of objects may be set/alterd interactively or specified exactly through the **Control Panel**.
- The orientation of zones on a triangular face may be “cycled”.

- Like objects may be grouped together; zones derived from these objects will retain that group assignment on generation.
- Multiple forms of control are available for zoning (see the next topic below).
- A library of pre-built existing shapes and geometries is available.

As mentioned above, the present topic is only intended to offer a narrative overview of *Building Blocks* capabilities. See the [Building Blocks Pane](#) topics for details and instructions for using the capabilities listed above.

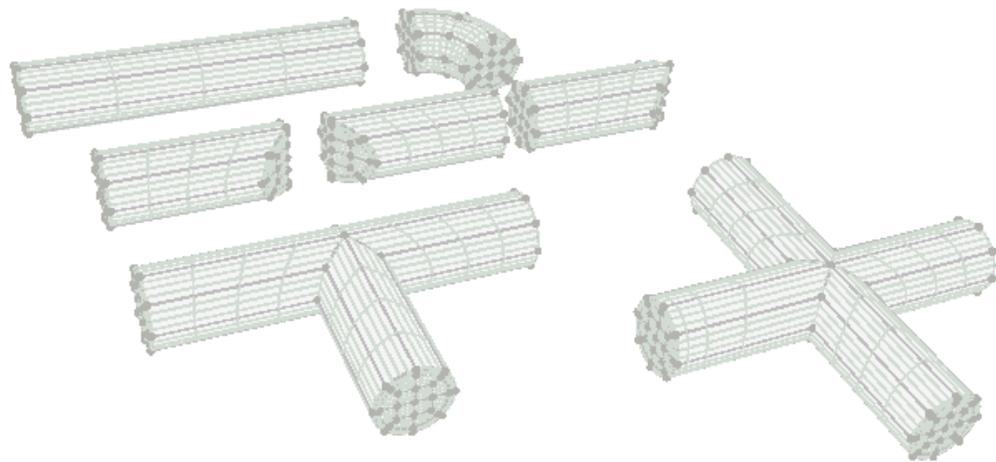


Figure 20: The “Cylinder Parts” blocks from the pre-built blocks library.

Zoning

Zoning is set along a block edge. A “Zones” attribute indicates how many zones should exist along the edge. A ratio or factor can be used to set finer sizing of zones toward the end of the edge where smaller zone sizes are needed. A block may be set to accept a multiplier, which will increase any existing zones within the block by the factor supplied. An auto-zoning facility allows for setting the zoning for an entire building-blocks set at once. Options for auto-zoning are: set a uniform zone size throughout; specify a number of zones to be used across the largest dimension of the model; or specify a total number of zones in the set.

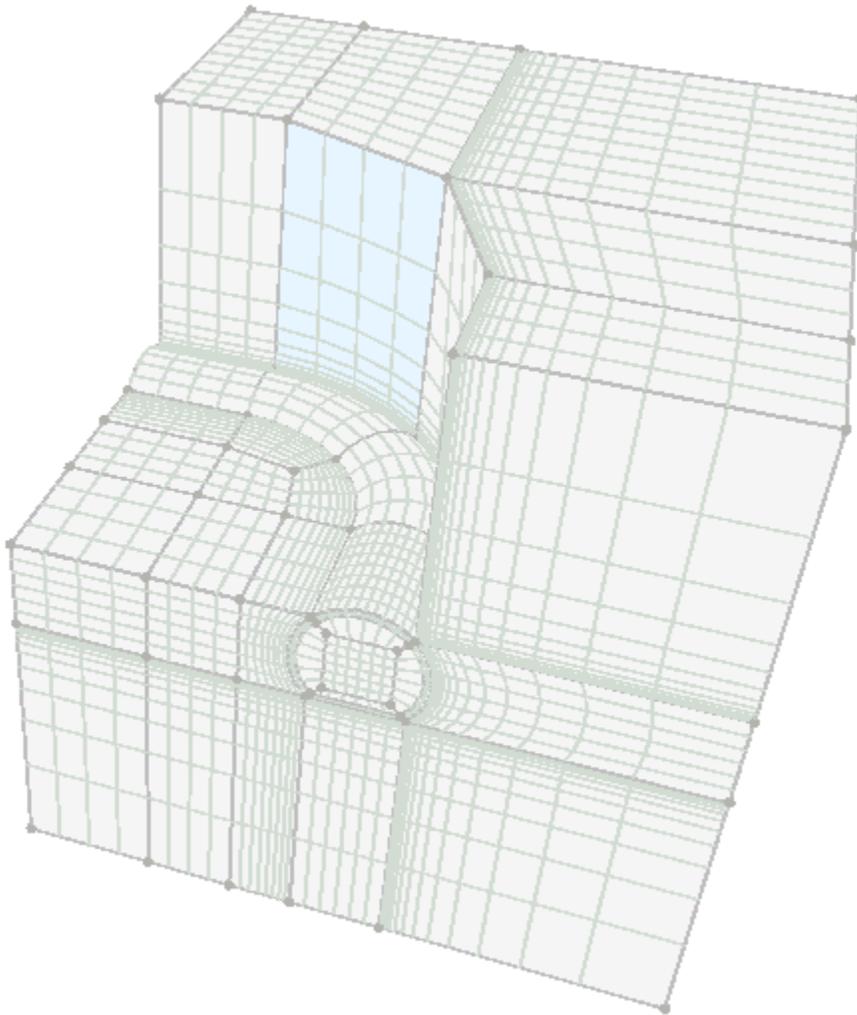


Figure 21: The “Cylinder Tunnel with Wall 90 Degree Turn” pre-built blocks. Ratios have been used to increase zone refinement around the tunnel. Blocks above the tunnel have been hidden to allow access to/visibility of the interior of the block set.

Grids from Outside *FLAC3D*

In addition to the built-in methods of mesh generation, meshing created by third-party applications outside *FLAC3D* can be imported. Currently, *FLAC3D* supports direct import of three different file formats:

- *FLAC3D* grid files (*.f3grid)
- ANSYS files (*.lis) (Note that files in this format normally come in pairs: a node file that is loaded first followed by an element file.)
- ABAQUS files (*.inp)

***FLAC3D* grid files is a format developed by Itasca, that has two primary benefits from the other formats:**

- It exports and preserves metadata, meaning group and extra variable assignments to zones, faces, and gridpoints.
- It has a binary version, resulting in smaller files that are faster to import.

These files can be imported into a *FLAC3D* model by using the `zone import` command, by using the File › Grid entry in the main menu, or in the specific case of *.f3grid files by selecting the file in the **Open into Project** dialog.

Once imported, the file should appear as an entry in the “Data Files” section of the **Project Pane** and will be added to any bundle made to archive or communicate the project.

Either the *ANSYS* or *ABAQUS* file formats (or both) are available as export options from almost all third-party meshing programs. If you have access to an advanced mesh generation, or just have a preference due to experience, you can continue to use these tools for creating *FLAC3D* models. Some examples include:

- CUBIT
- HyperMesh
- TrueGrid

Itasca offers a powerful mesh generation tool that works with the popular CAD program *Rhino*. This tool is called **Griddle**, and a brief overview of it can be found in **Grids from Rhino/Griddle**.

Grids from *Rhino/Griddle*

Rhinoceros (Rhino) is a NURBS-based curves and surface representation CAD package that can be used to create closed volumes composed of meshed surfaces in 3D.[1] *Griddle* is Itasca’s powerful automatic grid generator that creates all-tetrahedral or hex-dominant unstructured meshes for use in *FLAC3D* or *3DEC*. These tools are sold together but *separately* from *FLAC3D*. They represent the most advanced approach to creating complex meshes for *FLAC3D*. When the grid needed for a *FLAC3D* model exceeds the built-in capabilities provided with *FLAC3D*, these tools should be considered. This topic provides a brief introduction to *FLAC3D* users who may need to give such consideration.

The workflow for creating a complex mesh with *Rhino/Griddle* is straightforward. First, the model geometry is built or imported into *Rhino*. Next, *Rhino*'s surface meshing is applied. *Griddle* is then used to automatically remesh the surfaces to the desired density and type (triangles, quadrilaterals). The *Griddle* volume mesher is then called to fill the geometry with zones (all-tetrahedral or hex-dominant). The workflow is illustrated as follows.

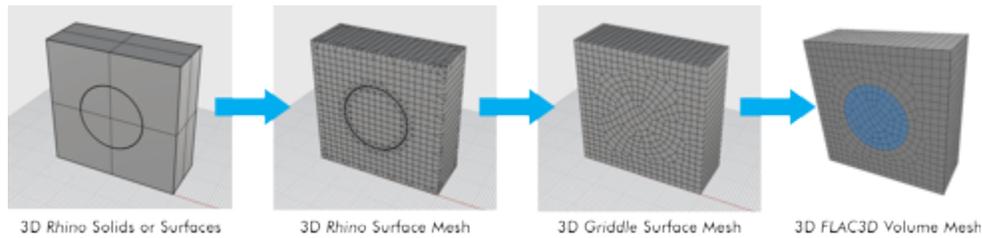
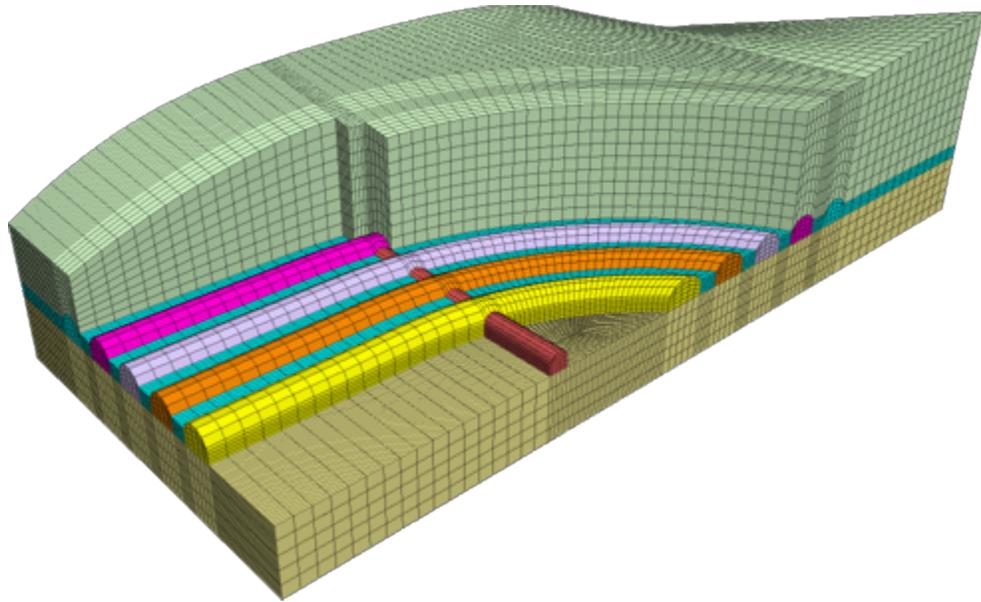


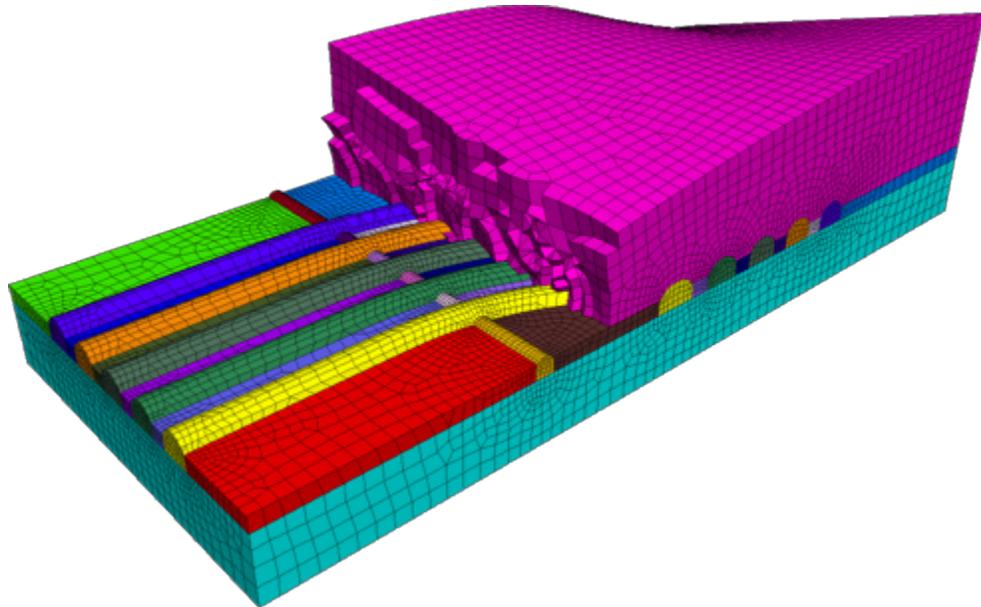
Figure 22: From left to right: an initial volume is described in *Rhino*; surfaces are given an initial mesh with *Rhino*'s own meshing tools; *Griddle* is used to remesh the surfaces according to user specifications; the remeshed surfaces are the basis for zoning the volume, which is then imported into *FLAC3D*.

All-hexahedral meshes are generally preferable for producing high-quality numeric solutions. However, the ability to produce hex-dominant meshes—that is, meshes composed mostly of hexahedral elements but also wedge, pyramid, and tetrahedral elements where necessary to conform to the geometry—with *Rhino/Griddle* can be a preferable trade-off when it radically reduces the challenges posed by grid complexity.

Consider the following two meshes. The first is all-hexahedral. In order to produce it, the base geometry has to be broken down (blocked) into hexahedral, tetrahedral, and prism solids. This step is time-consuming. Overall mesh generation time for this case was five hours.



The second mesh is hex-dominant. Using *Griddle*, just the bounding surfaces of the tunnels and the soil layers are used as input; decomposing the model to simple primitive shapes is not required. Consequently, the total mesh generation time for this mesh was less than one hour.



Endnote

[1] See https://en.wikipedia.org/wiki/Rhinoceros_3D.

Grid Generation: Additional Facilities

These additional mesh creation facilities have one thing in common: they rely on an existing mesh as a starting point for further modifications. Thus, they can be used in combination with any of the other forms of mesh generation.

The **Densify** operation subdivides zones into multiple zones, allowing the user to more finely discretize a coarse mesh in an area of particular interest.

By combining **Geometry-based filtering** with **Densification**, we can produce **Octree meshes**.

Finally, we can extrude new zones from an existing zone surface to an irregular geometric surface. This can be used to create complex layering (as long as the layers do not intersect) or irregular surface topography. See **Surface Topography and Layering**.

Densifying Grids

We can generate a simple grid and then densify the grid using the `zone densify` command. New zones and faces created in this manner copy group assignments and extra variable values held by the original, but lose all material model, stress, and other state information.

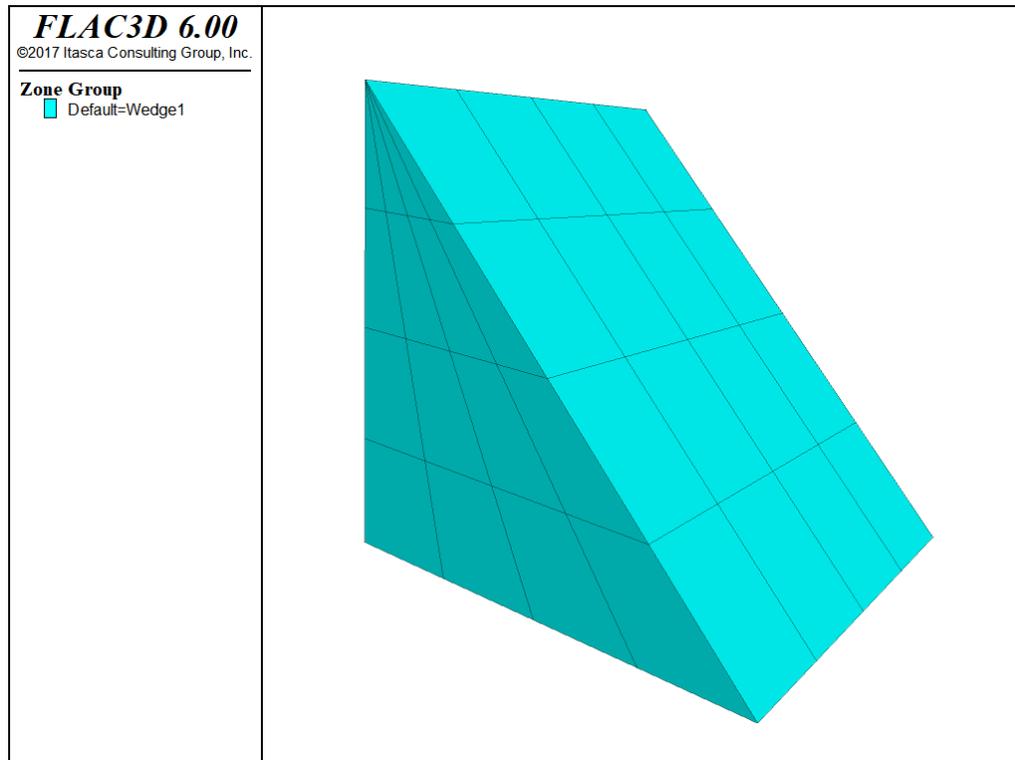


Figure 25: Original grid.

The following example shows commands to densify a simple grid by dividing two segments on each edge of existing zones. The above image shows the original grid; the figure following the example shows the densified grid.

Densify a grid by dividing at each edge of the original zones

```
model new
zone create wedge size 4 4 4
plot 'Wedge' export bitmap filename 'densify1.png'
;
zone densify segments 2
plot 'Wedge' export bitmap filename 'densify2.png'
```

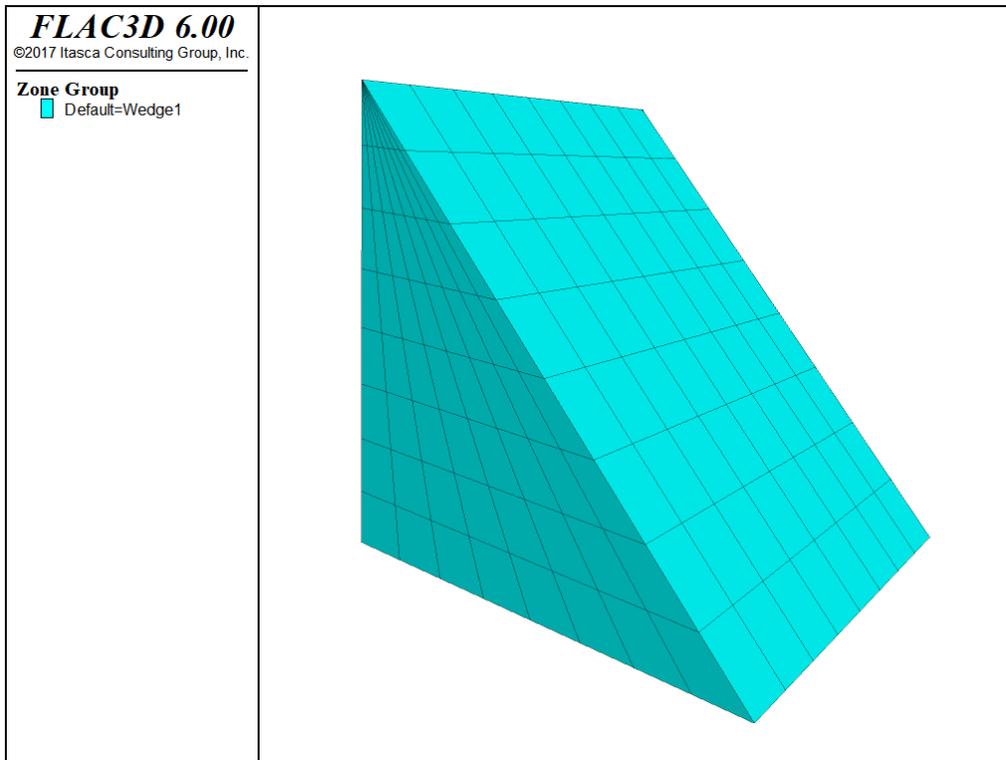


Figure 26: Densified grid by dividing segments on each edge of zones.

In addition to dividing the segments, we can also specify the maximum length of the refining zones by using the `maximum-length` keyword. No matter how `maximum-length` or `segments` is used, *FLAC3D* always densifies the grid by dividing the zone edges with a segment number of an integer. The next example shows how to densify a grid in the range of z -coordinates between 2 and 4 by setting the maximum length to be 0.5, 0.5, and 0.4 in the local x -, y -, and z -directions of the original zones.

Densify a grid by specifying the maximum size length

```

model new
zone create brick size 4 4 4
plot 'Brick' export bitmap filename 'densify3.png'
;
zone densify local maximum-length (0.5,0.5,0.4) range position-z 2 4
zone attach by-face
;
plot 'Brick' export bitmap filename 'densify4.png'

```

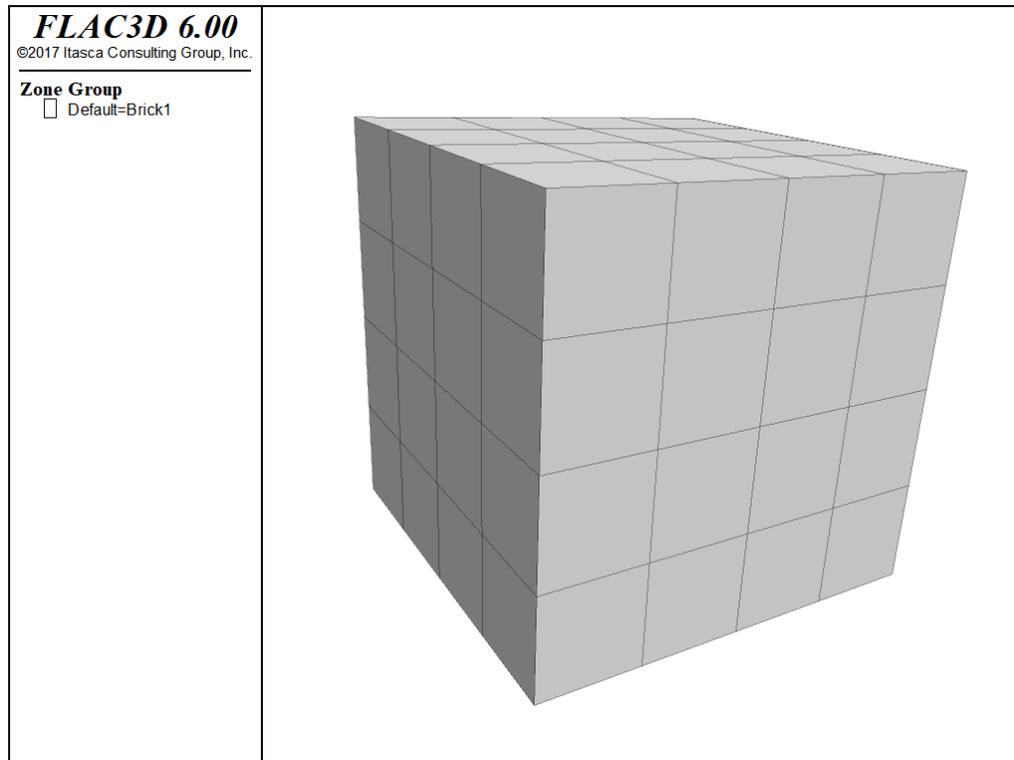


Figure 27: Original grid.

Please note that in the local z -direction, the maximum size length is 0.4. *FLAC3D* densifies the maximum length in this direction to be $1/3$ ($= 0.4$), so that the corresponding dividing segment (3, in this case) is an integer no less than the ratio of the original length (1.0, in this case) to the maximum length (0.4, in this case). The `zone attach by-face` command in this example is used to attach faces of sub-grids together rigidly to form a single grid. Always use the `zone attach by-face` command after the `zone densify` command if there are different numbers of gridpoints along faces of different zones. The image above is the original grid for this example; the image below is the densified grid.

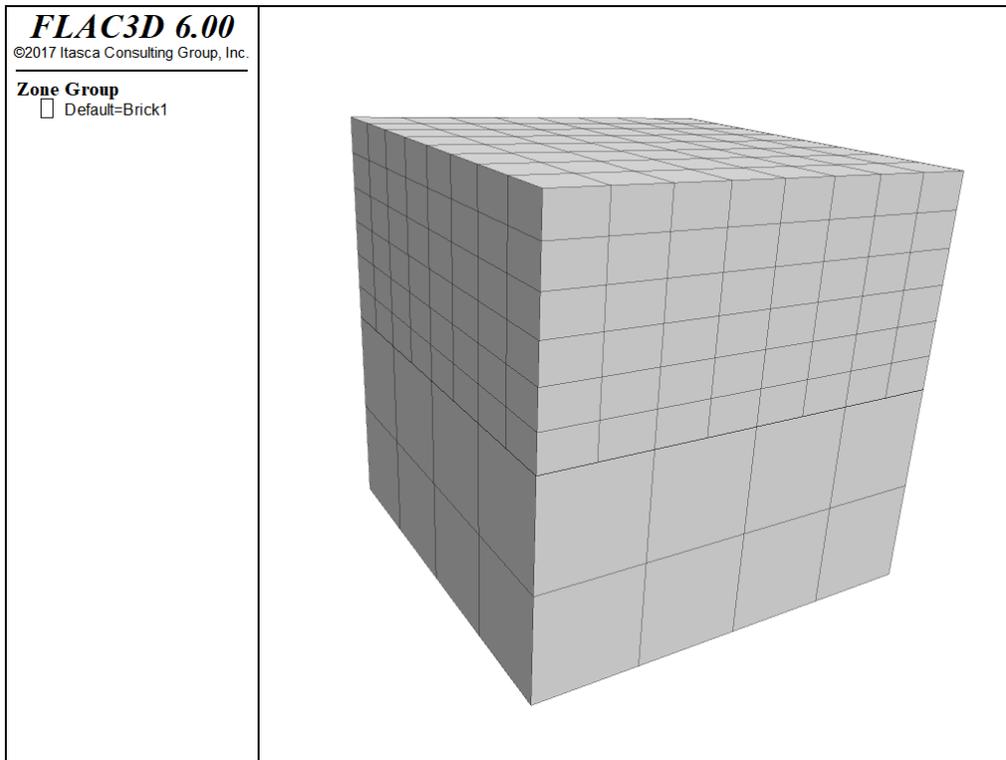


Figure 28: Densified grid by setting maximum lengths.

Sometimes we need to densify the grid in a space defined by geometric data contained in a geometric set. The next example presents a method to densify a space between two geometric sets. See [Working with Geometric Data](#) for a general overview and description of using geometric data to identify objects in the model.

In this example, each geometric set is defined by two polygons. The `zone densify` command uses a `geometry range` element that selects any zones with a centroid such that a ray with direction of $(0,0,1)$ will intersect once with any of the polygons making up `setA` or `setB`. See the `range geometry-space` documentation for details.

The `zone attach by-face` command here is used to attach faces of sub-grids together rigidly to form a single grid for all zones. The next two figures plot the original and densified grids.

Densify a grid using geometric information

```

model new
zone create brick size 10 10 10
;
geometry set "setA" polygon create ...
by-positions (0,0,1) ( 5,0, 1) ( 5,10, 1) (0,10,1)
geometry set "setA" polygon create ...
by-positions (5,0,1) (10,0, 5) (10,10, 5) (5,10,1)
geometry set "setB" polygon create ...
by-positions (0,0,5) ( 5,0, 5) ( 5,10, 5) (0,10,5)
geometry set "setB" polygon create ...
by-positions (5,0,5) (10,0,10) (10,10,10) (5,10,5)
plot 'Brick2' export bitmap filename 'densify5.png'

zone densify segments 2 range geometry-space "setA" set "setB" count 1
zone attach by-face
;
plot 'Brick2' export bitmap filename 'densify6.png'

```

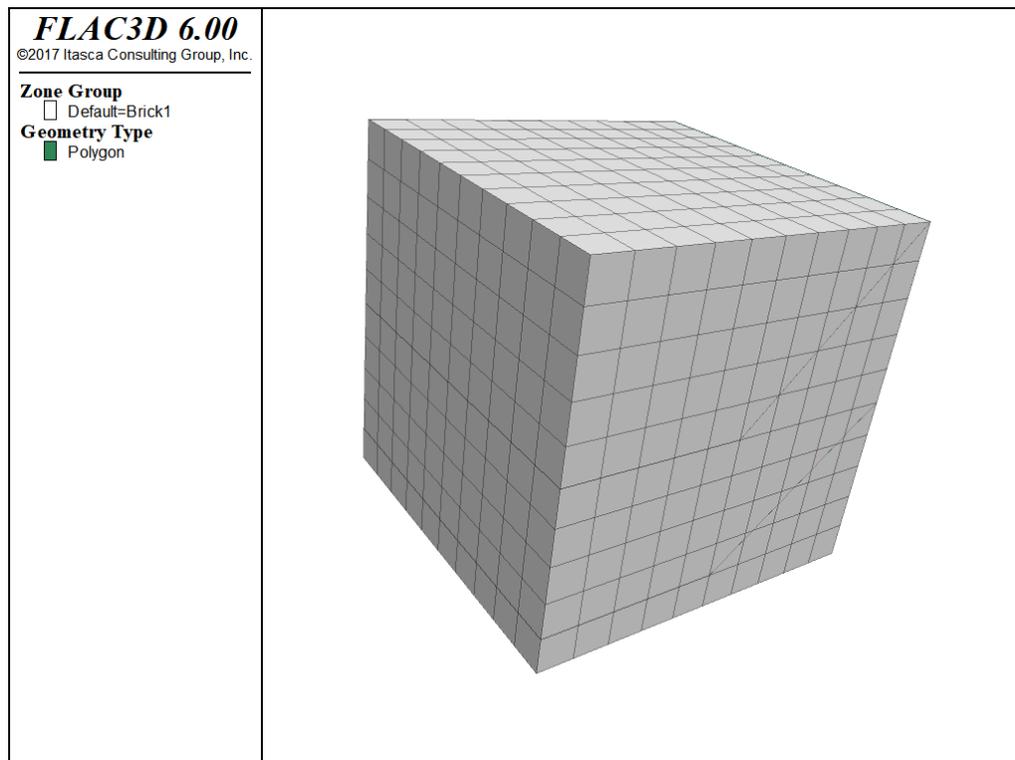


Figure 29: Original grid.

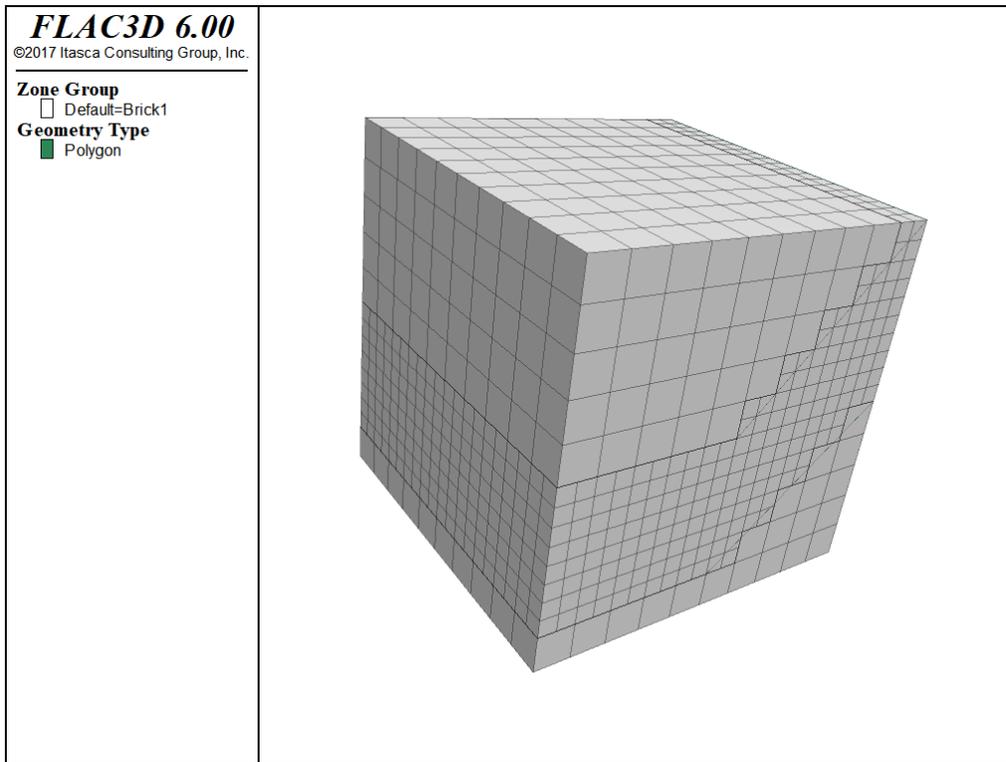


Figure 30: Densified grid using geometric data.

Geometry-Based Densification: Octree Meshing

Sometimes we need to densify the grid in a space defined by geometric data contained in a geometric set. This is often used to approximate material property changes on a very irregular boundary, when exact conformation to the surface is not important physically. Often orebody and other kinds of geologic structures fall into the category.

The command `zone densify` can be used to subdivide zones selected by their proximity to a geometry surface. Using it in combination with the `range geometry-distance` range element and using the `repeat` keyword allows the creation of an octree mesh in a single command.

For example, the command

```
zone densify segments 2 gradient-limit maximum-length 0.05 ...
  repeat range geometry-distance 'intcylinder' gap 0.0 extent
```

can be used to generate an octree mesh based on proximity to the “intcylinder” geometry set.

It is important to be clear about what each of the keywords invoked above is doing.

- The `segments` keyword indicates that each level of densification will subdivide a zone by 2 in each direction, resulting in eight new zones for every original.
- The `gradient-limit` keyword affects the zones tagged for densification. It ensures that the maximum difference in zone size from one zone to the next is one level of densification.
- The `maximum-length` keyword, in combination with the `repeat` described below, indicates that zones will be tagged for densification if they have an edge length longer than 0.05.
- The `repeat` keyword indicates that a single densification pass will be made over the zones, then if any zones were densified, a *new* list of zones will be selected for densification. This operation will be repeated until no zones are left to be densified, either because they do not fall within the range or because they are already smaller than the `maximum-length` specified.
- The `range geometry-distance` selects zones that fall within a `gap` distance from the `intcylinder` geometry set. Since the `gap` is zero in this case, it selects zones that actually intersect the surface.
- The `extent` keyword indicates that the distance from the surface should be judged by the *Cartesian extent* of the zone, rather than just the zone centroid. If this was not used, a non-zero `gap` would be required for there to be any chance of a zone falling within the range.

The results of this command are shown here.

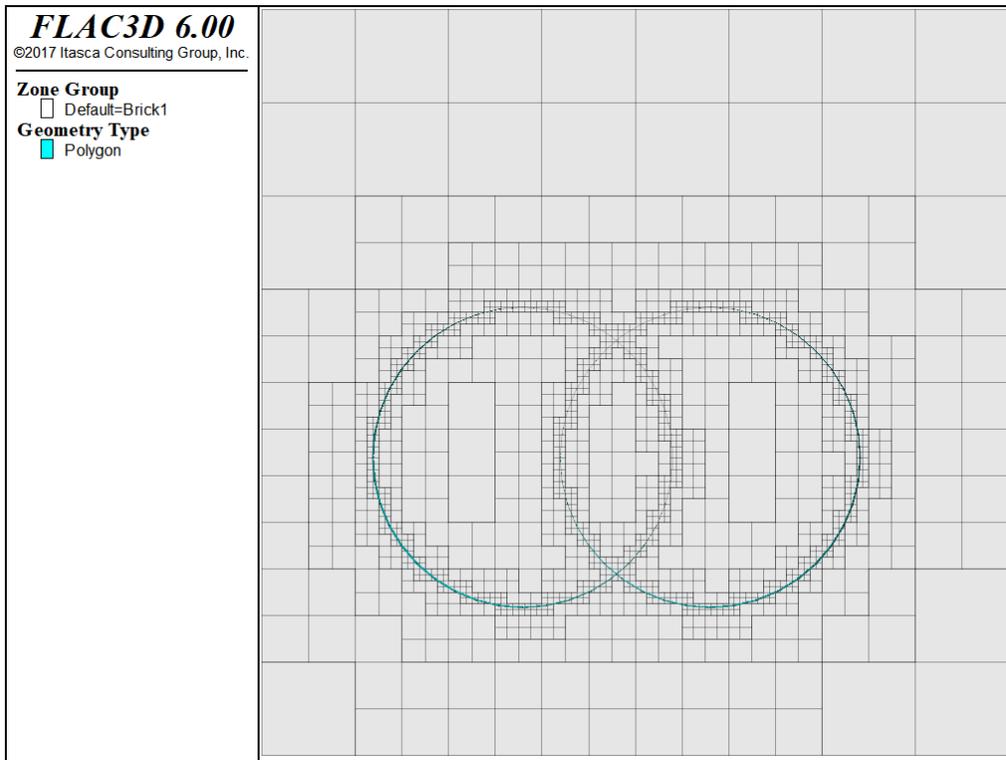


Figure 31: Octree mesh generated with *zone densify*.

Remember that this mesh should not be run until the `zone attach by-face` command is used to attach hanging nodes to adjacent faces. Also note that large gradients in zone size can increase the time it takes to reach equilibrium and change the path the model takes to get there.

For a more realistic example, the following data file takes a simple rectangular mesh and uses it as a starting point to create an octree mesh of a complex orebody.

```
model new
zone create brick size (10,15,10) point 0 (25000,20000,0) ...
                                point 1 (35000,20000,0) ...
                                point 2 (25000,35000,0) ...
                                point 3 (25000,20000,10000)
geometry import 'orebody.stl'
zone densify segments 2 gradient-limit maximum-length 50 repeat ...
                range geometry-distance 'orebody' gap 0.0 extent
zone group 'orebody' range geometry-space 'orebody' count odd
```

The result of that operation is shown in the figure below. The last `zone group` command uses the `range geometry-space range` element to select zones “inside” the geometric surface.

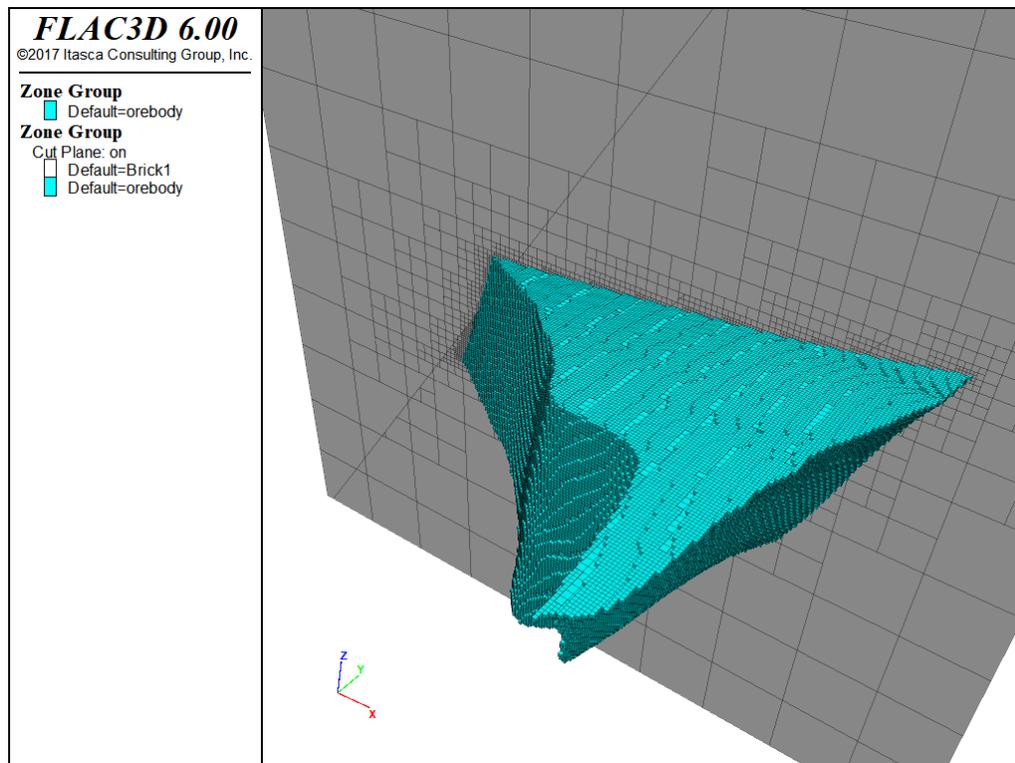


Figure 32: Octree mesh generated with `zone densify`.

Surface Topography and Layering

FLAC3D provides a `zone generate from-topography` command to generate a grid between the surface faces of an existing grid and a geometry set that characterizes the topography. The zones are created by extruding from faces in a specified range along a ray direction to a specified geometry set. This can be used to create a series of irregular non-intersecting layers, or most commonly, to add a layer of zones to the top that conforms to an irregular topographic surface. Some simple examples of its use follows.

To create a grid that conforms to a topographic surface, a `geometry set` must be specified. The easiest way to create a geometry set is via the `geometry import` command. **Additional commands** are available for manipulation of edges, polygons, and nodes, as well as other operations on geometric sets in their entirety.

An initial grid must exist before using the `zone generate from-topography` command. Only the zone faces satisfying all of the following criteria will be selected:

- The faces must be in the range.
- The faces must be on the existing grid surface, and any interior faces will be neglected.
- The outward normal vector of the faces must make an angle with the ray of less than 89.427 degrees. This will exclude the faces parallel to the ray (with a crossing angle of 90 degrees) or the faces facing a relatively opposite direction of the ray (with a crossing angle greater than 90 but not greater than 180 degrees).

The data file in the next example first imports geometry data from the STL format file `surface1.stl` and places it into a geometry set called `surface1`. (By default, the geometry set name is the same as the file name just imported.) The data file then creates a background grid with a group name `Layer1`. The geometry set and the background grid are shown in the next image below. It can be seen that if projected to the x - y plane, the geometry set will fully cover the background grid, so any node in a valid face in the range will have an intersection point with the geometry set if extruding along the ray direction.

Create a grid and a geometric set

```
model new
geometry import 'surface1.stl'
zone create brick size 50 40 5 ...
    point 0 (0, 0, -4000) point 1 (15000,0, -4000) ...
    point 2 (0,12000, -4000) point 3 ( 0,0, -2000) ...
group 'Layer1'
```

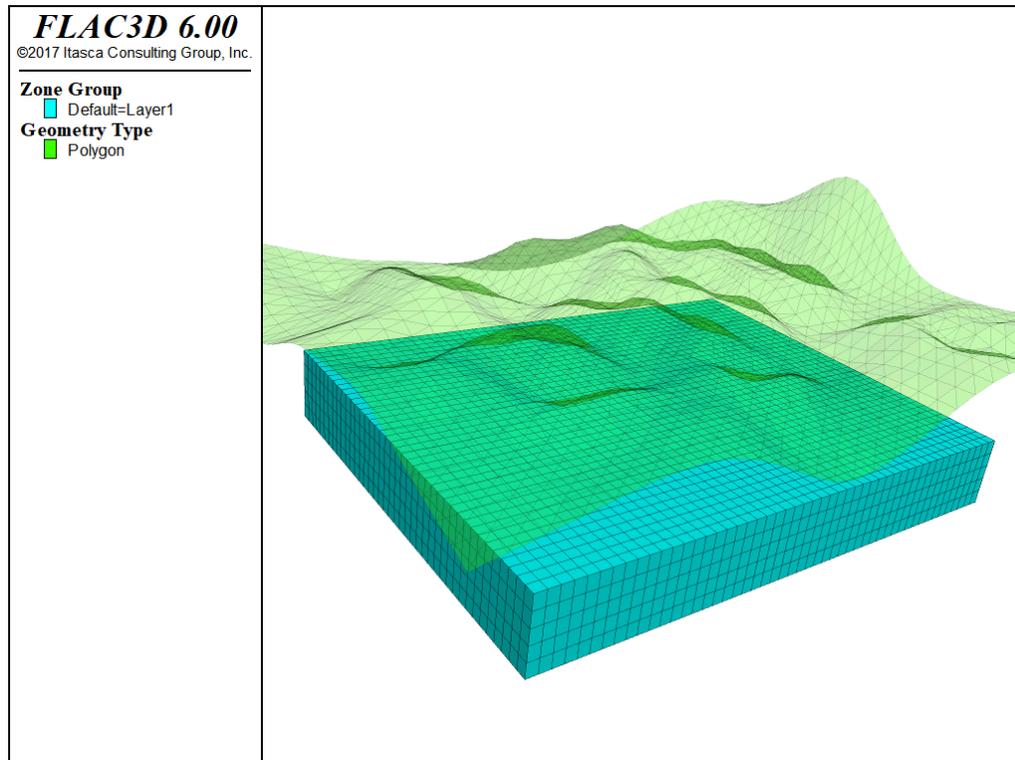


Figure 33: The existing grid and the geometry set.

Next the `zone generate from-topography` command is added. In the command, the `surface1` geometry set is specified. No ray direction is specified, so the default direction $(0, 0, 1)$ is used by default. The `segment` number is 8, so 8 zones will be created between the selected faces and the geometric surface. The newly created zones will be assigned the group name `Layer2` in slot `Default`. No range is assigned to the faces in this command, so all faces on the existing grid (group `Layer1`) surfaces are subjected to extrusion to the geometry set. However, the zone faces on the four side surfaces whose normal vector is vertical to the ray direction $(0, 0, 1)$, and the faces on the bottom whose normal vector is in the opposite direction of the ray (with a crossing angle of 180 degrees), are neglected in this example, so only the faces on the top surface of group `Layer1` will actually be used to create the new grid. The completed grid is shown in the next figure. The topography of the geometry set is correctly featured in the grid.

Add surface topography zones

```
model new
geometry import 'surface1.stl'
zone create brick size 50 40 5 ...
```

```

point 0 (0, 0,-4000) point 1 (15000,0,-4000) ...
point 2 (0,12000,-4000) point 3 ( 0,0,-2000) ...
group 'Layer1'
zone generate from-topography geometry-set 'surface1' ...
segments 8 group 'Layer2'

```

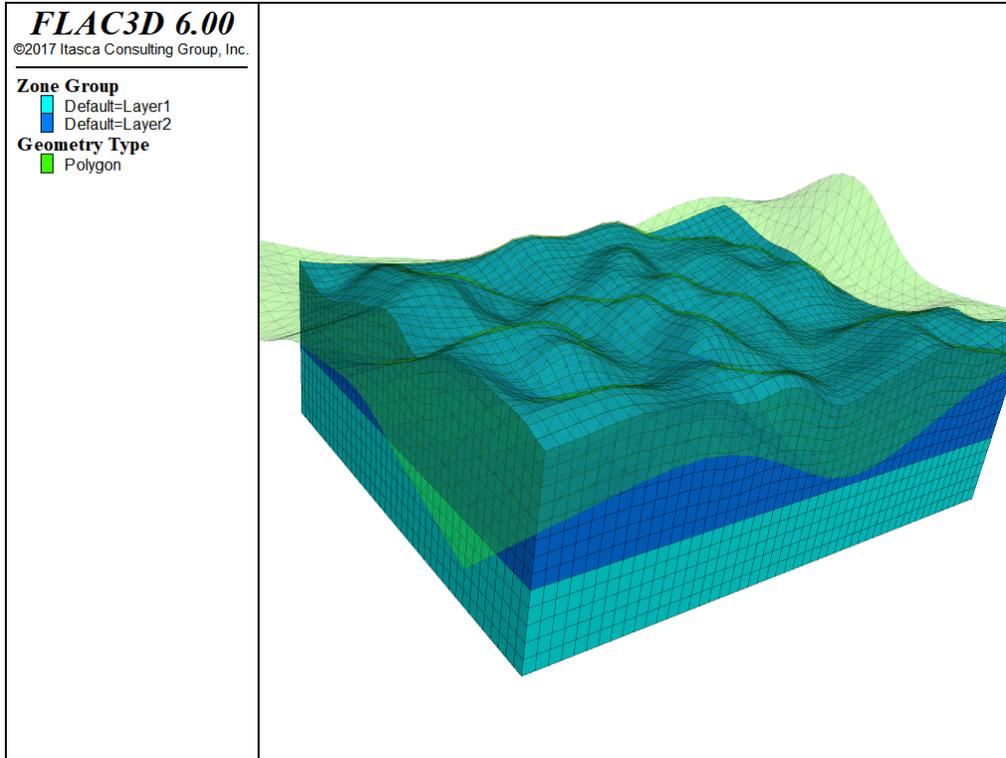


Figure 34: Grid created by the `zone generate from-topography` command.

The next example adds two new keywords to the `zone generate from-topography` command. The first is `ratio`, which is set to `0.6`. This changes the distribution of zone sizes as you approach the surface; in this case each zone is `0.6` time the size of the next. The second is `face-group`, which assigns a group name to the new zone surface created for ease of reference later.

Grade zones toward surface

```

model new
geometry import 'surface1.stl'
zone create brick size 50 40 5 ...
point 0 (0, 0,-4000) point 1 (15000,0,-4000) ...
point 2 (0,12000,-4000) point 3 ( 0,0,-2000) ...
group 'Layer1'

```

```

zone generate from-topography geometry-set 'surface1' ...
                        segments 8 ratio 0.6 group 'Layer2' ...
                        face-group 'Surface'

```

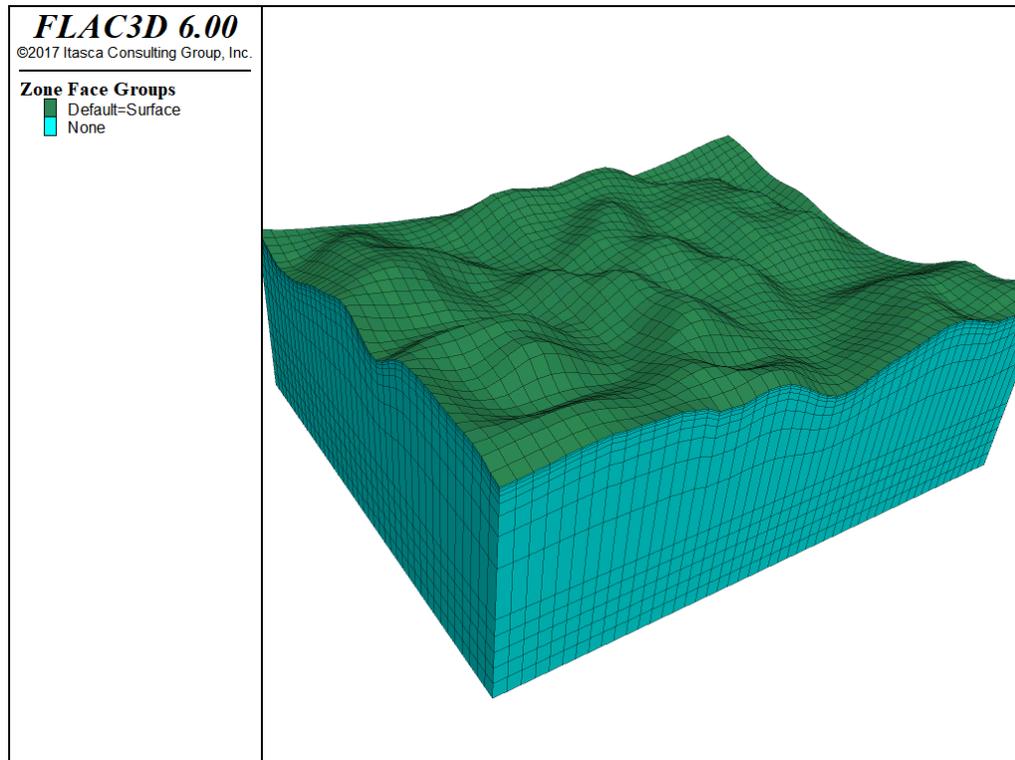


Figure 35: Grid created by the `zone generate from-topography` command (ratio=0.6).

In the next example, a range is applied to the faces on the background grid. This range is limited to only the faces whose centroid coordinates fall in the ranges $0 \leq x \leq 5000$ and $0 \leq y \leq 50000$. It should be emphasized that the range in `zone generate from-topography` is selecting **zone faces**.

Limit area of surface

```

model new
geometry import 'surface1.stl'
zone create brick size 50 40 5 ...
    point 0 (0, 0, -4000) point 1 (15000, 0, -4000) ...
    point 2 (0, 12000, -4000) point 3 (0, 0, -2000) ...
    group 'Layer1'
zone generate from-topography geometry-set 'surface1' ...
                        segments 8 ratio 0.6 group 'Layer2' ...
                        range position-x 0 5000 position-y 0 5000

```

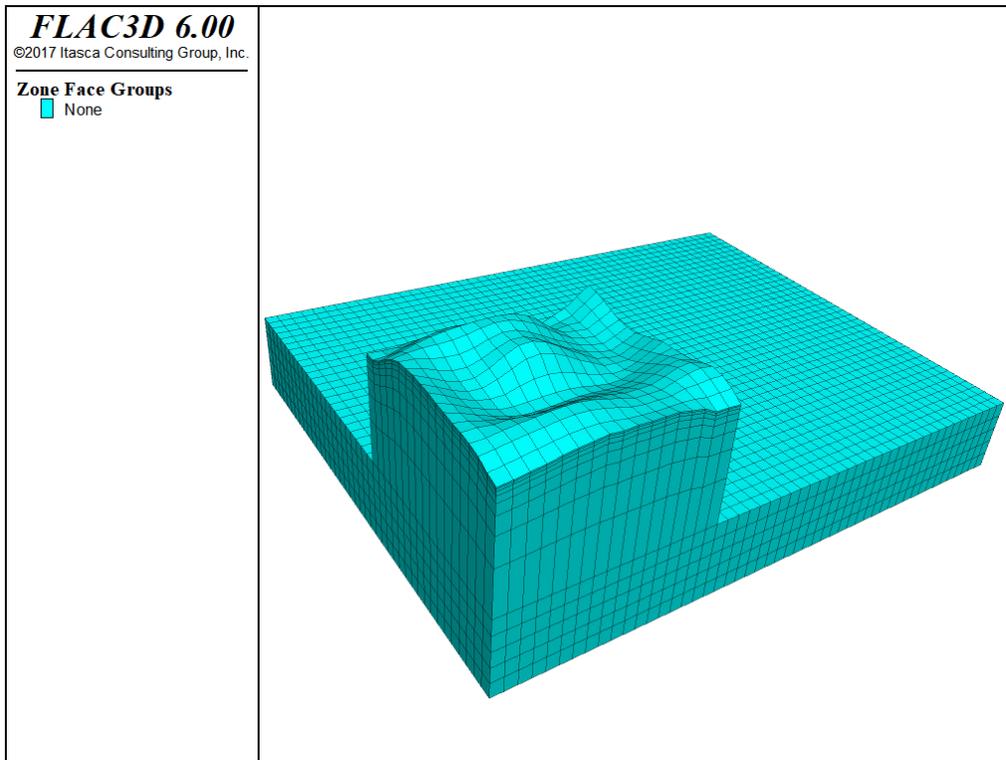


Figure 36: Grid created by the `zone generate from-topography` command (with a range applied to the faces).

In some cases, the geometric surface may not cover the projected area of the selected faces in the ray direction. To deal with this case, *FLAC3D* will first find a point A in the geometry set that is closest to the ray that starts at the relevant gridpoint, and then find a “virtual” intersection point B so that points A and B have the same distance to the projected plane. The completed grid is shown in the second figure below.

Surface that does not cover zone extent

```

model new
geometry import 'surface1.stl'
zone create brick size 50 40 5 ...
    point 0 (-10000,-10000,-4000) ...
    point 1 ( 15000,-10000,-4000) ...
    point 2 (-10000, 12000,-4000) ...
    point 3 (-10000,-10000,-2000) ...
    group 'Layer1'
zone generate from-topography geometry-set 'surface1' ...
    segments 8 ratio 0.6 group 'Layer2'

```

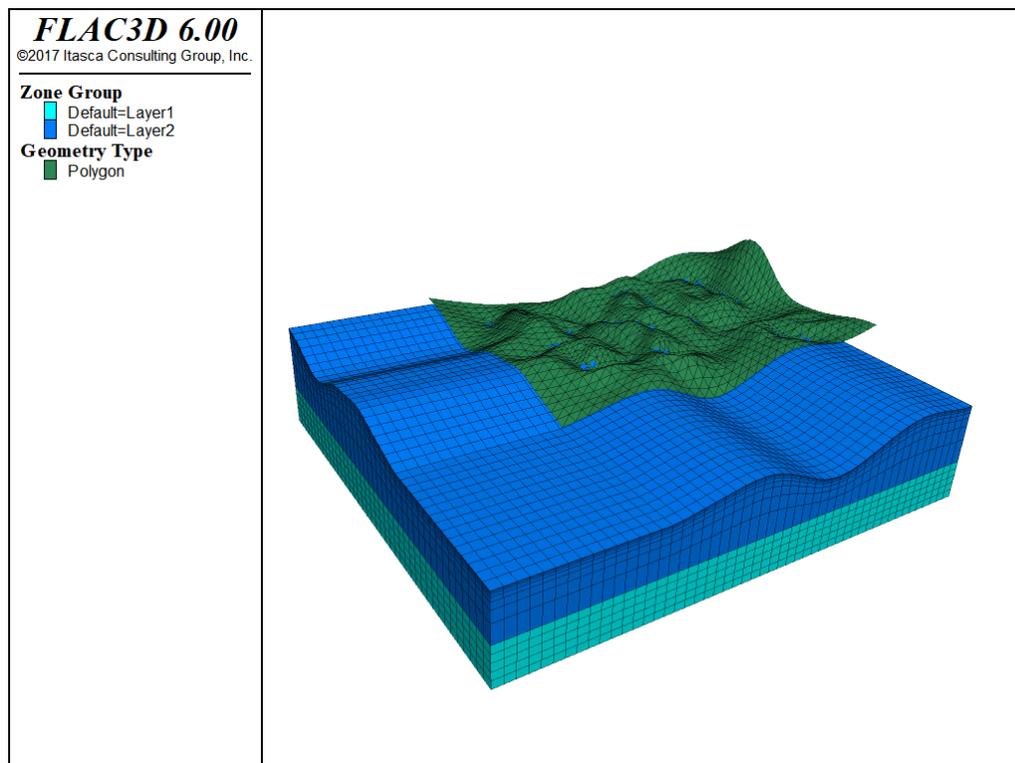


Figure 37: Grid created by the `zone generate from-topography` command (the geometry set does not fully cover the background grid if projected into a plane perpendicular to the ray).

The `zone generate from-topography` command can be used multiple times to create a grid with multiple layers. Each time the `zone generate from-topography` command is employed, the most updated grid will be regarded as the background grid. The `zone-face` keyword can also be used to mark the new zone surface faces in order to use them in the next layer.

In the next example, the second geometry set (called `surface2`) is created with a simple `geometry set` command. Then a quadrilateral polygon is created in it using `geometry polygon create`. Next, a second `zone generate from-topography` command is issued to fill the space between the irregular surface (labeled `Surface`) and the polygon in set `surface2`.

Note that the faces labeled `Surface` could subsequently be used to generate an interface, if desired.

Create a grid with multiple geometric sets

```

model new
geometry import 'surface1.stl'
zone create brick size 50 40 5 ...
      point 0 (0, 0, -4000) point 1 (15000,0,-4000) ...
      point 2 (0,12000,-4000) point 3 ( 0,0,-2000) ...
      group 'Layer1'
zone generate from-topography geometry-set 'surface1' ...
      segments 8 ratio 0.8 group 'Layer2' ...
      face-group 'Surface'

; Create the next layer
geometry select 'surface2'
geometry polygon create by-positions (0,0,3000) (15000,0,3000) ...
      (15000,12000,3000) (0,12000,3000)
zone generate from-topography geometry-set 'surface2' ...
      segments 8 ratio 1.2 group 'Layer3' ...
      range group 'Surface'

```

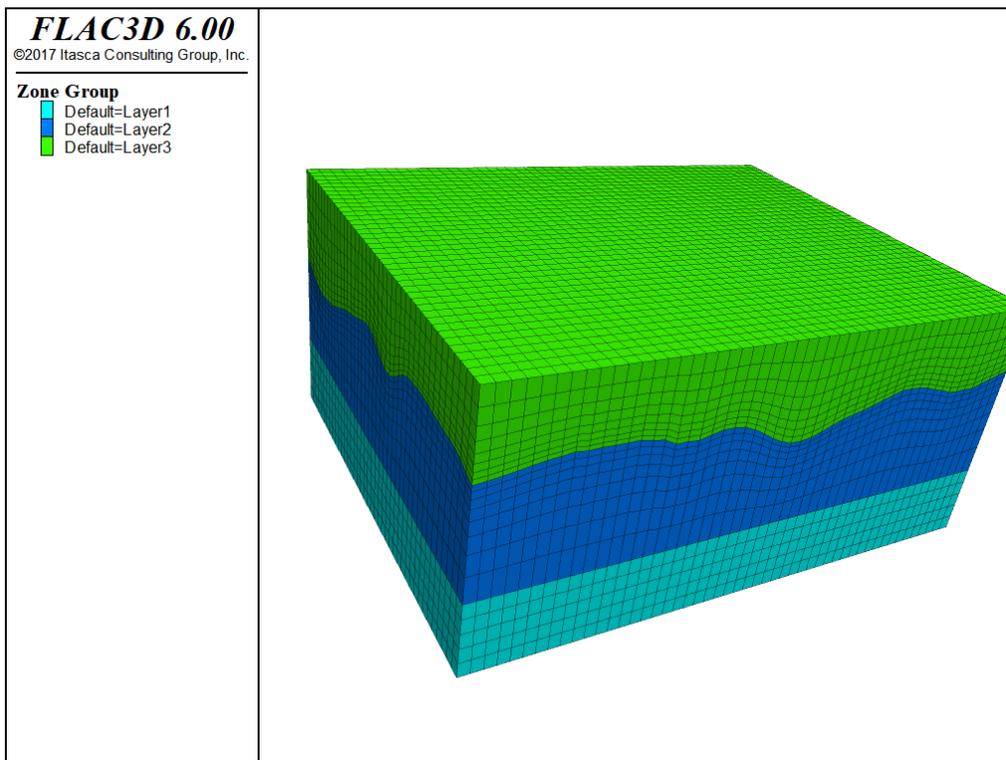


Figure 38: Grid created by multiple *zone generate from-topography* commands.

Identifying Regions of the Model

Once a mesh has been created (see [Grid Generation](#)), the next problem to confront a user is how to identify the zones and zone faces that make up areas of interest in the model. In a complex three-dimensional model, this can be particularly troublesome. In point of fact, in the past, a great deal of the difficulty perceived in creating a data file was simply the problem of how to specify the zones or faces desired for a given command.

Some of this will generally be done during grid generation—all of the built-in methods of grid generation allow the user to assign names to regions that will be carried over in to the zones generated within those regions. Most third-party tools have similar capabilities, and *Griddle* does as well.

If these labels exist, however, they are often automatically generated. They need to be collected into identifiable regions that relate to specific model features. Examples include areas where boundary conditions will be applied, areas where interfaces will be created, regions of different material properties, or regions to be excavated.

FLAC3D 6 contains both old and new features designed to make it easier than ever to identify regions of the model for later reference. Once this is done, the commands required to construct the rest of the model often become almost trivial. Some of these features are described in this section.

Groups and Slots

The primary means for identifying objects (zones, structural elements, fractures, geometry polygons, etc.) in *FLAC3D* is by *Groups* and *Slots*. The use of both terms is due to historical legacy, and they are perhaps not immediately descriptive. Think of an object (like a zone) as having a number of different locations on its surface on which one might place a label. Only one label will fit in a given location. The object might, however, have multiple locations available, each with a space for a different label. In *FLAC3D*, the label is a *Group Name* and the location is the *Slot*. Any number of group names may be specified, and up to 128 different slot names may be used in a given model.

Recall that a **Range** is the primary way commands use to identify which particular objects they should operate on. `range group` is the most common range element used in *FLAC3D*. It checks if the given group(s) and slot(s) described in the element match the set of group names and slots assigned to the object. More information on these command sub-structures, and others, is provided in the **Command Constructs** section.

If a group name is assigned to an object without a slot name, the slot name is set to `Default`. More information regarding the use of the group logic is described in **Group**. There are many contexts where it can be simpler, more efficient, or just easier to work with groups without troubling to specify slots. However, users should always remember that a subsequent group assignment within a given slot—including the `Default` slot—will overwrite an earlier one. Explicit use of slots is the easiest way to avoid potential conflicts that can come up.

The primary way to assign group names to objects is with the `group` keyword following the noun that represents the object. For example, zone group names are assigned with `zone group`, zone face group names with `zone face group`, and cable elements with `structure cable group`. The following example assigns the group name `Fred` to all zones whose centroid falls between 0 and 1.

```
zone group 'Fred' range position-x 0 1
```

Any time a group and slot are used, there are two methods of specifying it on the command line. One is the explicit use of a `group` keyword and a `slot` keyword. The other is to combine slot and group (in that order) using the syntactical construction `slotname=groupname` following a `group` keyword. Consider the following.

```
zone group 'mohr' slot 'models'
zone group 'models=mohr'
; the above two group assignments are equivalent
zone cmodel assign elastic range group 'mohr' slot 'models'
zone cmodel assign elastic range group 'models=mohr'
; above two ranges using the "mohr" group in slot "models" are the same
```

As may be inferred from the snippet above, one useful application for slots is the ability to create meaningfully named categorizations of groups—one could extend what is happening here by creating slots named `excavation_sequence`, `material_type`, `fracture_groups`, and so on. It is quite common to use one slot for

different materials that will be assigned different constitutive models and properties, and another for regions that will be excavated in a specific sequence. The **Model Pane** is particularly useful for this purpose.

Using the Group Range Element

- *Simple group range element*

Below is an example of the most common and simple use of the group range element: selecting objects that match a simple group name.

```
zone property bulk 3e8 range group 'Fred'
```

This command assigns the bulk modulus property to all zones that match the group name `Fred`. Not specifying a slot means that *FLAC3D* will check for `Fred` in all slots assigned to the zone. Even if more than one slot is being used, it is very common for group names to only be used in a single slot.

- *Multiple groups in one element*

Multiple group names can be specified in a single range element. The command

```
zone property bulk 3e8 range group 'Fred' or 'George'
```

will select zones that have been assigned *either* the group name `Fred` or the group name `George`. If the “slotname=groupname” syntax is used, `Fred` and `George` could be required to appear in different specific slots.

- *Connected objects*

The check to see if a group name is assigned to an object will often follow any object hierarchy that exists. For example, zone gridpoints are connected to zone faces, which are connected to zones. Checking for a group name on a gridpoint will automatically check if that name has been assigned to any of the zone faces connected to the gridpoint, or to any of the zones

connected to the gridpoint. The following example assigns the group name `One` to a single zone and then initializes the velocity of all the gridpoints connected to that zone.

```
zone group 'One' range id 1
zone gridpoint initialize velocity-x 1 range group 'One'
```

- *Faces between zone groups*

The hierarchy can be used with multiple range elements to select zone faces that fall between two groups. For example,

```
zone separate by-face range group 'Fred' group 'George'
```

will select those zone faces that fall on the boundary between zones with group `Fred` and those with group `George`. Recall that by default, a range is the *intersection* of all its range elements. That means that the selected face must belong to both group `Fred` and group `George`. Because these group names were applied to zones only in a single slot, this means that the face must be connected to a zone of group `Fred` on one side and a zone of group `George` on the other.

See `range group` for full documentation on the options available to the group range element.

Select and Hide

Selecting and hiding are both new concepts in *FLAC3D* 6.0. In the context of the *Model Pane*, their meanings are as one might expect: *Hiding* removes objects (zones, for instance) from the view so that other zones that may have been obscured can be seen and interacted with; and *Selecting* identifies which specific zones a subsequent operation will act upon.

However, they are not only visual and interactive concepts. The hidden and selected state of objects is changed through the command line, and is part of the command record. Operations that are affected or depend on the hidden or selected states of an object can be done using commands manually entered into a data file like any other.

It is important that one understand the possible side effects of setting and clearing these states on subsequent commands.

The `hide` Keyword

Many objects provide a `hide` command that allows you to set and remove the hidden state of an object. The hidden state of an object is a persistent part of that object. It is saved and restored from a model save file. Zones, for example, can be hidden with the command:

```
zone hide range group 'Fred'
```

The same group can be made visible (un-hidden) with the command:

```
zone hide off range group 'Fred'
```

Once hidden, objects remain hidden even if others are hidden later using a different command.

Important: Hiding has the effect of “suspending” the hidden objects from the model. Hidden objects are automatically skipped by commands that take a `range` keyword, even if no range was specified. Thus, hiding is another way—in addition to use of `range`, groups, and selection—to operate on partial rather than complete object collections. The `use-hidden` keyword may be used in a range phrase to suppress this behavior and select objects even if they are hidden.

```
zone hide range 'Fred'  
zone cmodel assign mohr-coulomb  
zone hide off
```

The example above has the effect of assigning the Mohr-Coulomb constitutive model to every zone *except* those assigned the group name `Fred`, because they were hidden before the command was issued.

If you use the `hide` command(s), it is advisable to give the command `zone hide off` (for example) and make certain that all zones are visible before you move on.

The `select` Keyword

As a rule, any object that can be hidden will also provide a `select` command word as well. The command is used both to select (default) or de-select (by setting the optional flag `off` in the command) the objects found in the `range` given with the command. Objects remain selected until they are de-selected. While a selection exists, additional selection commands on like objects will add to the current set of selected objects. For any object type, there can only be one selection at a time. However, there may be multiple different types with a current selection.

The `selected` Range Element

If there is a current selection of a given object type, it may be invoked as an object in a `range` with the `selected` keyword.

```
zone select range group 'Fred'
zone cmodel assign mohr range selected
zone select off
```

The above commands select zones assigned the group name `Fred`, use the selection in a `range` to assign the `mohr-coulomb` constitutive model, then de-select the zones.

There is also a `deselected` range keyword that can be used in the same way, but, as its name implies, it will only select objects *not* currently selected.

Some Additional Considerations

Selected objects may not be hidden. An attempt to hide objects in a set that partially overlaps the current set of like selected objects will successfully hide those objects *not* selected, but the objects that are selected will be unaffected.

Users should also note that behaviors of the interactive tools in the *Model* pane do not correspond exactly to behaviors observed in commands with regards to selection. When using a tool to select an object(s) in the *Model* pane, the selection of a currently unselected object will cause the current selection to become de-selected. This is not how the `select` command word operates. It will *add* newly selected object(s) to the current selection. In the case of the commands, the keyword `new` can be added to a `select` command, which will make the command

operate as the interactive tools do (new objects are selected and the current selection is de-selected). Conversely, the operation of interactive tools is made additive by including the `Ctrl` key when using the tools.

Using the Model Pane

The **Model Pane** provides an interactive facility for visualizing and selecting model objects. A selected set of objects can, in turn, be used as the target of a command directly (see the **selected Range Element**) or “stored” as needed as a group.

Actions in the **Model Pane** are emitted as commands. These commands are echoed in the **Console Pane**, and they are recorded in the **State Record Pane**. Users should be mindful of the possible need to capture the commands issued in the **Model Pane** for later use in a data file. If such a need exists, then the **State Record Pane** may be used to export, to a data file, all the commands that have been emitted in the **Model Pane** session (or anywhere else) since the model state was reset.

Use of the tools provided in the **Model Pane** are described in detail in the **Model Pane** topics found in the **FLAC3D Interface** section of this Help. The following material provides a narrative overview of kinds of operations available in the **Model Pane**.

Objects in the Model Pane

In version 6.0, the objects that can be shown in the *Model* pane are limited to zones or zone faces. Note that the view of the model will only show zones or zone faces, but not both at the same time. Later versions will incorporate additional objects for visualization and selection in the pane, such as structural elements, geometric data, user-defined data, and so on.

Visualizing Objects

The appearance of objects in the pane is controlled by the “Objects” control set in the **Control Panel**. Visualization, in turn, controls what can and cannot be selected, as described below. Visualization in the *Model* pane is comparable to visualization in a **View Pane**, with some differences: 1) the list of objects in the *Model* pane is automatically populated as objects are added to the model (the user

does not have to specify which objects appear); and 2) there is a smaller number of attributes available to modify rendering, and they are focused on differentiation of different selectable regions.

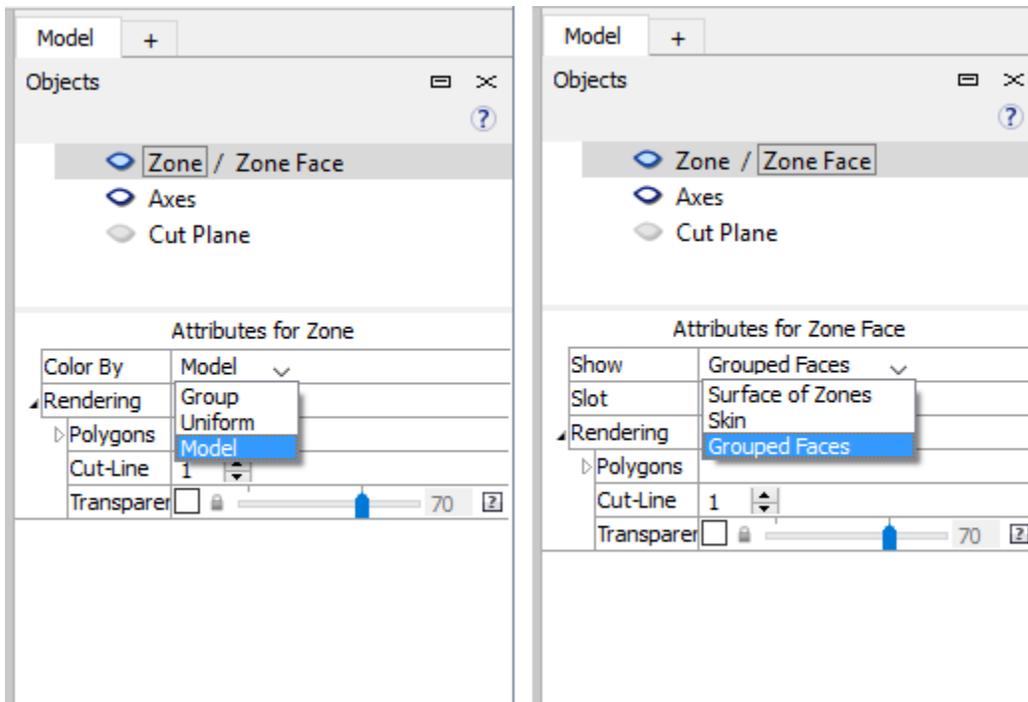


Figure 1: Left: the “Zone” object is active, with rendering options shown in the “Attributes” section; Right: the “Zone Faces” object is active, with the rendering options shown. Clicking on the label allows switching from one to the other.

Show/Hide tools (“Hide”, “Show Selected Only”, “Hide Selection”, “Show All”) facilitate cutting into the model to access interior spaces or reducing the model to exact areas of interest. In addition, “Range” elements may be applied as well. These operate similarly to the “Range” elements that *View* panes use for plotting. One significant difference is that range elements in the *Model* pane cannot be combined; they must be specified one element at a time.

Hiding vs. Selection

It is a self-evident but important rule of the interface that an object can only be selected if it is visible: while “zones” are rendered, it is not possible to select “zone faces”; while “zone groups” are set in the “ColorBy” attribute, it is not possible to select by constitutive model assignment, and so on. This applies to all tools in the pane. For instance, when the entire model is visible, the *assign groups names to faces automatically* tool can quickly return assigned group names to the

set of faces that constitute the outer boundaries of the model. However, if the model is reduced—through use of Range elements, for instance—to a much smaller set of visible zones, the *assign groups names to faces automatically* tool will operate on those zones in exactly the same way (the image below illustrates).

Selection

Selection is performed using the tools provided on the *Model* pane toolbar. Note the tools provided are contextual to the kind of object currently being rendered/selected (zones or zone faces). Also note that only base operations are described below; however, keyboard modifiers are available to alter/extend tool behavior. See [The Model Pane](#) section for detailed usage instructions.

The base *select* tool () will select all zones or zone faces of a like color, as determined by the “ColorBy” attribute in the “Objects” control set.

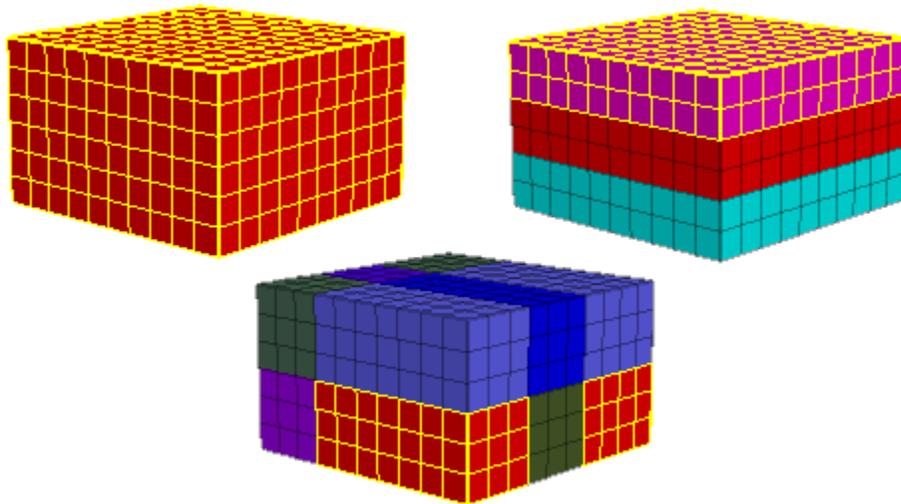


Figure 2: Simple selection by color. Left: *ColorBy > Uniform* — all zones selected. Middle: *ColorBy > Group* — zones in the picked group are selected (note these do not need to be contiguous); Right: *ColorBy > Model* — zones with the same constitutive model assignment are picked. Selection is always indicated with yellow highlighting.

The *disambiguate* tool () shows a list of all groups to which the object under the mouse belongs. “ColorBy” set to “Group” is often the easiest way to select objects, however, “finding” the desired group to select in models where there are numerous overlapping group assignments for a given zone (or zone face) can be difficult. This tool provides useful assistance in that circumstance.

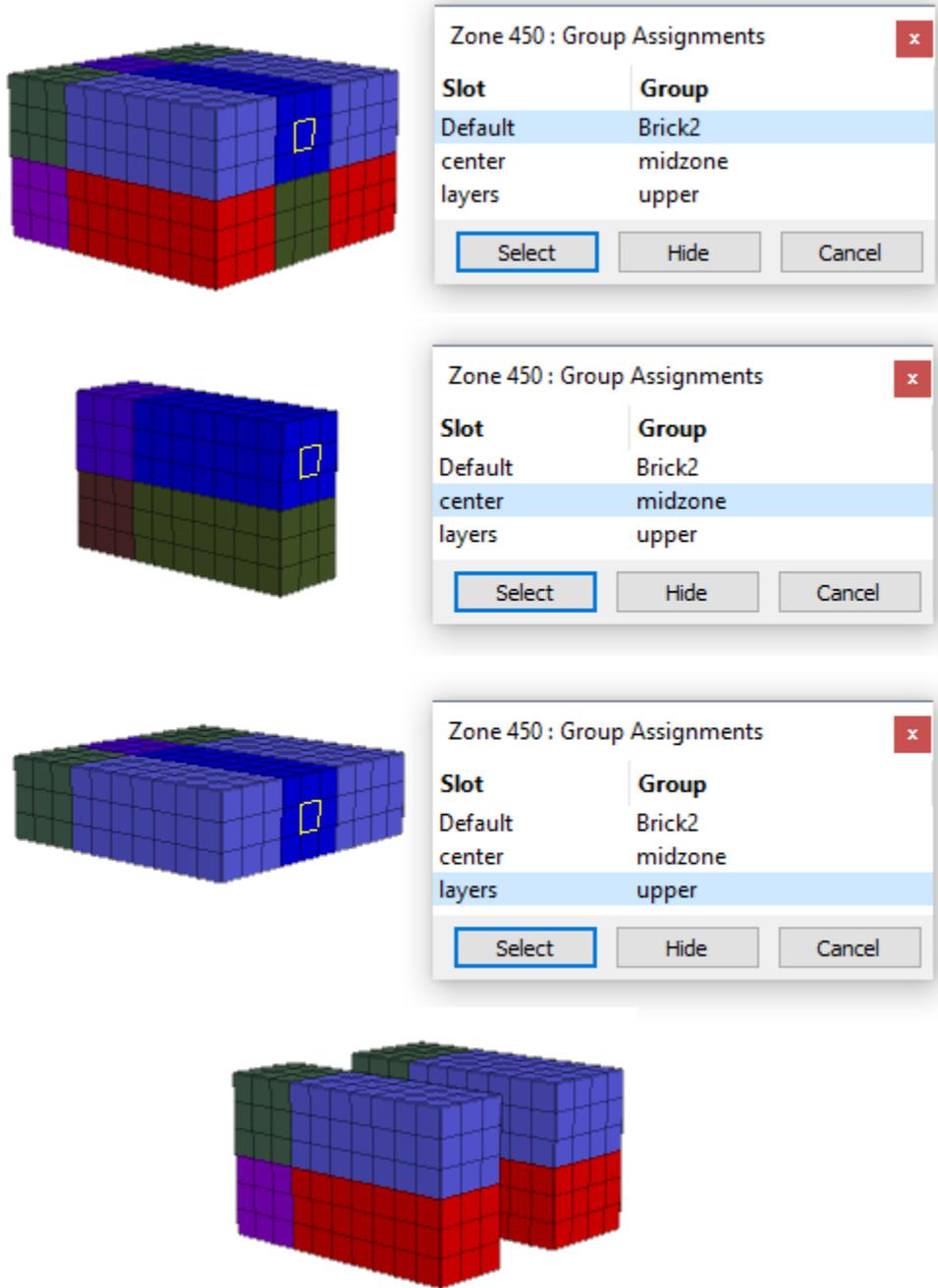


Figure 3: The disambiguate tool is used to show the three groups to which the highlighted zone is assigned. If the “Hide” button is clicked while the “center=midzone” group is highlighted, the result will be as shown in the final image at bottom.

When “zones” are rendered, the *assign group names to faces automatically* tool () can be used to “skin” the visible zones into face groups as determined by a break angle. The model is split into external (and, optionally, internal) face groups at boundaries that are determined by the break angle (that is, contiguous faces that form angles smaller than the break angle) plus the current “ColorBy” setting.

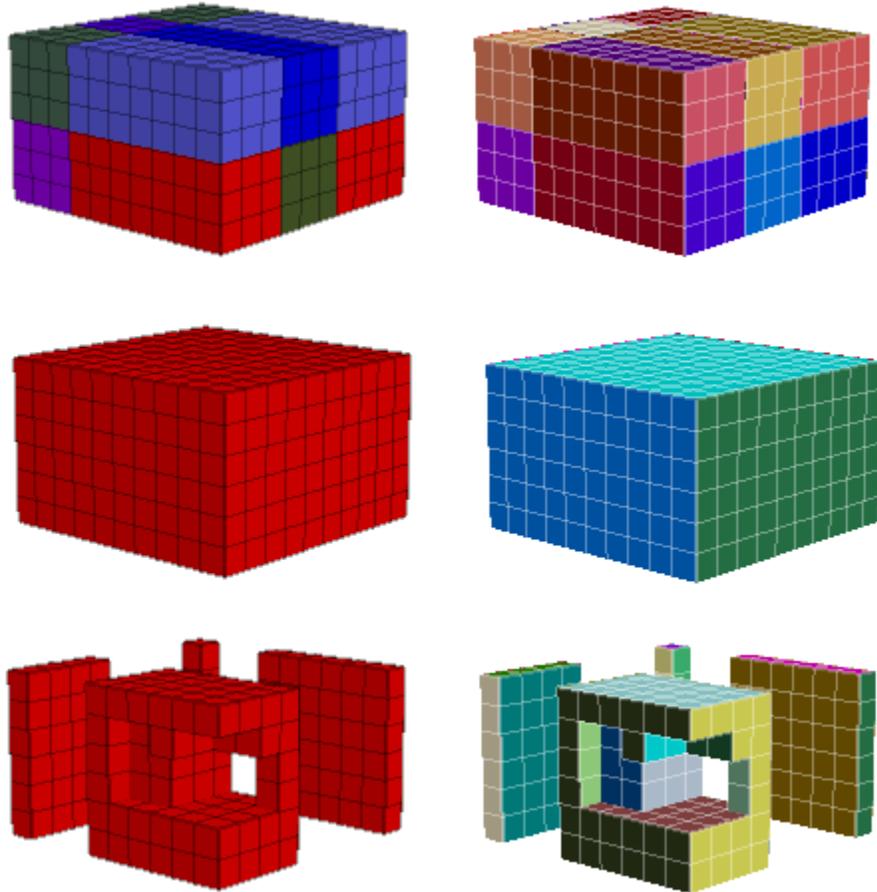


Figure 4: The same operation (Assign Group Names to Faces Automatically) three ways. Top: The model is colored by group, and the face groups that result are shown to the right. Middle: The model is colored uniformly (“uniform”), with resulting face groups shown on the right. Bottom: The model is colored uniformly, and zones have been cut away using the Range and other tools, resulting in the face groups shown on the right.

When “zone faces” are rendered, the *select by break angle* tool () is available to select contiguous faces as single selections (the *hide by break angle* tool () is also available).

Selection vs. Hiding

Another interface rule in the *Model* pane is that selected objects cannot be hidden (this would be the inverse of the rule that hidden objects cannot be selected). Attempting to use a hide tool on a selected object will fail (which is indicated in an *Information* dialog). The one exception to this is the *Hide the Selection* tool (), which, when pressed, will hide *and de-select* whatever is currently selected.

Groups

The *Assign a Group to the Selection...* command () is available on the toolbar to create a group from the current selection. When pressed, it provides a dialog where a group name and, optionally, a slot may be specified (see [Groups and Slots](#) and [Group](#) for explanations of groups and slots).

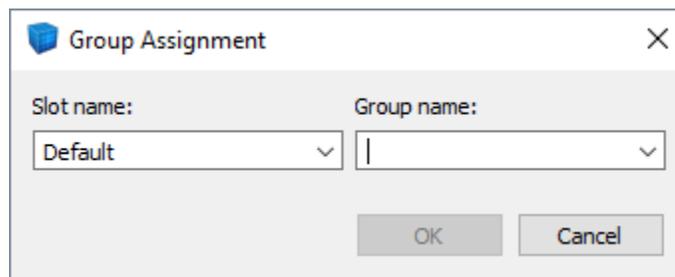


Figure 5: The dialog called when the *Assign a Group to the Selection...* command is used.

Additional Commands

When “zones” are the currently rendered objects, a *Densify the Selected Zones* command is available on the command button menu (). When zone faces are rendered, a *Create a 2D Structural Element* command () is available.

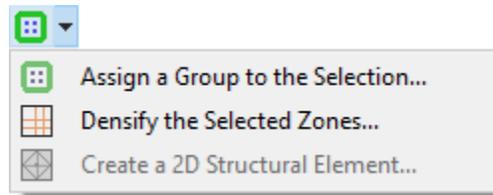


Figure 6: The menu of commands available in the Model pane (zones currently the rendered object).

Working with Geometric Data

FLAC3D has the capability to import and define arbitrary geometric data. For instance, CAD data from AutoCAD can be imported from a DXF file. This geometry information can be manipulated after creation, and it can also be used for reference in visualization. There are certain (as yet somewhat limited) commands that allow zones or building blocks to be generated from polygons forming a closed volume. In addition, the data can be used to filter the objects affected by a *FLAC3D* command and to identify objects in certain regions by assigning group names.

This section provides a high-level description of how geometric data can be used in the current version of *FLAC3D*. For details, including all available options, see the documentation of commands related to geometric data in the [Command and FISH Reference Index](#).

The data files and geometry data used to generate all the images in this section can be found in the project “UsersGuide\ProblemSolving\GeometricData\GeometricData.f3prj” in the “datafiles” folder of the *FLAC3D* distribution.

Note that a discussion of using geometric data to create densified “octree” grids can be found in [Grid Generation](#) under [Geometry-Based Densification: Octree Meshing](#).

Geometric Data

FLAC3D organizes geometric data into **sets**, which are named collections of polygons, edges, and nodes. The data are topologically connected. Polygons are defined by a series of edges, and edges are defined by two nodes.

The easiest way to create a geometric set is via the `geometry import` command. For example, the command

```
geometry import 'surface1.stl'
```

will import the data in the file `surface1.stl` and place it into a geometric set called `surface1`. Currently available formats are DXF, STL, and an Itasca-defined format that preserves added metadata (e.g., group names, *FISH* extra variables,

etc.). You can also simply go to the File › Open into Project... menu command (Ctrl + O) and select a geometry file with a recognized extension (STL, DXF, and GEOM) in the ensuing dialog.

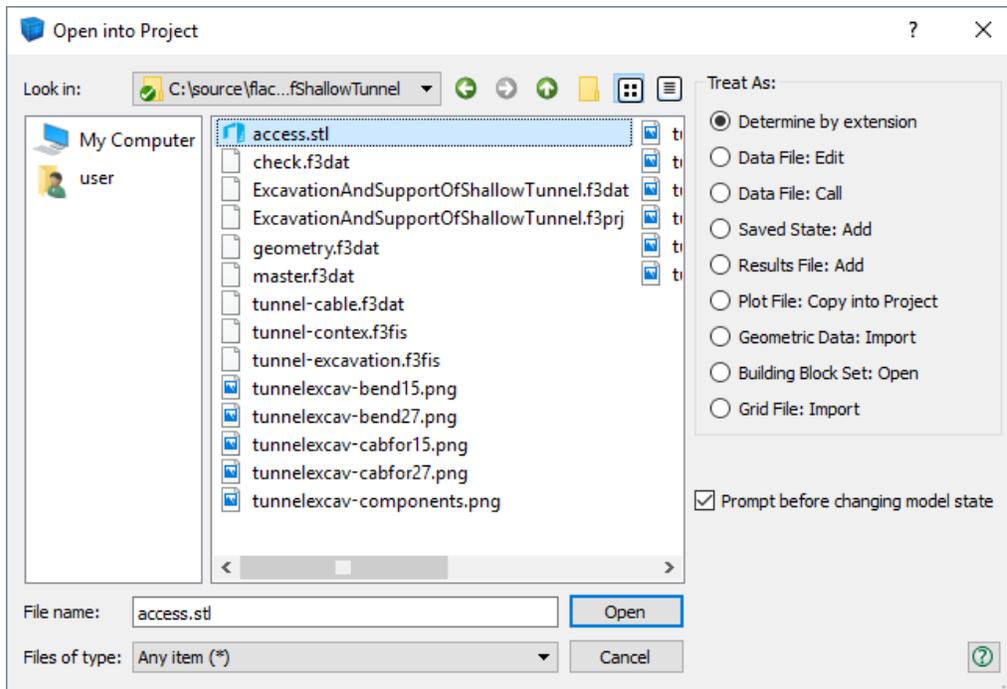


Figure 1: The *FLAC3D* Open into Project dialog.

Geometric data can also be created via commands. For instance, the `geometry polygon create` command can be used to directly add a polygon to a geometric set. For example,

```
geometry polygon create by-positions (0,0,0) (1,0,0) (1,1,0) (0,1,0)
```

will add a simple quadrilateral to the current geometry set.

In addition, all data in all sets are available through *FISH* (see the topic [Geometry FISH Functions](#) for a list of functions). *FISH* can be used to both modify and create geometric data. For instance, the following is a simple *FISH* function that creates a geometric set called “FISH Example” and then adds polygons forming a cylinder:

Example: “fish_cylinder.fis”

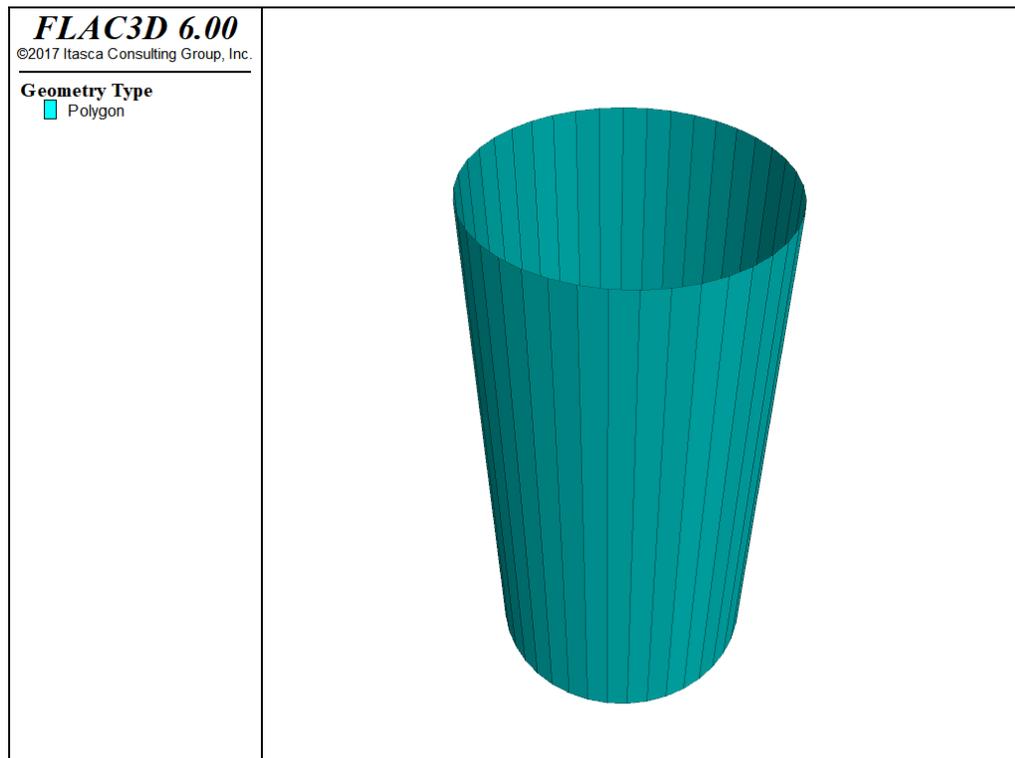
```
model new
```

```

fish define fish_cylinder(start,radius,height,segments)
  local gset = geom.set.find("FISH Example")
  if gset = null then
    gset = geom.set.create("FISH Example")
  endif
  loop local i (1,segments)
    local ang1 = float(i-1) * math.pi * 2.0 / float(segments)
    local ang2 = float(i) * math.pi * 2.0 / float(segments)
    local p1 = start + vector(math.cos(ang1),math.sin(ang1),0.0)
    local p2 = start + vector(math.cos(ang2),math.sin(ang2),0.0)
    local p3 = vector(p2->x,p2->y,height)
    local p4 = vector(p1->x,p1->y,height)
    local poly = geom.poly.create(gset)
    geom.poly.add.node(gset,poly,p1)
    geom.poly.add.node(gset,poly,p2)
    geom.poly.add.node(gset,poly,p3)
    geom.poly.add.node(gset,poly,p4)
    geom.poly.close(gset,poly)
  end_loop
end
@fish_cylinder((0,0,0),1.0,4.0,40)

```

The results are shown below.



Geometry Visualization

Geometric data can be visualized in a **View Pane**. When the Open into Project... dialog is used, the geometry will be automatically rendered in a new plot.

If the set is added by command at the command prompt, it will need to be added manually to a plot to be seen. To do so:

1. Create a plot (File ▸ Add New Plot... or Ctrl + Shift + N).
2. Use the *Build Plot* tool () to get the *Build plot* dialog, pick the “Geometry” item from the list in the “User Defined Data” category.
3. In “Attributes”, make sure that the desired set is selected in the “Sets” control. (Note: With this control, more than one set maybe visualized on the same plot item.)

Geometry Painting

To help visualize how model variables vary with respect to geometry data, data from the zones can be used to “paint” values on to geometric nodes. These values can then be contoured to visualize how model variables vary in relation to geometric features that might not be explicitly modeled.

By switching the “Type” attribute to “Contour”, and switching the “ContourBy” value to “FieldData”, you can select from the list of all zone field variables. This data is used only for plotting, and is not visible to the model state.

The `geometry paint-extra` command, however, can be used to “paint” zone field data values from the model to an extra variable stored in geometric nodes. For example, to paint the minimum principal stress in the model onto the active geometric set, you use the command

```
geometry paint-extra 2 stress quantity minimum
```

which will calculate minimum principal stresses at the location of all nodes in the active set, and assign that value to extra variable index 2. The complete list of zone field variables available can be found in the `geometry paint-extra` command documentation.

The result of painting can be visualized by using the *Geometry* plot item (with attributes “Type” = “Contour”, “Contour By” = “Node Extra”, and “Index” = “2”). The figure below is an example of painting the displacement magnitudes on a simple tunnel next to the main excavation.

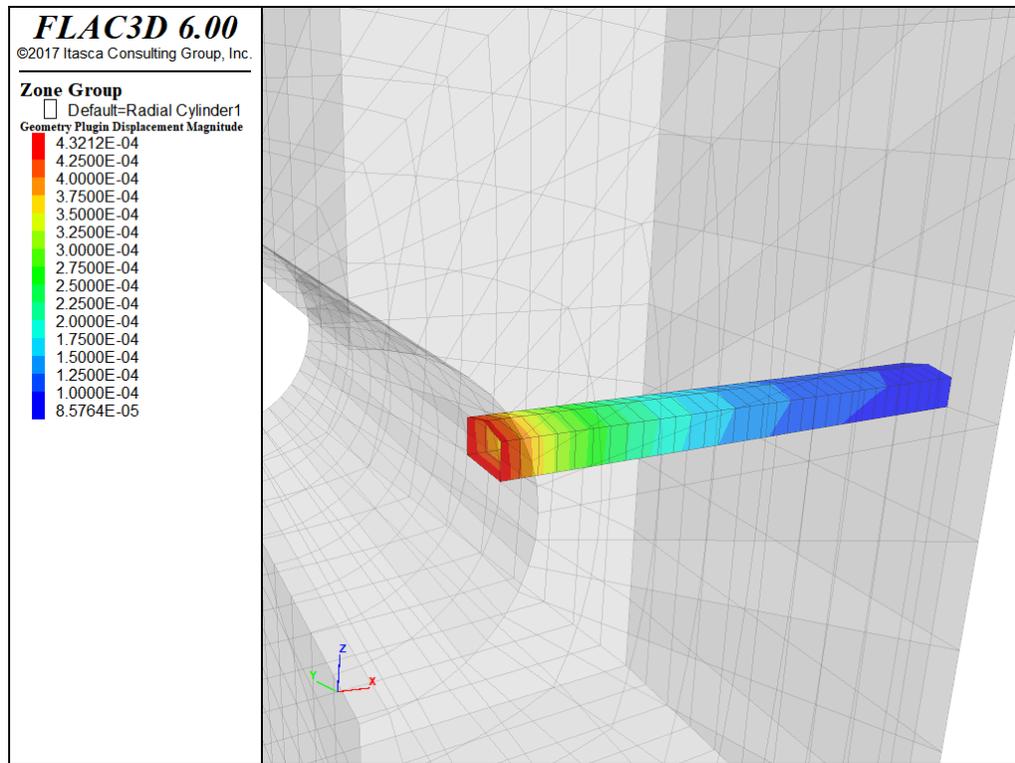


Figure 3: Displacement magnitude painted on geometry.

Geometric Filtering – Geometry Range Elements

The primary method for selecting objects in *FLAC3D* is to provide a filter that determines the objects affected by a given command. This is done using the `range` logic. There are two `range` elements that exist to filter objects based on their relation to data in geometric sets: `geometry-distance` and `geometry-space`.

The `geometry-distance` range element can be used to select objects that fall within a given distance of data in a geometric set. This includes polygons, free edges and free nodes. The minimum distance from the object centroid to any data in the set determines the distance value.

The `extent` keyword is valid for the `geometry-distance` range element, finding the minimum distance to the Cartesian extent of an object. This can be used with a distance value of zero to return an approximation of all objects that actually

intersect the geometric set. For example, the following command can be used with the above data to assign group name “Intersect” to all zones that have a Cartesian extent that intersects the geometric set “intcylinder.”

```
zone group "Intersect" range geometry-distance "intcylinder" gap 0.0 extent
```

The result of that operation is shown here.

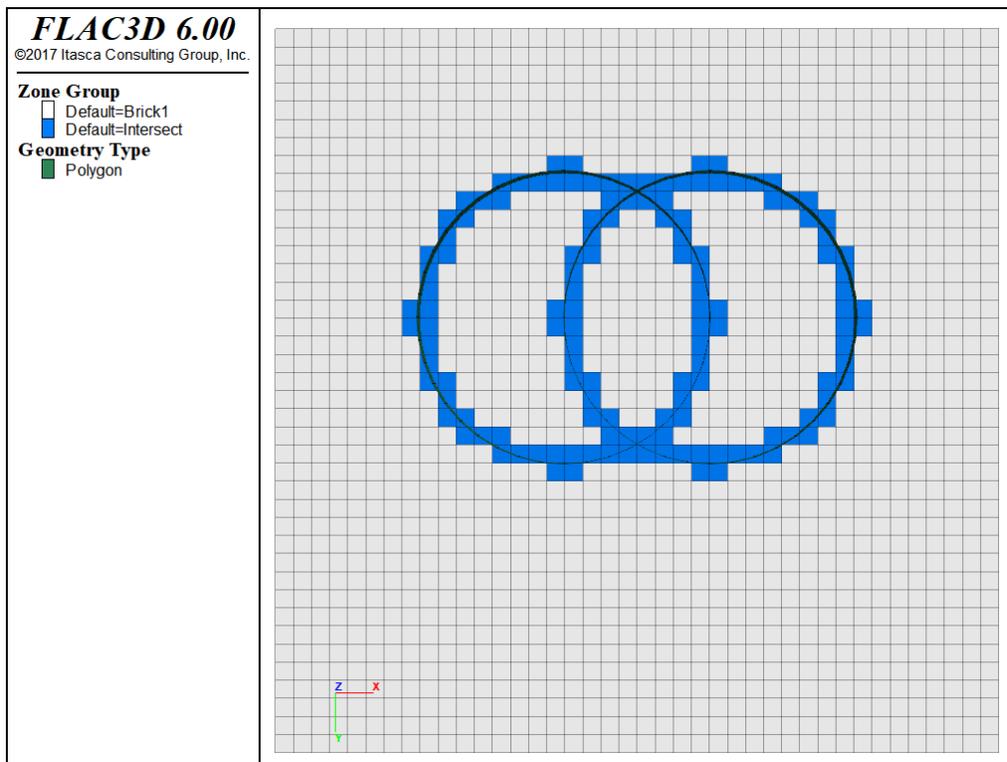


Figure 4: Zones selected by the geometry dist range element.

The `geometry-space` range element selects objects based on how many times a ray starting from the object intersects polygons from the geometric set. If the intersection count matches the number provided, it is considered selected. This can be used to

```
zone group "Count 1" ...
  range geometry-space "intcylinder" count 1 direction (1,0,0)
```

is given, the results can be seen here.

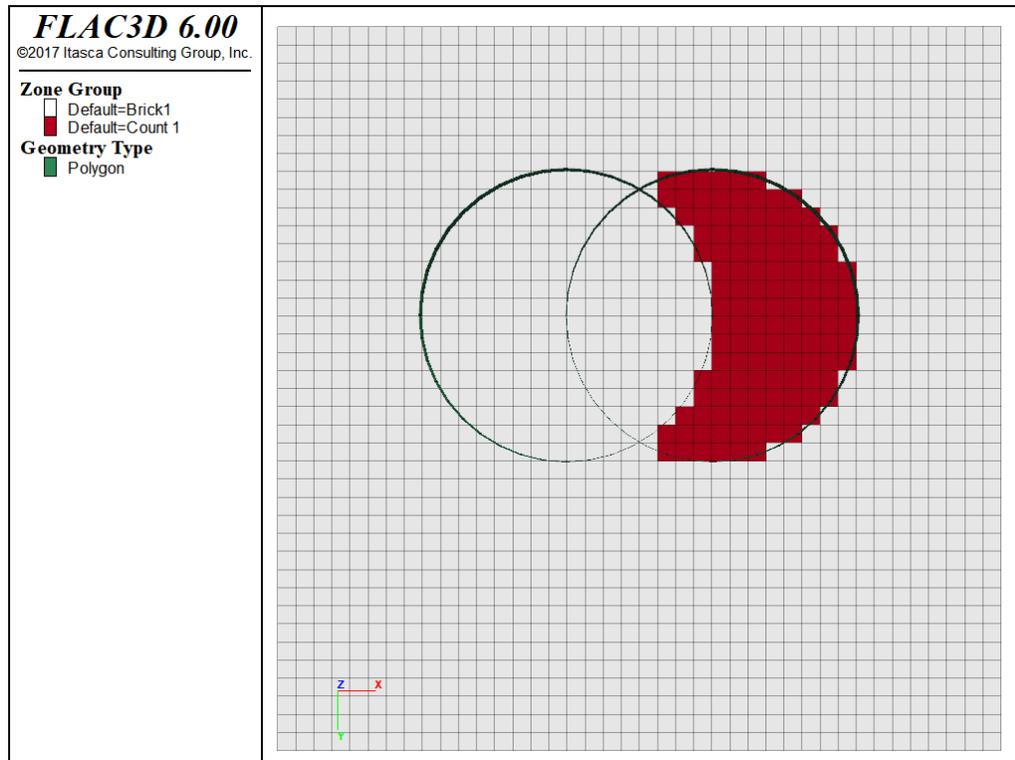


Figure 5: Zones selected by the geometry count range element.

As a special case, the keywords `odd` or `even` can be substituted in place of a count number in the geometry count range element. If the geometric set *effectively* forms a closed volume, the `odd` keyword will result in selecting all objects inside, and the `even` will select all objects outside. No checking is done to make certain the polygons actually form a single closed surface. The results of the command

```
zone group "Count Odd" ...
      range geometry-space "intcylinder" count odd direction (1,0,0)
```

are shown in in the next figure. Note that the intersecting cylinder geometry set does not form a single closed surface, and therefore, the range element does not select zones in the shared region.

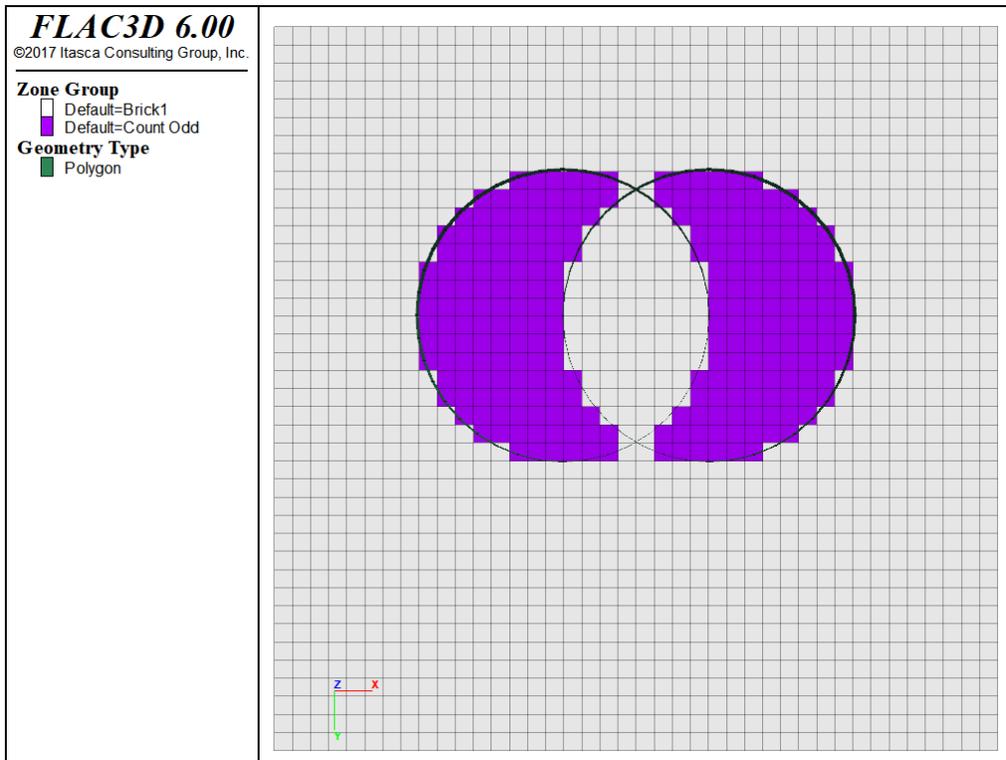


Figure 6: Zones selected by the geometry count range element using the *odd* keyword.

If the geometric set does form a perfect topologically closed volume, then the *inside* and *outside* keywords can be used to select objects instead of the *count* keyword.

Geometry Data and Group Assignment

There are occasions when you wish to select objects that fall into regions of space delineated by complex geometric data. Sometimes these data are not neatly separated into distinct surfaces that can be assigned to different geometric sets. Sometimes the relation between the data is complex, and manually identifying specific regions (e.g., above this surface, behind that, and ahead of this) is burdensome. The *geometry assign-groups* command lets you assign group names to objects based on how they relate to all these surfaces at once.

The operation works by first assigning a unique ID number to all polygons that form a single contiguous surface. Specifically, all polygons in a single set that are connected across edges will be given the same ID number. A ray is then passed from the centroid of the object through those polygons, and a group name is generated based on how many times the ray intersects polygons of each ID.

As a simple example, we will use a geometric set consisting of two intersecting cylinders. The `geometry assign-groups` command is used to assign group names to zones based on their relation to the surfaces using the command

```
geometry assign-groups zone projection (1,0,0) set 'intcylinder'
```

The figure below shows the group name assigned to zones in a cut-plane through the model. Each letter represents a different intersection region as defined by surface ID and intersection count.

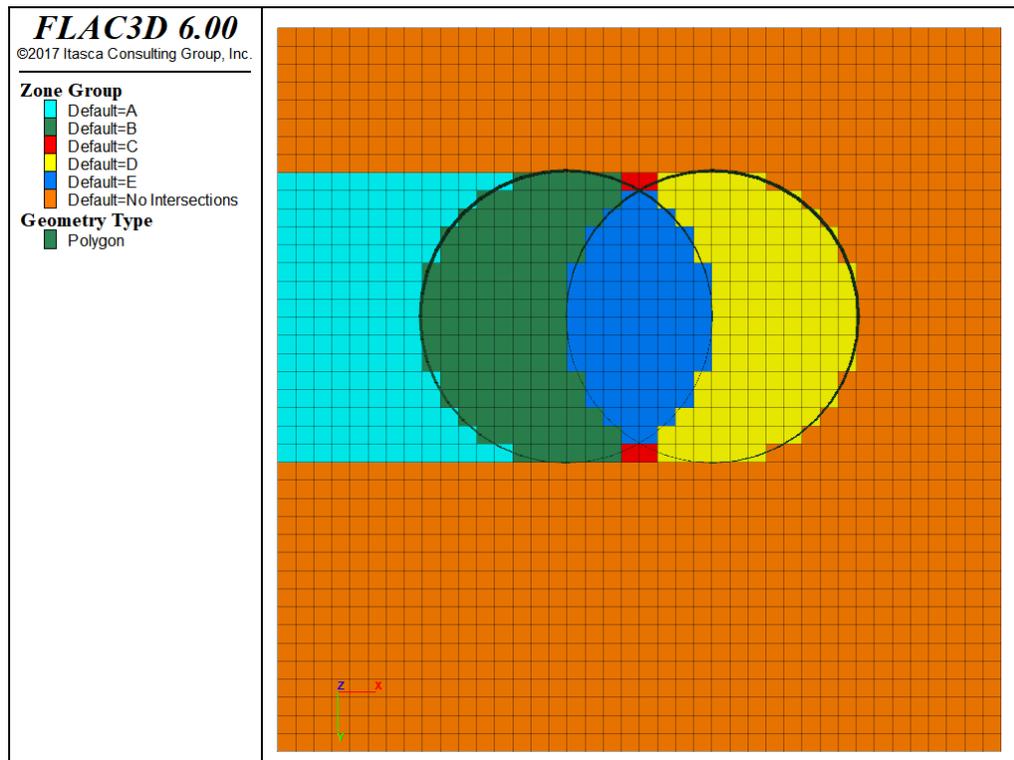


Figure 7: Groups assigned by the `geometry assign-groups` command.

This operation may be performed on zones, gridpoints, user-defined data and structural element objects (elements, nodes, and links).

While the `geometry assign-groups` command can quickly separate objects into separate geometric regions, it does have limitations. The user has no control over the ID numbers assigned to surfaces, and determining the group names assigned to desired regions has to be done via visual inspection. Often, multiple groups will need to be combined into the actual region of interest.

Choice of Constitutive Model

All zones in a model must be assigned a constitutive model. In turn, the properties of the model(s) that have been assigned must be set. These two steps make up the work involved in characterizing the material that makes up the model, and there are a number of considerations to be taken into account. This section examines the constitutive model selection and assignment process. The next, [Material Properties](#), covers the subject of property assignment.

All of the examples listed in this section are included in the project file “ConstitutiveModels.f3prj.”

Overview of Constitutive Models

This section provides an overview of the constitutive models in *FLAC3D* and makes recommendations concerning their appropriate application. The [Constitutive Models](#) section presents background information on the model formulations.

The built-in mechanical material models [1] in *FLAC3D* are:

1. Null;
2. (Isotropic) elasticity;
3. Orthotropic elasticity;
4. Anisotropic (transversely isotropic) elasticity;
5. Drucker-Prager plasticity;
6. Mohr-Coulomb plasticity;
7. Ubiquitous-joint plasticity;
8. Anisotropic-elasticity ubiquitous-joint plasticity;
9. Strain-hardening/softening Mohr-Coulomb plasticity;

10. Bilinear strain-hardening/softening ubiquitous-joint (SUBI) plasticity;
11. Double-yield plasticity;
12. Modified Cam-clay plasticity;
13. Hoek-Brown plasticity;
14. Hoek-Brown-PAC plasticity;
15. Cap-Yield (CYSoil) plasticity;
16. Simplified Cap-Yield (CHSoil) plasticity;
17. Plastic-Hardening (PH) plasticity;
18. Swell plasticity; and
19. Mohr-Coulomb Tension Crack plasticity.

Each model is developed to represent a specific type of constitutive behavior commonly associated with geologic materials. The **null** model is used to represent material that is removed from the model. The **isotropic elasticity** model is valid for homogeneous, isotropic, continuous materials that exhibit linear stress-strain behavior. The **orthotropic elasticity** model and the **anisotropic (transversely isotropic) elasticity** model are appropriate for elastic materials that exhibit well-defined elastic anisotropy. The **Drucker-Prager plasticity** model is a simple failure criterion in which the shear yield stress is a function of isotropic stress. The **Mohr-Coulomb plasticity** model is used for materials that yield when subjected to shear loading, but the yield stress depends on the major and minor principal stresses only; the intermediate principal stress has no effect on yield. The **ubiquitous-joint plasticity** model corresponds to a Mohr-Coulomb material that exhibits a well-defined strength anisotropy due to embedded planes of weakness. The **Anisotropic-elasticity ubiquitous-joint plasticity** model is extended from the ubiquitous-joint plasticity model by taking the anisotropic elasticity into account. The **strain-hardening/softening Mohr-Coulomb plasticity** model is based upon the Mohr-Coulomb model, but is appropriate for materials that show an increase or degradation in shear strength when loaded beyond the initial failure limit. The **SUBI plasticity** model is a generalization of the ubiquitous-joint model that allows the strength properties for the matrix and the

joint to harden or soften. The **double-yield plasticity** model is an extension of the strain-softening model to simulate irreversible compaction as well as shear yielding. The **modified Cam-clay plasticity** model accounts for the influence of volume change on deformability and on resistance to failure. The **Hoek-Brown plasticity** model is an empirical relation that is a nonlinear failure surface representing the strength limit for isotropic intact rock and rock masses. Its non-linear failure surface is continuously approximated by the Mohr-Coulomb tangent at the current minimum principle stress level. It can be used to perform factor of safety calculations. The **Hoek-Brown-PAC plasticity** model is based on the traditional Hoek-Brown nonlinear failure surface but also includes a plasticity flow rule that varies as a function of the confining stress level. The **CYSoil plasticity** model provides a comprehensive representation of the nonlinear behavior of soils considering both frictional and volumetric hardening. The **CHSoil plasticity** model is a simplified version of the CYSoil plasticity model by neglecting the volumetric hardening. The **Plastic-Hardening plasticity** model is a shear and volumetric hardening constitutive model for the simulation of soil behavior using a curved pre-failure stress-strain relation. The model is straightforward to calibrate using either conventional lab tests or in-situ tests. It is well established for soil-structure interaction problems, excavations, tunneling, and settlements analysis, among many other applications. The **Swell plasticity** model is based on the Mohr-Coulomb model with nonassociated shear and associated tension flow rules. The difference is that the wetting-induced deformations are taken into account by means of coupling wetting strains with the model state prior to wetting. The **Mohr-Coulomb Tension Crack plasticity** model is an extension of the Mohr-Coulomb constitutive model that considers tensile plastic strains to be reversible and prevents generation of compressive normal stresses (perpendicular to cracks) before cracks close.

The material models in *FLAC3D* are primarily intended for applications related to geotechnical engineering (e.g., underground construction, mining, slope stability, foundations, earth and rock-fill dams). When selecting a constitutive model for a particular engineering analysis, the following two considerations should be kept in mind:

1. What are the known characteristics of the material being modeled?
2. What is the intended application of the model analysis?

The table below presents a summary of the *FLAC3D* models along with examples of representative materials and possible applications of the models. *The Mohr-Coulomb model is the most applicable for general engineering studies.* Also, Mohr-Coulomb parameters for cohesion and friction angle are usually available more often than other properties for geo-engineering materials. The ubiquitous-joint, strain-softening, bilinear strain-softening/ubiquitous-joint, and double-yield plasticity models are actually variations of the Mohr-Coulomb model. These models will produce results identical to those for Mohr-Coulomb if the additional material parameters are set to high values. The Drucker-Prager model is a simpler failure criterion than Mohr-Coulomb, but it is not generally suitable for representing failure of geologic materials. It is provided mainly to allow comparison of *FLAC3D* to other numerical programs that have the Drucker-Prager model but not the Mohr-Coulomb model. Note that, at zero friction, the Mohr-Coulomb model degenerates to the Tresca model, while the Drucker-Prager model degenerates to the von Mises model.

The Drucker-Prager and Mohr-Coulomb models are the most computationally efficient plasticity models; the other plasticity models require increased memory and additional time for calculation. For example, plastic strain is not calculated directly in the **Mohr-Coulomb model**. If plastic strain is required, the strain-softening, bilinear ubiquitous-joint or double-yield model must be used. These three models are primarily intended for applications in which the post-failure response is important (e.g., yielding pillars, caving, or backfilling studies).

The Constitutive Models in *FLAC3D*

Table 1: *FLAC3D* Constitutive Models

Model	Representative Material	Example Application
null	void	holes, excavations, regions in which material will be added at later stage
elastic	homogeneous, isotropic continuum; linear stress-strain behavior	manufactured materials (e.g., steel) loaded below strength limit; factor of safety calculation
orthotropic elastic	materials with three mutually perpendicular	columnar basalt loaded below strength limit

Model	Representative Material	Example Application
	planes of elastic symmetry	
transversely isotropic elastic	thinly laminated material exhibiting elastic anisotropy (e.g., slate)	laminated materials loaded below strength limit
Drucker-Prager	limited application; soft clays with low friction	common model for comparison to implicit finite-element programs
Mohr-Coulomb	loose and cemented granular materials; soils, rock, concrete	general soil or rock mechanics (e.g., slope stability and underground excavation)
strain-hardening/ softening Mohr-Coulomb	granular materials that exhibit nonlinear material hardening or softening	studies in post-failure (e.g., progressive collapse, yielding pillar, caving)
ubiquitous-joint	thinly laminated material exhibiting strength anisotropy (e.g., slate)	excavation in closely bedded strata
anisotropic-elasticity ubiquitous-joint	thinly laminated material exhibiting stiffness and strength anisotropy (e.g., slate)	excavation in closely bedded strata
bilinear strain-hardening/ softening ubiquitous-joint	laminated materials that exhibit nonlinear material hardening or softening	studies in post-failure of laminated materials
double-yield	lightly cemented granular material in which pressure causes permanent volume decrease	hydraulically placed backfill
modified Cam-clay	clay	geotechnical construction on clay
Hoek-Brown	isotropic rock material	geotechnical construction in rock; factor-of-safety

Model	Representative Material	Example Application
		calculation
Hoek-Brown-PAC	isotropic rock material	geotechnical construction in rock
CYSoil and CHSoil	soils	excavation, tunnel, slope stability, embankment, foundation analysis
Plastic-Hardening	soils	excavation, tunnel, slope stability, embankment, foundation analysis
swell	soils with wetting-induced deformations	application on soils where wetting-induced deformations are significant
Mohr-Coulomb-Tension	rocks and soils	dynamic response or deformation of the overburden above the undercut

The tensile failure criterion is identical in the Drucker-Prager, Mohr-Coulomb, ubiquitous-joint, anisotropic-elasticity ubiquitous-joint, strain-softening, bilinear strain-softening/ubiquitous-joint, double-yield, and swell models. This criterion defines a tensile strength separately from the shear strength, and an associated flow rule for the onset of tensile failure. For the Drucker-Prager, Mohr-Coulomb, and ubiquitous-joint models, the value assigned to the tensile strength remains constant by default when tensile failure occurs, although the `flag-brittle` property can be used to reduce the tension limit to 0 after tensile failure. Tensile softening can be modeled with the strain-softening, bilinear strain-softening ubiquitous-joint and double-yield models. (see the [Post-Failure Properties](#) topic.) Note that no record is made of notional voids that may open after tensile failure and tensile strain; if the strain rate becomes compressive, all models start to take compressive load immediately—except for the Mohr-Coulomb-Tension model, which was explicitly formulated to overcome this.

The double-yield and modified Cam-clay models both take into account the influence of volumetric change on material deformability and failure characteristics. In both models, tangential bulk and shear moduli are functions of plastic volumetric deformation. The double-yield model was initially developed

to represent the behavior of mine backfill material, for which pre-consolidation pressures are low. The modified Cam-clay model is more applicable to soils such as soft clays for which pre-consolidation pressures can have a significant effect on material behavior. The comparison between the double-yield model and modified Cam-clay model is summarized [here](#).

The Hoek-Brown model approximates the non-linear failure surface by the Mohr-Coulomb tangent at the current minor compression principle stress level so that it keeps all merits of the traditional Mohr-Coulomb model, e.g., for factor-of-safety calculations. The Hoek-Brown-PAC model combines the generalized Hoek-Brown criterion with a plasticity flow rule that varies as a function of the confining stress level. At low confining stress, the volumetric expansion at yield is high, associated with axial splitting and wedging effects. At high confining stress, the material approaches a non-dilatant condition.

The CYSoil and PH models both consider friction and volumetric hardening, curved pre-failure behavior, and different unloading stiffness from the loading stiffness, so they are useful in the geotechnical application where the pre-failure and/or excavation-induced deformations are important, e.g., braced/anchored excavation wall design and tunnel construction. The comparison between the PH model and CYSoil model is summarized [here](#).

Selection of an Appropriate Model

A problem analysis should always start with the simplest material model; in most cases, an elastic model should be used first. This model runs the fastest and only requires two material parameters: bulk modulus and shear modulus (see [Material Properties](#)). The model provides a simple perspective of stress-deformation behavior in the *FLAC3D* grid, and can define locations where stress concentrations may develop. This may help to define zoning density for the grid.

It is often helpful to run a simple test of the selected material model before using it to solve the full-scale, boundary-value problem. This can provide insight into the expected response of the model compared to the known response of the physical material.

The following example illustrates the use of a simple test model. The problem application is the analysis of yielding mine pillars. A simple model is created to evaluate the implementation of the Mohr-Coulomb model versus the strain-

softening model. This test also illustrates the effect of the selected measurement location on the reported results. The model is a compression test performed on a cylindrical grid composed of Mohr–Coulomb material.

Compression test on Mohr–Coulomb material

```

model new
; Create zones
zone create cylinder point 0 (0,0,0) point 1 (1,0,0) ...
                    point 2 (0,2,0) point 3 (0,0,1) size 4 5 4
zone reflect normal (1,0,0)
zone reflect normal (0,0,1)
; Constitutive Model and properties
zone cmodel assign mohr-coulomb
zone property bulk 1.19e10 shear 1.1e10
zone property cohesion 2.72e5 friction 44 tension 2e5
; Boundary Conditions
zone face apply velocity (0, 1e-7,0) range position-y 0
zone face apply velocity (0,-1e-7,0) range position-y 2
; Histories
zone history displacement-y position (0,0,0)
zone history stress-yy position (0,1,0)
zone history stress-yy position (1,1,0)
model step 3000
model save 'Compression'

```

The axial (y -direction) stress–displacement response is monitored at the center and outer boundary in the grid. The results are shown in the figure below.

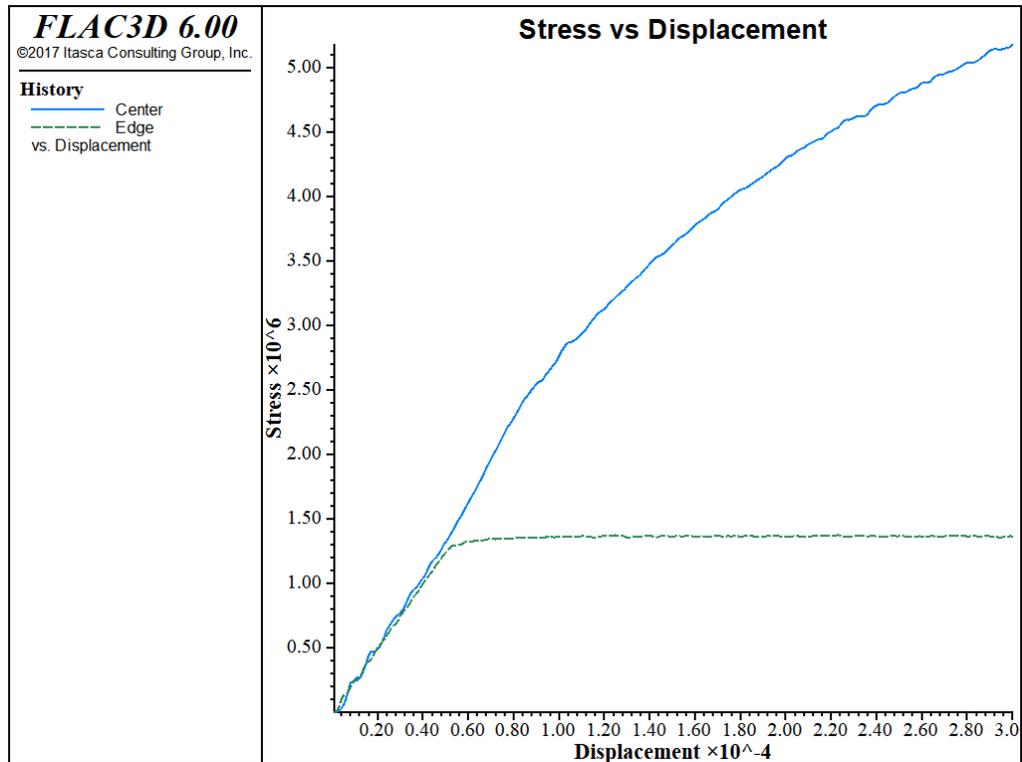


Figure 1: Stress–displacement plots for compression of Mohr–Coulomb material. Response is shown for an interior location (upper curve) and a boundary location (lower curve).

The test is now repeated with the strain–softening model.

Compression test on strain–softening material

```

model new
; Create zones
zone create cylinder point 0 (0,0,0) point 1 (1,0,0) point 2 (0,2,0) ...
                    point 3 (0,0,1) size 4 5 4
zone reflect normal (1,0,0)
zone reflect normal (0,0,1)
; Constitutive Model and properties
zone cmodel assign strain-softening
zone property bulk 1.19e10 shear 1.1e10
zone property cohesion 2.72e5 friction 44 tension 2e5
zone property table-cohesion 'coh' table-friction 'fri'
table 'coh' add (0,2.72e5) (1e-4,2e5) (2e-4,1.5e5) (3e-4,1.03e5) (1,1.03e5)
table 'fri' add (0,44) (1e-4,42) (2e-4,40) (3e-4,38) (1,38)
; Boundary Conditions
zone face apply velocity (0, 1e-7,0) range position-y 0
zone face apply velocity (0,-1e-7,0) range position-y 2
; Histories

```

```

zone history displacement-y position (0,0,0)
zone history stress-yy position (0,1,0)
zone history stress-yy position (1,1,0)
model step 3000
model save 'Softening'

```

The horizontal stress–displacement response is monitored again, as shown in the image below. This test produces distinct peak and residual failure stress levels.

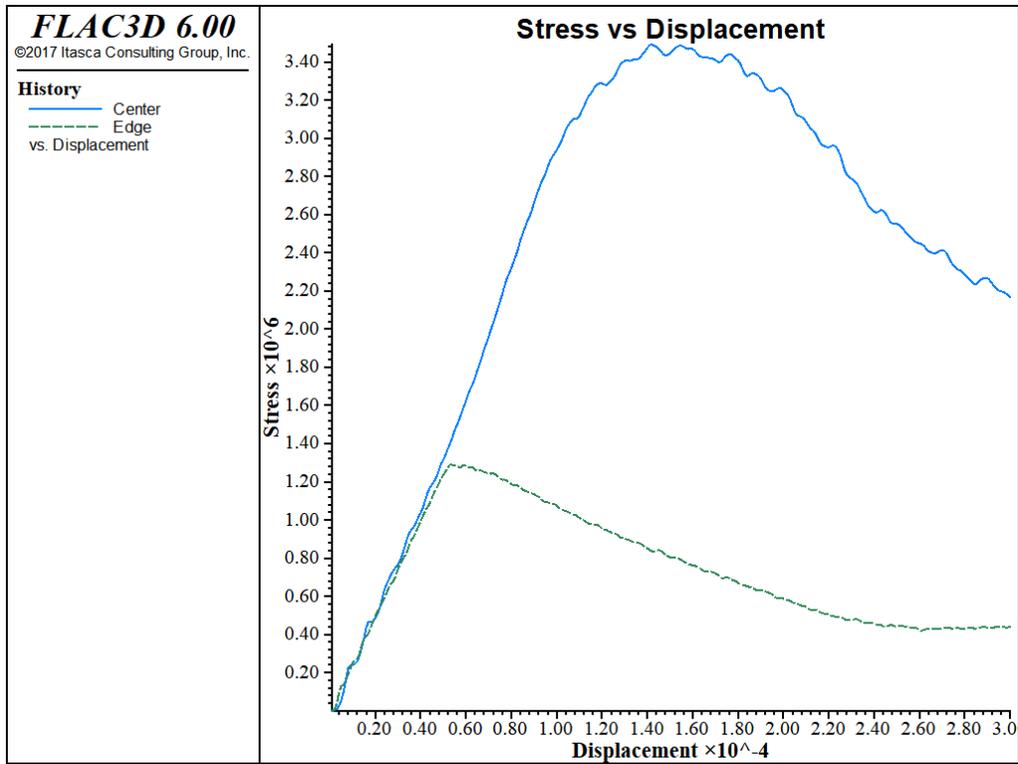


Figure 2: Stress–displacement for compression test of strain–softening material (similar monitoring points to those in the previous image).

The strain–softening model assumes both a brittle softening (due to reduction in cohesion) and a gradual softening (due to a reduction in friction angle). The selection of the properties is discussed further in [Material Properties](#).

Comparison of the two images above illustrates the different responses of the two models. The initial response up to the onset of failure is identical, but post–failure behavior is quite different. Clearly, more data are required to use the strain–softening model and, typically, the softening model must be calibrated for each specific problem.

The effect of confinement on the “measured” response is also demonstrated from these plots. The history recorded in the middle of the grid shows that a higher stress level develops in the center of the model than at the free side. The location of monitoring points should correspond as closely as possible to the location of measurements in the physical problem.

The Effect of Water

Geologic materials generally appear weaker when the pore spaces contain a pore fluid under pressure. This is represented in *FLAC3D* by the incorporation of an effective stress that accounts for the presence of pore pressure in a zone. The pore pressures in *FLAC3D* are taken to be positive in compression. Thus, the effective stress σ' is related to the total stress σ and pore pressure p by

$$\sigma' = \sigma + p$$

Effective stresses are used in all of the plasticity models.

The effect of water can be seen by repeating the [strain-softening example](#) in the preceding topic, but with constant pore pressure in the zones (i.e., an undrained compression test). Add the command

```
; Initial Conditions  
zone gridpoint initialize pore-pressure 1e5
```

to the example before stepping. The lower strength is seen below.

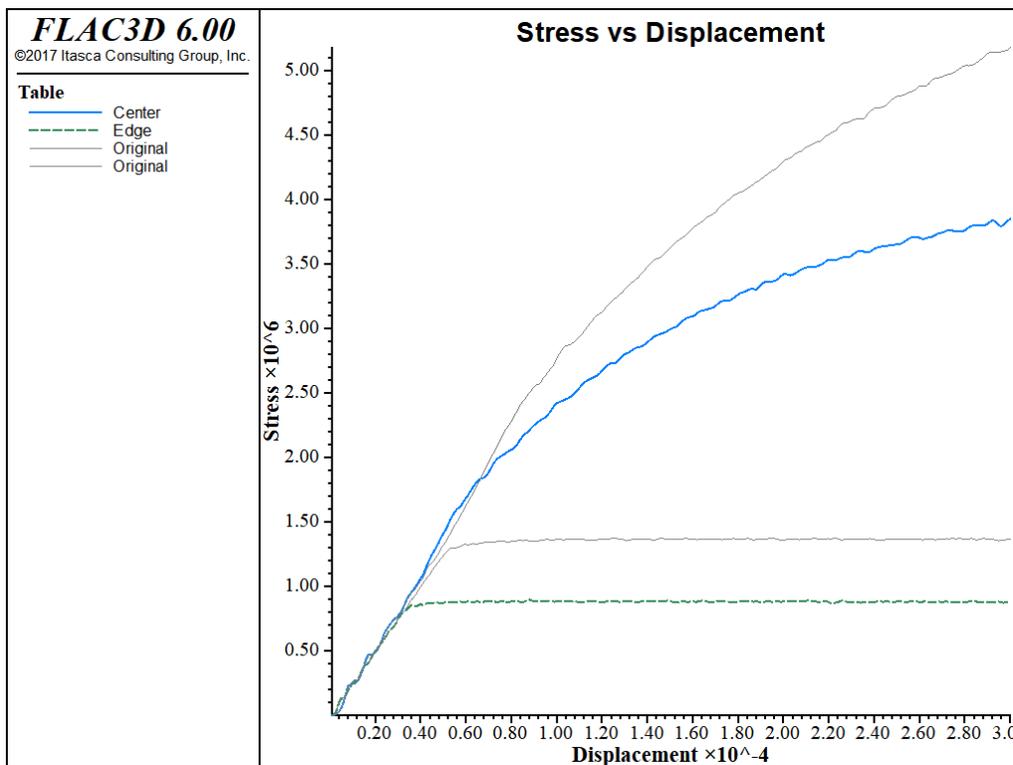


Figure 3: Stress–displacement plots for compression test of Mohr–Coulomb material at constant pore pressure.

Ways to Implement Constitutive Models

There are several different ways a constitutive model can be implemented in *FLAC3D*. The standard way is to invoke one of the built-in models with the `zone cmodel assign` command. The user-defined models described in [Writing New Constitutive Models](#) are also invoked with the `zone cmodel assign` command.

The source codes for the built-in models are included in the “\pluginfiles\cmodels” sub-directory, and are also copied to the application data location provided by the user the first time the program is launched. These files should be reviewed as examples for users who wish to write and implement their own models. User-defined models are written in C++ and are implemented as DLLs that are loaded with the *FLAC3D* executable file in the same manner as the built-in models.[2] (See [Writing New Constitutive Models](#).)

Often, it is desirable to modify an existing constitutive model (either a built-in model or a user-defined model) to make material properties dependent on other model parameters. There are three ways this can be done:

1. change properties of the built-in model via a *FISH* function that scans all the zones and is called at a specified step increment (say, every ten steps);[3]
2. change properties in a user-defined constitutive model function at every step by reference to a formula; or
3. change properties via look-up tables (with *table commands*) that modify strength properties as a function of plastic strain for the built-in strain-softening and double-yield models.

The third approach is the most efficient way to change properties in a *FLAC3D* model. The first approach is the least efficient.

Endnotes

- [1] Users can create their own constitutive models as DLLs by following the procedures given in *Writing New Constitutive Models*. Also available are nine optional models that simulate viscoelastic and viscoplastic, time-dependent (creep) behavior (see *Description of Creep Constitutive Models*), and two optional models to simulate dynamic pore-pressure generation (see *Dynamic Analysis*).
- [2] Note that this approach differs from the *FISH* constitutive model used in two-dimensional *FLAC*. With the DLL approach, user-defined models run at the same calculational speed as built-in models.
- [3] Note that increasing stiffness properties requires that a general update be performed in order to preserve stability. The *zone.do.update* function will force a general update to be performed at the start of the next cycle.

Material Properties

The material properties required in *FLAC3D* are generally categorized in one of two groups: elastic deformability properties and strength properties. This section provides an overview of the deformability and strength properties, and presents guidelines for selecting the appropriate properties for a given model.

Additionally, there are special considerations, such as the definition of post-failure properties, the extrapolation of laboratory-measured properties to the field scale, the spatial variation of properties and randomness of the property distribution, and the dependence of properties on confinement and strain. These topics are also discussed.

The selection of properties is often the most difficult element in the generation of a model because of the high uncertainty in the property database. When performing an analysis, especially in geomechanics, keep in mind that the problem will always involve a data-limited system: the field data will never be known completely. However, with the appropriate selection of properties based upon the available database, important insight to the physical problem can still be gained. This approach to modeling was discussed earlier in [Modeling Methodology](#).

Material properties are conventionally derived from laboratory testing programs. The following sections describe intrinsic (laboratory-scale) properties and list common values for various rocks and soils.

Intrinsic Deformability Properties

Isotropic Elastic Properties — All material models in *FLAC3D*, except for the transversely isotropic elastic and orthotropic elastic models, assume an isotropic material behavior in the elastic range described by two elastic constants: bulk modulus (K) and shear modulus (G). The elastic constants, K and G , are preferred rather than Young's modulus, E , and Poisson's ratio, ν . While both can be used in *FLAC3D*, it is believed that bulk and shear moduli correspond to more fundamental aspects of material behavior than do Young's modulus and Poisson's ratio. (See [tip 13. Use Bulk and Shear Moduli](#) in the topic [Tips and Advice](#) for justification of using (K, G) rather than (E, ν) .)

The equations to convert from (E, ν) to (K, G) are

$$K = \frac{E}{3(1 - 2\nu)}$$

$$G = \frac{E}{2(1 + \nu)}$$

The equations above should be used with caution when ν is near 0.5, because the computed value of K will be unrealistically high and convergence to the solution will be very slow. It is better to fix the value of K at its known physical value (estimated from an isotropic compaction test or from the p -wave speed), and then compute G from K and ν .

Some typical values for elastic constants are summarized in the next table for selected rocks and in the table following for selected soils.

Table 1: Selected Elastic Constants (laboratory-scale) for Rocks (adapted from Goodman 1980)

	Dry Density (kg/m ³)	E (GPa)	ν	K (GPa)	G (GPa)
sandstone		19.3	0.38	26.8	7.0
siltstone		26.3	0.22	15.6	10.8
limestone	2090	28.5	0.29	22.6	11.1
shale	2210 - 2570	11.1	0.29	8.8	4.3
marble	2700	55.8	0.25	37.2	22.3
granite		73.8	0.22	43.9	30.2

Table 2: Selected Elastic Constants (laboratory-scale) for Soils (adapted from Das 1994)

	Dry Density (kg/m ³)	Elastic Modulus E (MPa)	Poisson's Ratio
loose uniform sand	1470	10 - 26	0.2 - 0.4
dense uniform sand	1840	34 - 69	0.3 - 0.45

	Dry Density (kg/m ³)	Elastic Modulus E (MPa)	Poisson's Ratio
loose, angular-grained, silty sand	1630		
dense, angular-grained, silty sand	1940		0.2 - 0.4
stiff clay	1730	6 - 14	0.2 - 0.5
soft clay	1170 - 1490	2 - 3	0.15 - 0.25
loess	1380		
soft organic clay	610 - 820		
glacial till	2150		

Anisotropic Elastic Properties — For the special case of elastic anisotropy, the transversely isotropic, elastic model requires five elastic constants: E_1 , E_3 , ν_{12} , ν_{13} , and G_{13} ; and the orthotropic elastic model requires nine elastic constants: E_1 , E_2 , E_3 , ν_{12} , ν_{13} , ν_{23} , G_{12} , G_{13} , and G_{23} . These constants are defined in **Orthotropic Elastic Model**.

Transversely isotropic elastic behavior is commonly associated with uniformly jointed or bedded rock. Several investigators have developed expressions for the elastic constants in terms of intrinsic isotropic elastic properties and joint stiffness and spacing parameters. A short summary of typical values for anisotropic rocks is given in the following table.

Table 3: Selected Elastic Constants (laboratory-scale) for Transversely Isotropic Rocks (Batugin and Nirenburg 1972)

Rock	E_x (GPa)	E_y (GPa)	ν_{yx}	ν_{zx}	G_{xy} (GPa)
siltstone	43.0	40.0	0.28	0.17	17.0
sandstone	15.7	9.6	0.28	0.21	5.2
limestone	39.8	36.0	0.18	0.25	14.5
gray granite	66.8	49.5	0.17	0.21	25.3
marble	68.6	50.2	0.06	0.22	26.6
sandy shale	10.7	5.2	0.20	0.41	1.2

Fluid Elastic Properties — Models created for groundwater analysis also require the bulk modulus of the water, K_f , if the calculation involves incompressible grains, or the Biot modulus, M , if the grains are compressible. The physical value of K_f is 2 GPa for pure water at room temperature. The value selected should depend on the purpose of the analysis. If only steady-state flow or an initial pore pressure distribution is required (see [Solving Flow-Only and Coupled-Flow Problems](#)), then as low a value of K_f as possible should be used, without compromising the physics. This is because the fluid timestep will be small for high K_f ; also, the mechanical convergence will be very slow for high K_f . The fluid timestep, Δt_f , used by *FLAC3D* is related to porosity, n , permeability, k' , and K_f :

$$\Delta t_f \propto \frac{n}{K_f k'}$$

We can determine the effect of changing K_f by deriving the coefficient of consolidation, C_v , for the case of a deformable fluid. (The fluid is assumed incompressible in most textbook solutions.)

$$C_v = \frac{k'}{m_v + \frac{n}{K_f}}$$

where

$$m_v = \frac{1}{K + 4G/3}$$

and

$$k = k' \gamma_f$$

where: k' = permeability used in *FLAC3D*;
 k = hydraulic conductivity in velocity units (e.g., m/sec); and
 γ_f = the unit weight of water.

Because the consolidation time constant is directly proportional to C_v , the expression allows us to see how much error is introduced by reducing K_f from its real value (of 2×10^9 Pa).

The fluid bulk modulus will also affect the rate of convergence for a model with no flow, but with mechanical generation of pore pressure (see [No Flow — Mechanical Generation of Pore Pressure](#)). If K_f is given a value comparable to the mechanical moduli, pore pressures will be generated as a result of mechanical

deformations. If K_f is much greater than K , then convergence will be slow, but often it is possible to reduce K_f without significantly affecting the behavior. In real soils, for example, pore water will contain some dissolved air, which substantially reduces its apparent bulk modulus.

In the case of no flow, the undrained saturated bulk modulus is

$$K_u = K + \frac{K_f}{n}$$

and the undrained Poisson's ratio is

$$\nu_u = \frac{3K_u - 2G}{2(3K_u + G)}$$

These values should be compared to the drained constants K and ν to evaluate the effect on the rate of convergence. It is important to remember that *drained* properties are used for coupled, mechanical fluid-flow calculations in *FLAC3D*.

For the case of compressible grains, the influence of Biot modulus, M , on the rate of convergence for a model is similar to that of fluid bulk modulus.

Intrinsic Strength Properties

The basic criterion for material failure in *FLAC3D* is the Mohr-Coulomb relation, which is a linear failure surface corresponding to shear failure:

$$f_s = \sigma_1 - \sigma_3 N_\phi + 2c\sqrt{N_\phi}$$

where: $N_\phi = (1 + \sin \phi)/(1 - \sin \phi)$;
 σ_1 = major principal stress (compressive stress is negative);
 σ_3 = minor principal stress;
 ϕ = friction angle; and
 c = cohesion.

Shear yield is detected if $f_s < 0$. The two strength constants, ϕ and c , are conventionally derived from laboratory triaxial tests.

The Mohr-Coulomb criterion loses its physical validity when the normal stress becomes tensile, but for simplicity, the surface is extended into the tensile region to the point at which σ_3 equals the uniaxial tensile strength, σ^t . The minor principal stress can never exceed the tensile strength, i.e.,

$$f_t = \sigma_3 - \sigma^t$$

Tensile yield is detected if $f_t > 0$. Tensile strengths from rock and concrete are usually derived for a Brazilian test. Note that the tensile strength cannot exceed the value of σ_3 , corresponding to the apex limit for the Mohr-Coulomb relation. This maximum value is given by

$$\sigma_{max}^t = \frac{c}{\tan \phi}$$

Typical values of cohesion, friction angle and tensile strength for a representative set of rock specimens are listed in the next table. Cohesion and friction angle values for soil specimens are given in the table following. Strength is often described in terms of the unconfined compressive strength, q_u . The relation between q_u and cohesion, c , and friction angle, ϕ , is given by

$$q_u = 2 c \tan(45 + \phi/2)$$

Table 6: Selected Strength Properties (laboratory-scale) for Rocks (adapted from Goodman 1980)

	Friction Angle (degrees)	Cohesion (MPa)	Tensile Strength (MPa)
Berea sandstone	27.8	27.2	1.17
Repetto siltstone	32.1	34.7	—
Muddy shale	14.4	38.4	—
Sioux quartzite	48.0	70.6	—
Indiana limestone	42.0	6.72	1.58
Stone Mountain granite	51.0	55.1	—
Nevada Test Site basalt	31.0	66.2	13.1

Table 7: Selected Strength Properties (drained, laboratory-scale) for Soils (Ortiz et al. 1980)

	Cohesion (kPa)	Friction Angle Peak (degrees)	Residual (degrees)
gravel	—	4	32
sandy gravel with few	—	35	32

	Cohesion (kPa)	Friction Angle Peak (degrees)	Residual (degrees)
fines			
sandy gravel with silty or clayey fines	1.0	35	32
mixture of gravel and sand with fines	3.0	28	22
uniform sand — fine	—	32	30
uniform sand — coarse	—	34	30
well-graded sand	—	33	32
low-plasticity silt	2.0	28	25
medium- to high-plasticity silt	3.0	25	22
low-plasticity clay	6.0	24	20
medium-plasticity clay	8.0	20	10
high-plasticity clay	10.0	17	6
organic silt or clay	7.0	20	15

Drucker-Prager strength parameters can be estimated from cohesion and friction angle properties. For example, assuming that the Drucker-Prager failure envelope circumscribes the Mohr-Coulomb envelope, the Drucker-Prager parameters q_ϕ and k_ϕ are related to ϕ and c by

$$q_\phi = \frac{6}{\sqrt{3}(3 - \sin \phi)} \sin \phi$$

$$k_\phi = \frac{6}{\sqrt{3}(3 - \sin \phi)} c \cos \phi$$

For further explanation on the relations between Drucker-Prager model parameters, see [Notes on Parameters](#).

The ubiquitous-joint model also requires strength properties for the planes of weakness. Joint properties are conventionally derived from laboratory testing (e.g., triaxial and direct shear tests). These tests can produce physical properties for joint friction angle, cohesion, dilation angle, and tensile strength.

Published strength properties for joints can be found, for example, in Jaeger and Cook (1969), Kulhawy (1975), and Barton (1976). Friction angles can vary from less than 10° for smooth joints in weak rock, such as tuff, to over 50° for rough joints in hard rock, such as granite. Joint cohesion can range from zero cohesion to values approaching the compressive strength of the surrounding rock.

It is important to recognize that joint properties measured in the laboratory typically are not representative of those for real joints in the field. Scale-dependence of joint properties is a major question in rock mechanics. Often, comparison to similar joint properties derived from field tests is the only way to guide the choice of appropriate parameters. However, field test observations are extremely limited. Some results are reported by Kulhawy (1975).

Post-Failure Properties

In many instances, particularly in mining engineering, the response of a material after failure has initiated is an important factor in the engineering design. Consequently, the post-failure behavior must be simulated in the material model. In *FLAC3D*, this is accomplished with properties that define four types of post failure response:

1. shear dilatancy;
2. shear hardening/softening;
3. volumetric hardening/softening; and
4. tensile softening.

These properties are only activated after failure is initiated, as defined by the Mohr-Coulomb relation or the tensile-failure criterion. Shear dilatancy is simulated with the Mohr-Coulomb, ubiquitous-joint and strain-softening Mohr-Coulomb and ubiquitous-joint models. Shear hardening/softening is simulated with the strain-softening Mohr-Coulomb and ubiquitous-joint models, and volumetric hardening/softening is simulated with the modified Cam-clay model. Tensile softening is simulated with the strain-softening Mohr-Coulomb and ubiquitous-joint models.

Shear Dilatancy

Shear dilatancy, or dilatancy, is the change in volume that occurs with shear distortion of a material. Dilatancy is characterized by a dilation angle, ψ , which is related to the ratio of plastic volume change to plastic shear strain. This angle can be specified in the Mohr-Coulomb ubiquitous-joint and strain-hardening/softening models in *FLAC3D*. Dilation angle is typically determined from triaxial tests or shear-box tests. For example, the idealized relation for dilatancy, based upon the Mohr-Coulomb failure surface, is depicted for a triaxial test in the figure below. The dilation angle is found from the plot of volumetric strain versus axial strain. Note that the initial slope for this plot corresponds to the elastic regime, while the slope used to measure the dilation angle corresponds to the plastic regime.

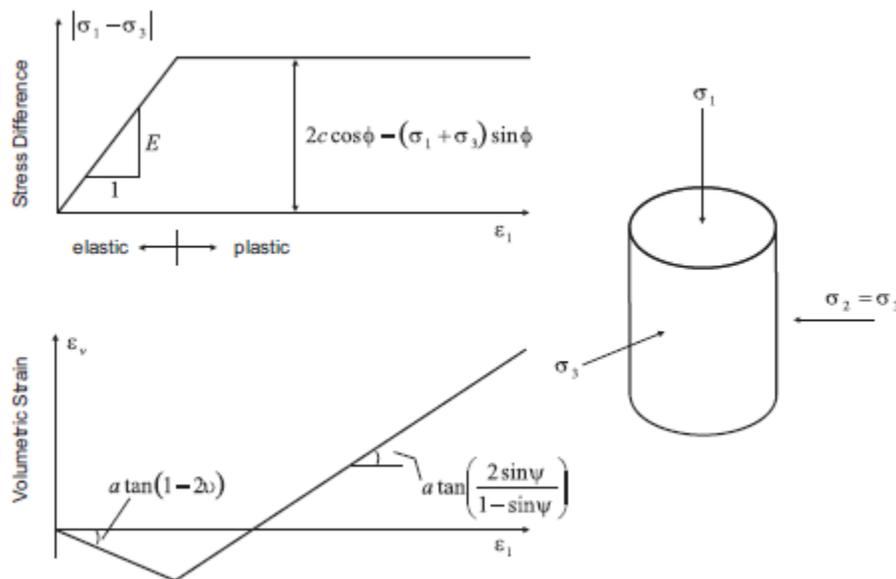


Figure 1: Idealized relation for dilation angle, ψ , from triaxial test results (Vermeer and de Borst 1984).

For soils, rocks, and concrete, the dilation angle is generally significantly smaller than the friction angle of the material. Vermeer and de Borst (1984) report the following typical values for ψ .

Table 8: Typical Values for Dilation Angle (Vermeer and de Borst 1984)

dense sand	15°
loose sand	< 10°
normally consolidated clay	0°
granulated and intact marble	12°–20°
concrete	12°

Vermeer and de Borst observe that values for the dilation angle are approximately between 0° and 20°, whether the material is soil, rock, or concrete. The default value for dilation angle is zero for all the constitutive models in *FLAC3D*.

Dilation angle can also be prescribed for the joints in the ubiquitous-joint model. This property is typically determined from direct shear tests, and common values can be found in the references discussed in [Intrinsic Strength Properties](#).

Shear Hardening/Softening

The initiation of material hardening or softening is a gradual process once plastic yield begins. At failure, deformation becomes more and more inelastic as a result of micro-cracking in concrete and rock, and particle sliding in soil. This also leads to degradation of strength in these materials and the initiation of shear bands. These phenomena, related to localization, are discussed in [Modeling Methodology](#).

In *FLAC3D*, shear hardening and softening are simulated by making Mohr-Coulomb properties (cohesion and friction, along with dilation) functions of plastic strain (see [Strain-Softening/Hardening Mohr-Coulomb Model](#)). These functions are accessed from the strain-softening model, and can be specified either with a [table](#) or via a *FISH* function.

Hardening and softening parameters must be calibrated for each specific analysis, and the values are generally back-calculated from results of laboratory triaxial tests. This is usually an iterative process. Investigators have developed expressions for hardening and softening. For example, Vermeer and de Borst (1984) propose the frictional hardening relation

$$\sin \phi_m = 2 \frac{\sqrt{e_p e_f}}{e_p + e_f} \sin \phi \quad \text{for } e_p \leq e_f$$

$$\sin \phi_m = \sin \phi \quad \text{for } e_p > e_f$$

where ϕ = ultimate friction angle;
 ϕ_m = mobilized friction angle;
 e_p = plastic strain; and
 e_f = parameter.

Cundall (1989) incorporates this relation into two-dimensional *FLAC* to study localization in a frictional material. This is accomplished by approximating the function with a table relating friction angle to plastic strain (i.e., table accessed from property `f table`). This approach is demonstrated for the **Compression test on strain-softening material** example described previously.

Numerical testing conditions can influence the model response for shear hardening/softening behavior. The rate of loading can introduce inertial effects. The results are also grid-dependent. Thus, it is important to evaluate the model behavior for differing zone size and grid orientation whenever performing an analysis involving shear hardening or softening.

The following example illustrates the application of a shear-softening material in a uniaxial compression test. A low velocity is applied to the top and bottom of the specimen, and the grid contains fine zoning. The softening response is shown in the stress-displacement plot in the next figure. The development of localization and shear bands can be observed in the second and third figure following. The plastic region is identified as hourglass-shaped with spiral structures radiating from the hourglass cones.

Uniaxial test of a shear-softening material

```
; Uniaxial test of strain-softening material
model new
model title 'Uniaxial test of strain-softening material'
; Create zones
zone create cylinder point 0 (0,0,0) point 1 (1,0,0) ...
                    point 2 (0,4,0) point 3 (0,0,1) ...
                    size 12 30 12
zone reflect normal (1,0,0)
zone reflect normal (0,0,1)
zone face skin
; Constitutive model and properties
zone cmodel assign strain-softening
```

```

zone property density 2500 bulk 2e8 shear 1e8
zone property cohesion 2e6 friction 45 tension 2e5 dilation 10
zone property table-friction 'fri' table-cohesion 'coh' table-dilation 'dil'
table 'fri' add (0, 45) (.05, 42) (.1, 40) (1, 40)
table 'coh' add (0,2e6) (.05,1e6) (.1,5e5) (1,5e5)
table 'dil' add (0, 10) (.05, 3) (.1, 0)
; Boundary Conditions
zone face apply velocity (0,-2.5e-5,0) range group 'North'
zone face apply velocity (0, 2.5e-5,0) range group 'South'
; Histories
history interval 1
zone history displacement-y position (0,0,0)
zone history displacement-x position (1,1,0)
; FISH to store gridpoints on y=0 surface 'South'
fish define FindSurface
  global surface = map
  loop foreach local gp gp.list
    if gp.isgroup(gp,'South') then
      surface(gp.id(gp)) = gp
    endif
  endloop
end
@FindSurface
; FISH to track AxialStress on y=0 surface 'South'
fish define AxialStress
  local str = 0
  loop foreach local gp surface
    str = str + gp.force.unbal.y(gp)
  endloop
  AxialStress = str / math.pi ; cylinder radius = 1
end
fish history @AxialStress
model step 5000
; Plot of plastic region as zones with strain > 0.2
fish define TagPlasticZones
  loop foreach local zp zone.list
    if zone.prop(zp,'strain-shear-plastic') > 0.2
      zone.group(zp) = 'Yield'
    else
      zone.group(zp) = 'Other'
    endif
  endloop
end
@TagPlasticZones

```

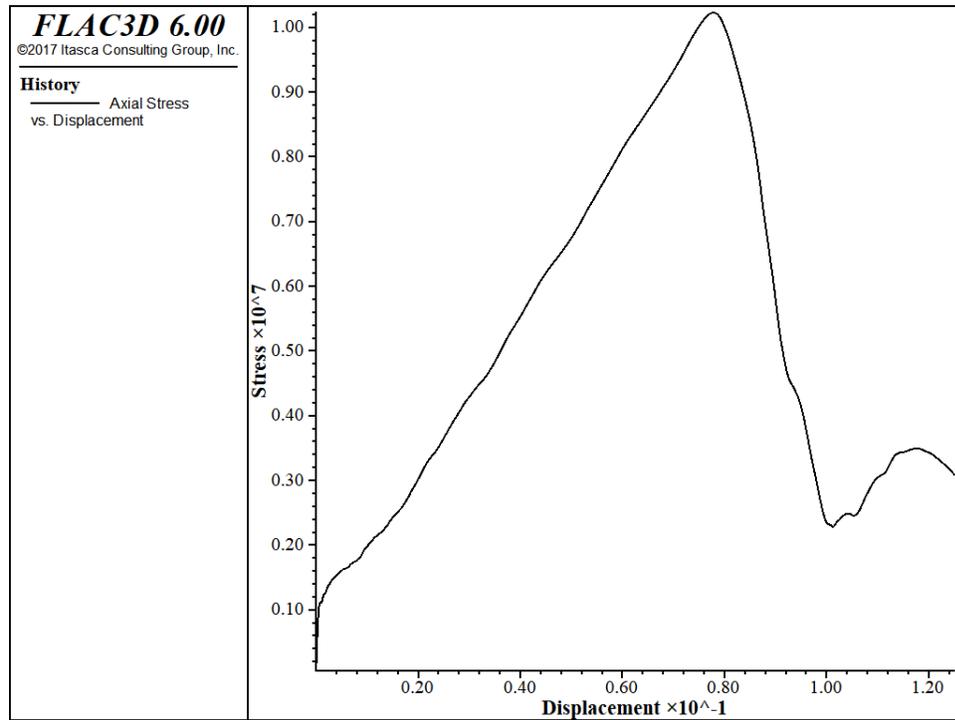


Figure 2: Stress–displacement plot for uniaxial test of shear–softening material

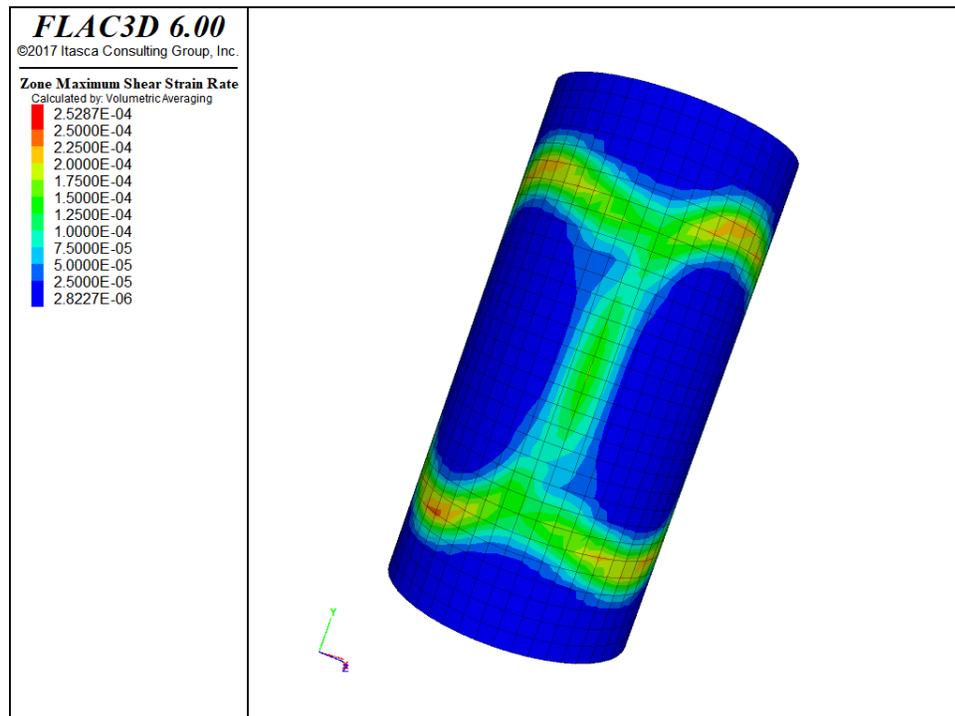


Figure 3: Contours of shear–strain rate indicating shear bands in strain–softening material.

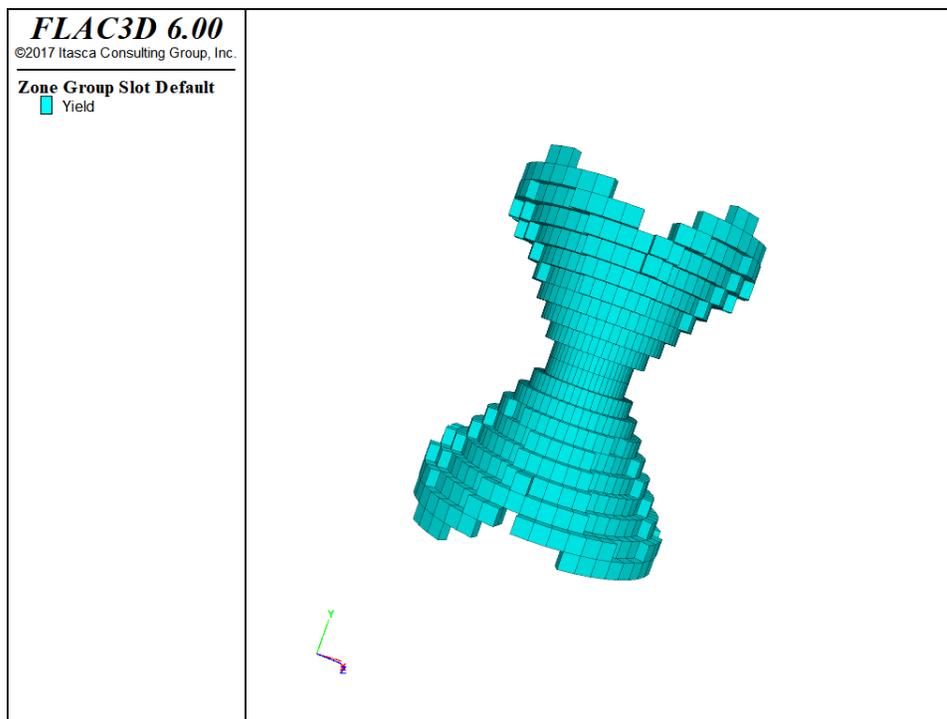


Figure 4: Yielded region identified as zones with plastic shear strain > 0.2 .

Volumetric Hardening/Softening

Volumetric hardening corresponds to irreversible compaction; increasing the isotropic pressure can cause permanent volume decrease. This behavior is common in materials such as lightly cemented sands and gravels and over-consolidated clays.

Volumetric hardening/softening may be simulated in the double-yield model and the Cam-clay model. The double-yield model assumes that the hardening depends on plastic volume strain, while the Cam-clay model treats volumetric hardening as a function of both shear and volume strain.

The double-yield model takes its hardening rule from either a *table* or a *FISH* function. [This example](#) describes a recommended test procedure to develop these parameters from triaxial tests performed at constant mean stress, and for loading in which axial stress and confining pressure are kept equal. Typically, though, these data are not available. An alternative is to back-calculate the parameters from a uniaxial strain (or oedometer) test.

The modified Cam-clay model is defined by initial elastic moduli plus parameters that prescribe the nonlinear elasticity and hardening/softening behavior. The properties include the following:

κ	slope of the elastic swelling line
λ	slope of the normal consolidation line
M	material constant
p_c	pre-consolidation pressure
p_1	reference pressure
v_0	initial specific volume
v_λ	specific volume at reference pressure on normal consolidation line

The definition of these properties can be found in soil mechanics texts (e.g., Wood 1990). The procedures for determining these properties from laboratory tests are described in [Determination of Input Parameters](#). It is recommended that single-zone tests be conducted with *FLAC3D* to exercise the modified Cam-clay model, and verify whether the choice of model properties is adequate. An example test is provided in the [isotropic consolidation example](#).

Tensile Softening

At the initiation of tensile failure, the tensile strength of a material will generally drop to zero. The rate at which the tensile strength drops, or tensile softening occurs, is controlled by the plastic tensile strain in *FLAC3D*. This function is accessed from the strain-softening model and can be specified either with a [table](#) or via a *FISH* function.

A simple tension test illustrates the tensile-softening behavior. The model is the same as that used previously in the [compression test on Mohr-Coulomb material](#) and the [compression test on strain-softening material](#) in the topic [Selection of an Appropriate Model](#). The ends of the cylindrical sample are now pulled apart at a constant velocity.

Tension test on tensile-softening material

```

model new
; Create zones
zone create cylinder point 0 (0,0,0) point 1 (1,0,0) ...
                    point 2 (0,2,0) point 3 (0,0,1) ...
                    size 4 5 4
zone reflect normal (1,0,0)
zone reflect normal (0,0,1)
zone face skin
; Assign constitutive model and properties
zone cmodel assign strain-softening
zone property bulk 1.19e10 shear 1.1e10

```

```

zone property cohesion 2.72e5 friction 44 tension 2e5
zone property table-tension 'ten'
table 'ten' add (0,2e5) (2e-5,0)
; Boundary conditions
zone face apply velocity (0, 1e-8,0) range group 'North'
zone face apply velocity (0,-1e-8,0) range group 'South'
; Histories
history interval 1
zone history displacement-y position (0,0,0)
zone history displacement-x position (1,1,0)
; FISH to store gridpoints on y=0 surface 'South'
fish define FindSurface
  global surface = map
  loop foreach local gp gp.list
    if gp.isgroup(gp,'South') then
      surface(gp.id(gp)) = gp
    endif
  endloop
end
@FindSurface
; FISH to track AxialStress on y=0 surface 'South'
fish define AxialStress
  local str = 0
  loop foreach local gp surface
    str = str + gp.force.unbal.y(gp)
  endloop
  AxialStress = str / math.pi ; cylinder radius = 1
end
fish history @AxialStress
model step 1500

```

The plot of axial stress versus axial displacement (the next figure) shows that the average axial stress through the center of the model drops to zero. (The stress will remain constant in the non-softening Mohr-Coulomb model.) The model decreases in diameter until tensile failure initiates, then expands as tensile softening occurs (see the second figure below). The brittleness of the tensile softening is controlled by the plastic tensile-strain function. As with the shear hardening/softening model, the tensile-softening model must be calibrated for each specific problem and grid size.

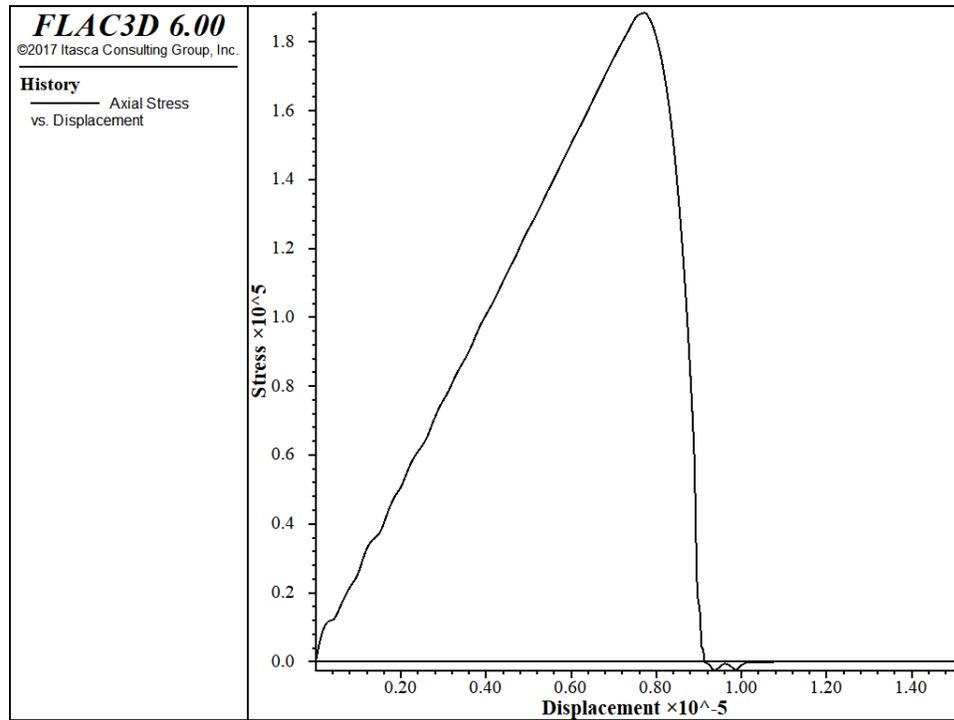


Figure 5: Axial stress versus axial displacement for tensile test of tension-softening material.

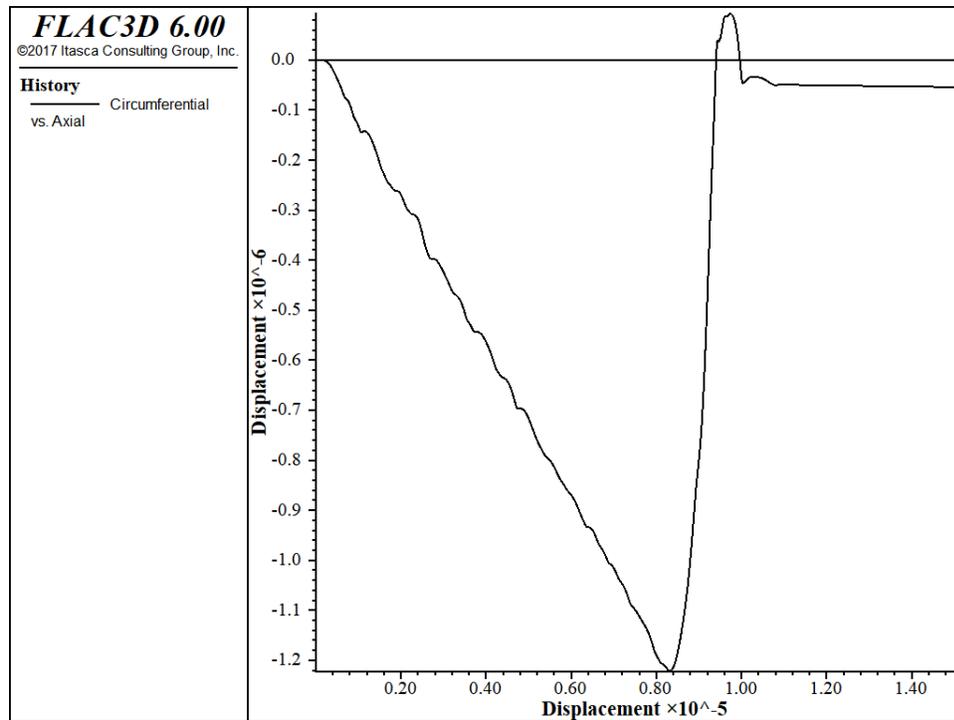


Figure 6: Circumferential displacement versus axial displacement for tensile test of tension-softening material.

Volume-Pressure Properties

The double-yield and modified Cam-clay models require material properties that relate pressure to volumetric change. For both models, the pre-consolidation pressure is the maximum past consolidation pressure. In the double-yield model, the relation between pressure and volumetric change is expressed by a table relating the cap pressure to plastic volume strain. In the Cam-clay model, the pressure-volume relation is expressed by the slopes of the initial compression line and the reloading-unloading line for the plot of volumetric strain versus natural log of pressure. The recommended procedure for selecting volumetric properties for the double-yield model is given in [Choice of Volumetric Properties](#); the procedure for the Cam-clay model is given in [Implementation Procedure](#).

Extrapolation to Field-Scale Properties

The material properties used in the *FLAC3D* model should correspond as closely as possible to the actual values of the physical problem. Particularly in rock, the laboratory-measured properties generally should not be used directly in a *FLAC3D* model for a full-scale problem. These properties should be scaled to account for the presence of discontinuities and heterogeneities present in a rock mass.

Several empirical approaches have been proposed to derive field-scale properties. Some of the more commonly accepted methods are discussed.

Deformability of a rock mass is generally defined by a modulus of deformation, E_m . If the rock mass contains a set of relatively parallel, continuous joints with uniform spacing, the value for E_m can be estimated by treating the rock mass as an equivalent transversely isotropic continuum. The following relation can then be used to estimate E_m in the direction normal to the joint set.

$$\frac{1}{E_m} = \frac{1}{E_r} + \frac{1}{k_n s}$$

where: E_m = rock mass Young's modulus;
 E_r = intact rock Young's modulus;
 k_n = joint normal stiffness; and
 s = joint spacing.

A similar expression can be derived for shear modulus:

$$\frac{1}{G_m} = \frac{1}{G_r} + \frac{1}{k_s s}$$

where: G = rock mass shear modulus;
 G_r = intact rock shear modulus; and
 k_s = joint shear stiffness.

The equivalent continuum assumption, when extended to three orthogonal joint sets, produces the following relations:

$$E_i = \left(\frac{1}{E_r} + \frac{1}{s_i k_{ni}} \right)^{-1} \quad (i = 1, 2, 3)$$

$$G_{ij} = \left(\frac{1}{G_r} + \frac{1}{s_i k_{si}} + \frac{1}{s_j k_{sj}} \right)^{-1} \quad (i, j = 1, 2, 3)$$

Several expressions have been derived for two- and three-dimensional characterizations and multiple joint sets. References for these derivations can be found in Singh (1973), Gerrard (1982a and b), and Fossum (1985).

In practice, the rock mass structure is often much too irregular, or sufficient data are not available, to use the preceding approach. It is common to determine E_m from a force-displacement curve obtained from an in-situ compression test. Such tests include plate-bearing tests, flatjack tests, and dilatometer tests.

Bieniawski (1978) developed an empirical relation for E_m based upon field test results at sites throughout the world. The relation is based upon rock mass rating (RMR). For rocks with a rating higher than 55, the test data can be approximately fit to

$$E_m = 2(\text{RMR}) - 100$$

The units of E_m are GPa.

For values of E_m between 1 and 10 GPa, Serafim and Pereira (1983) found a better fit, given by

$$E_m = 10^{\frac{\text{RMR}-10}{40}}$$

References by Goodman (1980) and Brady and Brown (1985) provide additional discussion on these methods.

The most commonly accepted approach to estimate rock mass strength is that proposed by Hoek and Brown (1980). The generalized Hoek-Brown failure criterion for jointed rock masses is defined by (Hoek and Brown 1997):

$$\sigma'_1 = \sigma'_3 + \sigma_{ci} \left(m_b \frac{\sigma'_3}{\sigma_{ci}} + s \right)^a \quad (1)$$

where σ'_1 and σ'_3 are the maximum and minimum effective stress at failure, respectively (compressive stresses are positive), m_b is the value of the Hoek-Brown constant m for the rock mass, s and a are constants that depend upon the characteristics of the rock mass, and σ_{ci} is the uniaxial compressive strength of the intact rock pieces.

The tensile strength of the rock mass is estimated by Hoek and Brown (1997) to be

$$\sigma_{tm} = \frac{\sigma_{ci}}{2} \left(m_b - \sqrt{m_b^2 + 4s} \right) \quad (2)$$

Three properties are required in order to use the Hoek-Brown criteria (Equations (1) and (2) above):

1. the uniaxial compressive strength, σ_{ci} , of the intact rock pieces;
2. the Hoek-Brown constant, m_i , for the intact rock pieces; and
3. the value of the Geological Strength Index, GSI, for the rock mass. This provides an estimate for the Hoek-Brown constants, s , m_b , and a .

For the intact rock pieces that compose the rock mass, Equation (1) becomes

$$\sigma'_1 = \sigma'_3 + \sigma_{ci} \left(m_i \frac{\sigma'_3}{\sigma_{ci}} + 1 \right)^{0.5}$$

The values for σ_{ci} and the constant m_i should be determined by statistical analysis of the results of a set of triaxial tests. The recommended procedure is given by Hoek and Brown (1997). This paper also presents estimates of σ_{ci} and m for preliminary design calculations when laboratory test results may not be available.

The Geological Strength Index (GSI) provides a system for estimating the reduction in rock mass strength. Two tables from Hoek and Brown (1997) can be used to estimate GSI (see the two tables below). Hoek and Brown (1997) then give the following relation between GSI and m_b , s and a :

$$m_b = m_i \exp\left(\frac{GSI - 100}{28}\right)$$

For GSI > 25,

$$s = \exp\left(\frac{GSI - 100}{9}\right)$$

and

$$a = 0.5$$

For GSI < 25,

$$s = 0$$

and

$$a = 0.65 - \frac{GSI}{200}$$

Hoek and Brown (1997) also provide tables for these equations to facilitate their use.

It is possible to estimate Mohr-Coulomb friction angle and cohesion from the Hoek-Brown criterion (see, for example, Hoek 1990). For a given value of σ_3 , a tangent to Equation (1) will represent an equivalent Mohr-Coulomb yield criterion in the form

$$\sigma_1' = N_\phi \sigma_3' + \sigma_c^M$$

where $N_\phi = \frac{1+\sin\phi}{1-\sin\phi} = \tan^2\left(\frac{\phi}{2} + 45^\circ\right)$.

By substitution, σ_c^M is

$$\sigma_c^M = \sigma_1 - \sigma_3 N_\phi = \sigma_3 + \sqrt{\sigma_3 \sigma_c m + \sigma_c^2 s} - \sigma_3 N_\phi = \sigma_3(1 - N_\phi) + \sqrt{\sigma_3 \sigma_c m + \sigma_c^2 s}$$

σ_c^M is the apparent uniaxial compressive strength of the rock mass for that value of σ_3 .

The tangent to Equation (1) is defined by

$$N_\phi(\sigma_3) = \frac{\partial\sigma_1}{\partial\sigma_3} = 1 + \frac{\sigma_c m}{2\sqrt{\sigma_3 \cdot \sigma_c m + s\sigma_c^2}}$$

The cohesion (c) and friction angle (ϕ) can then be obtained from N_ϕ and σ_c^M :

$$\phi = 2 \tan^{-1} \sqrt{N_\phi} - 90^\circ$$

$$c = \frac{\sigma_c^M}{2\sqrt{N_\phi}}$$

ROCK MASS CHARACTERISTICS FOR STRENGTH ESTIMATES		SURFACE CONDITIONS				
<p>Based upon the appearance of the rock, choose the category that you think gives the best description of the 'average' undisturbed in situ conditions. Note that exposed rock faces that have been created by blasting may give a misleading impression of the quality of the underlying rock. Some adjustment for blast damage may be necessary and examination of diamond drill core or of faces created by pre-split or smooth blasting may be helpful in making these adjustments. It is also important to recognize that the Hoek-Brown criterion should only be applied to rock masses where the size of individual blocks is small compared with the size of the excavation under consideration.</p>		VERY GOOD Very rough, fresh unweathered surfaces	GOOD Rough, slightly weathered, iron stained surfaces	FAIR Smooth, moderately weathered or altered surfaces	POOR Slickensided, highly weathered surfaces with compact coatings or fillings of angular fragments	VERY POOR Slickensided, highly weathered surfaces with soft clay coatings or fillings
		STRUCTURE				
STRUCTURE		DECREASING SURFACE QUALITY ▾				
	BLOCKY - very well interlocked undisturbed rock mass consisting of cubical blocks formed by three orthogonal discontinuity sets	B/VG	B/G	B/F	B/P	B/VP
	VERY BLOCKY - interlocked, partially disturbed rock mass with multifaceted angular blocks formed by four or more discontinuity sets	VB/VG	VB/G	VB/F	VB/P	VB/VP
	BLOCKY/DISTURBED- folded and/or faulted with angular blocks formed by many intersecting discontinuity sets	BD/VG	BD/G	BD/F	BD/P	BD/VP
	DISINTEGRATED - poorly interlocked, heavily broken rock mass with a mixture of angular and rounded rock pieces	D/VG	D/G	D/F	D/P	D/VP

Figure 7: Characterization of rock masses on the basis of interlocking and joint alteration—from Hoek and Brown (1997).

ROCK MASS CHARACTERISTICS FOR STRENGTH ESTIMATES		SURFACE CONDITIONS				
<p>Based upon the appearance of the rock, choose the category that you think gives the best description of the 'average' undisturbed in situ conditions. Note that exposed rock faces that have been created by blasting may give a misleading impression of the quality of the underlying rock. Some adjustment for blast damage may be necessary and examination of diamond drill core or of faces created by pre-split or smooth blasting may be helpful in making these adjustments. It is also important to recognize that the Hoek-Brown criterion should only be applied to rock masses where the size of individual blocks is small compared with the size of the excavation under consideration.</p>		<p>VERY GOOD Very rough, fresh unweathered surfaces</p>				
		<p>GOOD Rough, slightly weathered, iron stained surfaces</p>				
STRUCTURE		<p>FAIR Smooth, moderately weathered or altered surfaces</p>				
		<p>POOR Slickensided, highly weathered surfaces with compact coatings or fillings of angular fragments</p>				
		<p>VERY POOR Slickensided, highly weathered surfaces with soft clay coatings or fillings</p>				
		<p>DECREASING SURFACE QUALITY ▼</p>				
 <p>BLOCKY - very well interlocked undisturbed rock mass consisting of cubical blocks formed by three orthogonal discontinuity sets</p>	B/VG	B/G	B/F	B/P	B/VP	
 <p>VERY BLOCKY - interlocked, partially disturbed rock mass with multifaceted angular blocks formed by four or more discontinuity sets</p>	VB/VG	VB/G	VB/F	VB/P	VB/VP	
 <p>BLOCKY/DISTURBED- folded and/or faulted with angular blocks formed by many intersecting discontinuity sets</p>	BD/VG	BD/G	BD/F	BD/P	BD/VP	
 <p>DISINTEGRATED - poorly interlocked, heavily broken rock mass with a mixture of angular and rounded rock pieces</p>	D/VG	D/G	D/F	D/P	D/VP	
		<p>DECREASING INTERLOCKING OF ROCK PIECES ▼</p>				

Figure 8: Estimate of Geological Strength Index (GSI) based on geological descriptions—from Hoek and Brown (1997).

Spatial Variation and Randomness of Property Distribution

Material properties can be specified to adjust or to vary as a function of grid position. In fact, a different property can be assigned to every zone in a *FLAC3D* model, regardless of model size.

There are two commands available in *FLAC3D* to specify material properties. The first, `zone property`, allows multiple properties to be specified in a single command. This is convenient to completely specify properties in different regions, where that property does not vary within that region.

The second can only specify one property at a time, but has optional keywords that allow the property value to be automatically varied in space. This is the `zone property-distribution` command. Among the available keywords are `add`, `multiply`, `gradient`, and `vary`, which can be used to provide fixed or linear variations of properties with position. For example, the following command provides a linear variation of cohesion in the x -direction:

```
zone property-distribution cohesion 1e6 gradient (-1e5,0,0)
```

An initial profile of a property can also be assigned via *FISH*.

With *FLAC3D*, it is also possible to study the influence of nonhomogeneity in a material. Any type of statistical property distribution can be introduced, as each element may have a unique property value. Two optional keywords are available to apply a random distribution of a selected property with the `zone property-distribution` command. The keyword `deviation-gaussian` assumes a normal (Gaussian) distribution for the property, with a mean value, ν , and standard deviation, s . The keyword `deviation-uniform` assumes a uniform distribution with mean value, ν , and standard deviation, s . Be careful to ensure that properties do not acquire negative values if s is large. As an example, the following command would give a mean friction angle of 40° with a standard deviation of $\pm 5\%$:

```
zone property-distribution friction 45 deviation-gaussian 2
```

Use of the variation keywords in any command (not just for property variation) is defined in the topic [Value Modifiers \(Add, Multiply, Gradient, Vary\)](#).

Boundary Conditions

The boundary conditions in a numerical model consist of the values of field variables (e.g., stress and displacement) that are prescribed at the boundary of the numerical grid. Boundaries are of two categories: real and artificial. Real boundaries exist in the physical object being modeled (e.g., a tunnel surface or the ground surface). Artificial boundaries do not exist in reality, but they must be introduced in order to enclose the chosen number of zones. The conditions that can be imposed on each type are similar; these conditions are discussed first. Then (in [Artificial Boundaries](#)) some suggestions concerning the location and choice of artificial boundaries and the effect they have on the solution are made.

Mechanical conditions that can be applied at boundaries are of two main types: prescribed displacement or prescribed stress. A free surface is a special case of the prescribed-stress boundary. The two types of mechanical conditions are described in [Stress Boundary](#) and [Displacement Boundary](#).

All examples listed in this section can be found in the project file “BoundaryConditions.f3prj.”

Stress Boundary

By default, the boundaries of a *FLAC3D* grid are free of stress and any constraint. Forces or stresses may be applied to any boundary, or part of a boundary, by means of the `zone face apply` command. Individual components of the stress tensor are specified with the `stress-xx`, etc., keywords. For example, the commands

```
zone face apply stress-zz -1e5 range position-z 0
zone face apply stress-xz -.5e5 range position-z 0
```

apply the given σ_{zz} and σ_{xz} components of a stress tensor to all boundary zone faces with centroids that fall within the range $-0.1 \leq z \leq 0.1$. All other stress components are zero.

Stress can be applied either in the global model x,y,z -directions, or in directions normal and tangential to the local boundary face. The keyword `stress-normal` applies a normal stress to a face, while keywords `stress-dip` and `stress-strike` apply shear stresses to the face. `stress-dip` is the stress component applied in the dip direction of the local face, and `stress-strike` is the stress component applied in the strike direction. The orientation of the local face axes is illustrated in the next figure. Note that global (x,y,z) -axes stresses and local (d,s,n) -axes stresses cannot both be applied to the same face.

There are several things to note about this use of `zone face apply` command. First, only those faces inside the range defined by `range` will be affected by the `zone face apply` command, and as with all face selection geometric range elements, will select by the location of the face centroid. Second, compressive stresses have a *negative* sign, in accordance with the **general sign convention** for internal stresses in *FLAC3D*. Finally, *FLAC3D* actually applies the stress components as forces, or *tractions*, which result from a stress tensor acting on the given boundary plane; the tractions are computed whenever a `model solve` command is issued, and again every tenth step in large-strain mode.

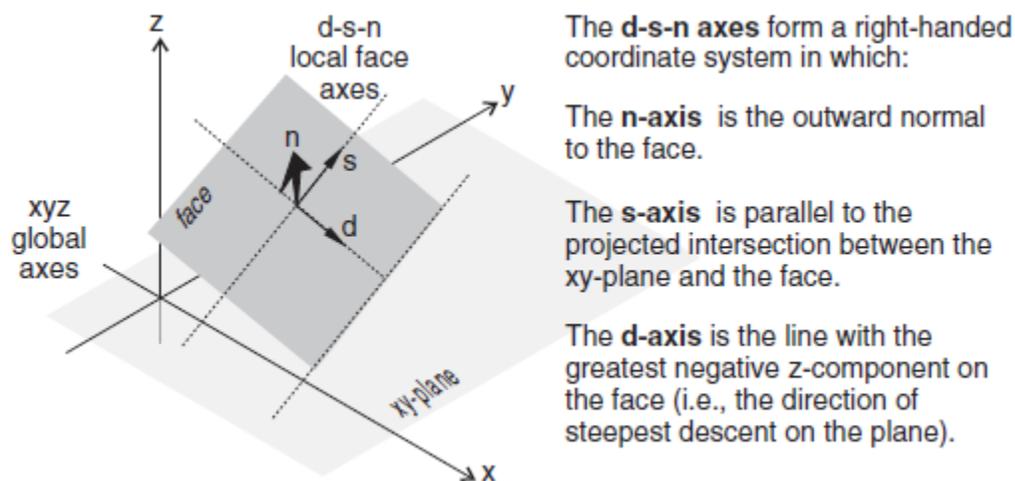


Figure 1: Local face axes defined by (d) dip direction, (s) strike direction, and (n) normal direction.

Individual forces can also be applied to the grid by using the `zone gridpoint fix` command and the `force-applied-x`, `force-applied-y`, and `force-applied-z` keywords, which specify the x -, y -, and z -components of an applied force vector

(all forces may be specified at once using `force-applied v`). In this case, no account is taken of the boundary face area; the specified forces are simply applied to the given gridpoints.

The applied forces, or tractions calculated from applied stresses, can be viewed with the *Zone Vector* plot item. It is necessary to perform a `model step 0` first in order to calculate the applied forces for viewing. For example, a stress boundary condition is applied normal to a boundary plane that has a dip angle of 60° and dip direction of 270° . Cut and paste the commands shown in the next example into a data file.

Name of example: Applying a normal stress to a dipping boundary

```
model new
zone create brick size (4,4,4) point 0 (0,0,0) point 1 (4,0,0) ...
                                point 2 (0,4,0) point 3 (2,0,3.464)
zone cmodel assign elastic
zone property bulk 1e8 shear .3e8
zone face apply stress-normal -1e6 ...
                                range plane dip 60 dip-direction 270 origin 0.1,0,0 above
```

The normal stress of -1×10^6 is applied to a boundary plane falling within the range defined by a plane with a dip direction of 270° , a dip angle of 60° , and above the position $x = 0.1$, $y = 0$, $z = 0$. The resulting applied forces can be seen by plotting a “Zone->Vectors” plot item, and setting the “Applied Force” option on the “Value” attribute.

Note that by using the `zone face skin` command, the desired boundary can be identified easily without the need to mathematically describe the plane, as seen below:

```
model new
zone create brick size (4,4,4) point 0 (0,0,0) point 1 (4,0,0) ...
                                point 2 (0,4,0) point 3 (2,0,3.464)
zone face skin
zone cmodel assign elastic
zone property bulk 1e8 shear .3e8
zone face apply stress-normal -1e6 range group 'West'
```

The plot below shows the forces applied to the grid as calculated from the normal stress and the boundary face areas. Note that the size of the force vectors is a function of the face areas over which the stress is applied.

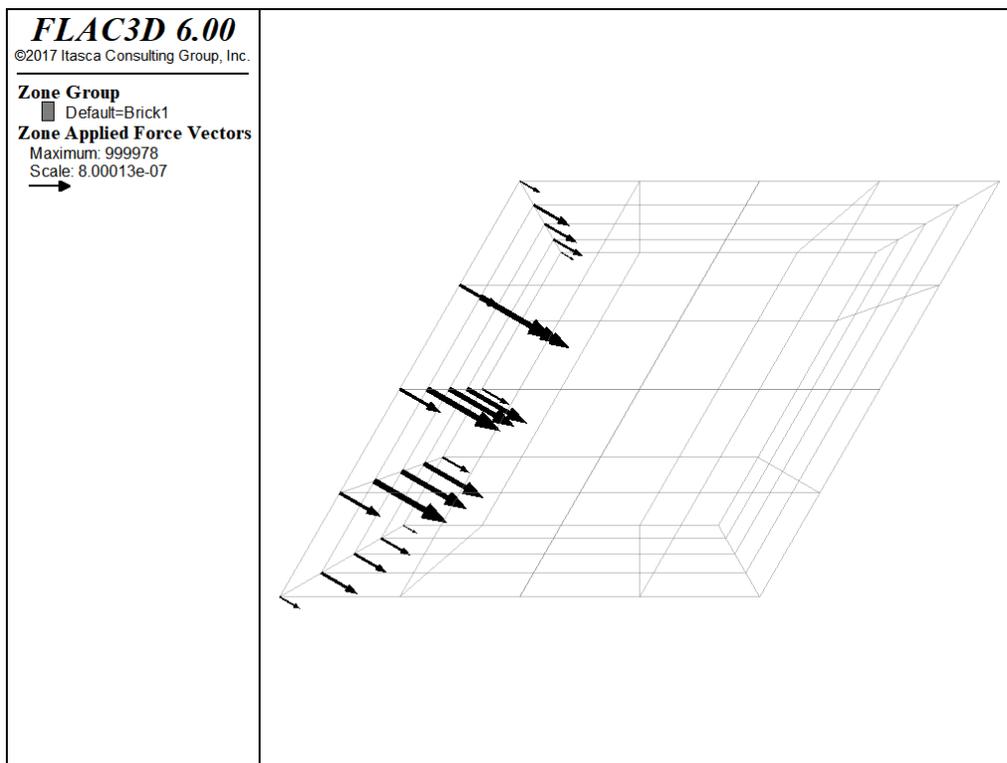


Figure 2: Applied forces resulting from zone face apply stress-normal.

Applied Stress Gradients

The `zone face apply` command may take the optional keyword `gradient`, which allows the applied stresses or forces to vary linearly over the specified range. The parameter following `gradient`, `v`, specifies the x -, y -, and z -variation, respectively, for the stress- or force-component. The value varies linearly with distance from the global coordinate origin at ($x = 0, y = 0, z = 0$):

$$S = S^{(o)} + g_x x + g_y y + g_z z$$

in which $S^{(o)}$ is the value at the global coordinate origin at ($x = 0, y = 0, z = 0$), and g_x, g_y , and g_z specify the variation of the value in the x -, y -, and z -directions.

The operation of this feature is best explained by an example:

```
zone face apply stress-xx -10e6 gradient (0,0,1e5) range position-z -100 0
```

The equation for the z -variation in stress-component σ_{xx} is

$$\sigma_{xx} = -10 \times 10^6 + (10^5)z$$

The σ_{xx} -stress at the origin ($x = 0, y = 0, z = 0$) is $\sigma_{xx} = -10 \times 10^6$; at $z = -100$, it is -20×10^6 . At points in between, the stress is linearly scaled to the relative z -distance from the origin.

Another option is to use the `vary` keyword, which may be more familiar from *FLAC*. See [Value Modifiers \(Add, Multiply, Gradient, Vary\)](#) for further discussion of both options.

Typically, applied stress gradients are used to reproduce the effects of increasing stress with depth caused by gravity. It is important to make sure that the applied gradient is compatible with the stress gradient specified with the `zone initialize-stresses` command. See [Initial Conditions](#) for further comments on stress gradients.

Changing Boundary Stress

It may be necessary to alter the values of applied stresses during the course of a *FLAC3D* simulation. For example, the load on a footing may change. To effect a sudden change in an existing applied stress or load, a new `zone face apply` command is given, with the range and stress component given *exactly as in the original command*, but with a changed value or variation. In this case, *FLAC3D* will detect a conflict in the `apply` condition already present and remove it. New values will replace existing values for any overlapping `zone face apply` commands. It may be necessary to first remove boundary conditions (with `zone face apply-remove`) before updating a boundary stress.

In many instances, it may be necessary to change a boundary stress gradually. This is often required to minimize the shock to a sensitive system, especially if “path-dependence” of the solution is important (see [Localization, Physical Instability and Path-Dependence](#)). The `zone face apply` command includes options to make gradual application (or removal) of boundary conditions simple and straightforward. For example, to increase the applied stress incrementally while maintaining quasi-equilibrium, you can use the `ramp` option to the `servo` keyword:

```
zone face apply stress-xx -1e5 servo ramp range group 'East' position-z 0,2
```

Apply changing stress boundary with a *FISH* function

The `servo` keyword and its various options is only one way to change the value of an applied boundary over time. Either *FISH* or `tables` can be used to add a multiplier to the value. Each step, the table is queried or the *FISH* function is called, and the value of the applied boundary will be multiplied by the return. A return of `1.0` will therefore have no effect, and a return value of `0.0` will effectively remove the boundary (in the case of stress boundaries). The following is a simple example of using a short *FISH* function to vary the stress applied sinusoidally.

```
fish define apply
  apply = math.sin(dynamic.time.total*10)
end
zone face apply stress-xx -1e5 fish @apply range group 'East' position-z 0,2
```

Apply changing stress boundary with table data

The following example creates a simple table that is used to gradually ramp the stress boundary up from nothing. Note that for a `table` multiplier, you must specify the time value to use as the *x* lookup. In this case, we specify `dynamic time`. By default, a table multiplier will use the current total step number as the *x* lookup value.

```
table 'ramp' add (0,0) (1,1)
zone face apply stress-xx -1e5 table 'ramp' time dynamic ...
  range group 'East' position-z 0,2
```

Gradual excavation of a circular tunnel

Because *FLAC3D* uses physics to inform the path it takes to static convergence, sudden changes to the model can have quasi-inertial effects that may artificially exaggerate damage in the area. One way to mitigate this is to gradually excavate regions, so that the effect of zone removal is less sudden. The `zone relax excavate` command was created for this purpose. This command is a zone-based apply condition that gradually reduces the stress, stiffness, and density of the zones in the range until they have effect on the model and are then set to the null constitutive model.

By default, `zone relax` uses a `servo` to automatically maintain quasi-equilibrium, however the command has any number of options for controlling the exact method of reducing the influence of zones tagged for excavation.

The following example creates a simple circular tunnel. This tunnel is both excavated instantly and gradually via `zone relax excavate`.

```

model new
; Create zones
zone create radial-cylinder point 0 0 0 0 point 1 add 18 0 0 ...
                             point 2 add 0 8 0 point 3 add 0 0 18 ...
                             dim 1.75 1.75 1.75 1.75 ratio 1.0 .83 1.0 1.2 ...
                             size 6 6 6 15 fill group 'exc'
zone create radial-cylinder point 0 0 8 0 point 1 add 18 0 0 ...
                             point 2 add 0 8 0 point 3 add 0 0 18 ...
                             dim 1.75 1.75 1.75 1.75 ratio 1.0 1.2 1.0 1.2 ...
                             size 6 6 6 15 fill

zone face skin
; Assign constitutive models and properties
zone cmodel assign mohr-coulomb
zone property bulk 33.33e9 she 25e9 fric 45 cohesion 15e6 ten 5e6
; Assign Boundary Conditions
zone face apply velocity-normal 0 range group 'West' or 'East'
zone face apply velocity-normal 0 range group 'South' or 'North'
zone face apply velocity-normal 0 range group 'Bottom' or 'Top'
; Initial Conditions
zone initialize stress xx -65e6 yy -65e6 zz -65e6
model solve
model save 'initial'
; instantaneous excavation
zone delete range group 'exc'
model solve
model save 'instant'
; gradual excavation
model restore 'initial'
zone relax excavate range group 'exc'
model solve
model save 'relax'

```

Note that the total number of steps taken to reach equilibrium is actually a little lower when relaxation is used. This is not uncommon.

An examination of the state flags for the zones surrounding the excavation shows that the relaxed version not only shows less damage, but is entirely lacking in tensile failure indicators.

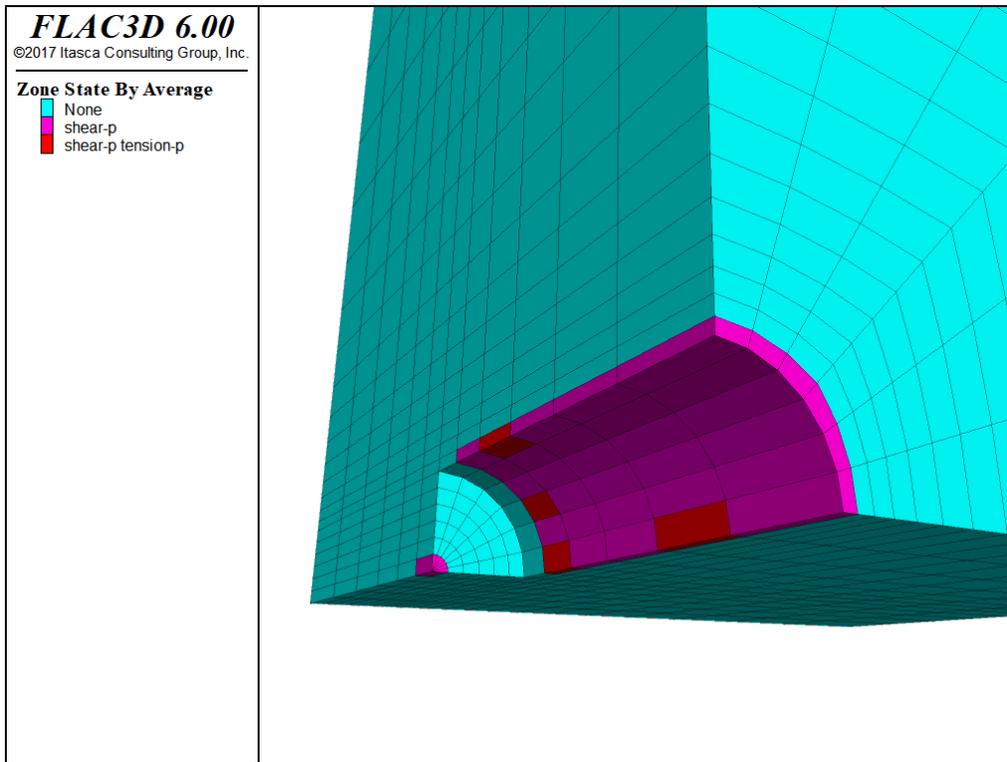


Figure 3: Plasticity state for instantaneous excavation of tunnel.

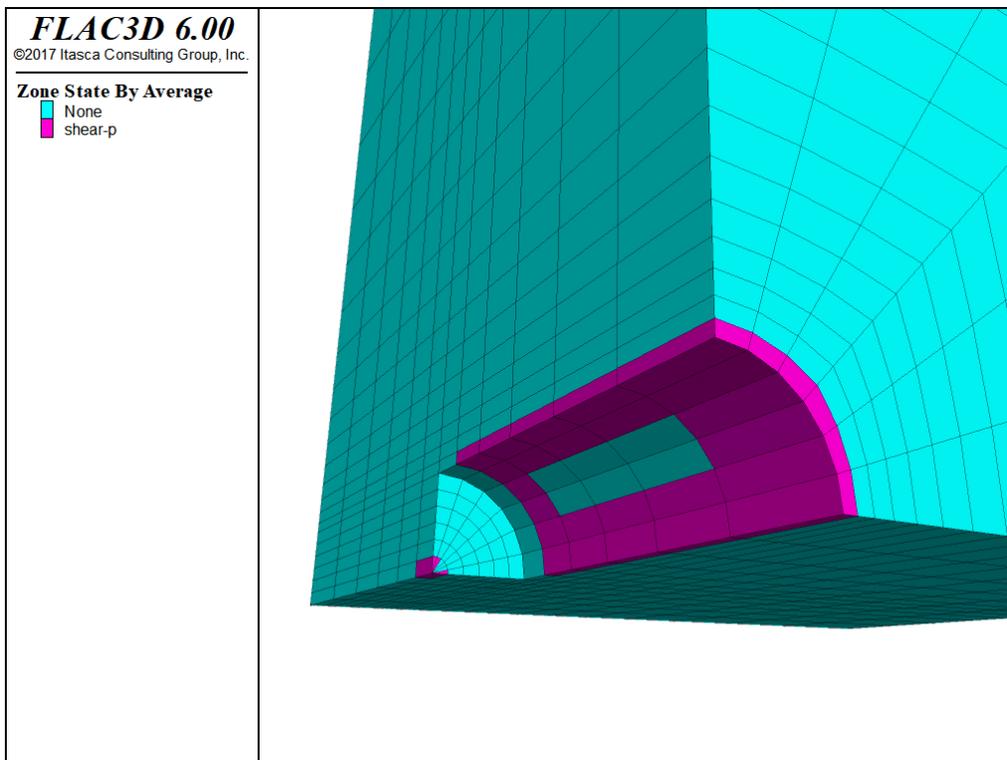


Figure 4: Plasticity state for relaxed excavation of tunnel.

Cautions and Advice

In this section, some miscellaneous difficulties with stress boundaries are described.

With *FLAC3D*, it is possible to apply stresses to the boundary of a body that has no displacement constraints (unlike many finite-element programs, which require some constraints). The body will react in exactly the same way as a real body would (i.e., if the boundary stresses are not in equilibrium, then the whole body will start moving). The following data file illustrates the effect.

Spin when grid is not in equilibrium

```
model new
zone create brick size 6,6,6 point 1 (6,0,-1)
zone cmodel assign elastic
zone property bulk 8e9 shear 5e9
zone face apply stress-xx -2e6 range position-x 0
zone face apply stress-xx -2e6 range position-x 6
model step 500
```

The plots produced from this example are given in the following figure. The applied σ_{xx} stress causes horizontal forces to act on the body. Because the body is tilted, these forces give rise to a moment that causes the body to spin.

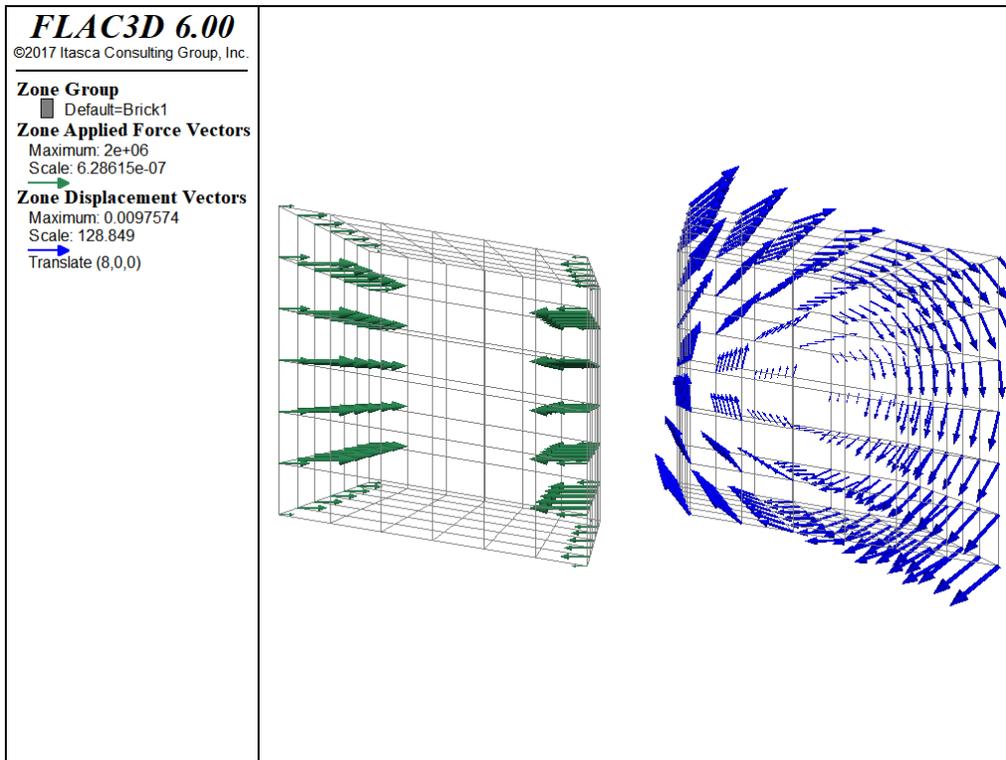


Figure 5: Applied horizontal forces and rotational displacement induced on tilted body.

A similar, but more subtle, effect arises when material is excavated from a body that is supported by a stress boundary condition: the body is initially in equilibrium under gravity, but the removal of material reduces the weight. The whole body then starts moving upward, as demonstrated by the following data file.

Uplift when material is removed

```

model new
; Create zones
zone create brick size 5,5,5
zone face skin
; Assign model and properties
zone cmodel assign elastic
zone property bulk 8e9 shear 5e9 density 1000
; Boundary Conditions
zone face apply velocity-normal 0 range group 'West' or 'East'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply stress-normal -5e4 range group 'Bottom'
; Initial Conditions
model gravity 10
zone initialize-stresses

```

```

model solve ; Starts at equilibrium
zone cmodel assign null range position (1,1,3) (4,4,5)
model step 100 ; body no longer in equilibrium

```

The uplift is shown in the next figure. The difficulty encountered in running this data file can be eliminated by fixing the bottom boundary, rather than supporting it with stresses. The **Artificial Boundaries** section contains some information relating to the location of such artificial boundaries.

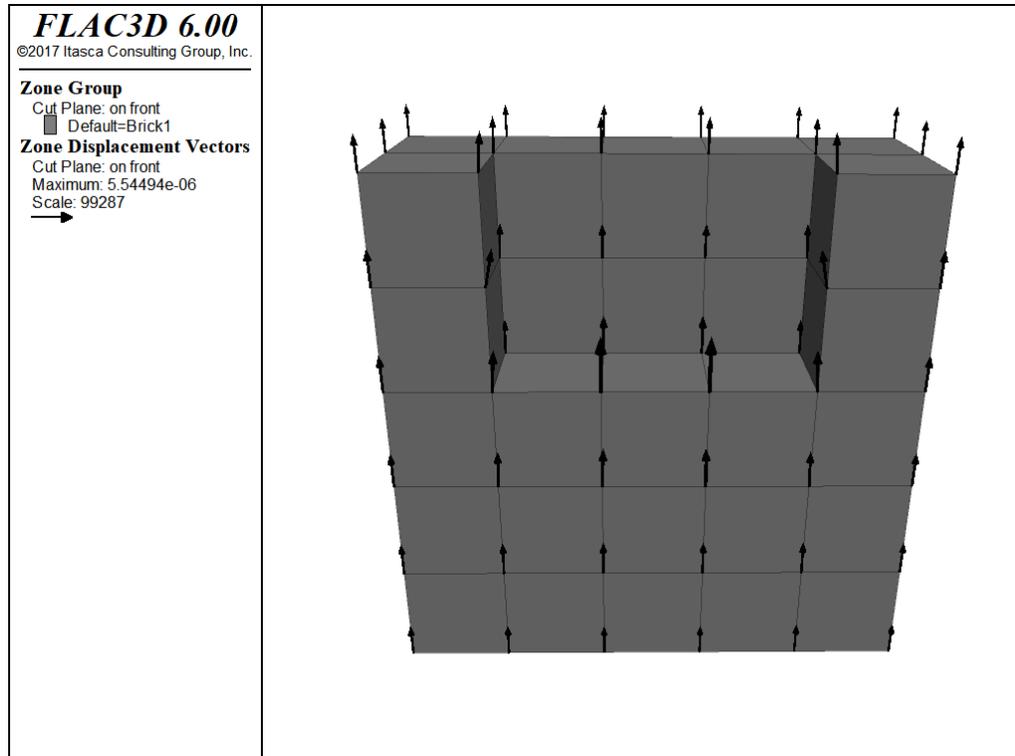


Figure 6: Uplift of model when material is removed.

Displacement Boundary

Displacements cannot be controlled directly in *FLAC3D*; in fact, they play no part in the calculation process. In order to apply a given displacement to a boundary, it is necessary to prescribe the boundary's velocity for a given number of steps. If the desired displacement is D , a velocity V over N steps (where $N = D/V$) may be applied. In practice, V should be kept small and N large, in order to minimize shocks to the system being modeled. The `zone face apply` command or the `zone gridpoint fix` and `zone gridpoint initialize` commands can be used to specify the velocities; gradients may also be specified.

Local System and Applied Velocities

Applied velocity conditions always refer to gridpoints, though they may be applied via zone-face commands (e.g., `zone face apply velocity-local`) or zone-gridpoint commands. When zone-face commands are used, all gridpoints connected to selected faces are selected.

Velocities can be applied in terms of the x,y,z -global axes (i.e., with keywords `velocity-x`, `velocity-y`, or `velocity-z`) or in terms of a local axes (with keywords `velocity-dip`, `velocity-strike`, or `velocity-normal`). All local or global velocity components may be supplied as a vector in one command (with `velocity-local v`, or `velocity v`, respectively). The local axes are defined by the normal vector at each gridpoint, using the convention in the figure below:

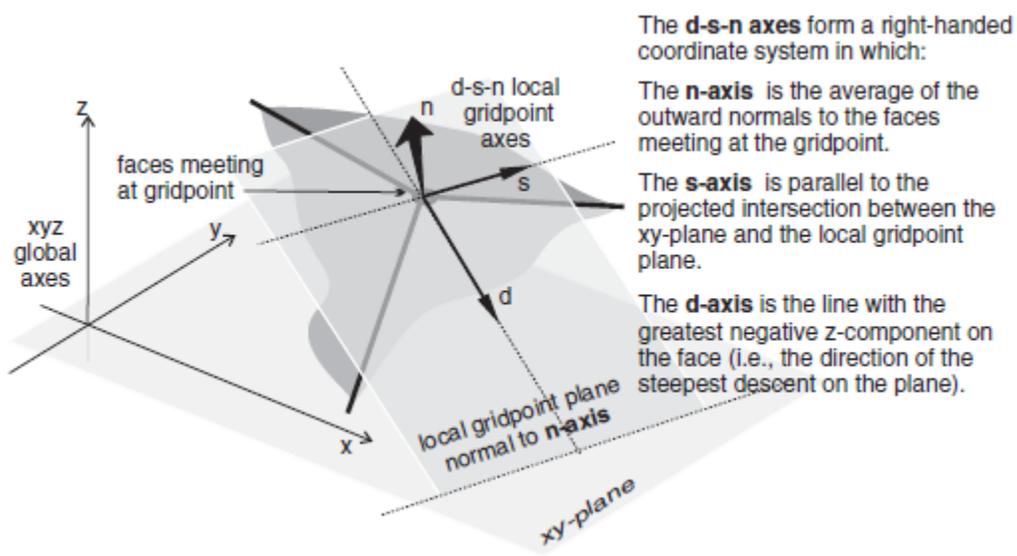


Figure 7: Local gridpoint axes defined by (d) dip direction, (s) strike direction, and (n) normal direction.

Each gridpoint has its own local coordinate system, which can be specified with the `zone gridpoint system` command. The `zone face apply` logic will over-write those values if necessary to apply the conditions requested. It will also automatically attempt to rotate and select appropriate local degrees-of-freedom to allow multiple velocity constraints to be present in the same gridpoint. If this is not possible, then a previous apply condition will be removed to make way for the new one. This means that it is not necessary to work out a compatible local

Surface Corners and Velocity Boundaries

The gridpoint normal direction used is the average of the normal vectors of the faces *included in that apply condition* that meet at a gridpoint. This means that care must be taken to select faces that result in the desired normal vector, and commonly, multiple apply commands are necessary to prescribe multiple conditions on gridpoints along boundaries. For example, if one attempted to apply roller boundaries to five sides of a mode in one command with:

```
zone face apply velocity-normal 0 ...
      range group 'North' or 'South' or 'East' or 'West' or 'Bottom'
```

then the result can be seen in the following plot of gridpoint fixity. The corners of the model have been fixed at a diagonal, because that is the normal vector calculated using the average of all faces *included in that apply* that is connected to a gridpoint.

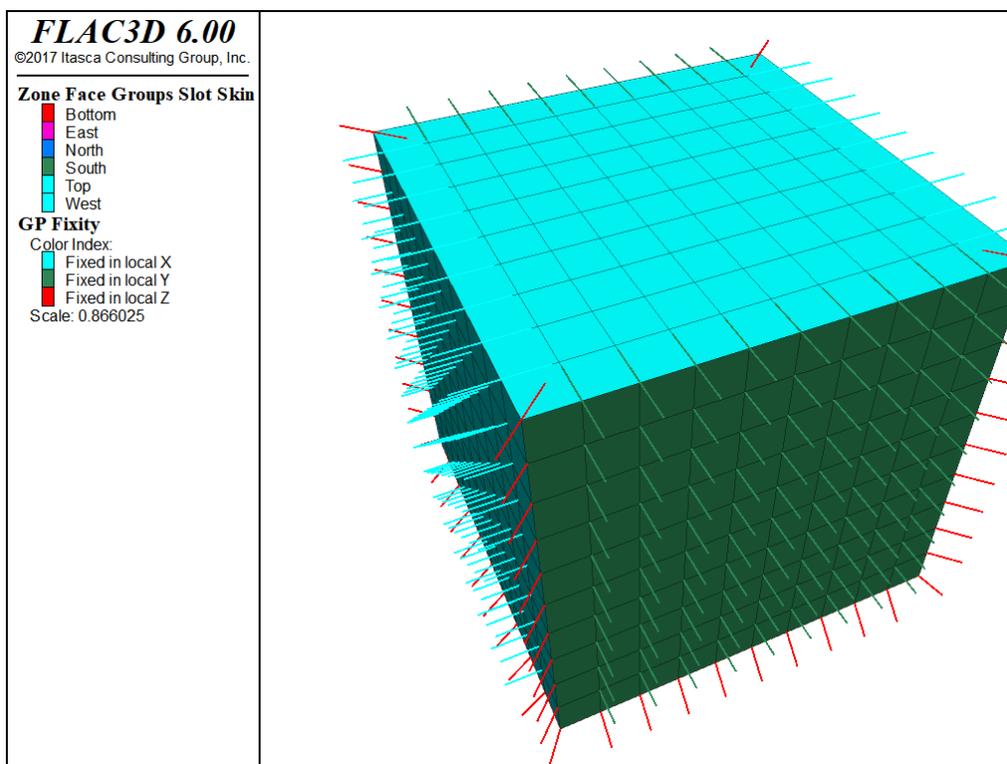


Figure 9: Velocity constraint directions from attempting to apply roller boundaries in a single command.

If the command were split into three, with the ‘East’ and ‘West’ done separately from the ‘North’ and ‘South’ and the ‘Bottom’, then the result is:

```
zone face apply velocity-normal 0 range group 'West' or 'East'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
```

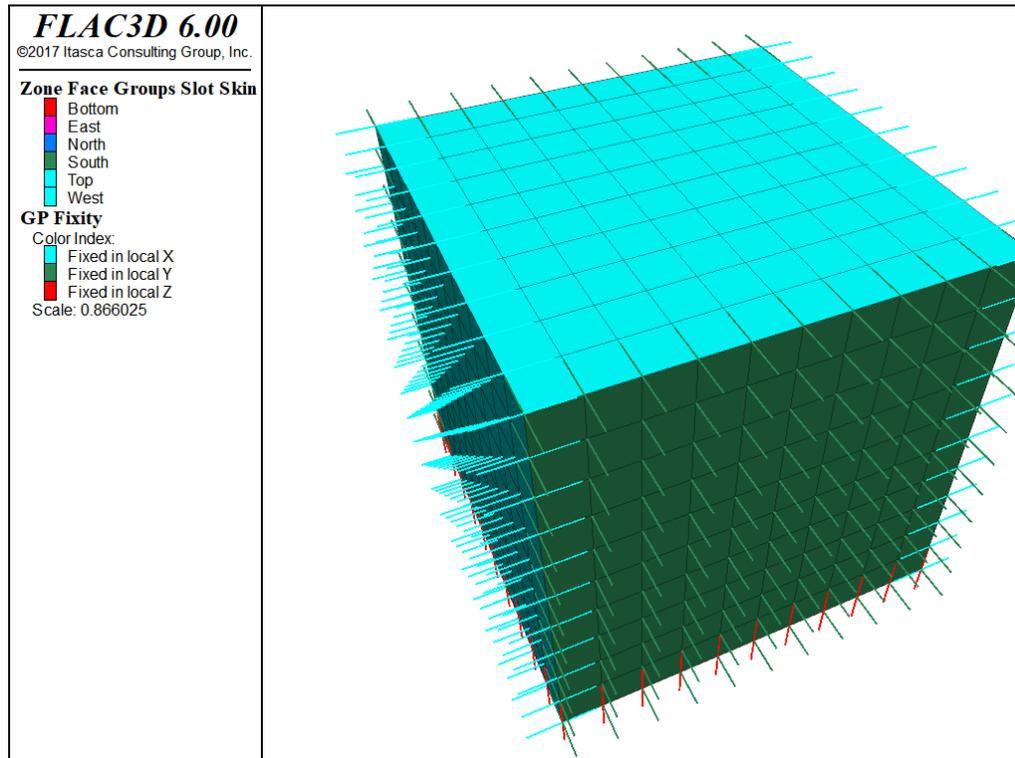


Figure 10: Velocity constraint directions when multiple apply commands are used.

This pattern is seen often in example problems in the *FLAC3D* documentation. Note that it is acceptable to combine ‘West’ and ‘East’ into one command because those surfaces do not share a boundary.

Velocity conditions can be applied at any orientation by using the local axes. A normal gridpoint direction can be specified arbitrarily for the local axes by using the `system` keyword; this will override the default normal direction.

Fix vs Apply

Another method of specifying velocities at gridpoints is to use the `zone gridpoint fix` command. For example, the following commands are equivalent. Both commands apply a fixed velocity of $(1e-5, 0, 0)$ to all gridpoints connected to faces of group `East`.

```
zone gridpoint fix velocity (1e-5,0,0) range group 'East'
zone face apply velocity (1e-5,0,0) range group 'East'
```

In general, using the `zone face apply` command is preferred because the apply logic uses the gridpoint local systems and fixity flags. Any values set there manually may be over-ridden by a later apply condition. Also, the apply logic calculates local systems and attempts to moderate between multiple constraints automatically. Note, however, that `zone face apply` can currently only be performed on surface faces, where `zone gridpoint fix` can be done on any gridpoints in the model.

Reaction Forces

When gridpoints are moved rigidly, the reaction forces at the gridpoints can be monitored using *FISH*. The sum of the reaction forces along a boundary may be obtained with a simple *FISH* function that adds up the values returned from the `gp.force.unbal` intrinsic over the desired range. (See [Tutorial: Working with FISH](#).)

Nonuniform Velocities

If nonuniform prescribed velocities are required, the `gradient` keyword may be used. For a more complicated velocity profile, or one that changes as the simulation proceeds, it will be necessary to write a *FISH* function. The next example demonstrates this for the model of a rotating retaining wall on the right-hand side of a block of soil.

Rotating retaining wall

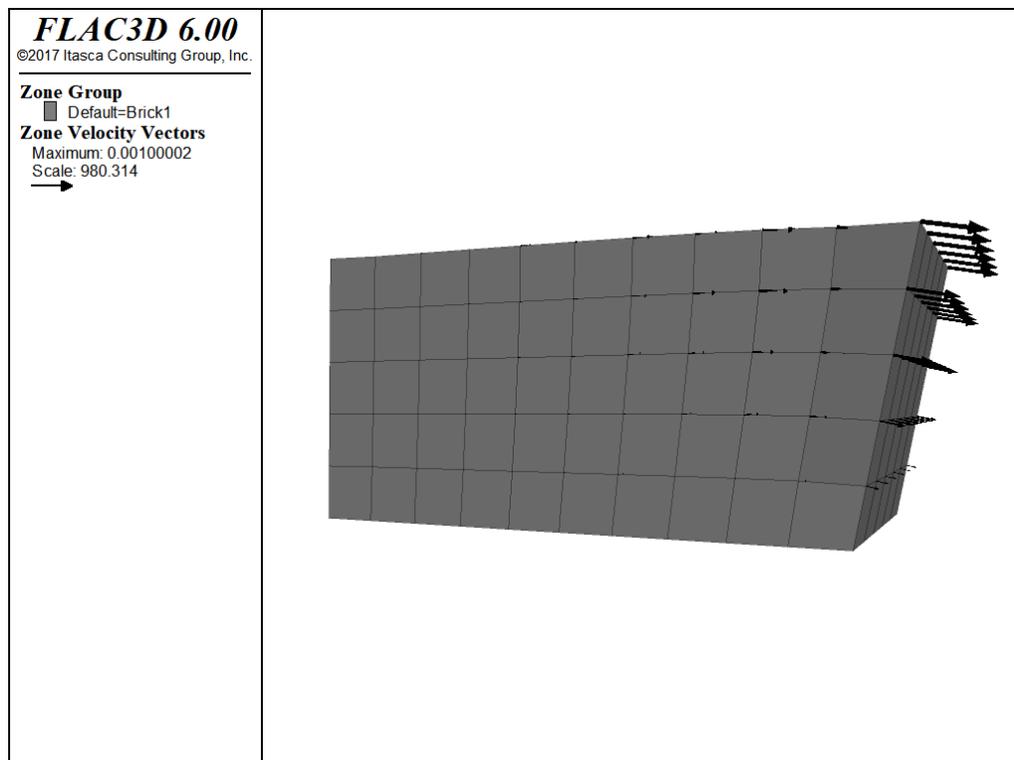
```
model new
; Create zones
zone create brick size 10 5 5
zone face skin
; Constitutive model and properties
zone cmodel elastic
```

```

zone property shear 1e8 bulk 2e8
; FISH function
fish define apply(gp,dum)
  local x = 1e-3 * gp.pos.z(gp) / 5.0
  local z = -1e-3 * (gp.pos.x(gp) - 10.0) / 5.0
  apply = vector(x,0,z)
end
; Boundary conditions
zone face apply velocity (0,0,0) range group 'West' or 'Bottom'
zone face apply velocity (1,1,1) fish-local @apply range group 'East'
; Setup
model largestrain on
; Cycle
model step 1000

```

In this example, the `fish-local` keyword is used to adjust the velocities at every gridpoint during cycling. The `FISH` function `apply` is called once for each gridpoint each cycle. The first argument `gp` is the pointer to the gridpoint. In this case, the second argument is unused. The location of the gridpoint is used to calculate a velocity that in large strain, will cause the right-hand boundary to rotate. The velocity profile of the wall is adjusted as the geometry changes in large-strain mode. Note that the given velocity in this example is much too high for a realistic simulation; it is for demonstration purposes only.



Real Boundaries — Choosing the Right Type

It is sometimes difficult to know which type of boundary condition to apply to a particular surface of the body being modeled. For example, in modeling a laboratory triaxial test, should the load applied by the platen be regarded as a stress boundary, or should the platen be treated as a rigid displacement boundary? Of course, the whole testing machine, including the platen, could be modeled, but that might be very time-consuming. (Remember that *FLAC3D* takes a long time to converge if there is a large contrast in stiffnesses.) In general, if the object applying the load is very stiff compared with the sample (say, more than 20 times stiffer), then it may be treated as a rigid boundary. If it is soft compared with the sample (say, 20 times softer), then it may be modeled as a stress-controlled boundary. Clearly, a fluid pressure acting on the surface of a body is of the latter category. Footings on soil can often be represented as rigid boundaries that move with constant velocity for the purpose of finding the collapse load of the soil. This approach has another advantage—it is much easier to control the test and obtain a good load/displacement graph. It is well-known that stiff testing machines are more stable than soft testing machines.

Artificial Boundaries

Artificial boundaries fall into two categories: planes of symmetry and planes of truncation.

Symmetry Planes

Sometimes it is possible to take advantage of the fact that the geometry and loading in a system are symmetrical about one or more planes. For example, if everything is symmetrical about a vertical plane, then the horizontal displacements on that plane will be zero. Therefore, we can make that plane a boundary and fix all gridpoints in the horizontal direction using the `zone face apply velocity-normal 0` command, with the range given as the boundary plane. In the case considered, the y - and z -components of velocity on the vertical plane of symmetry are not affected and should not be fixed. Similar considerations apply to other planes of symmetry. Planes of symmetry can also be set along boundaries that lie at angles to the x,y,z -coordinate axes.

Boundary Truncation

When modeling infinite bodies (e.g., tunnels underground) or very large bodies, it may not be possible to cover the whole body with zones, due to constraints on memory and computer time. Artificial boundaries are placed sufficiently far away from the area of interest that the behavior in that area is not greatly affected. It helps to know how far away to place these boundaries, and what errors might be expected in the stresses and displacements computed for the area of interest.

Several points should be considered when selecting the location for artificial boundaries:

1. A fixed boundary causes both stresses and displacements to be underestimated, while a stress boundary does the opposite.
2. The two types of boundary condition “bracket” the true solution, so that it is possible to do two tests with small boundaries and get a reasonable estimate of the true solution by averaging the two results.
3. The effect of boundary location is most noticeable for elastic bodies because the displacements and stress changes are more confined when plastic behavior is present; there is a natural cut-off distance within which most of the action occurs. The artificial boundary may be placed slightly closer without serious error. However, any artificial boundary must not be sufficiently close that it attracts plastic flow and thereby invalidates the solution.

It is always best to run several (coarse) models first, with different boundary locations, to evaluate the potential influence of the boundary on the calculated response, before performing the detailed analysis.

Initial Conditions

In all civil or mining engineering projects, there is an in-situ state of stress in the ground, before any excavation or construction is started. In *FLAC3D*, an attempt is made to reproduce this in-situ state by setting initial conditions. Ideally, information about the initial state comes from field measurements. But, when these are not available, the model can be run for a range of possible conditions. Although the range is potentially infinite, there are a number of constraining factors (e.g., the system must be in equilibrium, and the chosen yield criteria must not be violated anywhere).

In a uniform layer of soil or rock with a free surface, the vertical stresses are usually equal to $g\rho z$, where g is the gravitational acceleration, ρ is the mass density of the material, and z is the depth below the surface. However, the in-situ horizontal stresses are more difficult to estimate. There is a common—but erroneous—belief that there is some “natural” ratio between horizontal and vertical stress, given by $\nu/(1-\nu)$, where ν is the Poisson’s ratio. This formula is derived from the assumption that gravity is suddenly applied to a mass of elastic material in which lateral movement is prevented. This condition hardly ever applies in practice, due to repeated tectonic movements, material failure, overburden removal and locked-in stresses due to faulting and localization (see [Localization, Physical Instability, and Path-Dependence](#)). Of course, if we had enough knowledge of the history of a particular volume of material, we might simulate the whole process numerically to arrive at the initial conditions for our planned engineering works, but this approach is not often feasible. Typically, we compromise: a set of stresses is installed in the grid, and then *FLAC3D* is run until an equilibrium state is obtained. It is important to realize that there are an infinite number of equilibrium states for any given system.

In the following topics, we examine progressively more complicated situations and the ways in which the initial conditions may be specified. The user is encouraged to experiment with the various data files presented.

Uniform Stresses — No Gravity

For an excavation deep underground, the gravitational variation of stress from top to bottom of the excavation may be neglected because the variation is small in comparison to the magnitude of stress acting on the volume of rock to be modeled. The `model gravity` command may be omitted, causing the gravitational acceleration to default to zero. Initial uniform stresses can be installed with the `zone initialize` command. For example, the commands

```
zone initialize stress-xx -5e6
zone initialize stress-yy -1e7
zone initialize stress-zz -5e6
```

will set the stress components σ_{xx} , σ_{yy} , and σ_{zz} to compressive stresses of -5×10^6 , -10^7 , and -5×10^6 , respectively, throughout the grid. Range parameters may be added if the stresses are to be restricted to a sub-grid.

The `stress` keyword can be used to initialize the entire stress tensor at once. This can only be done for a uniform stress field. The following is almost the equivalent command:

```
zone initialize stress xx -5e6 yy -1e7 zz -5e6
```

The difference is that the `stress` keyword initializes *all* six components of stress for each zone in the range. Components that were not explicitly assigned a value are assumed to be zero.

The `zone initialize` command sets all stresses to the given values, but there is no guarantee that the stresses will be in equilibrium. There are at least two possible problems. First, the stresses may violate the yield criterion of a nonlinear model that has been assigned to the grid. In this case, plastic flow will occur immediately upon cycling and the stresses will readjust. This possibility should be checked by doing one trial step and examining the response (e.g., create zone plot item, set “ColorBy” to “Label”, and set “Label” to “State”). Second, the prescribed stresses at the grid boundary may not equal the given initial stresses. In this case, the boundary gridpoints will start to move as soon as cycling begins. Again, output should be checked (e.g., plot a zone-vectors plot item, set “Value” to “Velocity”) for this possibility. The example below shows a set of commands that produce initial stresses that are in equilibrium with prescribed boundary stresses.

Initial and boundary stresses in equilibrium

```

model new
zone create brick size 6 6 6
zone face skin
zone cmodel assign elastic
zone initialize stress xx=-5e6 yy=-1e7 zz=-2e7
zone face apply stress-xx=-5e6 range group 'East' or 'West'
zone face apply stress-yy=-1e7 range group 'North' or 'South'
zone face apply stress-zz=-2e7 range group 'Top' or 'Bottom'

```

Of course, if the boundary is velocity rather than stress-controlled, the initial stresses will be in equilibrium automatically; the `zone face apply` command is not necessary. See [Boundary Conditions](#) for more details on boundary conditions and the `zone apply` and `zone face apply` commands in particular.

Stresses with Gradients — Uniform Material

Near the ground surface, the variation in stress with depth cannot be ignored. The `model gravity` command is used to inform *FLAC3D* that gravitational acceleration operates on the grid. It is important to understand that the `model gravity` command does not directly cause stresses to appear in the grid; it simply causes *body forces* to act on all gridpoints. These body forces correspond to the weight of material surrounding each gridpoint. If no initial stresses are present, the forces will cause the material to move (during stepping) in the direction of the forces until equal and opposite forces are generated by zone stresses. Given the appropriate boundary conditions (e.g., fixed bottom, roller side boundaries), the model will, in fact, generate its own gravitational stresses that are compatible with the applied gravity. However, this process is inefficient, since many hundreds of steps may be necessary for equilibrium. It is better to initialize the internal stresses such that they satisfy both equilibrium and the gravitational gradient.

The `zone initialize-stresses` command will automatically initialize a stress field due to gravity. The `model gravity` command must have been given first, and `density` must have been assigned to all zones. `zone initialize-stresses` calculates stresses regardless of any irregularity in the geometry. It works by finding all changes in values that affect gravitational loading and tracking which ones are crossed as you move from a given zone against the direction of gravity.

Initial stress state with gravitational gradient

```
zone initialize-stresses ratio 0.5
zone face apply stress-normal [-1000*10*10*0.5] ...
      gradient (0,0,[1000*10*0.5]) ...
      range group 'North' or 'East' or 'South' or 'West'
```

The `ratio` keyword specifies a horizontal stress ratio K_0 that is used to determine the σ_{xx} and σ_{yy} stresses that in this case are equal. Note that if stress boundary conditions are used, they need to be compatible with the stresses calculated by `zone initialize-stresses`. In this case, in-line *FISH* can be used to make the load and gradient calculations clear and simple. The maximum compressive vertical stress is the height * density * gravity * ratio, and the gradient is density * gravity * ratio. Note that the simple model in question is a 10 m × 10 m × 10 m box of homogeneous material, and the top surface is free.

The following `zone initialize` command is, for this particular instance, functionally the same as `zone initialize-stresses`

```
zone initialize stress-xx [-1000*10*10*0.5] gradient (0,0,[1000*10*0.5])
zone initialize stress-yy [-1000*10*10*0.5] gradient (0,0,[1000*10*0.5])
zone initialize stress-zz [-1000*10*10] gradient (0,0,[1000*10])
```

In this example, horizontal stresses and gradients are equal to half the vertical stresses and gradients, but they may be set at any value that does not violate the yield criterion (Mohr–Coulomb, in this case). After preparing a data file such as the one shown here, one calculation step should be executed and the velocity field plotted; any failure to match internal stresses with boundary stresses will show up as a systematic movement at one or more boundaries. Small, chaotic velocities may be ignored (see [Interpretation: Gridpoint Velocities](#)). The unbalanced force after one step will be very small, but may not be exactly zero. This is a function of round-off error.

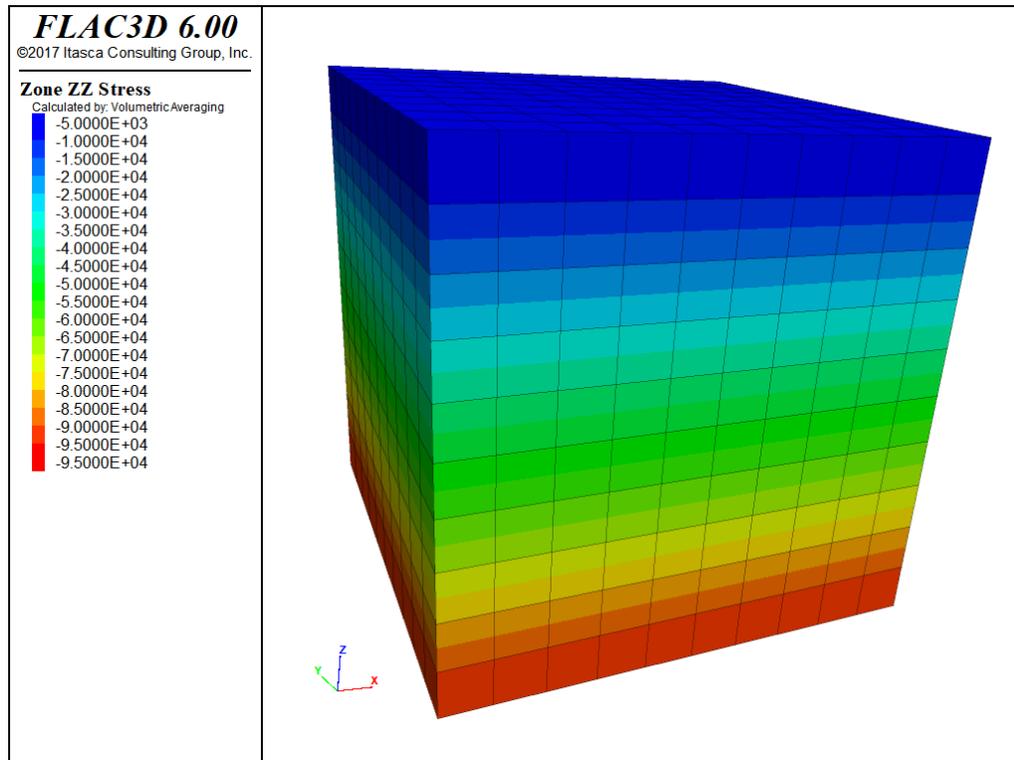


Figure 1: Uniform gradient equilibrium vertical stresses.

Note that the gradient specified with the `gradient` keyword associates stresses with zone centroids (i.e., the global location of the centroid determines the stress value from the gradient). Thus, although the applied σ_{xx} -stress varies from -5.0×10^6 to -4.5×10^6 in the example above, the actual zone stresses in this example vary from -4.975×10^6 to -4.525×10^6 . The stress-contours may be plotted with or without interpolation of the zone stress from the centroid to the boundary; the stress at the boundary is *not* exactly the stress in the boundary zone. To see the differences, plot a zone plot item, set “ColorBy” to “Contour”, “Value” to “Stress”, and “Quantity” to “zz”. Then change the selection on the “Method” attribute (options are “Constant”, “Volumetric Averaging”, “Inv. Distance Weighting”, and “Polynomial Extrapolation”) and note the result. *Actual* stresses in the zone are shown when the “Method” is set to “Constant”.

Stresses with Gradients — Nonuniform Material

`zone initialize-stresses` should generally be used to find initial stresses in more difficult models when materials of different densities are present. Consider a layered system with a free surface, enclosed in a box with roller side boundaries and fixed base. Suppose that the material has the following density distribution:

1600 kg/m³ from 0 to 10 m depth

2000 kg/m³ from 10 to 15 m

2200 kg/m³ from 15 to 25 m

An equilibrium state is produced by the following data file.

Initial stress gradient in a nonuniform material

```

model new
; Create Zones
zone create brick size 10 10 10 ...
                    point 0 (0, 0,0) point 1 (20,0, 0) ...
                    point 2 (0,20,0) point 3 ( 0,0,25)

zone face skin
; Constitutive model and properties
zone cmodel assign elastic
zone property bulk 5e9 shear 3e9
zone initialize density 1600 range position-z 0 10
zone initialize density 2000 range position-z 10 15
zone initialize density 2200 range position-z 15 25
; Boundary Conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
zone face apply stress-normal -1e6 range group 'Top'
; Initial Conditions
model gravity 10
zone initialize-stresses ratio 0.25 0.5 overburden -1e6
model solve

```

Without the use of `zone initialize-stresses`, the stress at each material interface must be calculated manually from the known overburden above it. This is somewhat tractable for a simple layered material, using the `gradient` formula to calculate the value following `stress-zz` on the `zone initialize` command. The `gradient` values are simply the variations across each layer due to the gravitational gradient. In this example, an overburden of $-1e6$ was added to the surface, and a different horizontal stress ratio was used in the σ_{xx} and σ_{yy} direction.

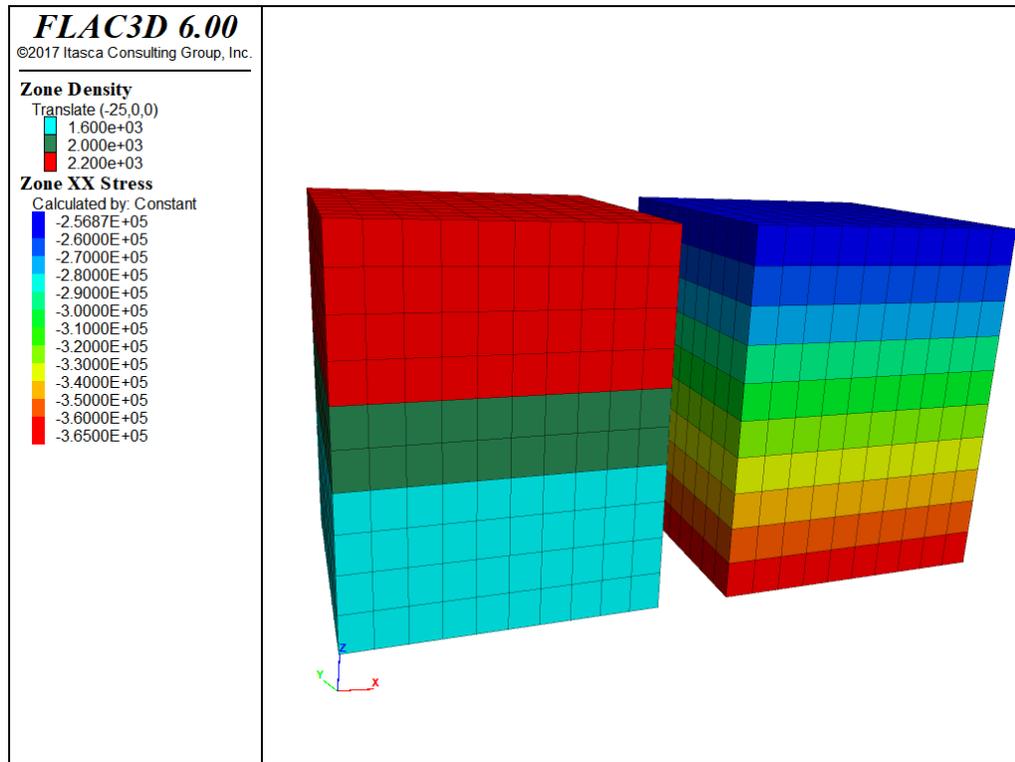


Figure 2: Nonuniform densities and resulting equilibrium vertical stresses.

In a more complicated situation, if `zone initialize-stresses` was not suitable for some reason, it would be necessary to use a *FISH* function to compute initial approximate stress values from a known material property distribution.

Stress Initialization in a Nonuniform Material

The existence of *internal* nonuniformities in a *FLAC3D* grid should not change the way stresses are initialized. However, some minor adjustment may be necessary, because forces do not balance exactly if the grid is irregular. For example, a grid may be distorted internally into the shape of a cylinder, with a view to “excavating” a tunnel at some later stage in the run. There will be a very slight initial imbalance in forces, but this may be relaxed with a few steps, as illustrated in this example.

Initial stress state for a nonuniform grid

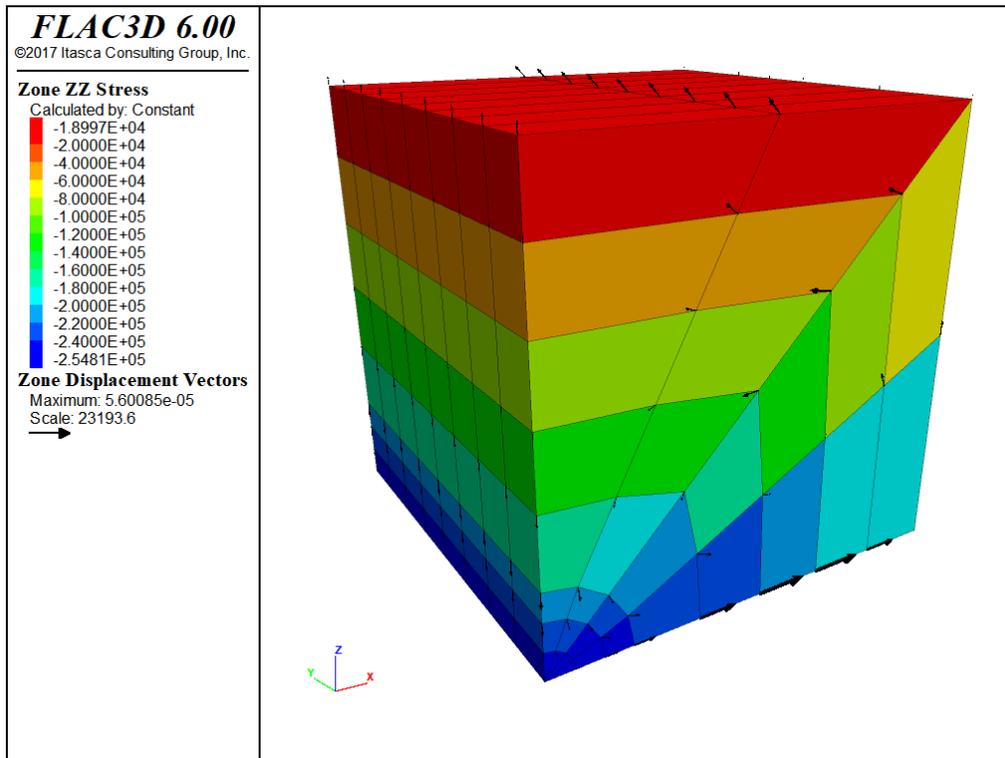
```
model new
; Create grid
zone create radial-cylinder size 3 8 4 5 fill ...
point 1 (10,0,0) point 2 (0,10,0) point 3 (0,0,10)
```

```

zone face skin
; Constitutive model and properties
zone cmodel elastic
zone property shear 3e8 bulk 5e8 density 2500
; Boundary conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
; Initial conditions
model gravity 10
zone initialize-stresses
model solve

```

At one calculation step, the unbalanced force ratio is approximately $3.5e-3$. It takes approximately 200 additional steps for this to be reduced to $1e-5$. The final zone stresses and displacements resulting from this adjustment are shown below.



More complications arise when a free surface has an irregular geometry. The next example produces the “mountain range” shown in the figure that follows, using the `zone generate from-topography` command.

Initial stress state for an irregular surface

```

model new
; Create Zones
zone create brick size 30 30 10 point 0 (0,0,0) point 1 (580,0,0) ...
                                point 2 (0,680,0) point 3 (0,0,100)

geometry import 'test.stl'
zone generate from-topography direction (0,0,1) ...
                                geometry-set 'test' segments 20

zone face skin
; Constitutive model and properties
zone cmodel assign elastic
zone property bulk 3e8 shear 2e8 density 1000
; Boundary conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
; Initial conditions
model gravity 10
zone initialize-stresses ratio 0.5
model solve elastic

```

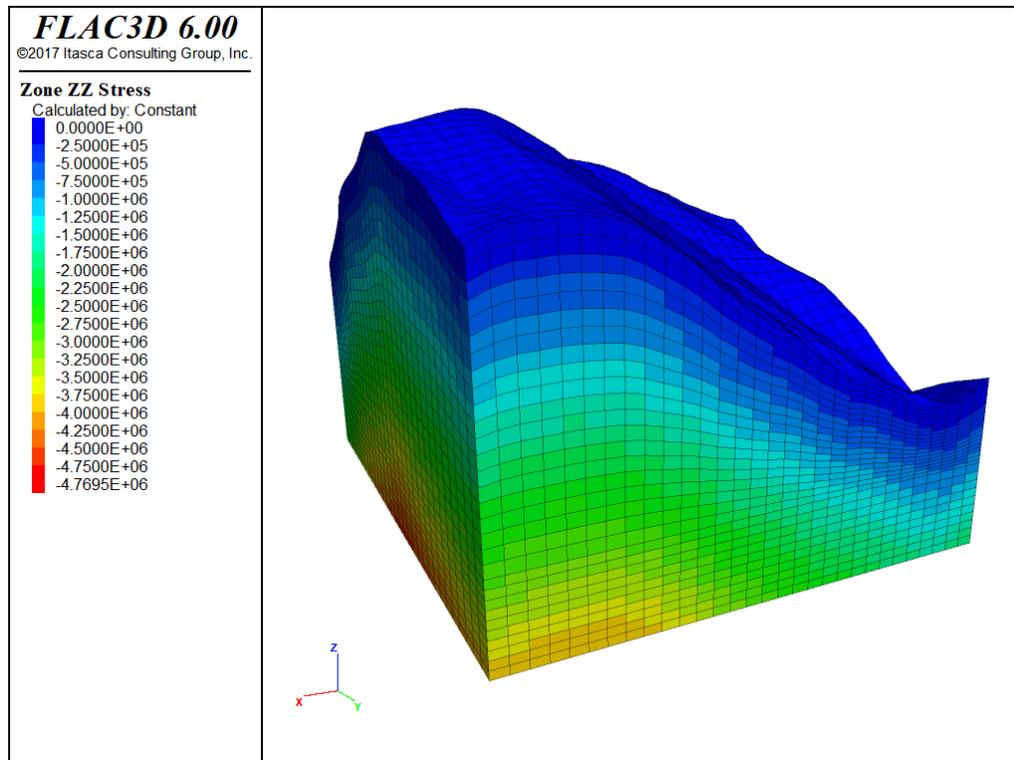


Figure 4: Initialized stresses in an irregular topological grid in FLAC3D.

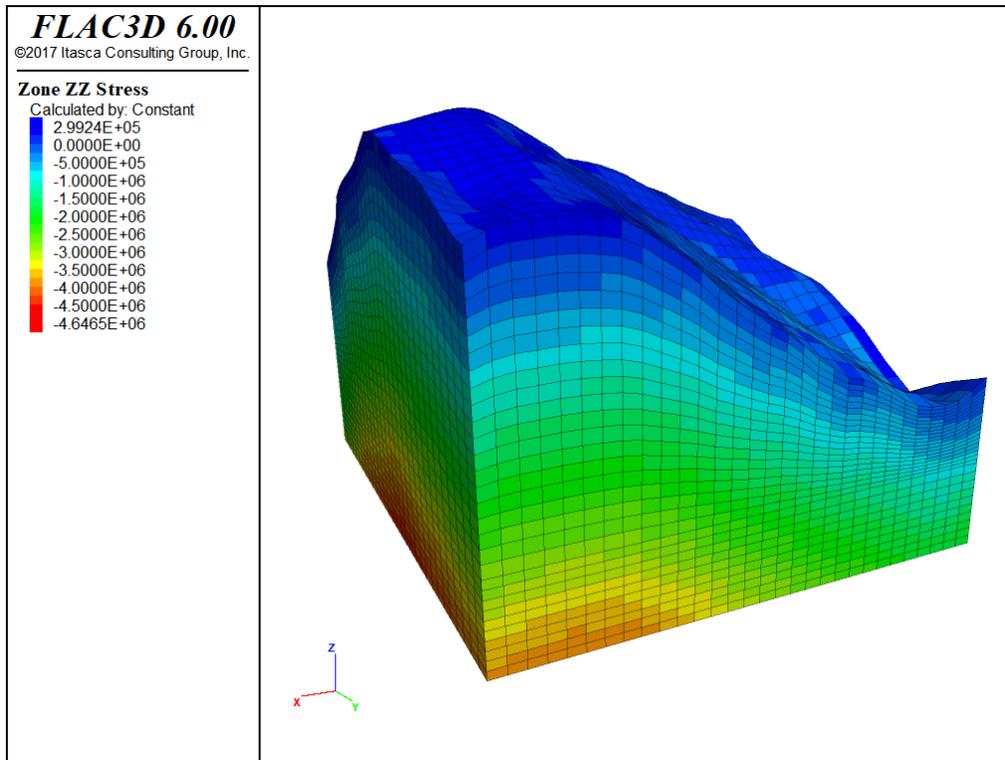


Figure 5: Stresses after cycling to equilibrium.

The `zone initialize-stresses` command calculates an approximate vertical stress distribution for the model, which is very close to actual equilibrium. If the horizontal stress ratio were set to θ , it would only take approximately 200 steps to reach full equilibrium after initialization. With a finite stress ratio in a surface topography like this, however, problems occur. A horizontal stress ratio that is perfectly correct near the base is generally too high when near the surface of a slope. This can take some time to come to equilibrium and can also cause unrealistic displacement or even damage on surface slopes.

There is no simple way to calculate initial stresses that are nearer equilibrium for this case. There are a number of different possible methods one could use to help, however:

1. Use multiple `zone initialize-stresses` commands at different elevations and regions, with different `ratio` values.
2. Solve the model elastically.

3. Reset displacement after reaching initial equilibrium (this is generally good practice in any case).
4. Allow plastic flow to occur, thus removing stress concentrations, and then re-assign constitutive models to reset state variables.

There are probably many other possible schemes, particularly for a nonlinear, path-dependent material. No initial state is the “correct one”: the choice may depend on the type of geological process that is believed to have occurred in the field.

Compaction within a Nonuniform Grid

Puzzling results are sometimes observed when a model is allowed to come to equilibrium under gravity, using a nonuniform grid. When a Mohr-Coulomb or other nonlinear constitutive model is used, the final stress state and displacement pattern are not uniform, even though the boundaries are straight and the free surface is flat. The data file in the following example illustrates the effect (see the figure following for the generated plot showing displacement vectors and vertical stress contours).

Nonuniform stress initialized in nonuniform grid

```

model new
; Create zones
zone create brick size 8 8 10 ratio 1.2 1 1
zone face skin
; Constitutive model and properties
zone cmodel assign mohr-coulomb
zone property bulk 2e8 shear 1e8 friction 30 density 2000
; Boundary Conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
model gravity 10
model save 'initial'

; -- the following lines will produce nonuniform patterns:
model solve
model save 'nonuniform'

; -- the following lines set stresses directly:
model restore 'initial'
zone initialize-stresses ratio 0.75
model solve
model save 'direct'

```

```

; -- the following lines initialize elastically:
model restore 'initial'
model solve elastic
model save 'uniform'

```

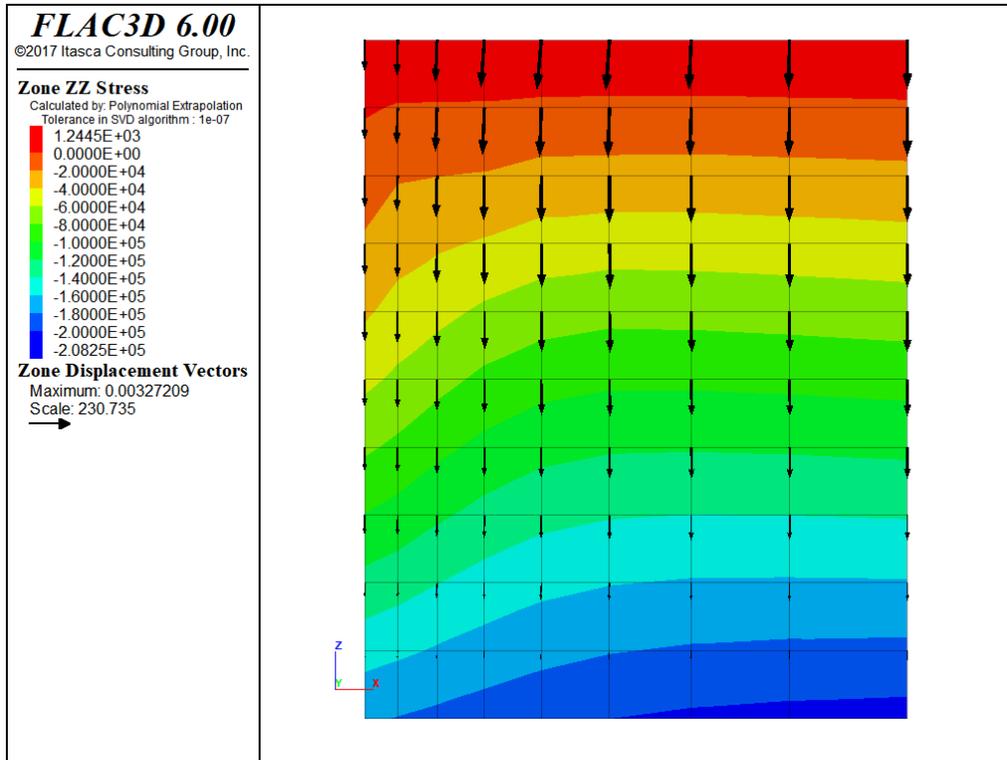


Figure 6: Nonuniform stresses and displacements.

Because we have roller boundaries on the vertical sides, we might expect the material to move down equally on these sides. However, the grid is finer near the origin. *FLAC3D* tries to keep the local timestep equal for all zones, so it increases the inertial mass for the gridpoints near the $x = 0$ boundary to compensate for the smaller zone sizes. These gridpoints then accelerate more slowly than those near the $x = 8$ boundary. This will have no effect on the final state of a linear material, but it causes nonuniformity in a material that is path-dependent. For a Mohr-Coulomb material without cohesion, the situation is similar to dropping sand from some height into a container and expecting the final state to be uniform. In reality, a large amount of plastic flow would occur because the confining stress does not build up immediately. Even with a uniform grid, this approach is not a good one because the horizontal stresses depend on the dynamics of the process.

The best solution is to use the `zone initialize-stresses` command to set initial stresses to conform to the desired K_0 value (ratio of horizontal to vertical stresses). For example, the `model solve` command in the previous data file would stop immediately if it was preceded by the line

```
zone initialize-stresses ratio 0.75
```

A stable state is achieved with $K_0 = 0.75$; no stepping is necessary.

If for some reason it is desirable to use *FLAC3D* to compute the final state, then the `model solve elastic` option could be used. This solves the model to equilibrium without allowing any plastic failure in the constitutive model, then re-activates failure and makes certain the model remains in equilibrium.

```
model solve elastic
```

The next figure shows the displacement vectors and vertical stress contours for this case. The material is prevented from yielding during the compaction process, but the original properties are restored when equilibrium is achieved.

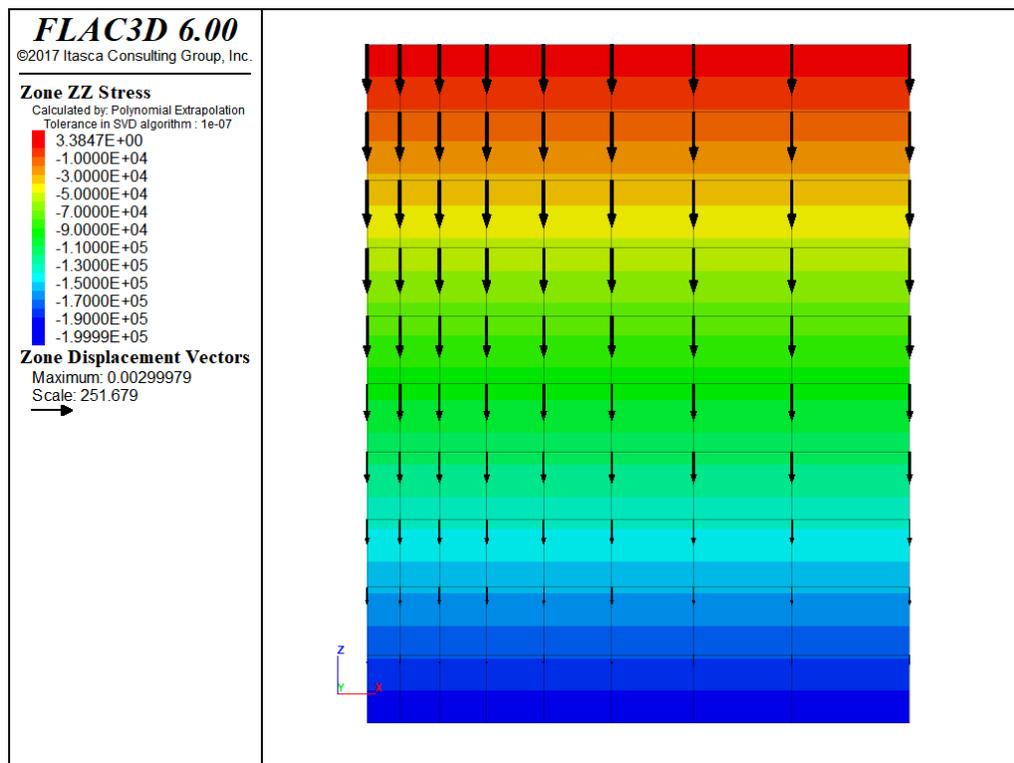


Figure 7: Uniform stresses and displacements.

Initial Stresses following a Model Change

There may be situations in which one constitutive model is used in the process of reaching a desired stress distribution, but another constitutive model is used for the subsequent simulation. If one cmodel is replaced by another non-null cmodel, the stresses in the affected zones are preserved, as in this example:

Initial stresses following a model change

```
model new
; Create zones
zone create brick size 5 5 5
zone face skin
zone group 'Switch' range position (0,0,0) (2,5,2)
; Constitutive models and properties
zone cmodel assign elastic
zone property shear 2e8 bulk 3e8 density 2000
; Boundary conditions
zone face apply velocity-normal 0 range group 'Bottom'
model gravity 10
zone initialize-stresses
model solve
zone cmodel assign mohr-coulomb range group 'Switch'
zone property shear 2e8 bulk 3e8 friction 35 range group 'Switch'
model step 1000
```

At this point in the run, the stresses generated by the initial elastic model still exist and act as initial stresses for the region containing the new Mohr-Coulomb model.

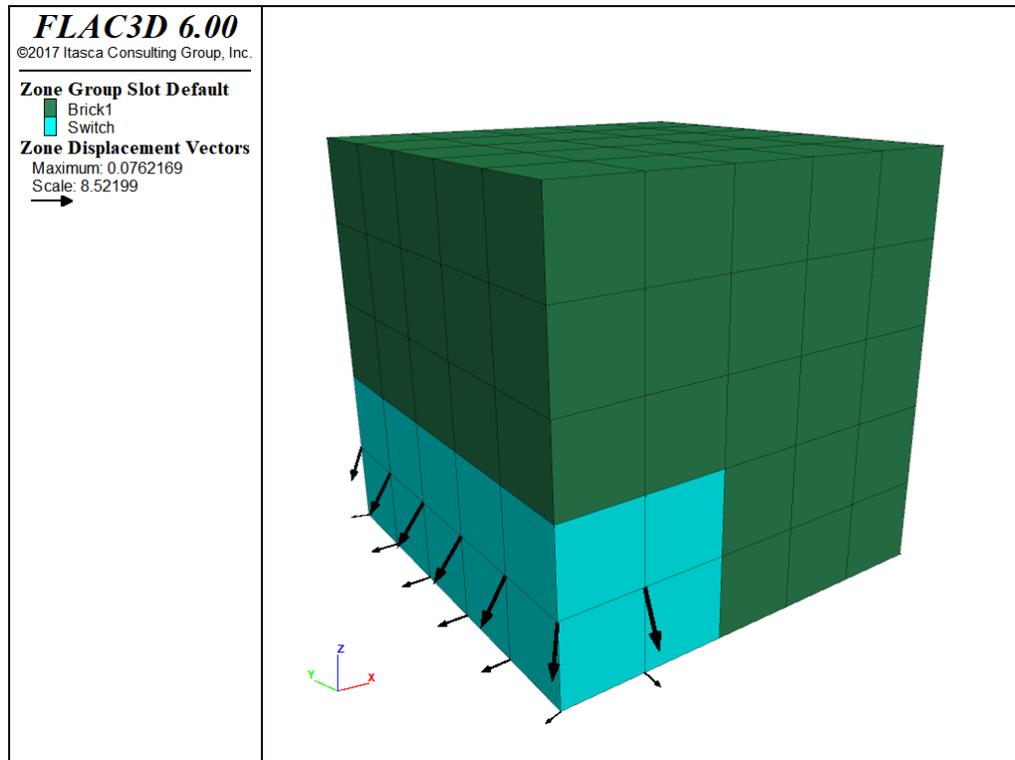


Figure 8: Area of constitutive model change and displacements resulting from failure.

Two points should be remembered:

First, if a null model is installed in any zones (even if it is subsequently replaced by another model), all stresses are removed from the affected zones.

Second, if one model is replaced by another and the stresses should physically be zero in the new model, then a `zone initialize` command must be used to reset the stresses to zero. This situation would occur if rock is mined out and replaced by backfill; the backfill would have its own stress state, unrelated to the rock it is replacing.

Stress and Pore-Pressure Initialization with a Phreatic Surface

Pore pressures are initialized in the same way as stresses. However, the *gridpoint* pore pressures, rather than the *zone* pore pressures, are initialized regardless of whether `model configure fluid` is specified. Zone pore pressures are derived from gridpoint pore pressures by averaging. Zone pore pressures are then used to calculate effective stresses needed by the constitutive models.

Manual initialization of a partially saturated grid can be confusing; it is very important to remember two things:

- If the grid has *not* been configured for groundwater calculations, then densities specified below the phreatic surface (pore-pressure = 0) must be **wet** densities and those above the phreatic surface must be **dry** densities. It is up to the user to supply proper values for densities in those regions.
- If the grid *has* been configured for groundwater calculations (`model configure fluid`), then *dry* densities are specified everywhere and the contribution from the fluid is calculated automatically from the porosity, saturation, and fluid density.

If the proper material properties for each case have been specified (dry density/porosity/saturation/fluid density for `model configure fluid`, wet/dry density distribution otherwise), then the `zone initialize-stresses` command will automatically calculate the correct vertical stress including the weight of fluid.

Note that the κ_0 calculated using the `ratio` keyword is done using *effective* stresses. Use the `total` keyword to calculate horizontal stress ratios in terms of total stresses.

Below are short examples illustrating each case.

Stress and pore pressure initialization with a phreatic surface — grid not configured for groundwater

```
model new
; Create zones
zone create brick size 8 5 10
zone face skin
; Mech constitutive model and properties
zone cmodel assign elastic
zone property bulk 1e9 shear 5e8 density 2000
```

```

zone property density 2500 range position-z 0 5 ; Wet Density
zone property density 2250 range position-z 5 6 ; Partially saturated density
; Boundary Conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
; Initial conditions
model gravity 10
zone water density 1e3
zone water plane origin (0,0,5) normal (0,0,1)
zone initialize-stress ratio 0.5
model solve ; immediately at equilibrium

```

In this example, the density was adjusted on and below the phreatic surface to partially saturated and wet values.

Stress and pore pressure initialization with a phreatic surface — grid configured for groundwater

```

model new
model config fluid
; Create zones
zone create brick size 8 5 10
zone face skin
; Mech constitutive model and properties
zone cmodel assign elastic
zone property bulk 1e9 shear 5e8 density 2000
; Fluid constitutive model and properties
zone fluid cmodel assign isotropic
zone fluid property porosity 0.5 permeability 1e-10
zone gridpoint initialize fluid-modulus 2e9
zone initialize fluid-density 1e3
; Boundary Conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
; Initial conditions
model gravity 10
zone water density 1e3
zone water plane origin (0,0,5) normal (0,0,1)
zone gridpoint initialize saturation 0 range position-z 5.1 10
zone initialize-stress ratio 0.5
model solve ; immediately at equilibrium

```

There are a few things to note in this example. First, the saturation needs to be explicitly initialized to zero above the phreatic surface. Second, as noted above, the dry density was given everywhere. Third, the `zone water` command was used to initialize pore pressures, which requires that a fluid density be given separately from the `zone initialize fluid-density` command.

Initialization of Velocities

Until now, we have concentrated on initialization of stresses. Normally, the velocities *inside* the grid are not set explicitly; they default to zero initially. If, however, a velocity loading condition is specified at the boundary of a body, it is sometimes beneficial to initialize the velocities throughout the body to minimize the shock to the system. For example, in a simulated triaxial test with rigid platens, the velocities can be initialized to achieve an initial linear gradient throughout the sample, as shown in the following example.

Velocity gradient in a triaxial test

```

model new
; Create zones
zone create cylinder point 0 (0,0,0) point 1 (1,0,0) ...
                    point 2 (0,2,0) point 3 (0,0,1) ...
                    size 4 5 4
zone reflect normal (1,0,0)
zone reflect normal (0,0,1)
zone face skin
; Constitutive model and properties
zone cmodel assign mohr-coulomb
zone property bulk 1.19e10 shear 1.1e10
zone property cohesion 2.72e5 friction 44 tension 2e5
; Boundary conditions
zone face apply velocity-normal -1e-4 range group 'North'
zone face apply velocity-normal 0 range group 'South'
zone face apply stress-normal -1e5 range group 'East'
; Velocities initialized to smooth response.
zone gridpoint initialize velocity-y 0 gradient (0,-1e-4,0)

```

If the linear gradient is not applied, the internal gridpoints will receive an initial shock because they must accelerate to produce a negative velocity through the sample. The subsequent motion of the internal gridpoints is not controlled, because only the ends are fixed.

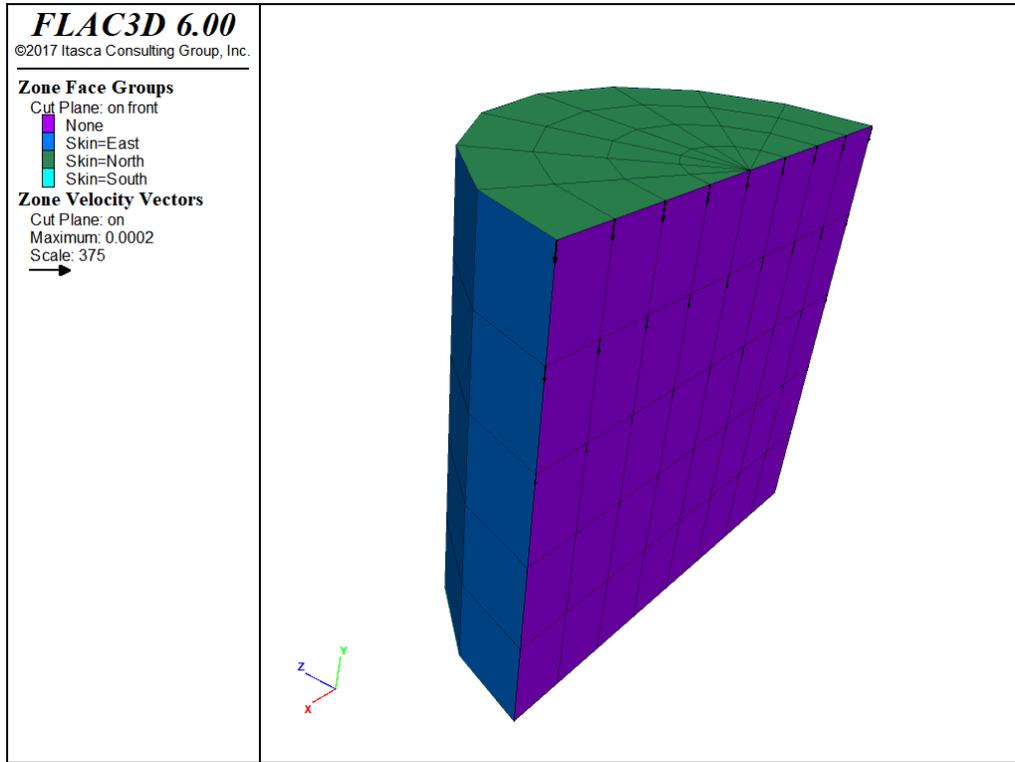


Figure 9: Velocities initialize in simple triaxial test sample.

Reaching Equilibrium

At the end of setting up initial conditions, at the end of the model, and perhaps during multiple intermediate stages, the model must reach “equilibrium.” In the examples so far, this is done with the simple command `model solve`. But how do you know when you have reached equilibrium? How can you tell if you have performed more cycles than is necessary, or if some specific part of the model still needs computation time?

Calculation cycles in *FLAC3D* can be performed manually with the `model step` or `model cycle` command. There are occasions where this is useful, most often when you have prescribed a velocity at a boundary and want to step until a specific total displacement has been reached. Generally, however, you wish to cycle until a certain condition has been reached—for example, a dynamic simulation could cycle until a certain amount of time has passed. The `model solve` command allows cycling to continue until one or more limits (specified using subsequent keywords) has been met. In a static model, the most common limit of interest is “until the model is in equilibrium.”

Because *FLAC3D* uses an iterative procedure informed by physics to reach static equilibrium, when we refer to being “in equilibrium,” we mean the same thing as being fully converged to a static solution. *FLAC3D* provides a number of different ways to measure and visualize how close you are to being converged or having reached an equilibrium state in your model. But, as always, the more you are aware of the issues involved, the more efficient and accurate your models are likely to be.

Convergence Criteria

FLAC3D has five different criteria available to determine when equilibrium is reached. Maximum Out-of-Balance Force, Local Force Ratio, Average Force Ratio, Maximum Force Ratio, and Convergence. Each of these options is described below (see [Out-of-Balance Force and Ratio Calculation](#) for more detailed formula).

Maximum Out-of-Balance Force

The first and simplest measure used for convergence is called the **maximum out-of-balance force**. While no longer used often, it is still available both as a `model solve criteria` and to the `zone.unbal FISH` intrinsic.

Every gridpoint (or structural node) in the model receives forces from zones, elements, gravity, etc. Each of these forces is added together, resulting in the net forces acting on a gridpoint. The maximum out-of-balance force is the maximum absolute value of any one component of the force remaining after all forces are summed together. This is the force that drives the gridpoint in the direction of equilibrium. If the force is zero, the gridpoint can be thought of as being converged, or at equilibrium.

As a measure of overall convergence, the major drawback of maximum out-of-balance force is that it is not a unitless measure. The appropriate target value is dependent on the model size, zone size, material properties, and stress magnitude. It is impossible to provide a single value to use as a rule of thumb. This is why it is no longer in common use as a solve criteria.

Local Maximum Force Ratio

The **local force ratio** is a unitless measure of *local* convergence at a gridpoint. The idea is simple: divide the out-of-balance component remaining after summing forces by a measure of the total forces being applied to the gridpoint. For performance purposes, we use the “Manhattan norm” or “Taxicab norm” of a vector, defined as:

$$\langle v_i \rangle = |v_x| + |v_y| + |v_z|$$

or the sum of the absolute values of the components of the vector. The local force ratio can then be defined as:

$$\frac{\langle \sum_i f_i \rangle}{\sum_i \langle f_i \rangle}$$

where f_i are all the force vectors being added to a gridpoint.

The **local maximum force ratio** is the maximum local ratio of any gridpoint in the model. This is a very conservative measure of the convergence of a model. However it has a weakness when there is less than two forces being applied to a gridpoint, or when all forces being applied to a gridpoint act in the same direction. In this circumstance, the only “equilibrium” value possible is for both numerator and denominator to converge toward zero; the ratio between them will tend toward 1.

FLAC3D will detect when only one force vector is being applied to a gridpoint and automatically remove it from consideration. But if multiple aligned forces are acting on a gridpoint, or if the second force is negligible compared to a main driving force, then this check is insufficient. These points will dominate the maximum local force ratio, and the model will *seem* to never effectively converge.

In practice this does not happen very often. The majority of models never see this effect. But it *does* happen, and so maximum local force ratio is not used as the default convergence criteria for `model solve` in order to guarantee that any given model will converge using the default criteria.

The last local force ratio calculated for a gridpoint is stored and is available to *FISH* as `gp.ratio` and to plot items as the Local Force Ratio.

Average Force Ratio

The **average force ratio** is defined as the sum of all out-of-balance force components at every gridpoint divided by the sum of all total forces applied at a gridpoint. Or:

$$\frac{\sum_j^n \langle \sum_i f_i \rangle}{\sum_j^n \sum_i \langle f_i \rangle}$$

This measure serves as a reliable measure of the overall convergence of the system. For models that are relatively uniform, it performs very well. It is bulletproof enough that the default convergence criteria for `model solve` is `ratio-average 1e-5`.

However, for very non-uniform models, the effects of localized convergence problems can be lost in the average, making the user unaware that an area that might be of most interest still needs

computation. This can occur either because of disparities in zone size, or because of large differences in stiffness. For large complex models, this measure should be checked carefully before being trusted. See the following discussion.

Maximum Force Ratio

The **maximum force ratio** is defined as the maximum out-of-balance force divided by the average total force acting on all gridpoints. The average total force is defined as:

$$\frac{\sum_j^n \sum_i \langle f_i^j \rangle}{N}$$

where N is the total number of gridpoints in the model.

Ratio

The three force ratio types—local, maximum, and average—can be referred by the single `ratio` keyword. By default, `ratio` maps to the average force ratio. The `zone ratio` command can be used to indicate that `ratio` should mean local or maximum force ratio instead. This allows the user to choose which criteria is driving various servo-driven capabilities of the code that are tied to the `ratio` value.

In many cases, you can also explicitly indicate which criteria you mean by using the `ratio-average`, `ratio-local`, or `ratio-maximum` keywords.

Convergence

The `convergence solve limit` is new in *FLAC3D* 6.0 and is still somewhat experimental. It shows promise, however, as a means of giving the user more control over the convergence criteria desired and overcoming the possible disadvantages of relying on the local force ratio.

First - a new gridpoint field variable called the Local Force Target is created. This value defaults to `1e-4` in every gridpoint and node. It can be changed with the `zone gridpoint initialize ratio-target` command and with the `gp.ratio.target FISH` intrinsic. This value indicates the local

force ratio that is to be considered “converged” *for that specific gridpoint*. This means that the local convergence criteria can vary in different locations of the grid.

We define a new quantity, unimaginatively called the **convergence**, that is the ratio of the *current* local force ratio to the *target* local force ratio. A convergence of 1.0 is therefore considered “converged”. The **maximum convergence** is then a measure of overall model convergence tied to the `convergence` keyword in `model solve`. The convergence value is available to plot items, and to *FISH* as the `gp.convergence` intrinsic. By default, a convergence of 1.0 is exactly the same as a `ratio-local` limit of $1e-4$.

In the event that the local ratio calculation is failing in corner cases, the `ratio-target` can be set to a high value in that gridpoint or node, effectively removing them from the overall convergence criteria. In addition, it is not unusual for a small region of the model to have an overly large effect on the time it takes to reach convergence—either because they are very small, or because they are very stiff, or both. For example, it is not unusual for a small number of very stiff structural supports to dominate the convergence time of a very large model. The `ratio-target` for those nodes could be set to $1e-3$ or even higher, relaxing the criteria for those nodes once it is determined that this will have negligible effect on the model results of primary interest.

Choosing Convergence Criteria

The default convergence criteria if none is specified for a `model solve` command is a `ratio` of $1e-5$. In our experience, an average force ratio of $1e-5$ corresponds to a local force ratio limit of around $1e-4$ for a model of average complexity. This is why $1e-4$ was chosen as a default `ratio-target`.

A maximum local ratio of $1e-4$ is generally sufficient, even conservative, for most engineering applications. A maximum local ratio of $1e-3$, or even as high as $1e-2$ may be sufficient for many purposes. This is especially true for intermediate stages (excavation sequences, etc.) of a model, where taking the time to reach “full” convergence after every step may have negligible effects on the final state of a model.

- In general, we recommend that one use a `convergence` of 1 based on a `ratio-target` of `1e-4` unless computation time is limited. Whether or not these target convergence criteria are raised, we recommend you take the time to evaluate the model to ensure that you have sufficient convergence for the results you are interested in. This is discussed in the next section.

Evaluating Equilibrium

For the purposes of illustration, we will demonstrate with a cartoonishly simple model of a surface excavation surrounded by a reinforcing wall. The basic geometry is shown in [Figure 1](#).

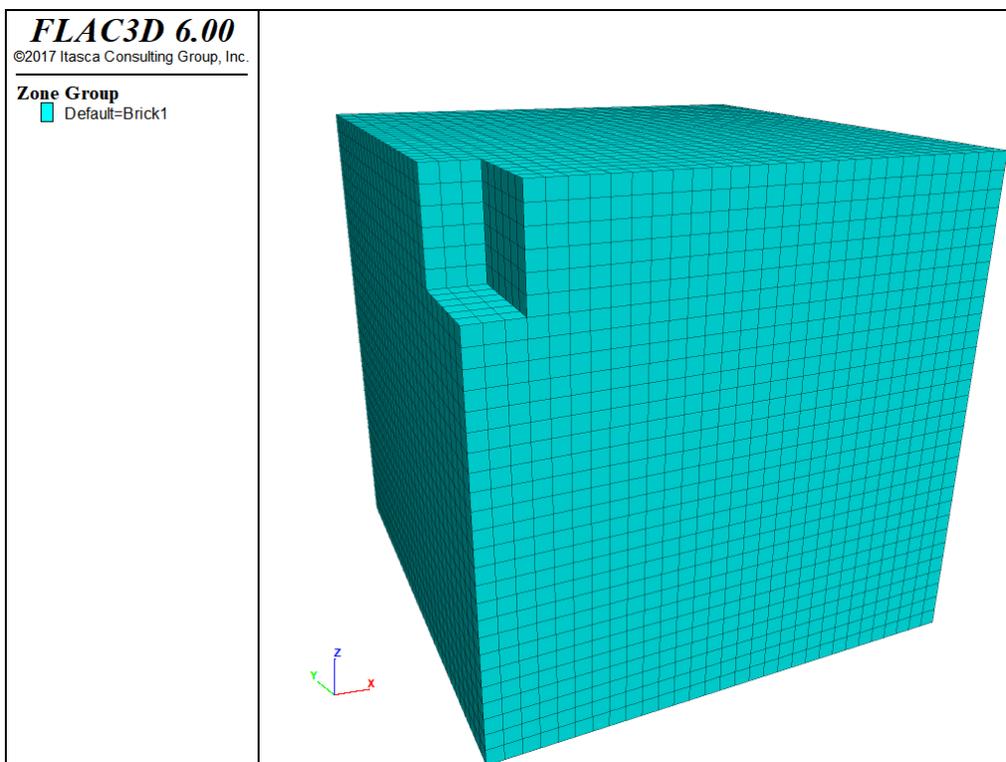


Figure 1: Cartoon model of a trench excavation.

The full data file for the first demonstration is below. Note that `zone relax excavate` was deliberately not used for the purposes of illustration.

```
model new
; Create zones
zone create brick size 30 30 30
zone face skin
```

```

zone group 'Trench' range position (0,0,24) (3,6,30)
; Constitutive model and properties
zone cmodel assign mohr-coulomb
zone property bulk 3e6 shear 2e6 density 1000
zone property cohesion 1e5 friction 25 tension 1e3
zone cmodel assign elastic range group 'Wall'
zone property bulk 3e6 shear 2e6 density 1000
; Boundary conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
; Initial conditions
model gravity 10
zone initialize-stresses
model solve
; Histories
model history mechanical ratio-average
zone history displacement-x position (3,0,30)
zone history displacement-z position (4,0,30)
; Excavate
zone delete range group 'Trench'
; Solve
model solve
model save 'simple'

```

There are several things that should always be done to check to see if your model has actually reached sufficient equilibrium. **First**, take histories of displacements and/or velocities in regions of primary interest. If these displacements are still changing relatively rapidly when cycling completes, the model is not done adjusting. **Second**, make plots (contours and isosurfaces) of Local Force Ratio or Convergence. An isosurface plot of Convergence is particularly useful to tell you where the model is taking the longest to converge, and where you should look closer in a contour plot (perhaps using cut-planes to see into the interior of the model).

Figure 2 shows the histories of averaged force ratio and displacements in the top of the trench. Note that the average force ratio is plotted on a log scale. For a well-behaved model, this sort of linear logarithmic (or exponential) convergence is expected. Failure or other non-linear behavior will show disturbances in convergence, and progressive uncontained failure will make the curve level out. The displacement history, again, shows a typical slightly underdamped response. It is clear that the top of the trench wall has stopped moving significantly and the model has effectively reached equilibrium.

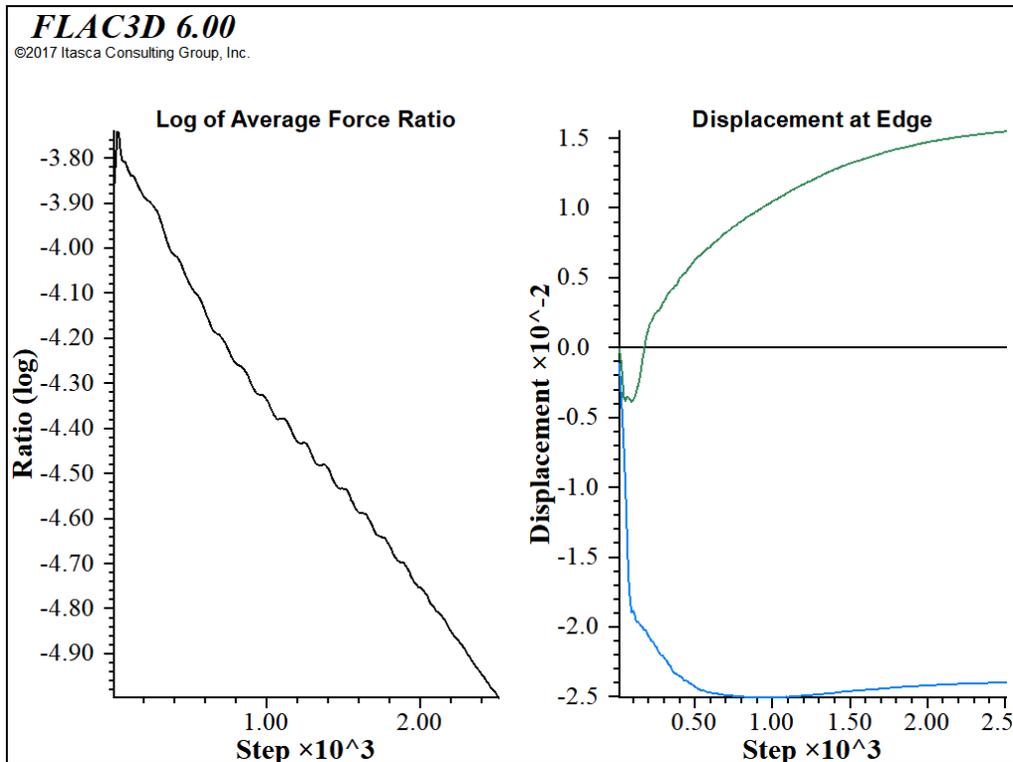


Figure 2: Histories of average unbalanced ratio and displacement for the simple trench.

Figure 3 shows the result of examining contours of convergence. While the bulk of the model has a convergence well below 1, a few gridpoints near the surface are still much higher. This is fairly typical of this sort of problem; the surface gridpoints have the same inertial mass resisting motion and smaller driving forces due to stress. The gridpoints at the bottom tend to (and need to) converge first, and small adjustments propagate against gravity, so the surface gridpoints converge last. If a stricter convergence criteria of `convergence 1` was used, the model would take half again as many steps to reach equilibrium. How important this is depends on the significance of relatively small adjustments in the surface gridpoints. If exact surface settlement values were critical, this might be a necessary change.

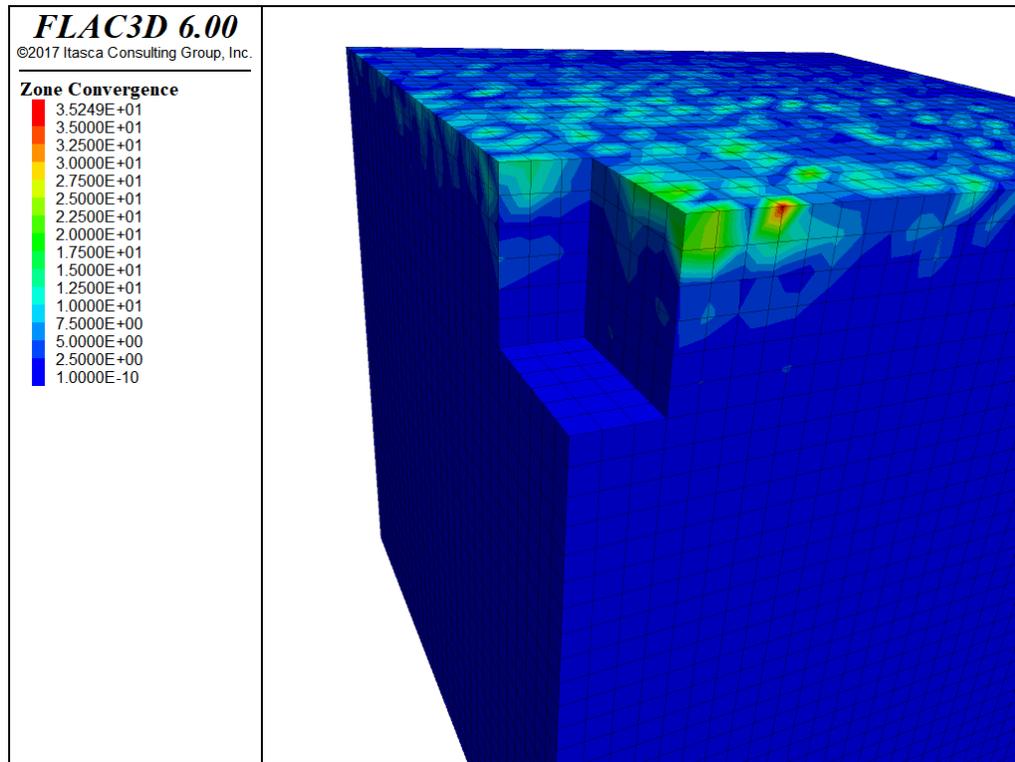


Figure 3: Contours of gridpoint convergence.

Effects of Large Stiffness Differences

To illustrate some of the possible difficulties, we introduce a large localized stiffness difference by adding a steel liner to the trench after excavation. The complete data file for this modification is shown below.

```

model new
; Create zones
zone create brick size 30 30 30
zone face skin
zone group 'Trench' range position (0,0,24) (3,6,30)
zone face group 'Wall' internal ...
      range group 'Brick1' group 'Trench' position-z 24.1 30
; Constitutive model and properties
zone cmodel assign mohr-coulomb
zone property bulk 3e6 shear 2e6 density 1000
zone property cohesion 1e5 friction 25 tension 1e3
; Boundary conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
; Initial conditions
model gravity 10
zone initialize-stresses

```

```

model solve
; Histories
model history mechanical ratio-average
zone history displacement-x position (3,0,30)
zone history displacement-z position (4,0,30)
; Excavate
zone delete range group 'Trench'
; Install liner
struct shell create by-face range group 'Wall'
struct shell property isotropic 10e9 0.1 thick 1
model solve
model save 'stiff'

```

The histories for the modified model are shown in Figure 4. While the average force ratio shows similar characteristics, the histories indicate that the model has not settled down into equilibrium. The issue is that the very stiff structural wall results in high inertial masses (see [Mechanical Timestep Determination for Numerical Stability](#)), while the forces driving settlement remain the same. This means the gridpoints connected to the shell elements move relatively slowly, and their convergence values lag behind the rest of the model. Eventually, this localized disturbance can be drowned out by the bulk of the model, and *on average*, the model will appear converged.

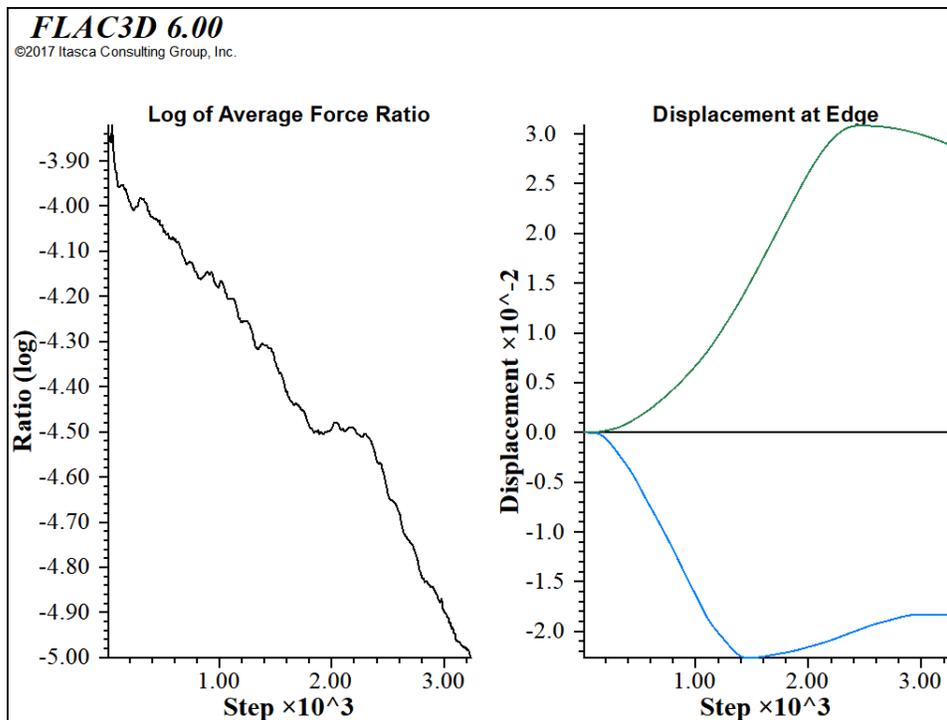


Figure 4: Histories of average unbalanced ratio and displacement for a trench with a stiff reinforcing wall.

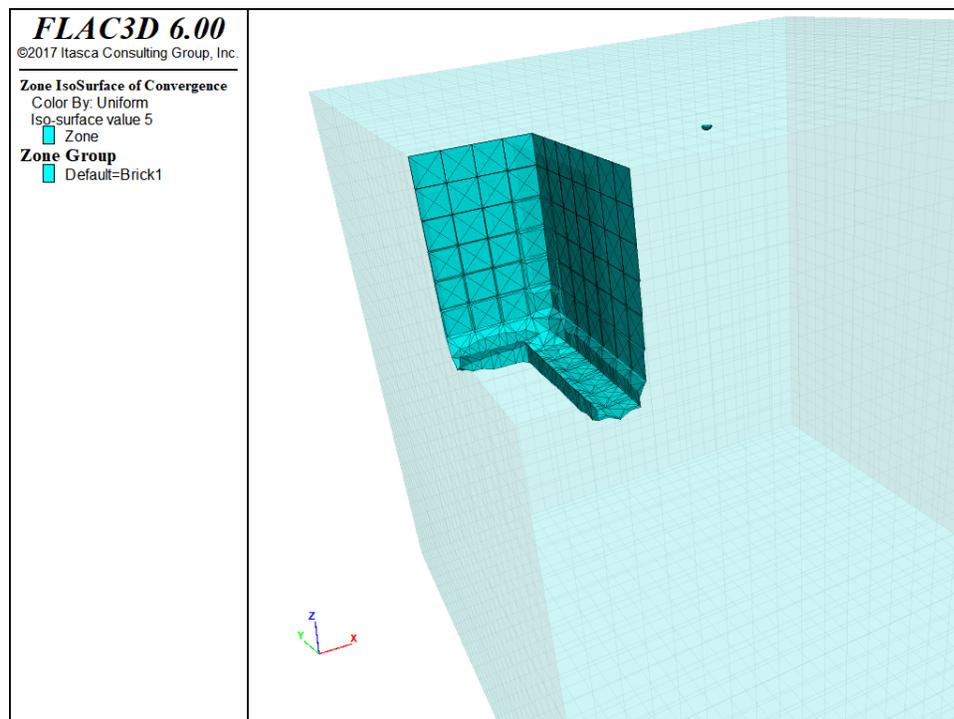
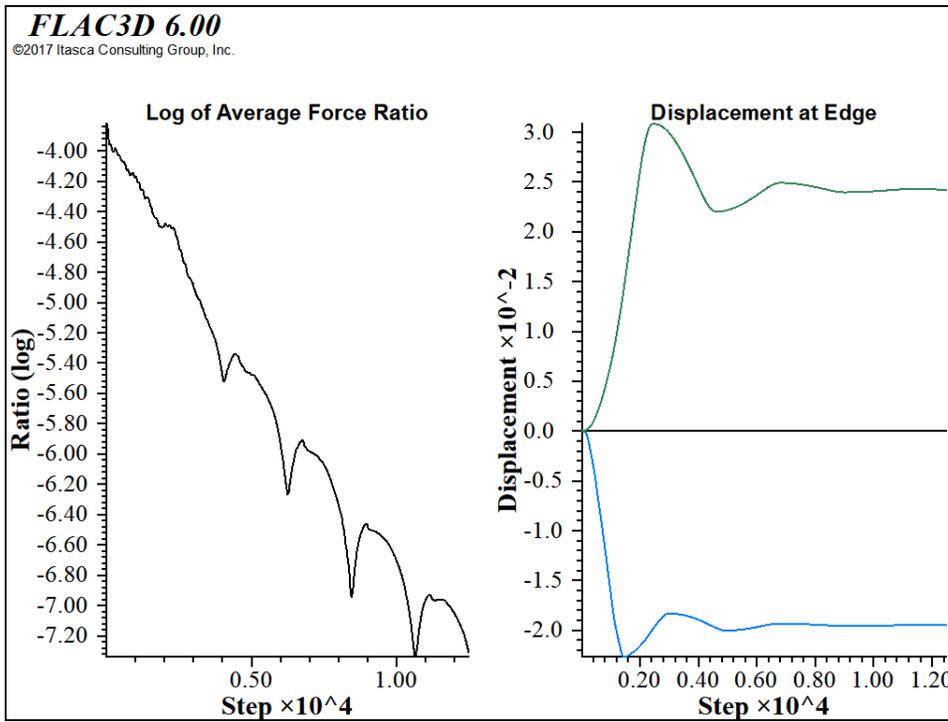


Figure 5: Isosurface of Convergence 5.

An isosurface plot of Convergence shown in Figure 5 reveals that the area around the wall is having trouble coming to equilibrium. If the convergence criteria used was changed to

```
model solve convergence 10
```

then the model would take much longer, but would also reach a state where there was some confidence that equilibrium had been achieved, as seen in Figure 6.



Loading and Sequential Modeling

By applying different model loading conditions at different stages of an analysis, it is possible to simulate changes in physical loading, such as sequences of excavation and construction. Changes in loading may be specified in a number of ways (e.g., by applying new stress or displacement boundaries, by changing the material model in zones to either a null material or to a different material model, or by changing material properties).

It is important to recognize that sequential modeling follows the stages of an engineering work (e.g., the stages in excavation and construction of a sheet pile wall). This modeling does *not*, however, include physical time as a parameter, and time-dependent behavior *cannot* be simulated.[1] Some engineering judgement must be used to estimate the effects of time. For example, a model parameter may be changed after a predetermined amount of displacement or strain has occurred. This displacement may be estimated to have occurred over a given period of time.

The following guidelines should be followed when performing loading changes or defining stages in a sequential analysis:

1. As discussed in the **Grid Generation** section, all regions to be excavated must correspond to zone boundaries established on creation. Zones and gridpoints can be moved after creation, but their stress states will *not* adjust to the new configuration, stresses are only modified as a result of velocity increments during cycling.
2. Zones and gridpoints can be created after cycling, to add material to a dam for example. But these zones are created with zero stresses and no constitutive model, as with all zones on creation.
3. In general, excavations should use `zone relax excavate` to minimize pseudo-inertial effects on the solution.

4. When material models are changed during a simulation sequence, all properties must be re-specified for the new model, even if the affected zones were previously assigned the same properties. Properties are lost when models are changed. Stresses in zones are preserved when models are changed, unless the zones are changed to null zones; in this case, all stresses in the affected zones are set to zero.
5. If the model is in equilibrium, a change in elastic properties will have no effect on the response of the model because elastic moduli are *tangent* moduli, not *secant* moduli. The model must be subjected to a change that causes unbalanced forces to develop. This may be caused, for example, by a change in strength properties if the current stresses exceed the new strength limit.

The recommended approach to sequential modeling is demonstrated in the example that follows this topic.

Example: Loading on Three Tunnels

This problem involves the analysis of the loading on three tunnels (a service tunnel and two main tunnels) that are sequentially excavated and lined. The grid generation for this problem was described previously in [Fitting the Grid to Simple Shapes](#).

The construction sequence to be analyzed consists of three modeling stages:

1. excavation of the first half of a service tunnel; then
2. installation of a liner in the first half, and excavation of the second half of the service tunnel; then
3. installation of the liner in the second half of the service tunnel, and excavation of the first half of the main tunnel.

The objective of the analysis is to investigate the influence of the main tunnel excavation on the response of the service tunnel.

The model is constructed to take advantage of symmetry in the problem. The service tunnel is located midway between the main tunnels, so a vertical plane of symmetry may be assumed to exist along the centerline of the service tunnel. The model grid is created in the following data file.

Sequential excavation and lining of tunnels — initial grid

```

model new
; main tunnel
zone create radial-cylinder point 0 (15, 0,0) point 1 (23,0,0) ...
                             point 2 (15,50,0) point 3 (15,0,8) ...
                             size 4 10 6 4 dim 4 4 4 4 rat 1 1 1 1 ...
                             group 'rock' fill group 'main1'
zone reflect dip 90 dip-direction 90 origin (15,0,0)
zone reflect dip 0 origin (0,0,0)
; service tunnel - create two gridpoints as reference location
zone gridpoint create (2.969848, 0.0,-0.575736) name='ref1'
zone gridpoint create (2.969848,50.0,-0.575736) name='ref2'
zone create radial-cylinder point 0 (0, 0,-1) point 1 (7, 0,0) ...
                             point 2 (0,50,-1) point 3 (0, 0,8) ...
                             point 4 (7,50, 0) point 5 (0,50,8) ...
                             point 6 (7, 0, 8) point 7 (7,50,8) ...
                             point 8 gridpoint 'ref1' ...
                             point 10 gridpoint 'ref2' ...
                             size 3 10 6 4 dim 3 3 3 3 rat 1 1 1 1 ...
                             group 'rock' fill group 'service1'
zone create radial-cylinder point 0 (0, 0,-1) point 1 (0, 0,-8) ...
                             point 2 (0,50,-1) point 3 (7, 0, 0) ...
                             point 4 (0,50,-8) point 5 (7,50, 0) ...
                             point 6 (7, 0,-8) point 7 (7,50,-8) ...
                             point 9 gridpoint 'ref1' ...
                             point 11 gridpoint 'ref2' ...
                             size 3 10 6 4 dim 3 3 3 3 rat 1 1 1 1 ...
                             group 'rock' fill group 'service1'

; outer boundary
zone create radial-tunnel point 0 ( 7, 0, 0) point 1 (50, 0, 0) ...
                          point 2 ( 7,50, 0) point 3 (15, 0,50) ...
                          point 4 (50,50, 0) point 5 (15,50,50) ...
                          point 6 (50, 0,50) point 7 (50,50,50) ...
                          point 8 (23, 0, 0) point 9 ( 7, 0, 8) ...
                          point 10 (23,50, 0) point 11 ( 7,50, 8) ...
                          size 6 10 3 10 rat 1 1 1 1.1 group 'rock'
zone create brick point 0 ( 0,0, 8) point 1 ( 7, 0, 8) point 2 (0,50, 8) ...
                  point 3 ( 0,0,50) point 4 ( 7,50, 8) point 5 (0,50,50) ...
                  point 6 (15,0,50) point 7 (15,50,50) ...
                  size 3 10 10 rat 1 1 1.1 group 'rock'
zone reflect dip 0 origin (0,0,0) ...
                  range position-x 0 23 position-y 0 50 position-z 8 50
zone reflect dip 0 origin (0,0,0) ...
                  range position-x 23 50 position-y 0 50 position-z 0 50
zone group 'main2' range group 'main1' position-y 25 50
zone group 'service2' range group 'service1' position-y 25 50

```

```

zone face group 'liner1' internal range group 'service1' group 'rock'
zone face group 'liner2' internal range group 'service2' group 'rock'
zone face skin
model save 'grid'

```

The resulting grid is shown in the next figure. The grid is manipulated such that the invert of the service tunnel is at the same elevation as that of the main tunnel (see [Fitting the Grid to Simple Shapes](#)).

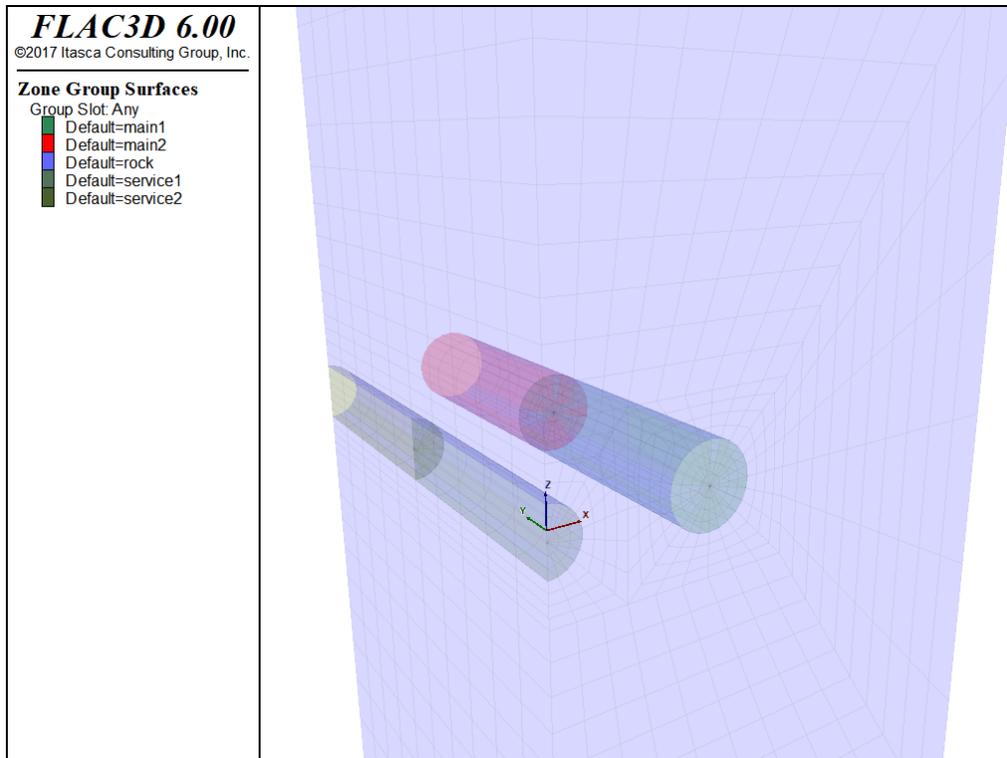


Figure 1: *FLAC3D* grid of main tunnels and service tunnel.

A Mohr–Coulomb material model is initially defined for all zones. The plane of symmetry is specified, and an isotropic initial stress is assigned to the model. The model is in force equilibrium before an excavation is made. Note that only one calculation step is taken when the `model solve` command is issued, because the model is in equilibrium for the specified boundary and initial conditions. The next listing shows the commands for this stage.

Sequential excavation and lining of tunnels — initial state

```

model restore 'grid'
; mohr-coulomb model
zone cmodel assign mohr-coulomb
zone property shear 0.36e9 bulk 0.6e9 cohesion 1e5 friction 20 tension 1e5
; Boundary conditions
zone face apply velocity-normal 0 range group 'Bottom'
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply stress-normal -1.4e6 range group 'Top'
; Initial Conditions
zone initialize stress xx -1.4e6 yy -1.4e6 zz -1.4e6
; Histories
model history name='conv' mechanical convergence
zone history name='s-f' displacement-x position (3,0,-1)
zone history name='c-f' displacement-z position (0,0,2)
zone history name='s-m' displacement-x position (3,25,-1)
zone history name='c-m' displacement-z position (0,25,2)
model solve convergence 1
model save 'initial'

```

In the first stage, the solution is found with a 25 m section of the service tunnel excavated. The excavation is made using the `zone relax excavate` command. [2] The stage 1 commands are listed below.

Sequential excavation and lining of tunnels — stage 1

```

; excavate 25 m section of service tunnel
model restore 'initial'
zone relax excavate range group 'service1'
model solve convergence 1
model save 'stage1'

```

The gridpoint history plots in the next figure for displacement at the springline and crown of the service tunnel indicate that the model has reached equilibrium within approximately 2500 calculation steps for stage 1.

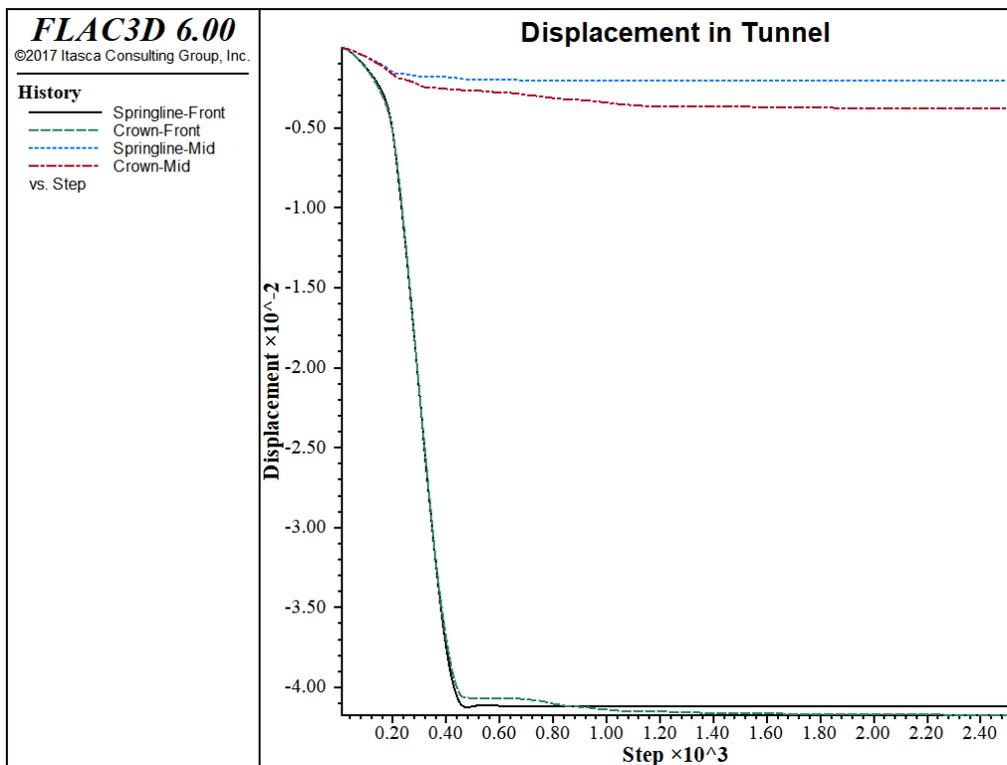


Figure 2: Displacement histories at crown and springline of service tunnel for stage 1.

In the second stage, the lining is installed along the excavated section of the service tunnel. The lining is modeled using structural shell elements, and an elastic liner is assumed. Fixity conditions are specified for structural element nodes located on the symmetry plane of the grid to correspond to the same symmetry condition for the liner.

The second 25 m section of the service tunnel is then excavated, and the solution is calculated for support provided by the lining, as seen here:

Sequential excavation and lining of tunnels — stage 2

```

model restore 'stage1'
zone gridpoint initialize displacement (0,0,0)
history purge
; Excavate second half of service tunnel
zone relax excavate range group 'service2'
; Install liner in first half of service tunnel
struct shell create by-face range group 'liner1'
struct shell property isotropic (25.3e9,0.266) thick 0.5
; Shell symmetry boundary conditions
struct node fix velocity-y rotation-x rotation-z range position-y 0

```

```

struct node fix velocity-x rotation-y rotation-z range position-x 0
; Solve to equilibrium
model solve convergence 1
model save 'stage2'

```

We initialize the displacements in the model so that we can focus on the response due only to the excavation in the second stage. The displacement histories at the two ends of the lined section of the service tunnel (i.e., at $y = 0$ and $y = 25$) indicate a nonsymmetric response of the service tunnel as a result of excavating the remainder of the service tunnel.

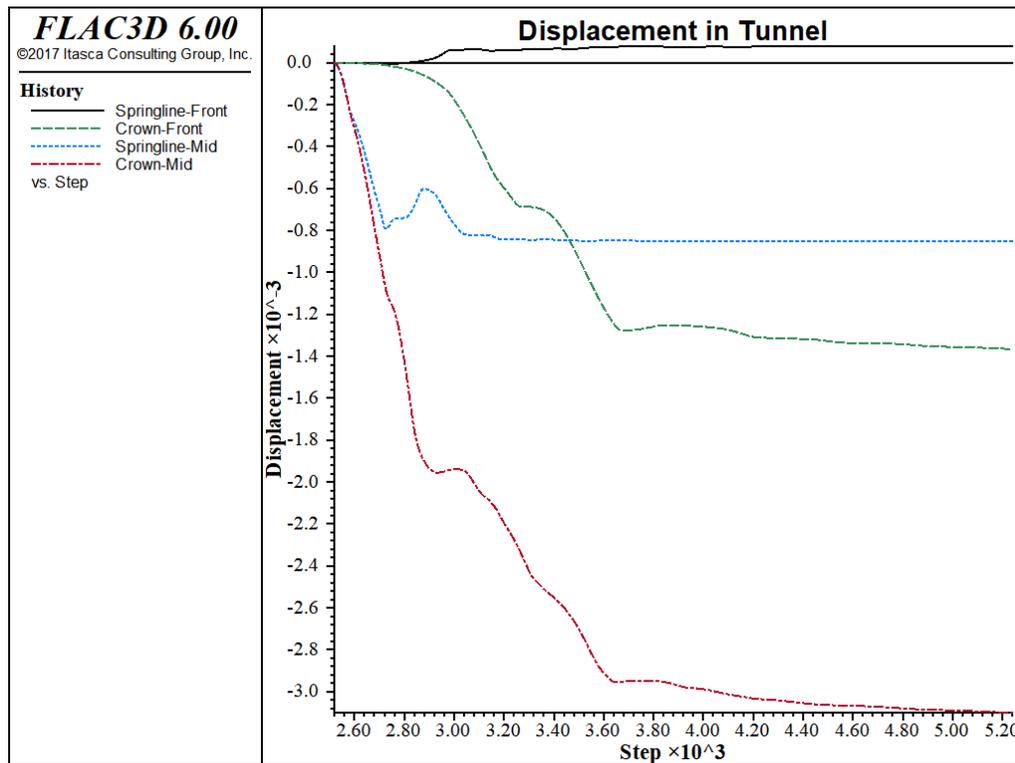


Figure 3: Displacement histories at crown and springline of service tunnel for stage 2.

Next, the second section of the service tunnel is lined, and the first section of the main tunnel is excavated.

Sequential excavation and lining of tunnels — stage 3

```

model restore 'stage2'
zone gridpoint initialize displacement (0,0,0)
history purge
; Excavate first half of main tunnel

```

```

zone relax excavate range group 'main1'
; Install liner in second half of service tunnel
struct shell create by-face range group 'liner2'
struct shell property isotropic (25.3e9,0.266) thick 0.5
; Shell symmetry boundary conditions
struct node fix velocity-y rotation-x rotation-z range position-y 50
struct node fix velocity-x rotation-y rotation-z range position-x 0
; Solve to equilibrium
model solve convergence 1
model save 'stage3'

```

Again the displacements are initialized so we can see the effects of the excavation of the main tunnel. Significantly more displacement is measured in the service tunnel than was seen in the first two stages, and a greater asymmetry in the response. This is also evident in the contour plot of the displacement magnitude in the grid, and the minimum (i.e., major) principal stress (σ_1) in the lining (see the second figure below).

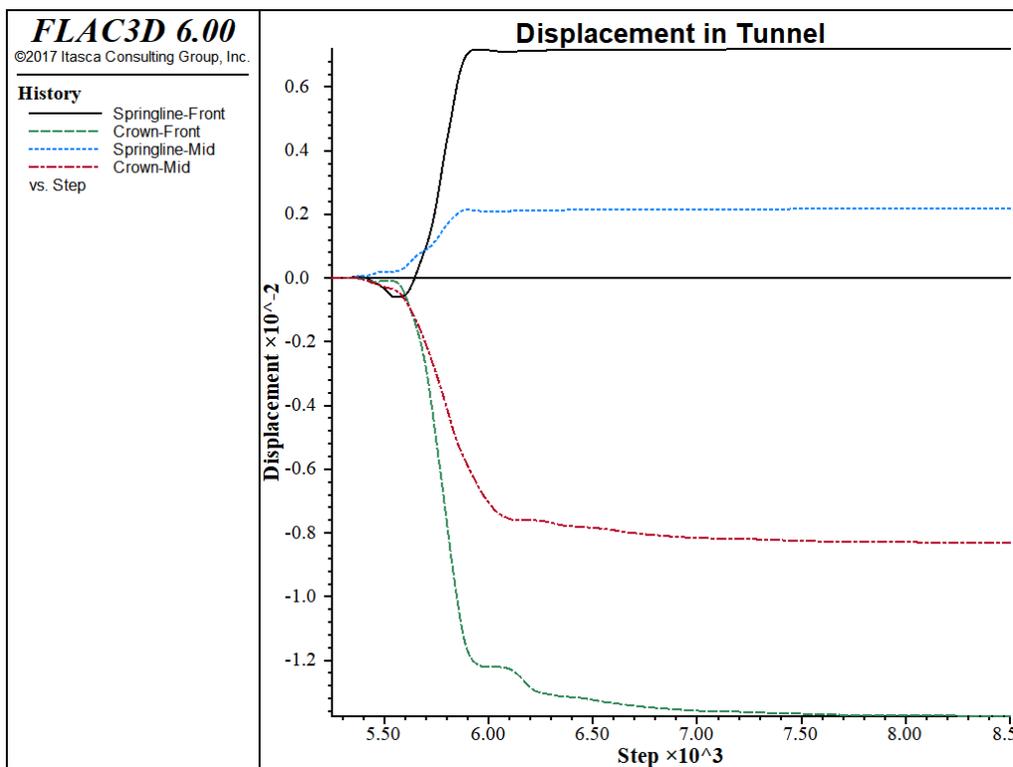


Figure 4: Displacement histories at crown and springline of service tunnel for stage 3.

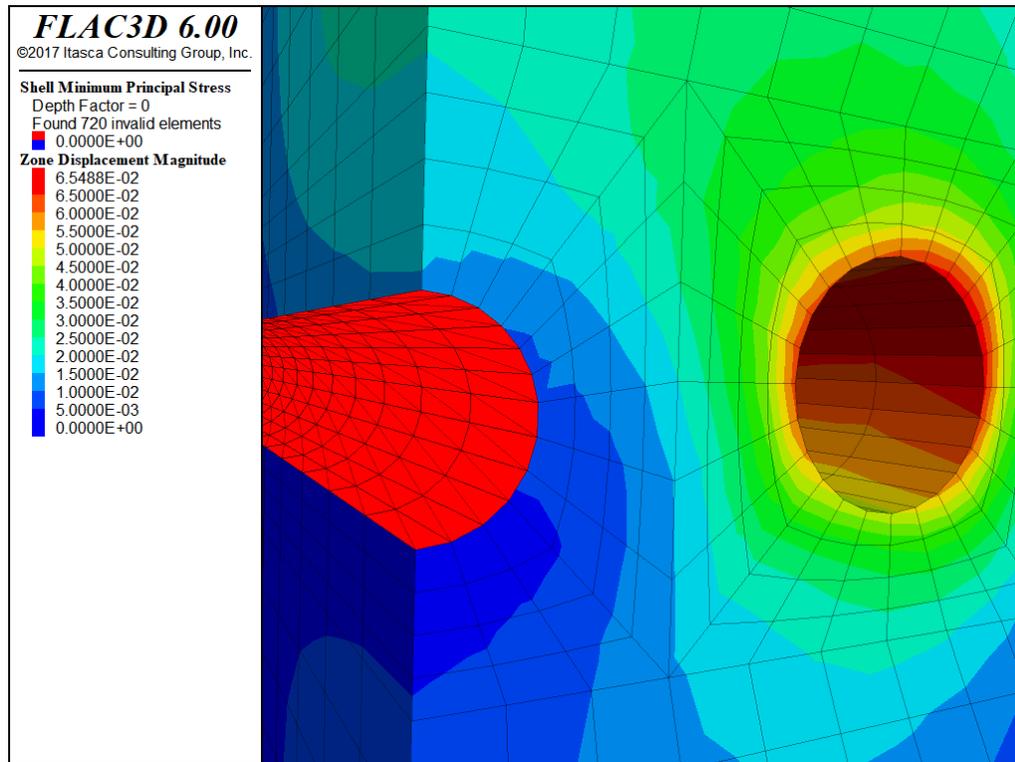


Figure 5: Contour plot of displacement magnitude in the grid and minimum principal stress in the lining after partial excavation of the main tunnel.

The modeling sequence may be repeated with different conditions of material properties, or locations of main tunnels relative to the service tunnel. If new tunnel locations are investigated, the grid must be regenerated and the model brought to an initial equilibrium state again. If different material properties are used, the model must be solved first for the response of the unlined service tunnel. Always remember that the model *must* be at an equilibrium state when the loading change is made.

Endnotes

- [1] This approach is a simplistic simulation of the progressive advancement of the tunnel. For a more realistic model of sequential tunnel construction, see [Excavation and Support for a Shallow Tunnel](#).
- [2] Of course, here we are referring to static mechanical processes. Transient calculations can be performed for groundwater flow and creep and thermal analyses with *FLAC3D*. Dynamic analysis can be performed with the dynamic option described in [Dynamic Analysis](#).

Structural Support

An overview of the structural element logic in *FLAC3D* and the types of support available can be found in [Structural Elements](#).

Interfaces

Often it is necessary in a model to represent planes on which sliding or separation can occur. The classic example is a geologic fault, but this can be also be desired at the boundary between a structure and material like soil.

Where this is necessary, interfaces can be created on zone faces to represent such a fault. Interfaces use a Coulomb sliding with tensile and shear bonding model. While *FLAC3D* can handle multiple interfaces, if the number or complexity of the surfaces becomes too high, it is recommended that *3DEC* be considered instead.

For a complete discussion of *FLAC3D* interfaces and their most effective use, see [Interfaces in *FLAC3D* Theory and Background](#). This section will focus on a simple first introduction to interfaces and the issues you need to be most aware of.

Interfaces are most often created with the `zone interface create by-face` command:

```
zone interface 'flt' create by-face separate range position-z 5
zone interface 'flt' node property stiffness-normal 1e8 stiffness-shear 1e8
```

This command creates an interface named `flt` by separating all zone faces found at $z=5$ and placing interface elements and nodes on one side. Contacts are found (and forces generated) by interface nodes coming into contact with zone surface faces that are not part of the interface itself.

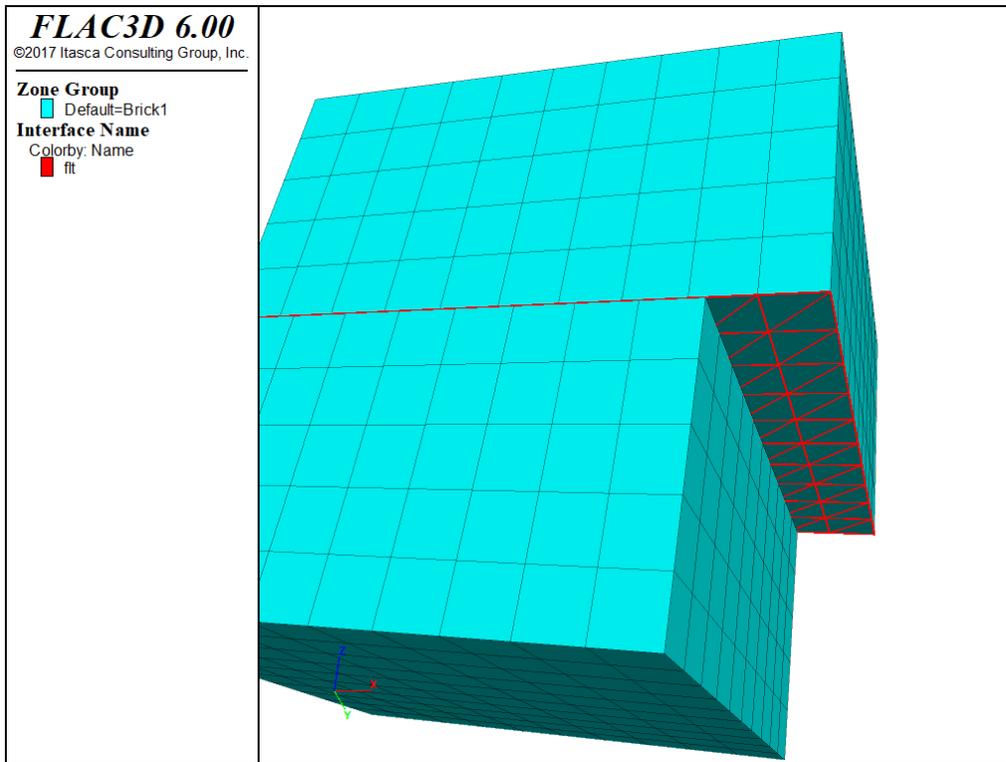


Figure 1: Discontinuity resulting from interface slip.

Every reference to an interface uses a name (like the `table` command) to indicate which interface is being referred to. An interface will not come into contact with itself, so you cannot necessarily use the same interface name for all contacting surfaces in the model.

You may be familiar with interfaces from *FLAC*, but the *FLAC3D* interface logic differs in many respects. There are a few important peculiarities of the *FLAC3D* interface logic that you must become familiar with.

Interfaces in *FLAC3D* are one-sided objects, unlike the interfaces in *FLAC*. Interface *elements* are (in general) created attached to zone surface faces, which in turn create interface *nodes*. All contact detection is between interface nodes and zone surface faces.

Creating interface elements on only one side of an existing discontinuity in the mesh can sometimes be difficult. When faces are selected using geometric range elements, faces on both sides will fall in the exact same location. If the zones on either side have different group names, then this can be used to select a side. Also, the `range orientation` range element can be used to select faces with a normal vector in a selected direction.

In the event the discretization on either side of the fault is not the same, interface elements should be applied to the side with the smaller faces.

Most commonly, interfaces need to be created starting with a mesh that is continuous. In that case, it is easiest to separate the mesh and create the interfaces all in one command using the `separate` keyword (as shown in the example above). The `new-side-origin` keyword can be used to determine which side of the separation interface elements are applied to. The side *opposite* the location of the origin will be selected. In the example above, to select the other side, change the command to:

```
zone interface 'flt' create by-face separate new-side-origin (0,0,10) ...
range position-z 5
```

Absolute Penetration

Unlike almost all other systems in *FLAC3D*, the calculation of normal stress due to interface penetration is *absolute* instead of *incremental*. This means that moving a gridpoint relative to an interface in the normal direction will have an immediate effect on the normal stress calculated. This is *not* true of accumulated shear stress, which remains incremental.

In small-strain mode, gridpoints do not actually move their location in response to accumulated velocities. Absolute penetration could be determined from the displacement field, but in *FLAC3D*, displacements are an output value only and can be reset at any time. Therefore, in support of small-strain interfaces, we introduce what is called the “small-displacement” field. This is a vector field that is accumulated from the velocity field every step in small-strain mode only. For the purposes of calculating interface penetration, a gridpoint location is based on the current location plus the small-strain displacement. The small-strain displacement may be initialized using the `zone gridpoint initialize displacement-small` command, and this is sometimes useful to reset the effective penetration of an interface after some grid motion.

To initialize interface stresses without actually moving gridpoints, we introduce a *normal stress increment* that is added to the normal stress calculated from absolute penetration. It is this value that is set when the `zone interface node initialize-stresses` command is given. It can also be set explicitly using `zone interface node stress-normal-increment`.

Single Target

A given interface node can only be in contact with one target surface face at a time.

For a single fault in a material, one target face per interface node is all that is necessary to accurately model a fault. However, if multiple interfaces intersect, there are not enough contacts per node at the intersection line to resolve the system. See the [Interfaces](#) section for further discussion on ways to mitigate this issue. However, in the end, if the exact behavior around multiple intersecting faults is critical to be modeled, it may be better to use *3DEC*.

Tips and Advice

When problem solving with *FLAC3D*, it is important to optimize the model for the most efficient analysis. This section provides several suggestions on ways to improve a model run, as well as some common pitfalls to be avoided when preparing a *FLAC3D* calculation.

1. Check Model Runtime

The solution time for a *FLAC3D* run is proportional to $N^{4/3}$, where N is the number of zones. This formula holds for elastic problems solved for the equilibrium condition. The runtime will vary somewhat, but not substantially, for plasticity problems, and it may be much larger if continuing plastic flow occurs. It is important to check the speed of calculation on your computer for a specific model. An easy way to do this is to run the benchmark test [below](#). Then use this speed to estimate the speed of calculation for the specific model based on interpolation from the number of zones.

2. Effects on Runtime

FLAC3D will take longer to converge if:

- a. there are large contrasts in stiffness in zone materials or between zones, structural members and interfaces; or
- b. there are large contrasts in zone sizes.

The code becomes less efficient as these contrasts become greater. The effect of a contrast in stiffness should be investigated before performing a detailed analysis. For example, a very stiff loading plate can be replaced by a series of fixed gridpoints that are given a constant velocity. (Remember that the `zone face apply` or `zone gridpoint fix` commands may be used to fix velocities, *not* displacements.) The inclusion of groundwater will act to increase the apparent mechanical bulk modulus (see the [Fluid–Mechanical Interaction](#) section).

It is possible that the overall model response of interest may not be significantly affected by requiring full convergence of small stiff regions (due to material properties or zone size). In this case, using the `convergence` solve criteria and increasing the `ratio-target` of the area may significantly decrease the required run time. As always, carefully analyze the model response to be satisfied that this is an acceptable additional error.

3. Considerations for Zoning Density

FLAC3D uses constant-strain elements. If the stress/strain gradient is high, you will need many zones to represent the varying distribution. Run the same problem with different zoning densities to check the effect.

Try to keep the zoning as uniform as possible, particularly in the region of interest. Avoid long, thin zones with aspect ratios greater than 5:1, and avoid jumps in zone size (i.e., use smoothly graded grids). Make use of the `ratio` keyword with the `zone create` command to grade zone sizes smoothly from regions with fine zoning to regions with coarse zoning. If zone densification has been done with `zone densify`, we recommend using the `gradient-limit` keyword to minimize the zone size gradient.

4. Automatic Detection of an Equilibrium State

As discussed in [Reaching Equilibrium](#), we recommend you use the `convergence 1.0` limit criteria when using the `model solve` command. However, you should always take displacement and velocity histories of the areas of most interest, and determine if sufficient equilibrium will be reached sooner, or if further cycling is still necessary. Initially solving to a `convergence` of 100 or more and then examining the response is good practice. Making isosurface plots of convergence to see where the model is having trouble can give insight into where model adjustments in the interest of efficiency are best placed.

The default ratio limit is also used to detect the steady-state solution for thermal and fluid-flow calculations. For thermal calculations, unbalanced and applied heat flux magnitudes, rather than unbalanced and applied mechanical forces, are evaluated. For fluid-flow calculations, the unbalanced and applied fluid flow magnitudes are evaluated.

5. Considerations for Selecting Damping

The default mechanical damping for static analysis is *local damping* (see the topic [Mechanical Damping](#)), which is most efficient for removing kinetic energy when the velocity components of most gridpoints pass through zero periodically. This is because the mass-adjustment process depends on velocity sign changes. Local damping is a very efficient algorithm for achieving a static-equilibrium solution without introducing erroneous damping forces (see Cundall 1987).

If the problem involves significant regions of the grid having nonzero components of velocity at the final state of solution, then the default damping may be insufficient to reach an equilibrium state. A different form of damping, known as *combined damping*, provides better convergence to the steady-state than local damping for situations in which significant rigid-body motion of the grid is occurring. This may occur, for example, in a creep simulation or in a very stiff structural reinforcement attached to soft soil moving in response to an excavation. Use the command `zone mechanical damping combined` to invoke combined damping. Combined damping is not as efficient at removing kinetic energy, so be careful to minimize dynamic excitation of the system (see the [gradual excavation of a circular tunnel](#) example). It is possible to switch back to the default damping with the command `zone mechanical damping local`.

6. Check Model Response

FLAC3D shows how a similar physical system would behave. Make frequent simple tests to verify that you are actually doing what you think you are doing. For example, if a loading condition and geometry are symmetrical, make sure that the response is symmetrical. After making a change in the model, execute a few calculation steps (say, 50 or 100) to verify that the initial response is of the correct sign and in the correct location. Do back-of-the-envelope estimates of the expected order of magnitude of stress or displacements, and compare to the *FLAC3D* output.

If you apply a violent shock to the model, you will get a violent response. If you do nonphysically reasonable things to the model, you must expect strange results. If you get unexpected results at a given stage of an analysis, review the steps you followed up to that stage.

Critically examine the output before proceeding with the model simulation. If, for example, everything appears reasonable except for large velocities in one corner zone, do not go on until you understand the reason. In this case, you may not have fixed a boundary gridpoint properly.

7. Initializing Variables

It is common practice to initialize the displacements of the gridpoints between runs to aid in the interpretation of a simulation in which many different excavation stages are performed. This can be done because the code does not require the displacements in the calculation sequence — they are determined from the velocities of the gridpoints as a convenience to the user.

Initialization of the velocities, however, is a different matter. If the velocities of gridpoints are fixed at a constant value, they will continue to have this value until set otherwise. Therefore, do not initialize the velocities of the grid to zero simply to clear them—this will affect the simulation results. However, sometimes it is useful to set velocities to zero (for example, to remove all kinetic energy).

8. Minimizing Transient Effects on Static Analysis

For static analyses, transient waves can cause plastic models to follow poorly defined stress paths (unrealistic failure could be triggered). For this reason, it is often important to approach the solution (at each stage) gradually (i.e., make the solution more “static” by minimizing the effects of transient waves when problem conditions are changed suddenly.[1]

The recommended method to make a *FLAC3D* solution more static is to use `zone relax excavate` to excavate regions, and the `servo` option on `zone face apply` when applying or reducing loads. When a sudden change is made (e.g., by nulling zones to simulate excavation), it is *not* advisable to set the strength properties to high values and step to equilibrium (and then reset the properties to realistic values). This latter method does not clearly define a direction in stress space to bring a zone within the yield surface (the entire process is nonphysical).

9. Changing Material Models

FLAC3D does not have a limit on the number of different material models you may use during a simulation. The code has been dimensioned to allow the user to have a different material for each zone (if desired) for the maximum size grid for your version of *FLAC3D*.

10. Running Problems with In-Situ Field Stresses and Gravity

There are a number of problems in which in-situ field stresses and gravity must be applied to the model. An example of such a problem is deep cut-and-fill mining, in which the rock mass is subjected to high in-situ stress fields (i.e., gravity stresses for the limited mesh size can be ignored), but in which the emplaced backfill pillars develop gravitational stresses that could collapse under the load. The important thing to note in these simulations (as in any simulation in which gravity is applied) is that at least three points on the grid must be fixed in space; otherwise, the entire grid will translate due to gravity. If you ever notice the entire grid translating in the direction of the gravitational acceleration vector, you have probably forgotten to fix the grid in space (e.g., see [uplift when material is removed](#)).

11. Determining Collapse Loads

In order to determine a collapse load, it is often better to use “strain-controlled” rather than “stress-controlled” boundary conditions (i.e., apply a constant velocity and measure the reaction forces, rather than apply forces and measure displacements). A system that collapses becomes difficult to control as the applied load approaches the collapse load. This is true of a real system as well as a model system.

12. Determining Factor of Safety

Factor of safety can be determined in *FLAC3D* for any selected parameter by taking the ratio of the calculated value under given conditions to the value that results in failure. For example:

$$F_L = \frac{\text{applied load to cause failure}}{\text{design load}}$$

$$F_{\phi} = \frac{\tan(\text{actual friction angle})}{\tan(\text{friction angle at failure})}$$

Note that the larger value is always divided by the smaller value (assuming that the system does not fail under the actual conditions). *The definition of failure must be established by the user.* A comparison of this approach, which is based on strength reduction for determining factor of safety, to that based upon limit analysis solutions is given by Dawson and Roth (1999) and Dawson et al. (1999). Also see the example application [Influence of Slope Curvature on Stability](#).

The strength reduction method for determining factor of safety is implemented in *FLAC3D* through the `model factor-of-safety` command. This command implements an automatic search for factor of safety using the bracketing approach, as described in Dawson et al. (1999). The `model factor-of-safety` command may be given at any stage in a *FLAC3D* run, provided that every non-null zone contains a constitutive model that supports property scaling. During the solution process, properties will be changed, but the original state will be restored on completion, or if the `factor-of-safety` search is terminated prematurely with the `Esc` key.

When *FLAC3D* is executing the `model factor-of-safety` command, the bracketing values for F are printed continuously to the screen so that you can tell how the solution is progressing. If terminated by the `Esc` key, the solution process terminates, and the best current estimate of `fos` is displayed. A save file that corresponds to the last nonequilibrium state is produced so that velocity vectors, and so on, can be plotted. This allows a visualization of the failure mode.

The example below, based upon an example stability analysis from Dawson et al. (1999), illustrates how the approach works. The corresponding project file is “FactorOfSafety.f3prj.” The run stops at $F = 1.06$. The figure that follows plots shear strain-rate contours and velocity vectors, which allow the failure surface to be identified.

Factor of safety calculation for slope stability analysis

```

model new
; Create zones
zone create brick point 0 (0, 0,0) point 1 (2,0,0) ...
                point 2 (0,0.5,0) point 3 (0,0,2) ...
                size 3 1 3
zone create brick point 0 (2, 0,0) point 1 (13.4,0,0) ...
                point 2 (2,0.5,0) point 3 ( 2,0,2) ...
                size 8 1 3

```

```

zone create brick point 0 (13.4, 0,0) point 1 (20 ,0,0) ...
                    point 2 (13.4,0.5,0) point 3 (13.4,0,2) ...
                    size 6 1 3
zone create brick point 0 ( 2 , 0, 2) point 1 (13.4, 0, 2) ...
                    point 2 ( 2 ,0.5, 2) point 3 (12 , 0,12) ...
                    point 4 (13.4 0.5, 2) point 5 (12 ,0.5,12) ...
                    point 6 (16 , 0,12) point 7 (16 ,0.5,12) ...
                    size 8 1 17
zone create brick point 0 (13.4, 0, 2) point 1 (20, 0,2 ) ...
                    point 2 (13.4,0.5, 2) point 3 (16, 0,12) ...
                    point 4 (20 ,0.5, 2) point 5 (16,0.5,12) ...
                    point 6 (20 , 0,12) point 7 (20,0.5,12) ...
                    size 6 1 17

zone face skin
; Constitutive model and properties
zone cmodel assign mohr-coulomb
zone property density 2000.0 bulk 1e8 shear 3e7 cohesion 12380 ...
                    tension 1e10 friction 20 dilation 20
; Boundary conditions
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity (0,0,0) range group 'Bottom'
; Initial conditions
model gravity 10
zone initialize-stresses
; Histories
model history name='conv' mechanical convergence
; Solve
model solve convergence 1
model factor-of-safety file 'slope3dfos' associated convergence 1

```

It is recommended that the `model factor-of-safety` command be given at an equilibrium state of a model (to improve solution time and reduce possible pseudo-inertial effects), but this is not essential.

The procedure used by *FLAC3D* during execution of `model factor-of-safety` is as follows. First, the code finds a “representative number of steps” (denoted by N_r), which characterizes the response time of the system. N_r is found by setting the cohesion to a large value, making a large change to the internal stresses, and finding how many steps are necessary for the system to return to equilibrium. Then, for a given factor of safety, F , N_r steps are executed. If the equilibrium condition is met, then the system is in equilibrium. If the condition is not met, then another N_r steps are executed, exiting the loop if the condition is met. The equilibrium condition value, averaged over the current span of N_r steps, is compared with the mean force ratio over the previous N_r steps. If the difference is less than 10%, the system is deemed to be in nonequilibrium, and the loop is exited with the new nonequilibrium, F . If the above-mentioned difference is

greater than 10%, blocks of N_p steps are continued until: 1) the difference is less than 10%; 2) 6 such blocks have been executed; or 3) the equilibrium condition is met. The justification for the first case is that the mean force ratio is converging to a steady value that is greater than that corresponding to equilibrium; the system must be in continuous motion. The *FISH* intrinsic variable `global.fos` allows access to the current value of F during `model factor-of-safety execution` (see example).

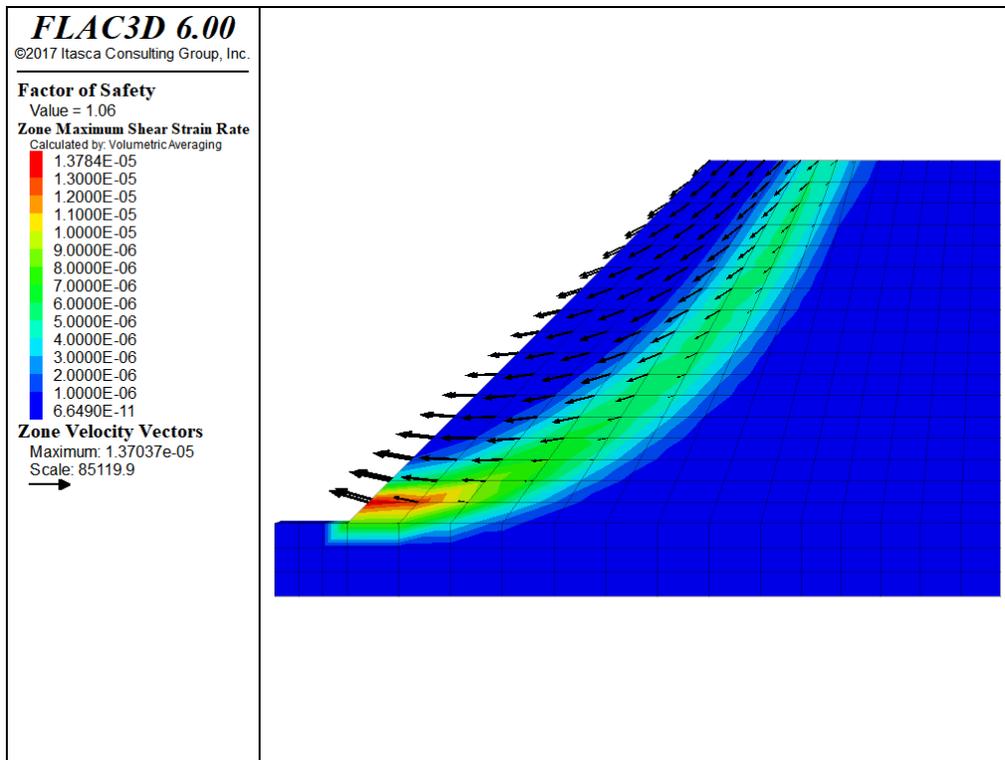


Figure 1: Shear strain-rate contours and velocity vectors in slope model at last nonequilibrium state.

13. Use Bulk and Shear Moduli

It is better to use bulk modulus, K , and shear modulus, G , than Young's modulus, E , and Poisson's ratio, ν , for elastic properties in *FLAC3D*.

The pair (K, G) makes sense for all elastic materials that do not violate thermodynamic principles. The pair (E, ν) does not make sense for certain admissible materials. At one extreme, we have materials that resist volumetric change but not shear; at the other extreme, materials that resist shear but not volumetric change. The first type of material corresponds to finite K and zero G ,

and the second to zero K and finite G . However, the pair (E, ν) is not able to characterize either the first or the second type of material. If we exclude the two limiting cases (conventionally, $\nu = 0.5$ and $\nu = -1$), the equations

$$3K(1 - 2\nu) = E$$

$$2G(1 + \nu) = E$$

relate the two sets of constants. These equations hold, however close we approach (but not reach) the limiting cases. We do not need to relate them to physical tests that may or may not be feasible; the equations are simply the consequence of two possible ways of defining coefficients of proportionality. Suppose we have a material in which the resistance to distortion progressively reduces, but the resistance to volume change remains constant. ν approaches 0.5 in this case. The equation $3K(1 - 2\nu) = E$ must still be satisfied. There are two possibilities (argued on *algebraic*, not physical, grounds): either E remains finite (and nonzero) and K tends to an arbitrarily large value; or K remains finite and E tends to zero. We rule out the first possibility because there is a limiting compressibility to all known materials (e.g., 2 GPa for water, which has a Poisson's ratio of 0.5). This leaves the second, in which E is varying drastically, even though we supposed that the material's principal mode of elastic resistance was unchanging. We deduce that the parameters (E, ν) are inadequate to express the material behavior.

FLAC3D Runtime Benchmark

FLAC3D has been tested on a number of different computers. The calculation rates are compared here for a 624,000-zone model footing in a Mohr-Coulomb material. The model is run for 100 steps, and the rate is calculated by a *FISH* function. The data file is given in the example below; the table that follows summarizes the calculation rates for different computers.

Benchmark Data File — “timing-test.f3dat”

```

model new
; Create zones
zone create cylindrical-shell point 0 (0, 0, 0) point 1 ( 0,15, 0) ...
point 2 (0, 0,15) point 3 (15, 0, 0) ...
point 4 (0,15,15) point 5 (15, 0,15) ...
point 8 (0, 3, 0) point 9 ( 3, 0, 0) ...
point 10 (0, 3,15) point 11 ( 3, 0,15) ...
size 100 80 60 30 ...
ratio 1.05 1.1 1 0.875 fill group 'cyl'
```

```

zone face skin
zone face group 'slab' slot 'app' range group 'Top' group 'cyl'
; Constitutive model and properties
zone cmodel assign mohr-coulomb
zone property bulk 2e8 shear 1e8 cohesion 1e5
zone property friction 20 dilation 20 tension 1e10
; Boundary Conditions
zone face apply velocity-normal 0 range group 'West'
zone face apply velocity-normal 0 range group 'South'
zone face apply velocity-normal 0 range group 'Bottom'
zone face apply velocity-normal 0 range group 'North'
zone face apply velocity-normal -2e-5 range group 'slab'
; Initial conditions
zone initialize stress xx -1e6 yy -1e6 zz -1e6
; Measure rate
[global start = time.clock]
model step 100
[global rate = 10 * zone.num / (time.clock-start) ]
[io.out('Calculation rate = '+string(rate)+' kilo-zones/sec')]

```

Endnote

- [1] This is not always the case. “Path-dependency” of a changing nonlinear system can be important. See the discussion in [Localization, Physical Instability, and Path-Dependence](#).

Interpretation

Since *FLAC3D* models a nonlinear system as it evolves in time, the interpretation of results may be more difficult than with a conventional finite-element program that produces a “solution” at the end of its calculation phase. There are several indicators that can be used to assess the state of the numerical model (e.g., whether the system is stable, unstable, or in steady-state plastic flow). The various indicators are described below.

Unbalanced Force and Convergence

Each gridpoint is connected to zones that contribute forces to the gridpoint. At equilibrium, the algebraic sum of these forces is almost zero (i.e., the forces acting on one side of the gridpoint nearly balance those acting on the other). If the unbalanced forces approach a constant nonzero value, this indicates that failure and plastic flow are occurring within the model.

As discussed in [Reaching Equilibrium](#), there are five methods of determining model convergence. Each of these is based on the out-of-balance force acting on the gridpoints. Each of these can be saved as a history and viewed as a graph. These convergence criteria are important in assessing the state of the model.

A rule of thumb is that an average force ratio of $1e-5$, an local force ratio of $1e-4$, or a convergence of 1.0 all indicate overall convergence. However, values 10 or even 100 times larger may be acceptable depending on the specifics of the model and the degree of precision required (e.g., a much larger value larger may be good enough for an intermediate stage in a sequence).

Note that a low convergence value only indicates that forces balance at all gridpoints. However, steady plastic flow may be occurring without acceleration. In order to distinguish between this condition and “true” equilibrium, other indicators (such as those described below) should be examined.

Gridpoint Velocities

The grid velocities may be assessed either by plotting out the whole field of velocities (plot a “zone-vectors” item and set “Value” to “Velocity”), or by selecting certain key points in the grid and tracking their velocities with histories (e.g., `zone history velocity-x`, or `velocity-y`, `velocity-z`, or `velocity`). Both types of plots are useful. Steady-state conditions are indicated if the velocity histories show horizontal traces in their final stages. If they have all converged to near-zero (in comparison to their starting values), then absolute equilibrium has occurred; if a history has converged to a nonzero value, then steady plastic-flow is occurring at the gridpoint corresponding to that history. If one or more velocity history plots show fluctuating velocities, then the system is likely to be in a transient condition. Note that velocities are expressed in units of displacement divided by number of steps.

The plot of the field of velocity vectors is more difficult to interpret, since both the magnitudes and the nature of the pattern are important. As with gridpoint forces, velocities never decrease precisely to zero. The magnitude of velocity should be viewed in relation to the displacement that would occur if a significant number of steps (e.g., 1000) were to be executed. For example, if current displacements in the system are of the order of 1 cm, and the maximum velocity in the velocity plot is 10^{-8} m/step, then 1000 steps would produce an additional displacement of 10^{-5} m, or 10^{-3} cm, which is 0.1% of the current displacements. In this case, it can be said that the system is in equilibrium even if the velocities all seem to be “flowing” in one direction. More often, the vectors appear to be random (or almost random) in direction and (possibly) in magnitude. A random velocity field of low amplitude is an infallible indicator of equilibrium and no plastic flow.

If the vectors in the velocity field are coherent (i.e., there is some systematic pattern) and their magnitude is quite large (using the criterion described above), then either plastic flow is occurring or the system is still adjusting elastically (e.g., damped elastic oscillation is taking place). To confirm that continuing plastic flow is occurring, a plot of plasticity indicators should be examined, as described below. If, however, the motion involves elastic oscillation, then the magnitude should be observed in order to indicate whether such movement is significant. Seemingly meaningful patterns of oscillation may be seen but, if amplitude is low, then the motion has no physical significance.

Plastic Indicators

For the plasticity models in *FLAC3D*, stresses that satisfy the yield criterion may be shown by plotting a zone plot item, setting “ColorBy” to “Label” and setting “Label” to “State”. Such an indication usually denotes that plastic flow is occurring, but it is possible for an element simply to “sit” on the yield surface without any significant flow taking place. It is important to look at the whole pattern of plasticity indicators to see whether a *mechanism* has developed.

Two types of failure mechanisms are indicated by a standard Mohr–Coulomb plasticity state plot: shear failure and tensile failure. Each type is designated by a different color on the plot.^[1] The plot also indicates whether stresses within a zone are currently on the yield surface (i.e., the zone is at active failure now, -n), or the zone has failed earlier in the model run, but now the stresses fall below the yield surface (the zone has failed in the past, -p). Initial plastic flow can occur at the beginning of a simulation, but subsequent stress redistribution unloads the yielding elements so that their stresses no longer satisfy the yield criterion, indicated by *shear-p* or *tension-p* (on the plasticity state plot).

A failure mechanism is indicated if there is a contiguous line of active plastic zones (indicated by either *shear-n* or *tension-n*) that join two surfaces. The diagnosis is confirmed if the velocity plot also indicates motion corresponding to the same mechanism.

If there is no contiguous line or band of active plastic zones between boundaries, two patterns should be compared before and after the execution of, say, 5000 steps. Is the region of active yield increasing or decreasing? If it is decreasing, then the system is probably heading for equilibrium; if it is increasing, then ultimate failure *may* be possible.

If a condition of continuing plastic flow has been diagnosed, one further question should be asked: Does the active flow band(s) include zones adjacent to artificial boundaries? The term “artificial boundary” refers to a boundary that does not correspond to a physical entity, but one that exists simply to limit the size of the grid that is used (see [Artificial Boundaries](#)). If plastic flow occurs along such a boundary, then the solution is not realistic, because the mechanism of failure is influenced by a nonphysical entity. This comment only applies to the final steady-state solution; intermediate stages may exhibit flow along boundaries.

Histories

In any problem, there are certain variables that are of particular interest (e.g., displacements may be of concern in one problem, but stresses may be of concern in another). Liberal use should be made of the `zone history` command to track these important variables in the regions of interest. After some timestepping has taken place, plots of these histories often provide a way to find out what the system is doing.

Endnote

- [1] For the ubiquitous-joint model, shear failure along the joint plane is designated by **shear** and tensile failure by **tension** on the plasticity plot. Other models may define other failure state indicators.

Project Completion

Upon “completion” of a project, there are several steps that should generally be taken.

Make certain your project file is saved and includes any and all plot views that were used to generate important images.

Then, using the Tools · Bundle · Pack menu option, create a project archive bundle. This creates a single file that includes the project file, all data files, and the state record from all save states. This file *does not* include the save states themselves, as they would in general be quite large.

If your available storage permits, archiving at least one save state representing the final result of your model can save a lot of computation time, if going back to these results are necessary.

Lastly, if you want to be certain to be able to recover the exact model results, archive the *FLAC3D* installation version you used to run the model. Future updates may change results, especially for models that exhibit strong sensitivity to changes in initial conditions.

We strongly recommended that the project bundle and the *exact* version number of *FLAC3D* used to create the model (if not an actual copy of the installation files) be stored along with any report or other final products from your project.

References

- Barton, N. "The Shear Strength of Rock and Rock Joints," *Int. J. Rock Mech. Min. Sci. & Geotech. Abstr.*, **13**, 255-279 (1976).
- Batugin, S. A., and R. K. Nirenburg. "Approximate Relation between the Elastic Constants of Anisotropic Rocks and the Anisotropy Parameters," *Soviet Mining Science*, **8**(1), 5-9 (1972).
- Bieniawski, Z. T. "Determining Rock Mass Deformability: Experience from Case Histories," *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr.*, **15**, 237-247 (1978).
- Brady, B. H. G., and E. T. Brown. *Rock Mechanics for Underground Mining*. London: George Allen & Unwin. (1985).
- Cundall, P. A. "Distinct Element Models of Rock and Soil Structure," in *Analytical and Computational Methods in Engineering Rock Mechanics*, Ch. 4, pp. 129-163. E. T. Brown, ed. London: Allen & Unwin. (1987).
- Cundall, P. A. "Numerical Experiments on Localization in Frictional Material," *Ingenieur-Archiv*, **59**, 148-159 (1989).
- Cundall, P. A. "Numerical Modelling of Jointed and Faulted Rock," in *Mechanics of Jointed and Faulted Rock*, pp. 11-18. Rotterdam: A. A. Balkema(1990).
- Cundall, P. A."Shear Band Initiation and Evolution in Frictional Materials," in *Mechanics Computing in 1990s and Beyond (Proceedings of the Conference, Columbus, Ohio, May 1991)*, Vol. 2: Structural and Material Mechanics, pp. 1279-1289. New York: ASME (1991).
- Das, B. M. *Principles of Geotechnical Engineering*, 3rd Ed. Boston: PWS Publishing Company (1994).
- Dawson, E. M., and W. H. Roth. "Slope Stability Analysis with FLAC," in *FLAC and Numerical Modeling in Geomechanics (Proceedings of the International FLAC Symposium on Numerical Modeling in Geomechanics, Minneapolis, Minnesota, September 1999)*, pp. 3-9. Rotterdam: A.A. Balkema (1999).
- Dawson, E. M., W. H. Roth and A. Drescher. "Slope Stability Analysis with Finite Element and Finite Difference Methods," *Géotechnique* **49**(6), 835-840 (1999).

- Fossum, A. F. "Technical Note: Effective Elastic Properties for a Randomly Jointed Rock Mass," *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr.*, 22(6), 467-470 (1985).
- Gerrard, C. M. "Elastic Models of Rock Masses Having One, Two and Three Sets of Joints," *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr.*, 19, 15-23 (1982a).
- Gerrard, C. M. "Equivalent Elastic Moduli of a Rock Mass Consisting of Orthorhombic Layers," *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr.*, 19, 9-14 (1982a).
- Gleick, J. *Chaos: Making a New Science*. New York: Penguin Books (1987).
- Goodman, R. E. *Introduction to Rock Mechanics*. New York: John Wiley and Sons (1980).
- Hoek, E. "Estimating Mohr-Coulomb Friction and Cohesion Values from the Hoek-Brown Failure Criterion," *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr.*, 27(3), 227-229 (1990).
- Hoek, E., and E. T. Brown. "The Hoek-Brown Failure Criterion — a 1988 Update," in *Rock Engineering for Underground Excavation (Proceedings of 15th Canadian Rock Mechanics Symposium, Toronto, October 1988)*, pp. 31-38. Toronto: University of Toronto (1988).
- Hoek, E., and E. T. Brown. "Practical Estimates of Rock Mass Strength," *Int. J. Rock Mech. Min. Sci.*, 34(8), 1165-1186 (1997).
- Hoek, E., and E. T. Brown. *Underground Excavations in Rock*. London: IMM (1980).
- Jaeger, J. C., and N. G. W. Cook. *Fundamentals of Rock Mechanics*, 2nd Ed. London: Chapman and Hall (1969).
- Kulhawy, F. H. "Stress Deformation Properties of Rock and Rock Discontinuities," *Eng. Geol.*, 9, 327-350 (1975).
- Ortiz, J. M. R., J. Serra and C. Oteo. *Curso Aplicado de Cimentaciones*, 3rd Ed. Madrid: Colegio Oficial de Arquitectos de Madrid (1986).
- Rudnicki, J. W., and J. R. Rice. "Conditions for the Localization of the Deformation in Pressure-Sensitive Dilatant Materials," *J. Mech. Phys. Solids*, 23, 371-394 (1975).

- Serafim, J. L., and J. P. Pereira. "Considerations of the Geomechanical Classification of Bieniawski," in *Proceedings of the International Symposium on Engineering Geology and Underground Construction (Lisbon, 1983)*, Vol. 1, pp. II.33-42. Lisbon: SPG/LNEC (1983).
- Singh, B. "Continuum Characterization of Jointed Rock Masses: Part I — The Constitutive Equations," *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr.*, **10**, 311-335 (1973).
- Starfield, A. M., and P. A. Cundall. "Towards a Methodology for Rock Mechanics Modelling," *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr.*, **25**(3), 99-106 (1988).
- Thompson, J. M. T., and H. B. Stewart. *Nonlinear Dynamics and Chaos*. New York: John Wiley & Sons (1986).
- Vardoulakis, I. "Shear Band Inclination and Shear Modulus of Sand in Biaxial Tests," *Int. J. Numer. Anal. Meth. in Geomech.*, **4**, 103-119 (1980).
- Vermeer, P. A., and R. de Borst. "Non-Associated Plasticity for Soils, Concrete and Rock," *Heron*, **29**(3), 1-64 (1984).
- Wood, D. M. *Soil Behaviour and Critical State Soil Mechanics*. Cambridge: Cambridge University Press (1990).