FLAC3D Gitbook

Kyeong Sun Kim

Civil and Environmental Engineering

Seoul National University

Sept, 2021

Contents

There is grandeur in this view of life, with its several powers, having been originally breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.

Charles Darwin(1809-1882)

1 KAIST Model

1.1 Initial Configuration

```
import itasca as it
import numpy as np
np.set_printoptions(threshold=20)
it.command("python-reset-state false")
from itasca import zonearray as za
from itasca import gridpointarray as gpa

# PARAMETERS #

_D_shaft = 1
_H_shaft = 7.45
_T_plate = 1.5
_B_footing = 3.25
```

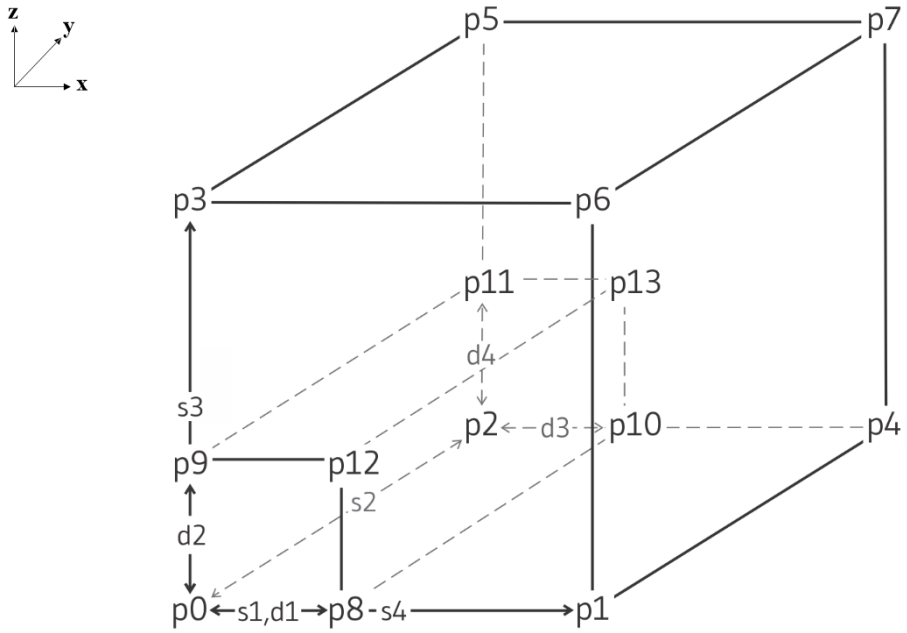
```
_D_footing = _H_shaft + _T_plate
_B_soil = _B_footing*5
_D_soil = _D_footing+3

_radial = 50
_perimeter = 2*_radial
_axial = 2*_radial
_outer = 2*_radial

_bulk = 13.9e9
_shear = 10.4e9

_E_o = 1e7
_const = 1e8
_poisson = 0.25
```

1.2 Zones

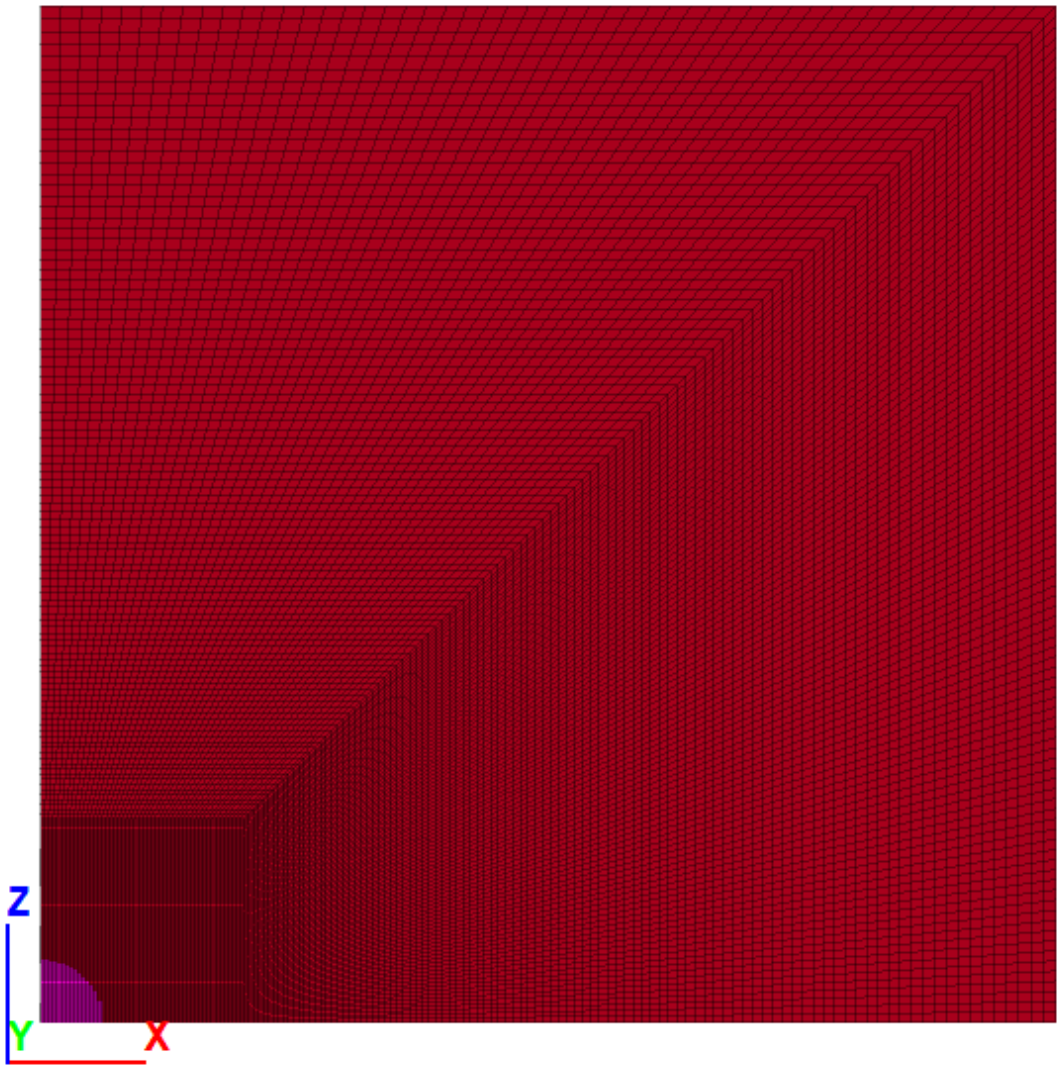


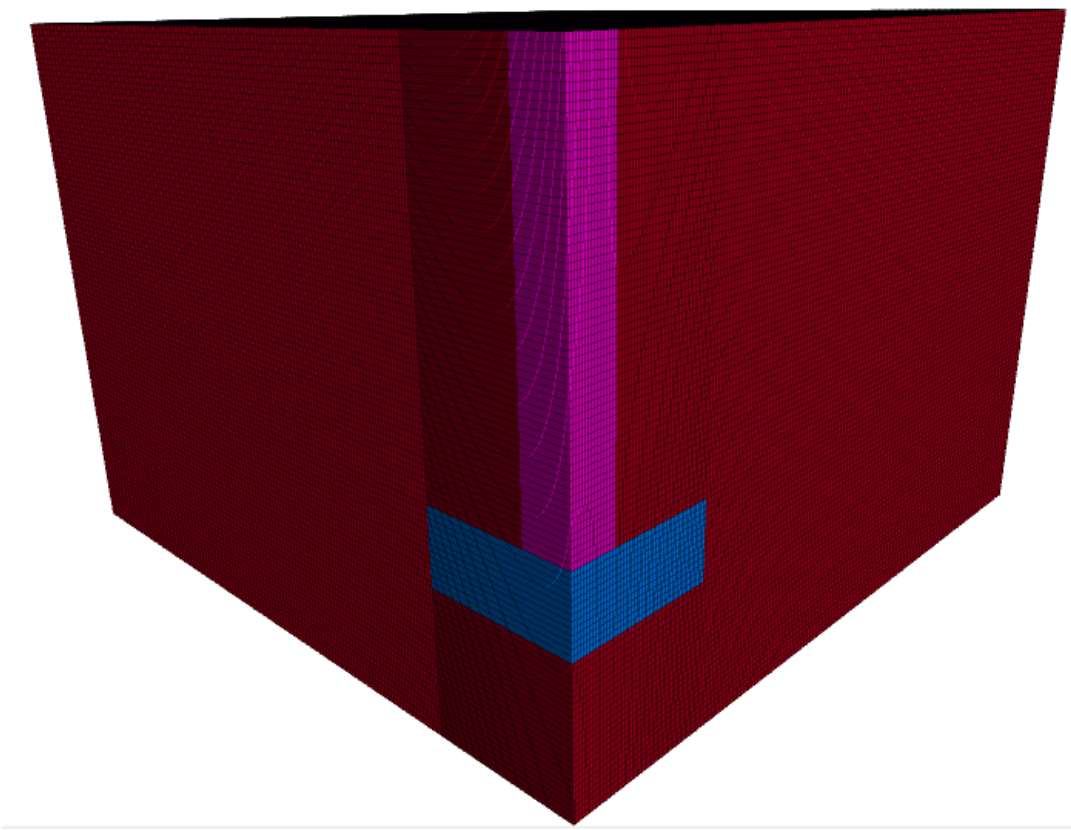
1. KAIST Model

```
# ZONE #
command_zone = """
model new
z crea r-t p 0 (0,0,0) ...
           p 1 ({B_soil},0,0) ...
           p 2 (0,{D_soil},0) ...
           p 3 (0,0,{B_soil}) ...
           p 4 ({B_soil},{D_soil},0) ...
           p 5 (0,{D_soil},{B_soil}) ...
           p 6 ({B_soil},0,{B_soil}) ...
           p 7 ({B_soil},{D_soil},{B_soil}) ...
           p 8 ({B_footing},0,0) ...
           p 9 (0,0,{B_footing}) ...
           p 10 ({B_footing},{D_soil},0) ...
           p 11 (0,{D_soil},{B_footing}) ...
           p 12 ({B_footing},0,{B_footing}) ...
           p 13 ({B_footing},{D_soil},{B_footing}) ...
           size {radial} {axial} {perimeter} {outer} ...
           rat 1 1 1 1.01 ...
           fill
"""

command = command_zone.format(
B_footing = _B_footing,
B_soil = _B_soil,
D_soil = _D_soil,
radial=_radial,
axial = _axial,
perimeter = _perimeter,
outer = _outer)
```

```
it.command(command)
```





1.3 Group

```
# GROUP #
p = za.pos()
x,y,z = p.T
print(it.zone.count(), "zones in whole model")

shaft = reduce(np.logical_and, (np.sqrt(x**2+z**2)<_D_shaft, y<_H_shaft))
za.set_group(shaft, "shaft") # set the zones with shaft = true have "shaft" and "
print(za.in_group("shaft").sum(), "zones in shaft group.") #output how many zones

plate = reduce(np.logical_and, (x<_B_footing,z<_B_footing, y>_H_shaft,y<_D_footing)
za.set_group(plate, "plate") # set the zones with plate = true have "plate" and "
print(za.in_group("plate").sum(), "zones in plate group.") #output how many zones
```

1.4 Constitutive Model

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 are constants. We write a FISH function to install appropriate values of bulk and shear modulus in the grid, as in this example:

```
# CONSTITUTIVE MODEL
it.command("""
zone cmodel assign elastic range group "Radial Tunnel1"
zone cmodel assign elastic range group 'shaft'
zone cmodel assign elastic range group 'plate'

fish define fname(E_o,const)
loop foreach pnt zone.list
z_depth = zone.pos.y(pnt)
E = E_o+const*math.sqrt(z_depth)
zone.prop(pnt,'young')=E
end_loop
end
@fname({E_o_},{const_})
```

```
zone property poisson {poisson_}
plot item create zone contour property name 'young'
""".format(E_o_=_E_o,const_=_const,poisson_=_poisson))

it.command("""
zone property bulk {bulk}  shear {shear} range group 'shaft'
zone property bulk {bulk}  shear {shear} range group 'plate'
""".format(bulk=_bulk,shear=_shear))
```

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 intrinsics. (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.

```
# SOFTENING MODEL #
#it.command("""
#zone cmodel assign strain-softening range group "Radial Tunnel1"
#zone property density 2500 bulk 2e8 shear 1e8 range group "Radial Tunnel1"
#zone property cohesion 2e6 friction 45 tension 2e5 dilation 10 range group "Radi
#zone property table-friction 'fri' table-cohesion 'coh' table-dilation 'dil' ran
#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)
#""")
```

1. KAIST Model

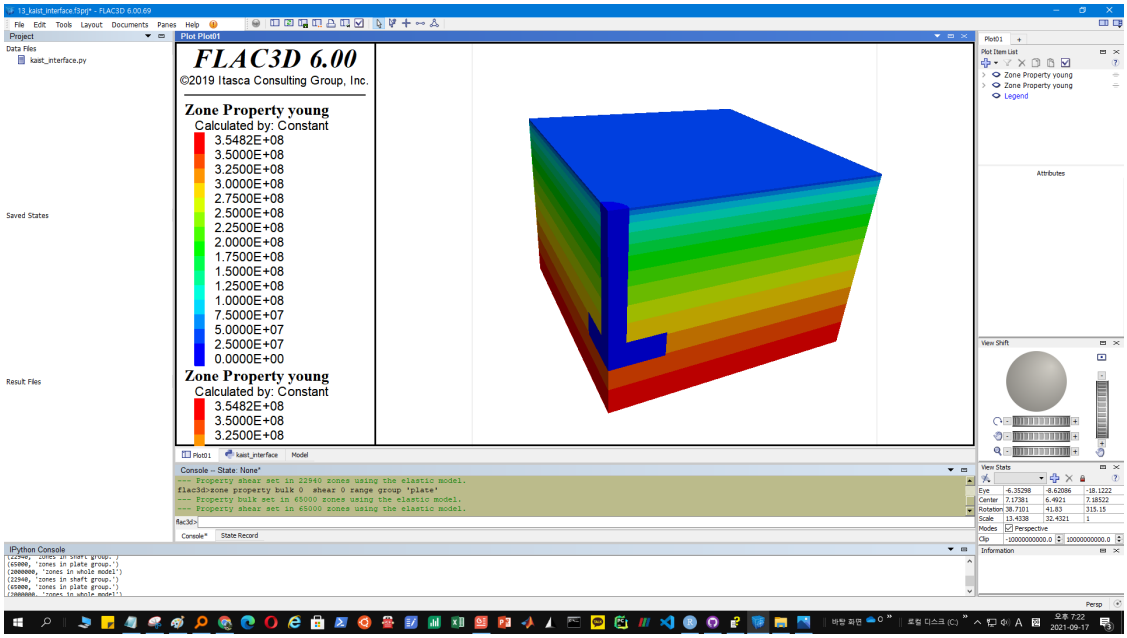


Table 1: Material Properties for a Concrete Pile Foundation in Clay

	Concrete Pile	Clay
Dry density	2500 kg/m^3	1230 kg/m^3
Wet density	–	1550 kg/m^3
Elastic Properties:		
Young's modulus	25.0 GPa	100.0 MPa
Poisson's ratio	0.20	0.30
Bulk modulus	13.9 GPa	83.33 MPa
Shear modulus	10.4 GPa	38.46 MPa
Strength Properties:		
Cohesion	–	30 kPa
Friction angle	–	0.0

1.5 Soil-Structure Interface

```
# INTERFACE
```

1.6 Boundary Conditions

1.7 Initial Equilibrium

1.8 Alterations

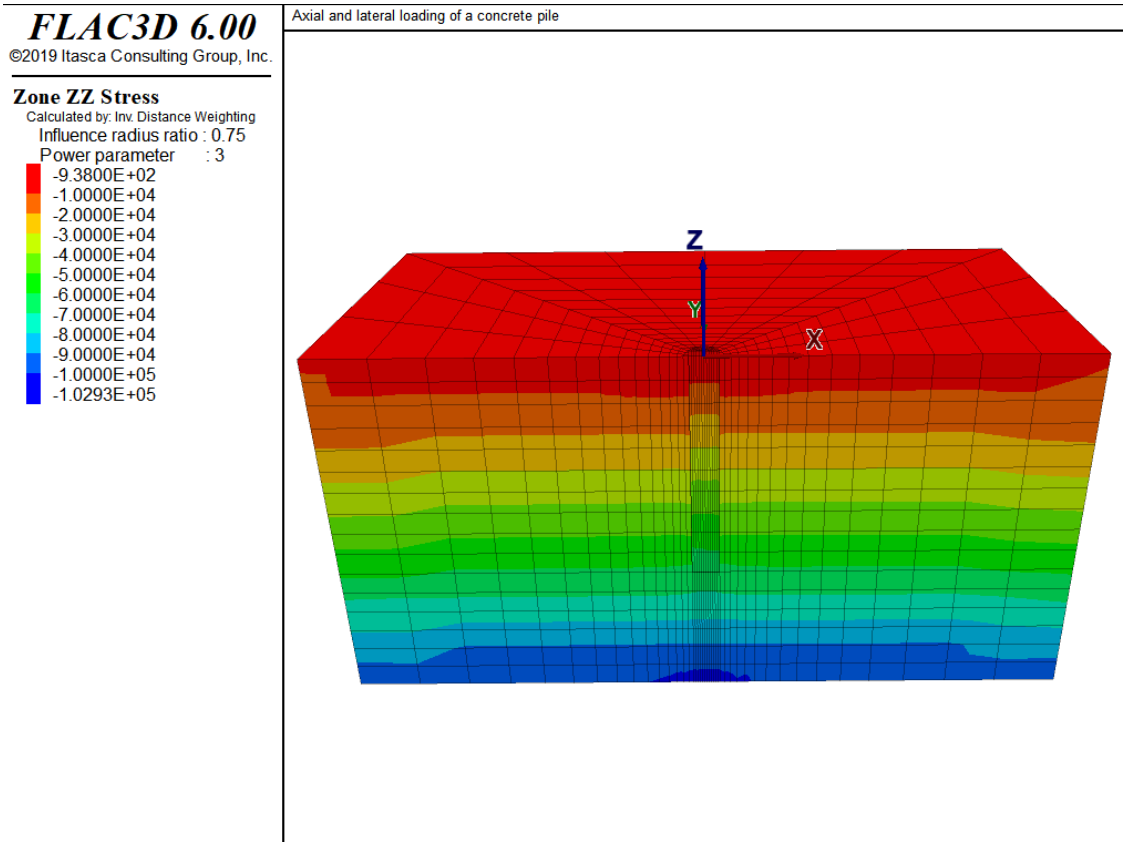
1.9 Results

My modelling choices
Are founded in logic.
Can I convince you?
If you look down,
Are you surprised to see
Only your feet?
The train keeps to the tracks
But the country around
Remains unexplored.
The excellent map
misses no detail, but it is
as large as the world.

Muir Wood(2004)

2

Axial Concrete Pile



2.1 Problem Description

2.1.1 Problem Statement

The pile is subjected to an axial load of 100 kN, and then the top of the pile is moved horizontally for a displacement of 4 cm. The goal is to determine relation of axial loading to the ultimate bearing capacity. And, lateral load-deflection curve is calculated.

1) origin at the top of the pile, z upward.
2) z=0: free surface
3) z=-8: fixed in z-eirection
4) x=+8, -8, y = 8: roller
5) skin friction is modeled by placing an interface between pile concrete wall and clay. In it, fric angle of 20 and c=30kPa are assumed.
6) toe interface is placed between pile tip and clay *note: Zone faces are separated in a previous command so that the gridpoints common to both will be separated as well.*
   note: include Figure of grid (geometry)

2.1.2 Main Parameters

Diameter = 0.6 m

Length = 5

Clay

GWT = 5.5m

2.2 Modeling Procedure

1) equil. stress state under gravity load before install.
   1-1) water table is created at z=5.5

1-2) wet density of clay is assigned below this water table.

2) equil. stress state after installation.

2-1) change properties of pile zones from those

representing clay to those representing concrete.

2-2) vertical equil. stress distribution at this equil. state is shown in

note: include Figure of contours of vertical stress at ini state incld. pile weight

3) apply vertical velocity at top of pile

"ramp" = boundary condition is increased linearly

note: critical timestep is controlled by high stiffness of concrete

If velocity is sudden, inertial effects will dominate and renders difficulty to identification of steady state response of system

table "ramp" is used to apply velocity to pile top gridpoints.

note: FISH FUNCTION vert_load calculates axial stress at the top of pile and stores value as a history

For efficiency, gridpoints on cap surface are stored in symbol "cap" as a map

note: include plot of axial stress vs axial displ. at pile toe. ramp = (0,5e-8), step number = 30000

note: combined damping is used to remove kinetic energy for prescribed loading condition. This is because mass-adjustment process depends on velocity sign-changes..

note: FISH FUNCTION tot_reac monitors soil reaction along pile as a func of lateral displ. tot_reac creates tables of soil reaction (p) vs. lateral displ (y) at diff. locations along pile to generate p-y curve.

note: include Figure of p-y curve at 11 equidistant points along pile

2.3 Zones

```
model new
model title 'Axial and lateral loading of a concrete pile'
; create grid interactively from the extruder tool,
; exported to geometry.f3dat from State Record pane.
call 'geometry' suppress
zone generate from-extruder
; Reflect the grid to get a 1/2 space instead of a 1/4 space
zone reflect dip-direction 270 dip 90
```

2.4 Groups

```
; Name intersections of things named in the two extruder views
zone group 'clay' range group 'clay-c' or 'clay-s' or 'wetclay-s'
zone group 'pile' range group 'pile-c' group 'pile-s' or 'remove-s'
zone group 'remove' range group 'remove-s' group 'pile-c' not ;
zone face group 'wall' internal range group 'wall-c' group 'pile'
zone face group 'base' internal range group 'base-s' group 'pile'
zone face skin ; Name far field boundaries
; Delete the area marked for removal
zone delete range group 'remove'
;
; setup interfaces
; separate using zone separate
; all at once so common nodes are separated
zone separate by-face new-side group 'iwall' slot 'int' ...
     range group 'wall' or 'base'
; Want two different interfaces for proper normal direction at corner
```

```
zone interface 'side' create by-face range group 'wall' and 'iwall'
zone interface 'base' create by-face range group 'base' and 'iwall'
; Save initial geometric state
model save 'geometry'
```

2.5 Properties

```
; Initialize gravity, pore-pressures, density, and stres state
model gravity 10
; water table information
zone water density 1000
zone water plane origin (0,0,-5.5) normal (0,0,-1)
zone initialize density 1230
zone initialize density 1550 range group 'wetclay-s' ; Wet density
; assign properties to the soil and interfaces - temporarily remove pile cap
zone cmodel assign mohr-coulomb ...
     range group 'clay'
zone property bulk 8.333e7  shear 3.846e7  cohesion 30000 fric 0 ...
     range group 'clay'
zone cmodel assign elastic                 range group 'pile'
zone property bulk 8.333e7  shear 3.846e7  range group 'pile'
zone cmodel assign null                    range group 'remove-s'
zone interface 'side' node property stiffness-normal 1e8 ...
                           stiffness-shear 1e8 friction 20 cohesion 30000
zone interface 'base' node property stiffness-normal 1e8 ...
                           stiffness-shear 1e8 friction 20 cohesion 30000
```

2.6 B.C. and I.C.

```
; boundary and initial stress 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 initialize-stress ratio 0.4286
zone interface 'side' node initialize-stresses
zone interface 'base' node initialize-stresses
```

2.7 Initial Equilibrium

```
; Solve to initial equilibrium
zone ratio local
model solve ratio 1e-4
model save 'initial'
```

2.8 Alterations

2.8.1 install the pile

```
; install the pile
model restore 'initial'
zone cmodel assign elastic                          range group 'pile'
zone property bulk 13.9e9 shear 10.4e9 density 2500 range group 'pile'
model solve ratio 1e-4
model save 'install'
```

2.8.2 vertical loading

```
; vertical loading
zone initialize state 0
zone gridpoint initialize displacement (0,0,0)
zone gridpoint initialize velocity     (0,0,0)
table 'ramp' add ([global.step],0) ([global.step+30000],-5e-8) ...
      ([global.step+58000],-5e-8) ; Increase velocity applied to pile
                                  ; over 30,000 steps
zone face apply velocity-normal 1 table 'ramp' range group 'Top'
history interval 250
zone history name 'disp' displacement-z position (0,0,0)
call 'load'
fish history name 'load' @vert_load
zone mechanical damping combined
model step 58000
model save 'vertical-loading'
```

2.8.3 vertical then lateral loading

```
; vertical loading then lateral loading
model restore 'install'
zone initialize state 0
zone gridpoint initialize displacement (0,0,0)
zone gridpoint initialize velocity     (0,0,0)
zone face apply stress-zz [-1.0e5/(math.pi*0.3*0.3)] range group 'Top'
model solve ratio 1e-4
model save 'lateral-load-start'

; apply lateral loading as x-velocity on cap
```

```
zone initialize state 0
zone gridpoint initialize displacement (0,0,0)
zone gridpoint initialize velocity     (0,0,0)
zone face apply velocity-x 1e-7 range group 'Top'
zone history name 'disp' displacement-x position 0,0,0
call 'p-y' suppress ; Calculates p-y curve for pile, when tot_reac is called
@make_pydata        ; Generate p-y curve calculation data
@output_structure   ; Sanity check of p-y curve data
fish history name 'load' @tot_reac
model step 416500
model save 'lateral-load'
```

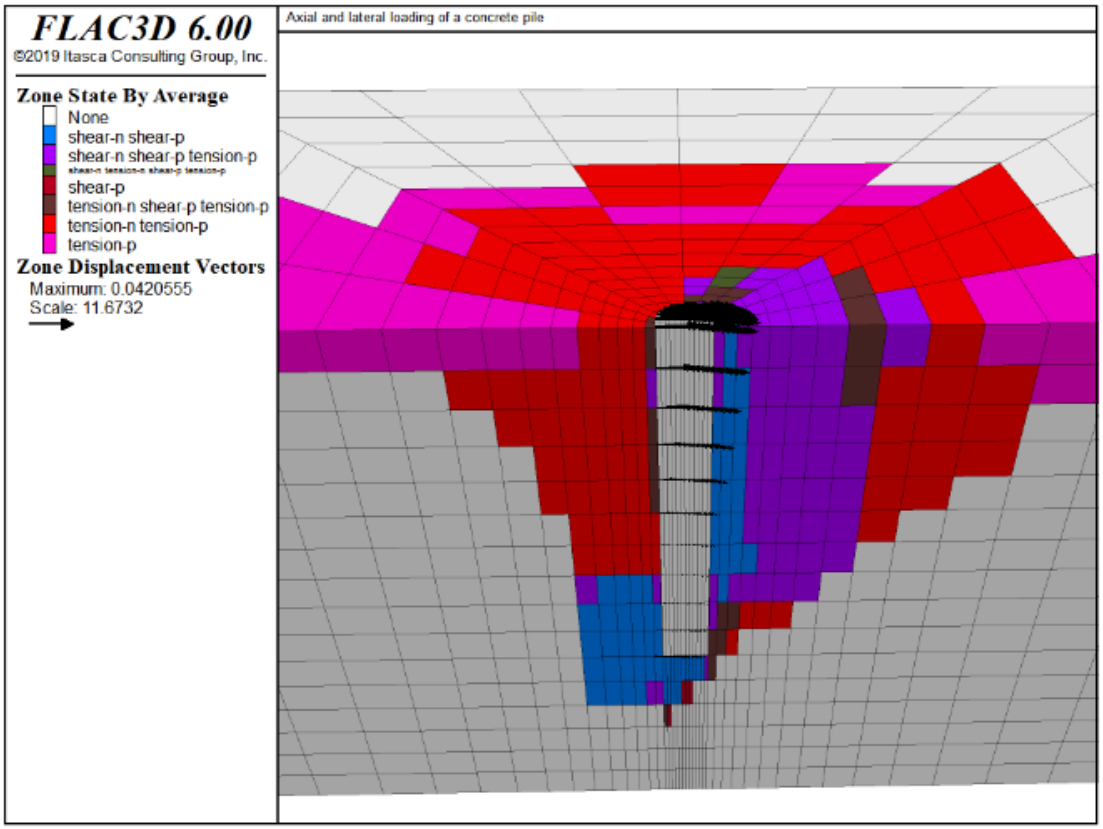


Figure 5: Pile displacement and plasticity state of soil at 4 cm lateral deflection.

In spite of their obvious deficiencies and unreliability, pile driving formulas still enjoy great popularity among practicing engineers, because the use of these formulas reduces the design of pile foundations to a very simple procedure. The price one pays for this artificial simplification is very high.

Karl Terzaghi(1943)

3

Pull-Tests

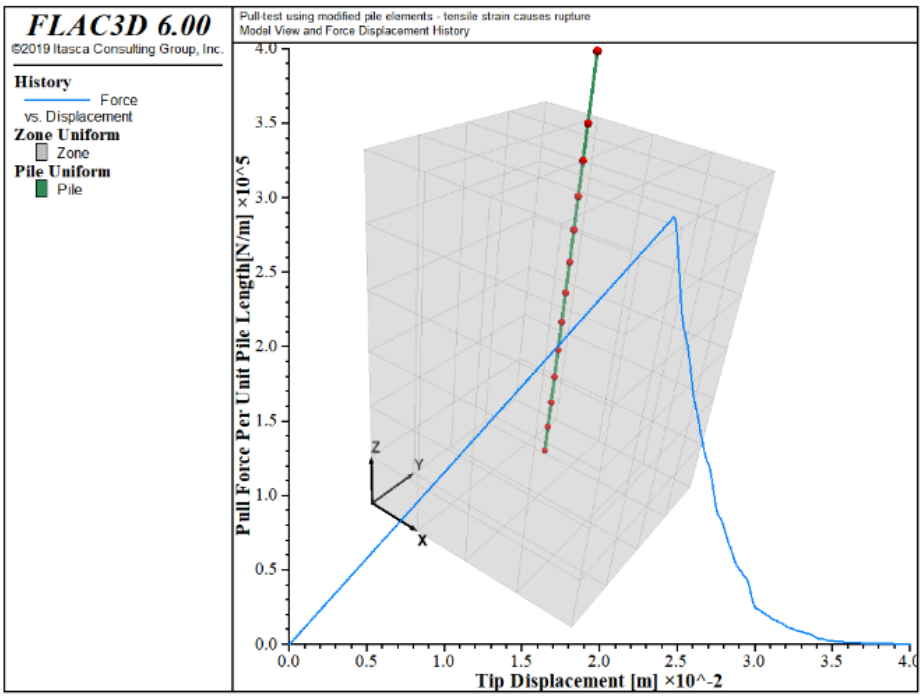


Figure 14: Rockbolt pull force in N/m versus rockbolt axial displacement in meters for the case of a single 25 mm grouted bolt—with tensile rupture.

Problem Description *note: FISH function force is used to sum the raction forces and monitor nodal displacement generated by the pull-test*

note: free length of bolt that extends out of block + larger diameter

Perfectly plastic behavior of grout = max cohesion is exceeded +post-peak weakening of shear bond strength

note: bond strength softening of the grout is defined with keyword coupling-cohesion-table (see Rockbolt Behavior)

The relation btw shear disp. and cohesion weakening is prescribed

thru table cct. softening of friction of grout canbe defined using keyword coupling-friction-table.

3.1 Zones

```
; ================================================================
;   Simulation of pull-test for grouted reinforcement
;   using modified pile elements - Softening of cohesion
; ================================================================
model new
fish automatic-create off
model title 'Pull-test using modified pile elements - cohesion softening'
; Create a single rock block and set its material properties.
zone create brick size 4 4 6 point 1 (0.4,0,0) point 2 (0,0.4,0) ...
                             point 3 (0,0,0.6)
```

3.2 Properties

```
zone cmodel assign elastic
zone property bulk 5e9 shear 3e9
zone face apply velocity-normal 0.0 range position-z 0.6
; Create a pile element and assign properties
```

```
struct pile create by-line (0.2,0.2,0.1) (0.2,0.2,0.7) segments 12
struct pile property rockbolt-flag on
struct pile property young 200e9 poisson 0.25 cross-sectional-area 5e-4 ...
                     perimeter 0.08
struct pile property tensile-yield   2.25e5  ; ultimate tensile strength
struct pile property moi-y 2.0e-8 moi-z 2.0e-8 moi-polar 4.0e-8 ; 0.25*pi*r^4
struct pile property coupling-cohesion-shear 1.75e5 ...
                     coupling-stiffness-shear 1.12e7
struct pile property coupling-cohesion-normal 1.75e5 ...
                     coupling-stiffness-normal 1.12e7
struct pile property coupling-cohesion-table 'cct'
; change in cohesion with relative shear displacement
table 'cct' add (0,1.75e5) (0.025,1.75e4)
```

3.3 Initial Equilibrium

```
struct node fix velocity-x range position-z 0.7
struct node initialize velocity-x 1e-6 local range position-z 0.7
call 'pileforce' suppress ; FISH function calculates reaction force on zones
```

3.4 Alterations

```
; Set up histories for monitoring model behavior
history interval 10
fish history name 'force' @force
struct node history name 'disp' displacement-z position (0.2,0.2,0.7)
; Achieve a total displacement of 4.0 cm
model cycle 40000
```

```
;
model save 'pull-5'
```

3.5 Some other notes

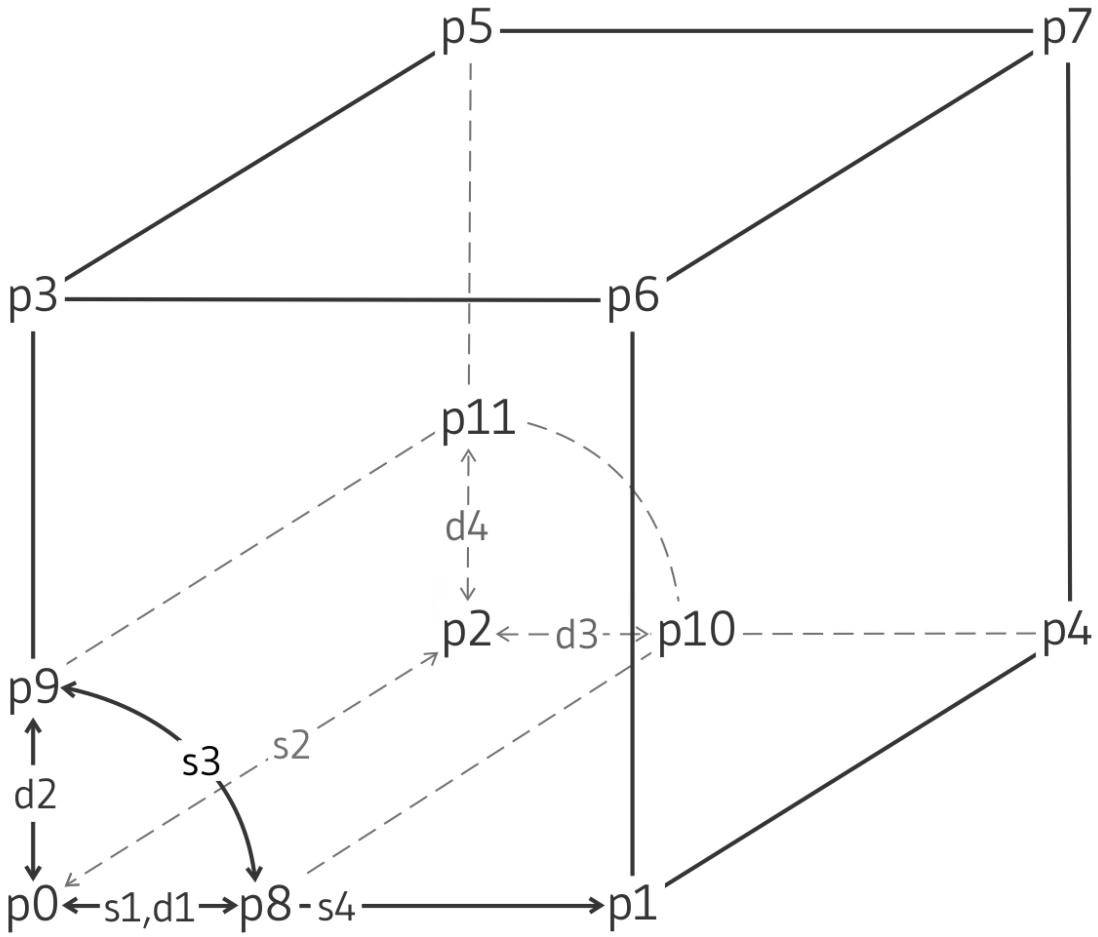
2.3. pull test with confinement “Pulltest06.f3dat” +modified pile logic.(see Behavior of Shear Coupling Springs) linear law is implemented.whereby reinforcement shear strength is defined as constant
*(coupling-cohesion-shear)+ effective pressure*perimeter*fric angle(coupling-friction-shear)*
This pressure dependence is activated automatically by issuing reinforcement properties(perimeter) and

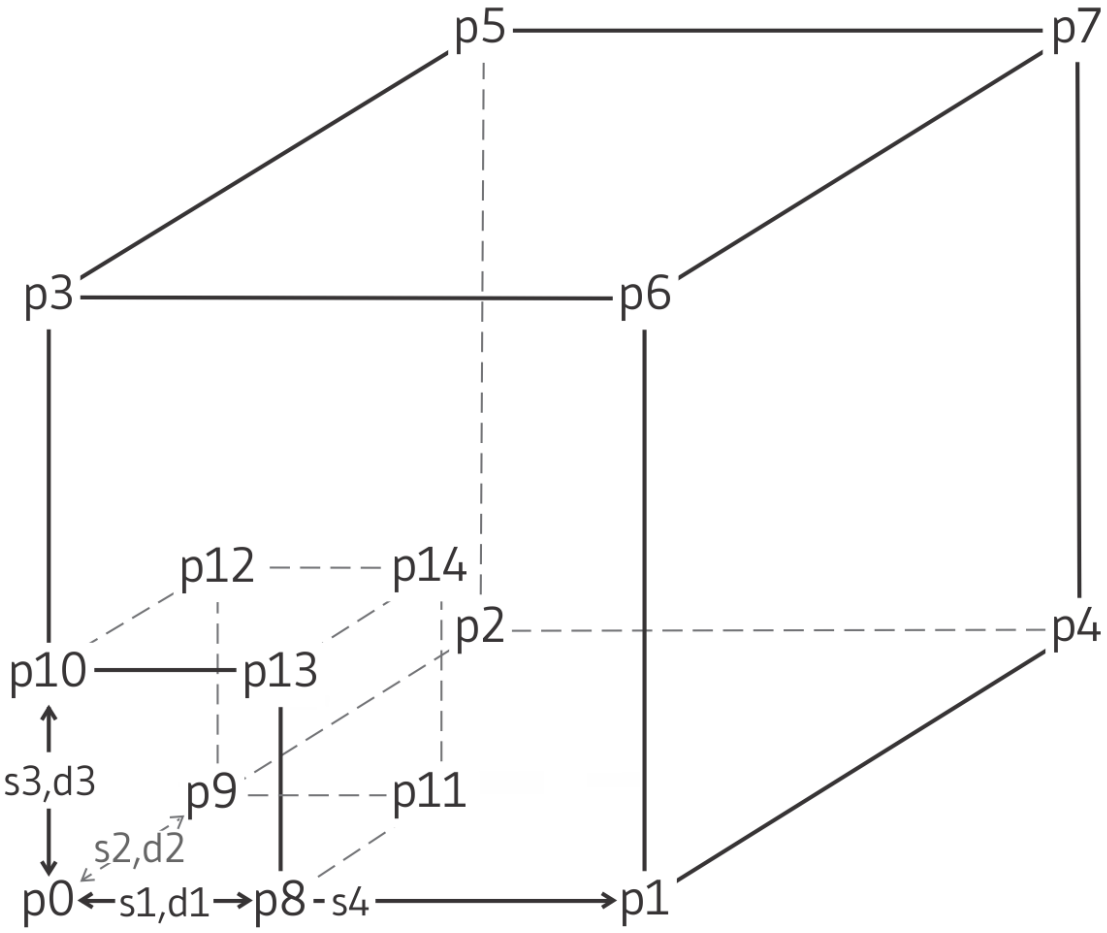
2.5. pull test with tensile rupture “Pulltest08.f3dat” *note: tensile-yield, tensile-failure-strain: for limiting axial yield force and limiting axial strain for rockbolt*

4

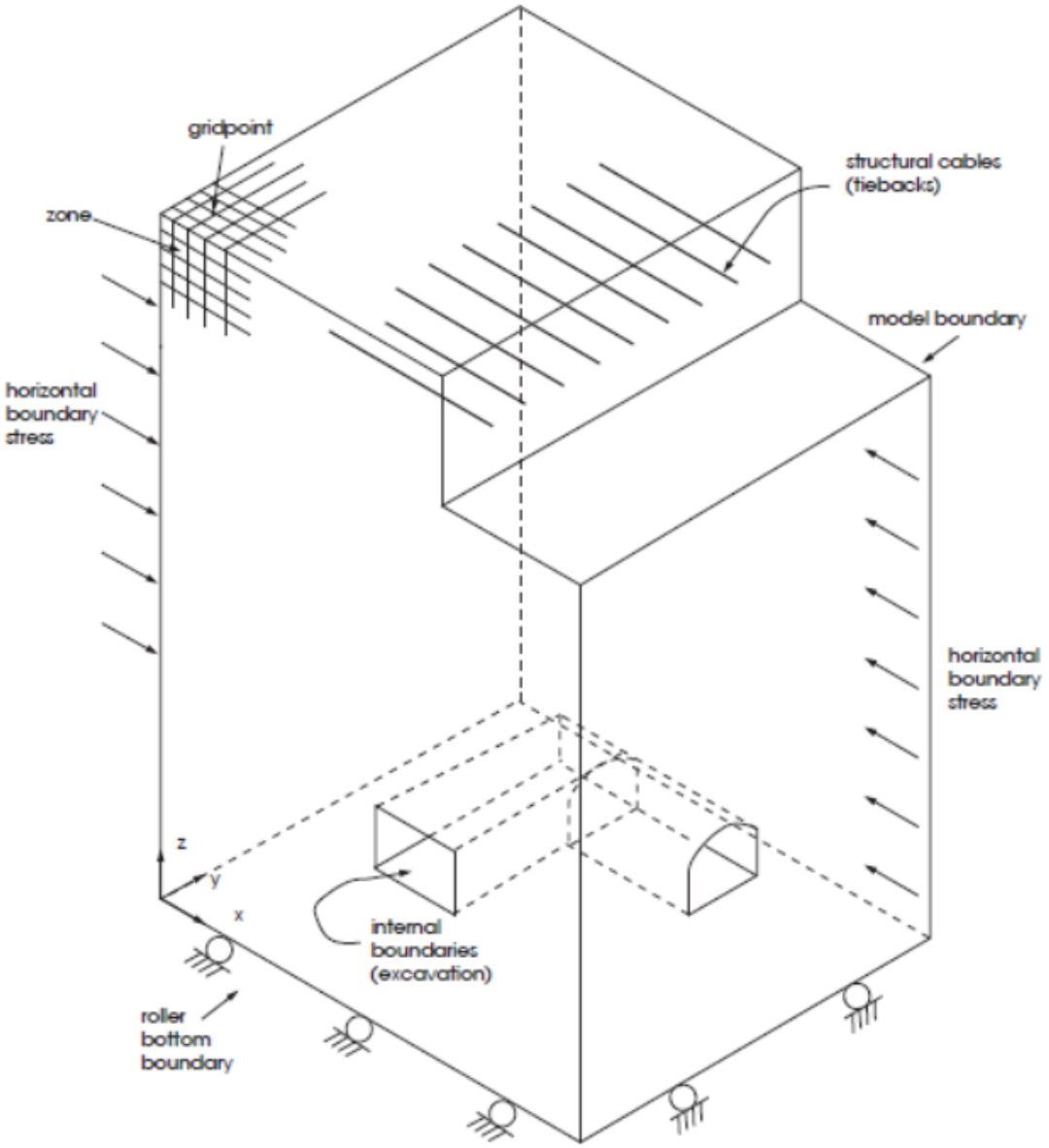
Grid

4.1 Primitive Shapes




p5
p7
p3
p6
p12
p14
p2
p4
p10
p13
s3,d3
p9
p11
s2,d2
p0
s1,d1
p8
s4
p1

4. Grid



note: zone create generates primitive grid

note: zone gridpoint create puts gridpoints at specific locations

note: zone gridpoint merge ensures separate primitives are connected properly

note: zone attach connects primitive meshes of different zone sizes.

```
zone create radial-cylinder size 5 10 6 12 fill
zone create radial-cylinder size 5 10 6 12 ratio 1 1 1 5
each size is controlled by a ratio (geometric ratio of 1.2 times preceding zone)
```

ex) 5 along innder radius of cylindrical tunnel,

10 along axis

6 along circumference of tunnel

12 between periphery of tunnel and outer boundary of model

note: size keyword defines the number of zones in the grid.

keywords for zone create:

- dimension - edge - fill - point (boundary dimensions) - ratio (coarser toward edge) - size

4.2 several primitive shapes connected:

```
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)
; here, model boundary dimensions are 100, 200, 100
; boundary coord are defined using point keyword

zone reflect dip 90 dip-direction 270 origin (0,0,0)
```

this adds symmetric part.

note: 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."

third option, the zone gridpoint create command, is available to position single points in the model region.
note: zone gridpoint create is used for positioning reference points of primitives

During execution of a zone create command, a check is made for each boundary gridpoint against the boundary gridpoints of zones that already exist.
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,
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.

Example: (zone attach) - 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
```

Example: (zone densify)

```
zone create brick size 4 4 4
zone densify segments 2 range position-x 2 4
```

the first two command lines can be changed to 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.

4.3 Structural Element Operation

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.

4.4 Densifying grid by specifying max 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'
```

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)
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.

4.4.1 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'
```

5

Syntax and Grammar

5.1 Introduction

```
import itasca as it
it.command("python-reset-state false")
it.command("""
model new
zone create brick size 10 10 10
zone cmodel assign elastic
zone property density 2950 young 12e9 poisson 0.25
cycle 1
""")
it.zone.count()
z=it.zone.find(1)
print z
z.pos()
```

```
volume_sum = 0.0
for z in it.zone.list():
  volume_sum += z.vol()

print volume_sum
print z.vol() * it.zone.count()
assert volume_sum == z.vol() * it.zone.count()

z = it.zone.near ((5,5,5))
z.pos()
```

5.2 Zones

```
it.zone.count() # 1000
z = it.zone.find(1)
for z in it.zone.list():
  z = it.zone.near((5,5,5))
z.pos()
z.vol()
```

5.3 Properties

```
z.props() or z.props()['bulk']
z.prop('shear')
z.set_prop('bulk', 8.5e9)
```

5.4 Gridpoints

```
gp = it.gridpoint.near((2,2,2))
for gp in it.gridpoint.list():
  total_mass = gp.mass_gravity()
z.vol()*z.density()*1000
```

5.5 Structural Elements

```
it.structure.list()
it.structure.find(1)
it.structure.near((0,2,2))
it.structure.node.find(1)
s_node.links()[0]
```

5.6 Extra Variables

```
z.set_extra(1, 1.23)
z.set_extra(2, "a test string")
z.set_extra(1, gp.pos())
```

5.7 Groups and B.C.

```
if z.group("default") == "lower":
  gp.set_fix(0, True)
  gp.set_fix(1, True)
  gp.set_force_load((1e6,2e6,1e6))
it.zone.near((5,5,5)).stress()
```

```
it.zone.near((5,5,5)).strain()
"""
```

5.8 Parameteric Studies

```
"*note: for modulus in [6e9, 8e9, 10e9, 12e9]:"
it.command("""
model restore 'before_cycling'
zone prop young {}
model solve
""".format(modulus))
vertical_disp = it.gridpoint.near((5,5,10)).disp_z()
print "~~~".format(modulus,vertical_disp)
```

5.9 Setting FISH variables

```
import itasca as it
it.command('python-reset-state false')
it.fish.set('x', 10)
x = it.fish.get('x') yields 10
```

5.9.1 Issuing Command

```
import itasca as it
import numpy as np
data = np.loadtext('brick-data.txt')
command_template = ;;;
zone create brick
zone cmodel assign elastic
```

```
zone property density {density} young {young} poisson {poisson}
;;;
density = data[0]
young = data[1]
poisson = data[2]

command = command_template.format(density=density, young=young, poisson=poisson)
it.command(command)
```

5.10 String

```
"The value of x is {:.2f}".format(0.3872)
"The value of x is {:.2e}".format(0.3872)
"My name is Sasha"
"My name is {}".format("Sasha")
"My name is {name}".format(name="Sasha")
```

5.11 Fish Syntax

5.11.1 Use of ...

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:

```
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
```

5.11.2 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)
```

5.11.3 Traditional for loop in FISH

standard for loop is also available in FISH to provide for additionalloop control.

```
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
```

5.11.3.1 Controlled loop

```
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.

5.11.4 if else endif construct

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 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 sqaure brackets [] of inline FISH, by giving the function name prepended with the 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:

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

5.11.5 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.

5.11.5.1 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.

5.11.5.2 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.

5.11.6 Fish Function

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]
```

6

Code Block

6.1 Some Code

```
import itasca as it
import numpy as np
np.set_printoptions(threshold=20)
it.command("python-reset-state false")
from itasca import zonearray as za
from itasca import gridpointarray as gpa

" GROUPS AND MASK ARRAYS "
it.command("zone group \"lower\" range position-z 0 5")
za.in_group("lower")
za.in_group("lower").sum(), "zones in lower group."
corner_mask = reduce(np.logical_and, (x<3, y<3, z<3))
za.set_group(corner_mask, "corner", "geometry")
```

6. Code Block

```
print za.in_group("corner", "geometry").sum(), "zones in corner group."


" GRIDPOINTS ARRAY FUNCTIONS "
gpos = gpa.pos()
gx, gy, gz = gpos.T
print gz
f = gpa.fixity()
print f
f[:,][gz==0] = True, True, True
print f
gpa.set_fixity(f)


top_gridpoints = gz==10
radial_distance = np.sqrt((gx-5)**2+(gy-5)**2)
central_gridpoints = radial_distance < 5
mask = np.logical_and(top_gridpoints, central_gridpoints)
print "boundary load applied to {} gridpoints".format(mask.sum())
fapp = gpa.force_app()
print fapp
fapp[:,2] = mask*1e6*(5.0-radial_distance)/5.0
gpa.set_force_app(fapp)


print "zone centroids: "
print za.pos()
za.gridpoints()
za.faces()
za.ids()
print za.neighbors()
```

6. Code Block

```
" ========================================== "
" ================RESULTS=================== "
" ========================================== "

it.command("model solve")
print "gridpoint displacements:"
print gpa.disp()
print "gridpoint displacement magnitudes: "
mag = np.linalg.norm(gpa.disp(), axis=1)
print mag
max_index = np.argmax(mag)
print "Maximum displacement: {} at location {}".format(gpa.disp()[max_index],
                                                       gpa.pos()[max_index])

print "Vertical displacement along the vertical line x=5, y=5: from z=0 to z=10"
print gpa.disp()[np.logical_and(gx==5, gy==5)][:,2]

za.stress()

za.stress_flat()



" ========================================== "
" =========REFERENCE EXAMPLES=============== "
" ========================================== "


""" Some Numpy Operation Examples
np.array([1,2,3,4,5])
```

```
np.linspace(0,1,15)
np.zeros((4,4))
a = np.linspace(0,1,15)
b = np.ones_like(a)
np.sin(a)
print a[0]
a[0] = 20.2
print a
c = np.array(((1,2,3),(4,5,6),(7,8,9),(10,11,12)))
print c
c[0][0]
c[:,0]
"""
""" SOME GRIDPOINTS EXAMPLES
z = it.zone.near((5,5,5))
print "central zone id: {}, position: {}".format(z.id(), z.pos())

for gp in z.gridpoints():
    print "gridpoint with id: {} at {}".format(gp.id(), gp.pos())
"""
```

6.2 KIAST Model

```
import itasca as it
import numpy as np
np.set_printoptions(threshold=20)
it.command("python-reset-state false")
from itasca import zonearray as za
from itasca import gridpointarray as gpa
```

6. Code Block

```
# PARAMETERS #

_D_shaft = 1
_H_shaft = 7.45
_T_plate = 1.5
_B_footing = 3.25
_D_footing = _H_shaft + _T_plate
_B_soil = _B_footing*5
_D_soil = _D_footing+3

_radial = 50
_perimeter = 2*_radial
_axial = 2*_radial
_outer = 2*_radial

_bulk = 13.9e9
_shear = 10.4e9

_E_o = 1e7
_const = 1e8
_poisson = 0.25

# ZONE #

command_zone = """
model new
z crea r-t p 0 (0,0,0) ...
           p 1 ({B_soil},0,0) ...
           p 2 (0,{D_soil},0) ...
```

```
            p 3 (0,0,{B_soil}) ...
            p 4 ({B_soil},{D_soil},0) ...
            p 5 (0,{D_soil},{B_soil}) ...
            p 6 ({B_soil},0,{B_soil}) ...
            p 7 ({B_soil},{D_soil},{B_soil}) ...
            p 8 ({B_footing},0,0) ...
            p 9 (0,0,{B_footing}) ...
            p 10 ({B_footing},{D_soil},0) ...
            p 11 (0,{D_soil},{B_footing}) ...
            p 12 ({B_footing},0,{B_footing}) ...
            p 13 ({B_footing},{D_soil},{B_footing}) ...
            size {radial} {axial} {perimeter} {outer} ...
            rat 1 1 1 1.01 ...
            fill
"""

command = command_zone.format(
B_footing = _B_footing,
B_soil = _B_soil,
D_soil = _D_soil,
radial=_radial,
axial = _axial,
perimeter = _perimeter,
outer = _outer)

it.command(command)

# GROUP #

p = za.pos()
```

6. Code Block

```
x,y,z = p.T
print(it.zone.count(), "zones in whole model")

shaft = reduce(np.logical_and, (np.sqrt(x**2+z**2)<_D_shaft, y<_H_shaft))
za.set_group(shaft, "shaft") # set the zones with shaft = true have "shaft" and "
print(za.in_group("shaft").sum(), "zones in shaft group.") #output how many zones

plate = reduce(np.logical_and, (x<_B_footing,z<_B_footing, y>_H_shaft,y<_D_footing)
za.set_group(plate, "plate") # set the zones with plate = true have "plate" and "
print(za.in_group("plate").sum(), "zones in plate group.") #output how many zones

# CONSTITUTIVE MODEL #

#it.command("""
#zone cmodel assign strain-softening range group "Radial Tunnel1"
#zone property density 2500 bulk 2e8 shear 1e8 range group "Radial Tunnel1"
#zone property cohesion 2e6 friction 45 tension 2e5 dilation 10 range group "Radi
#zone property table-friction 'fri' table-cohesion 'coh' table-dilation 'dil' ran
#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)
#""")

it.command("""
zone cmodel assign elastic range group "Radial Tunnel1"
zone cmodel assign elastic range group 'shaft'
zone cmodel assign elastic range group 'plate'

fish define fname(E_o,const)
loop foreach pnt zone.list
```

6. Code Block

```
z_depth = zone.pos.y(pnt)
E = E_o+const*math.sqrt(z_depth)
zone.prop(pnt,'young')=E
end_loop
end
@fname({E_o_},{const_})
zone property poisson {poisson_}
plot item create zone contour property name 'young'
""".format(E_o_=_E_o,const_=_const,poisson_=_poisson))

it.command("""
zone property bulk {bulk}  shear {shear} range group 'shaft'
zone property bulk {bulk}  shear {shear} range group 'plate'
""".format(bulk=_bulk,shear=_shear))

#soil = za.in_group("Radial Tunnel1")
# list(gp.list)
# gp.isgroup(::gp.list,"North")
# gp.force.unbal(::ad)
# [ad = list(gp.list)(gp.isgroup(::gp.list,"North))]
```

7

Theory

7.1 Interface

There are several instances in geomechanics in which it is desirable to represent planes on which sliding or separation can occur. For example:

1. joint, fault, or bedding planes in a geologic medium;
2. an interface between a foundation and the soil;
3. a contact plane between a bin or chute and the material that it contains;
4. a contact between two colliding objects; and
5. a planar "barrier" in space, which represents a fixed, non-deformable boundary at an arbitrary position and orientation.

FLAC3D provides interfaces that are characterized by Coulomb sliding and/or tensile and shear bonding. Interfaces have the properties of friction, cohesion, dilation,

normal and shear stiffnesses, and tensile and shear bond strength. Although there is no restriction on the number of interfaces or the complexity of their intersections, it is generally not reasonable to model more than a few simple interfaces with FLAC3D because it is awkward to specify complicated interface geometry. The program 3DEC (Itasca 2007) is specifically designed to model many interacting bodies in three dimensions; it should be used instead of FLAC3D for the more complicated interface problems.

Interfaces may also be used to join regions that have different zone sizes. In general, the zone attach command should be used to join grids together. However, in some circumstances, it may be more convenient to use an interface for this purpose. In this case, the interface is prevented from sliding or opening because it does not correspond to any physical entity.

7.2 Formulation

FLAC3D represents interfaces as collections of triangular elements (interface elements), each of which is defined by three nodes (interface nodes). Interface elements can be created at any location in space. Generally, interface elements are attached to a zone surface face; two triangular interface elements are defined for every quadrilateral zone face. Interface nodes are then created automatically at every interface element vertex. When another grid surface comes into contact with an interface element, the contact is detected at the interface node and is characterized by normal and shear stiffnesses, and sliding properties.

Each interface element distributes its area to its nodes in a weighted fashion. Each interface node has an associated representative area. The entire interface is thus divided into active interface nodes representing the total area of the interface. Figure 1 illustrates the relation between interface elements and interface nodes, and

the representative area associated with an individual node.

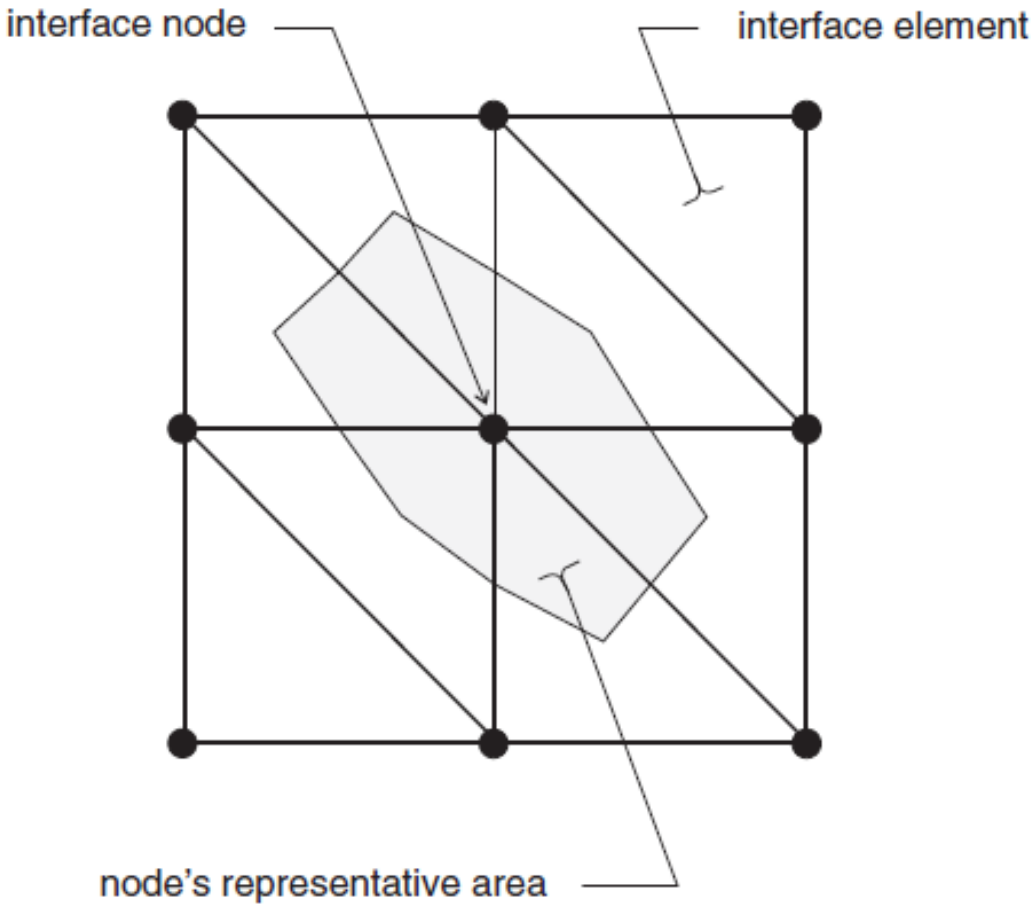


Figure 1: Distribution of representative areas to interface nodes.

It is important to note that interfaces are one-sided in FLAC3D. (This differs from the formulation of two-sided interfaces in two-dimensional FLAC (Itasca 2011).) It may be helpful to think of FLAC3D interfaces as "shrink-wrap" that is stretched over the desired surface, causing the surface to become sensitive to interpenetration with any other face with which it may come into contact.

The fundamental contact relation is defined between the interface node and a zone surface face, also known as the target face. The normal direction of the interface force is determined by the orientation of the target face.

During each timestep, the absolute normal penetration and the relative shear velocity are calculated for each interface node and its contacting target face. Both of these values are then used by the interface constitutive model to calculate a normal force and a shear-force vector. The constitutive model is defined by a linear Coulomb shear-strength criterion that limits the shear force acting at an interface node, normal and shear stiffnesses, tensile and shear bond strengths, and a dilation angle that causes an increase in effective normal force on the target face after the shear-strength limit is reached. By default, pore pressure is used in the interface effective stress calculation. This option can be activated/deactivated using the command zone interface effective command by setting effective = on/off. Figure 2 illustrates the components of the constitutive model acting at interface node (P):

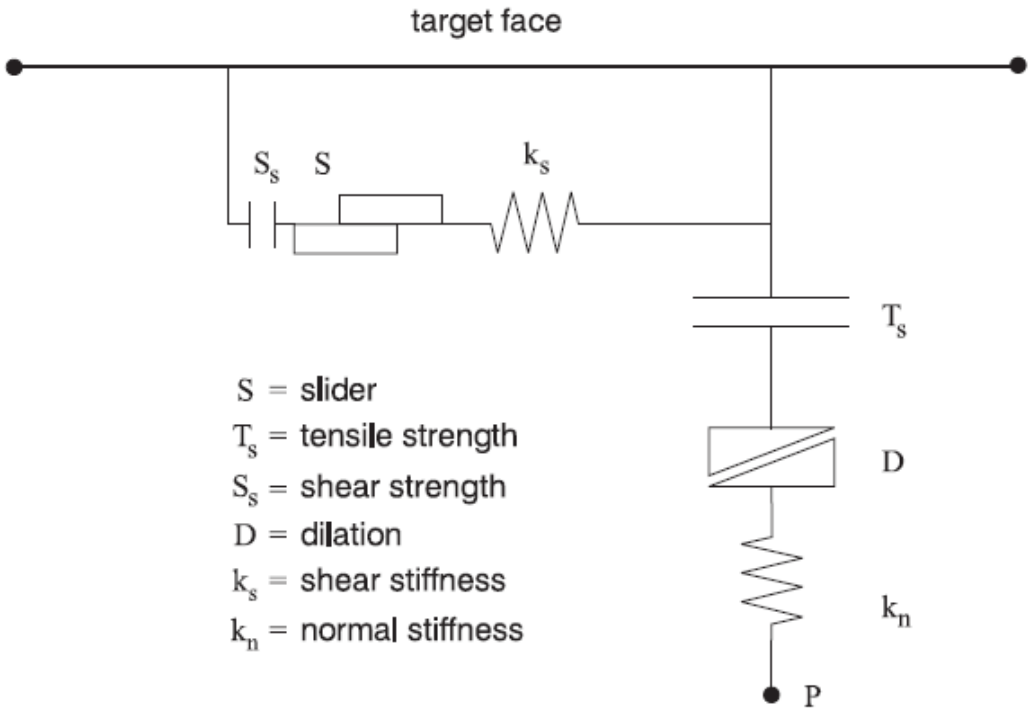


Figure 2: Components of the bonded interface constitutive model.

The normal and shear forces that describe the elastic interface response are determined at calculation time $(t + \Delta t)$ using the relations:

$$F_n^{(t+\Delta t)} = k_n u_n A + \sigma_n A$$

$$F_{si}^{(t+\Delta t)} = F_{si}^{(t)} + k_s \Delta u_{si}^{(t+0.5\Delta t)} A + \sigma_{si} A$$

The inelastic interface logic works in the following way:

1. Bonded interface — The interface remains elastic if stresses remain below the bond strengths; there is a shear bond strength, as well as a tensile bond strength. The normal bond strength is set using the tension interface property keyword. The command zone interface node property shear-bond-ratio = sbr sets the shear bond strength to sbr times the normal bond strength. The default value of property shear-bond-ratio (if not given) is 100.0. The bond breaks if either the shear stress exceeds the shear strength, or the tensile effective normal stress exceeds the normal strength. Note that giving property shear-bond-ratio alone does not cause a bond to be established—the tensile bond strength must also be set.

2. Slip while bonded — An intact bond, by default, prevents all yield behavior (slip and separation). There is an optional property switch (bonded-slip) that causes only separation to be prevented if the bond is intact (but allows shear yield, under the control of the friction and cohesion parameters, using (F_n) as the normal force). The command to allow/disallow slip for a bonded interface segment is zone interface node and by setting bonded-slip on or off.

The default state of bonded-slip (if not given) is off.

3. Coulomb sliding — A bond is either intact or broken. If it is broken, then the behavior of the interface segment is determined by the friction and cohesion (and of course the stiffnesses). This is the default behavior, if bond strengths are not set (zero). A broken bond segment cannot take effective tension (which may occur under compressive normal force, if the pore pressure is greater). The shear force is zero (for a nonbonded segment) if the effective normal force is tensile or zero.

The Coulomb shear-strength criterion limits the shear force by the relation:

$$F_{smax} = cA + tan\phi(F_n - pA)$$

During sliding, shear displacement may cause an increase in the effective normal stress on the joint, according to the relation:

$$\sigma_n := \sigma_n + \frac{|F_s|_o - F_{smax}}{Ak_s} tan\psi k_n$$

On printout (see the zone interface node list command) the value of tension denotes whether a bond is intact or broken (or not set) — nonzero or zero, respectively.

The normal and shear forces calculated at the interface nodes are distributed in equal and opposite directions to both the target face and the face to which the interface node is connected (the host face). Weighting functions are used to distribute the forces to the gridpoints on each face. The interface stiffnesses are added to the accumulated stiffnesses at gridpoints on both sides of the interface in order to maintain numerical stability.

Interface contacts are detected only at interface nodes, and contact forces are transferred only at interface nodes. The stress state associated with a node is assumed to be uniformly distributed over the entire representative area of the node. Interface properties are associated with each node; properties may vary from node to node.

By default, the effect of pore pressure is included in the interface calculation by using effective stress as the basis for the slip condition. (The interface pore pressure is interpolated from the target face.) This applies in model configure fluid mode, or if pore pressures are assigned with the zone water or zone gridpoint initialize pore-pressure command without specifying model configure fluid. The user can switch options for interface s by using the zone interface effective command and by setting effective on or off. By default in the FLAC3D logic, fluid flow (saturated or unsaturated) is carried across an interface, provided the interface keyword maximum-edge is not used for that particular interface. The permeable interface option can be deactivated/reactivated for interface s by using the zone interface permeability command and by setting effective on or off. Note that if the keyword maximum-edge is used after the zone interface element command, and permeability is on for a particular interface, a warning is issued to inform the user that this interface will be considered as impermeable to fluid flow. (Note that for fluid flow calculation only, a mechanical model must be present. Also, the model cycle 0 command with model mechanical active on should be used to initialize the weighting factors used to transfer fluid flow information across the interface.) No pressure drop normal to the joint and no influence of normal displacement on pore pressure is calculated.

Also, flow of fluid along the interface is not modeled.

7.3 Creation of Interface Geometry

Interfaces are created with the zone interface create command. For cases in which an interface between two separate grids in the model is required, the zone interface create by-face command should be used to attach an interface to one of the grid surfaces. This command generates interface elements for interface s along all surface zone faces with a center point that fall within a specified range. Any surfaces on which an interface is to be created must be generated initially; it must be possible to specify an existing surface in order to create the interface elements. A gap must be specified between two adjacent surfaces, unless the zone interface create by-face command and the separate keyword are given. In this case, the separate sub-grids may have surface gridpoints at the same location in space.

By default, two interface elements are created for each zone face. The number of interface elements can be increased by using the zone interface s element maximum-edge v command. [1] This causes all interface elements with edge lengths larger than v to subdivide into smaller elements until their lengths are smaller than v. This command can be used to increase the resolution and decrease arching of forces in portions of a model that have large contrasts in zone size across an interface.

Several rules should be followed when using interface elements in FLAC3D:

1. If a smaller surface area contacts a larger surface area (e.g., a small block resting on a large block), the interface should be attached to the smaller region.

2. If there is a difference in zone density between two adjacent grids, the interface should be attached to the grid with the greater zone density (i.e., the greater number of zones within the same area).

3. The size of interface elements should always be equal to or smaller than the target faces with which they will come into contact. If this is not the case, the interface elements should be subdivided into smaller elements.

4. Interface elements should be limited to grid surfaces that will actually come into contact with another grid.

   A simple example illustrating the procedure for interface creation is provided in "DippingJoint.f3dat". The corresponding project file, "DippingJoint.f3prj", is located in the folder "datafiles InterfaceDippingJoint." The example is a block specimen containing a single joint dipping at an angle of 45ř

```
; Create interface elements on the top surface of the base
zone interface 'joint' create by-face separate range group 'Top' group 'Base'
model save 'int1'
return
```

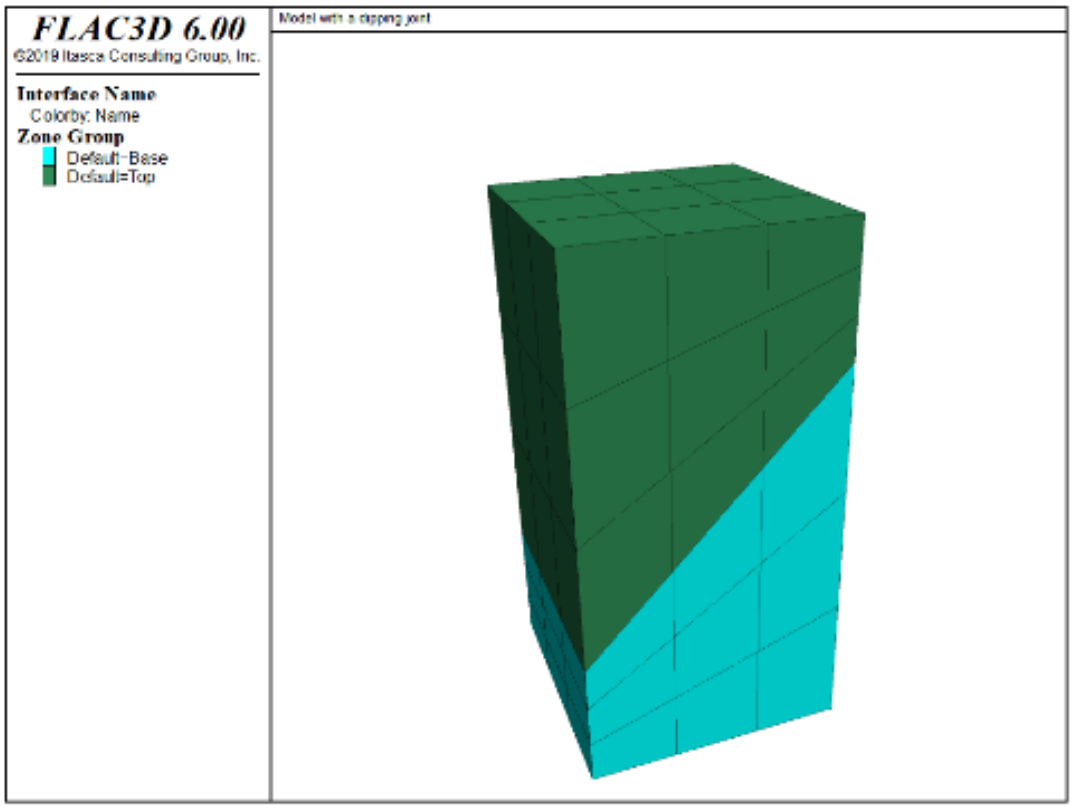


Figure 1: Initial geometry before creation of the interface.

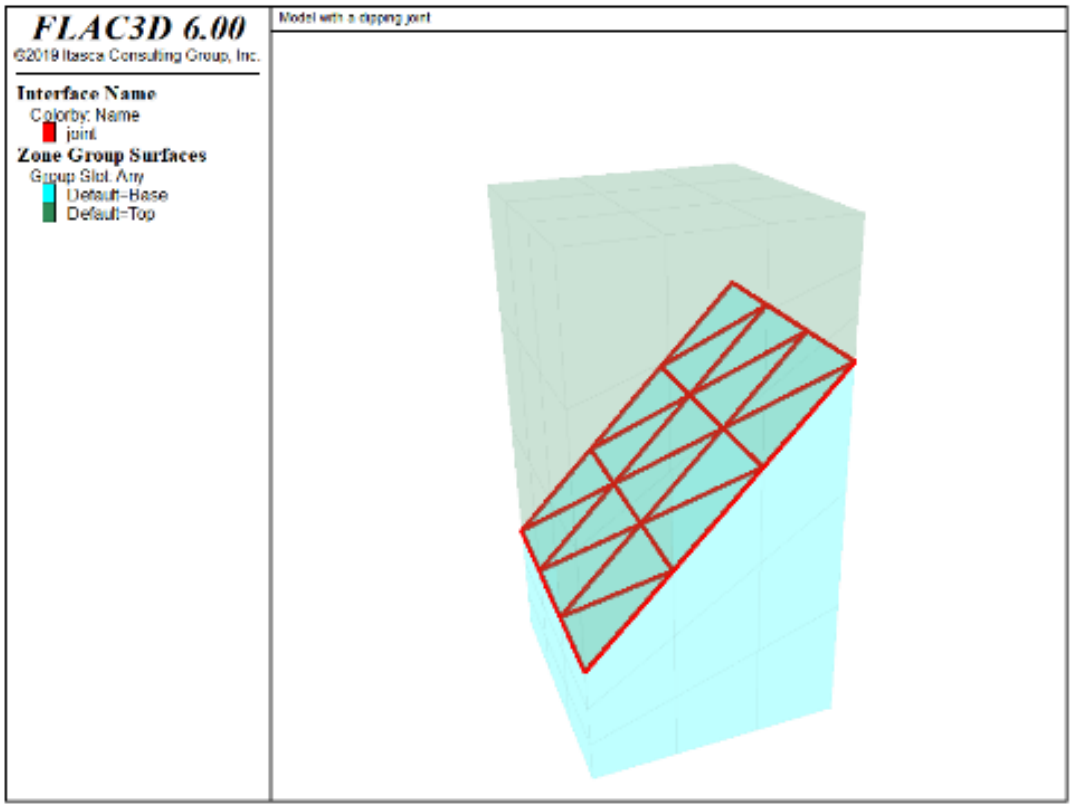


Figure 2: Final geometry.

8

Command

8.1 Interface

8.1.1 zone face group

```
zone face group s keyword ... <range>
```

Example:

```
zone face group 'wall' internal range group 'wall-c' group 'pile'
```

Adds a group name ‘wall’ to zone faces included in the optional range. Also can be used with zone face group “slotname = groupname” Zone face may belong to many groups, up to 128. keywords: internal, or, remove.
- internal: to select all faces.
If specified, then the range looks at internal faces as well as surface faces.

- or: considered inside the surface if any of those conditions has a non-null model.

- remove: group is removed from the zone face.

8.1.2 zone face skin

```
zone face skin
```

Generates contiguous face groups by skinning.

"skinning" refers to automatic face group generation, which occurs as follows: the program looks for contiguous faces and puts them into automatically named groups within the slot named skin. Two faces are not considered contiguous if the angle between two faces exceeds the break angle, or if the two faces belong to different groups according to the command's slot specification. Unless internal is specified, the operation of zone face skin is restricted to external faces. Only zone faces that have no zone on the other side, a hidden zone on the other side are considered surface faces. When use-hidden-zones is used, within specified range will be included for consideration.

group names are generated as follows: sEast, sWest, sNorth, sSouth, sTop, sBottom

Internal faces are assigned names of the group on either side that changed in order to create them.

8.1.3 zone separate

```
zone separete by-face new-side group 'iwall' slot 'int' range group 'wall' or 'base
```

separates internal faces specified by the range. The gridpoints of the face are duplicated, and new face is created. New faces and gridpoints get copies of all group and extra variable assignments belonging to the original face and gridpoint.

Example: If Fred and George are group names assigned to zones, then

```
range group 'Fred' group 'George'
```

will select faces that are connected to zone of both group Fred and George.

keyword: - by-face: attempts to separate all internal faces in the range - new-side group: newly created faces will be assigned the group name in the specified slot. The default slot, is named Default.

8.1.4 zone interface create

```
zone interface 'side' create by-face range group 'wall' and 'iwall'


zone interface 'side' node property stiffness-normal 1e8 stiffness-shear 1e8 fricti
```

creates nodes or elements on interface side

There are two techniques for creating an interface. 1. To derive an interface from a range of zone faces using by-face keyword. 2. To specify a triangular interface element from 3 points. 3. Command may also be used to construct an interface node that may be used with the element keyword.

keyword: - by-face: interface elements are created on all surface zone faces that are within the specified range. An error will occur if interface elements from that interface already exist on the selected faces. If the optional separate keyword is used, then internal zone faces are selected by the range. The list of selected faces is then automatically separated (as with zone separate) and interface elements placed on one side. The following are for the separate keyword: new-side-origin, clear-attach.

- element point i keyword . . . : create a triangular interface element by specifying 3 points. Three verticies must be specified by the following keyword. Each point can be created and located in space using the position keyword, or an existing interface node with ID i may be specified. The interface element that is created is not attached to a grid face even if the location corresponds to that of the face. the element is fixed in space. The active side of the element is defined by walking around the edge of the element, from point 1 to point 2 to point 3; the active side is up when walking in a clockwise direction.

8. Command

For each point of the element, one of the following two keywords must be supplied: node i, position v.

- node v : this creates an interface node at position v. If a node already exists at the selected location, an error is reported. The created node is fixed in space.

Appendices

A

1. Template

A.1 Problem Description

A.2 Modeling Procedure

A.3 Zones/Groups

A.4 Properties

A.5 B.C. and I.C.

A.6 Initial Equilibrium

A.7 Alterations

A.8 Results

B

Reference Collective

B.0.1 Uplift Resistance of Anchor Plate

B.0.1.1 Before 1968

- Coulomb
- Mohr
- Kotter's equation
- Balla (1961)
- Mors
- Matsuo

B.0.1.2 Post-1968

- Meyerhof, G.G., and Adams, J.I. 1968
- Meyerhof, G.G. 1973
- Das, B.M., and Seeley, G.R. 1975
- Rowe, R.K., and Davis, H. 1982
- Dickin, E.A., and Leung, C.F. 1983

- Murray, E.J., and Geddes, J.D. 1987
- Dickin, E.A. 1988
- Koutsabeloulis, N.C., and Griffiths, D.V. 1989 #### Post-2000
- Merifield, R.S., and Sloan, S.W. 2006

B.0.2 Numerical Analysis

B.0.3 Standards

- IEEE 2001
- DS 1110, DS 1111

B.0.4 Textbook

- Das, B. M. 2013. Earth Anchors

B.0.5 Ph.D Thesis

B.0.6 Award Lecture