ITASCA™

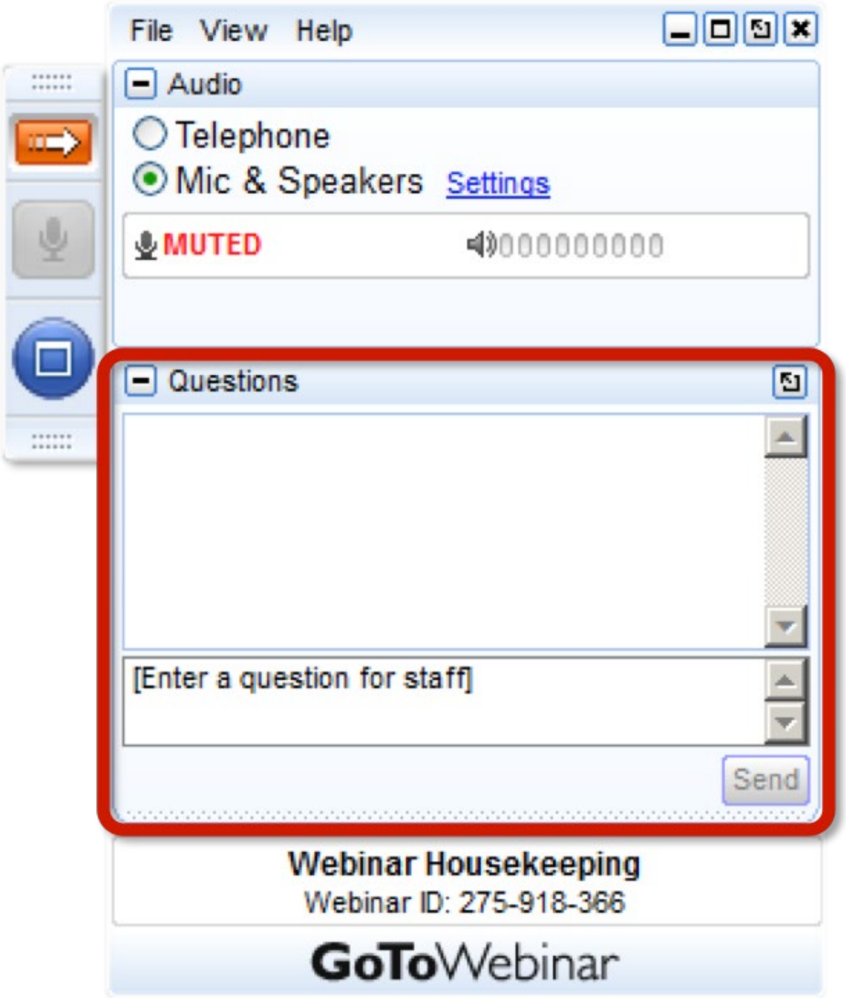
40 YEARS

# New Features in *PFC* 7.0

# Information

To type your questions, please use **Questions** dialog in the **GoTo**Webinar window.

Questions will be answered at the end of the webinar.






ITASCA™

# Major New Features

## Multithreaded *FISH*

List splitting/filtering, *FISH* operators,…

## Particle Inlets

Generate particles at a specified flow-rate during cycling

## New Contact Models

*FISH* model | Adhesive models (JKR, EEPA) | SpringNetwork model

## Stress Installation Schemes

Ball and rigid block packings

## Feature Enhancements

Clumping Logic | Rigid blocks | Structural Elements

## Linux Version

# Multithreaded *FISH*

List Splitting/Filtering

*FISH* Operators

# List Splitting

- Splitting can be used as an alternative to loop statements to perform actions on many objects in a very clear and concise manner
- In order to make a split call, give the split operator "::" prefix to one or more arguments of the function

```
fish define hmax
    local tmp = -1e20
    loop foreach local b ball.list
        tmp = math.max(tmp,ball.pos(b)->y+ball.radius(b))
    endloop
    hmax = tmp
end
[hmax]
```

```
[hmax = list.max(ball.pos(::ball.list)->y+ball.radius(::ball.list))]
```

- If splitting is performed on a function that is tagged as thread-safe, the splitting will be done on all available threads automatically

# List Filtering

- Splitting in combination with boolean list filtering can be used to quickly find a sub-list of objects selected by a specific criteria

```
[ravg = list.sum(ball.radius(::ball.list)) / ball.num]

[allBalls = list(ball.list)]
[check = ball.isgroup(::ball.list,'large')]
[largeBalls = allBalls(check)]

[ravg2 = list.sum(ball.radius(::largeBalls)) / list.size(largeBalls)]
```

compute average radius over all balls

list subset of balls in the group “large”

compute average radius over large balls only

# *FISH* Operators

- *FISH* operators are a special class of function designed to be executed in a multi-threaded environment.
- Operators are created using the **FISH OPERATOR** command, with arguments following just like **FISH DEFINE**. The *FISH* lines in the definition are the same as for a normal function.

```
fish define sort
    loop foreach local b ball.list
        if ball.radius(b) > ravg then
            ball.group(b) = 'large'
        endif
    endloop
end
[sort]
```

```
fish operator sort(b)
    if ball.radius(b) > ravg then ball.group(b) = 'large'
end
[sort(::ball.list)]
```

- Because the functions need to be safe when multiple threads are running simultaneously, they operate in a restricted environment. See documentation for further details.
- Speedup increases with function complexity and threads availability

# Implications of Multithreaded *FISH*

```
fish define add_fluidforces
  global vf = 0.0
  loop foreach ball ball.list
    local vi = 0.0
    local d1 = ball.pos.z(ball) - ball.radius(ball)
    if ball.pos.z(ball) - ball.radius(ball) >= zf_
      ; above water level
      ball.force.app(ball) = vector(0.0,0.0,0.0)
    else
      local fdrag = -6.0*math.pi*etaf_*ball.radius(ball)*ball.vel(ball)
      local vbal = 4.0*math.pi*ball.radius(ball)^3 / 3.0
      if ball.pos.z(ball) + ball.radius(ball) <= zf_ then
        ; totally immerged
        vi = 4.0*math.pi*ball.radius(ball)^3 / 3.0
      else
        ; partially immerged
        if ball.pos.z(ball) >= zf_ then
          local h = ball.radius(ball) - (ball.pos.z(ball)-zf_)
          local vcap = math.pi*h^2*(3*ball.radius(ball) - h) /3.0
          vi = vcap
        else
          h = ball.radius(ball) - (zf_ - ball.pos.z(ball))
          vcap = math.pi*h^2*(3*ball.radius(ball) - h) /3.0
          vi = vbal - vcap
        endif
      endif
      local fb = -1.0*rhof_*vi*global.gravity
      ball.force.app(ball) = fb + (vi/vbal) *fdrag
    endif
    vf = vf + vi
  endloop
end
```

Speedup ~5

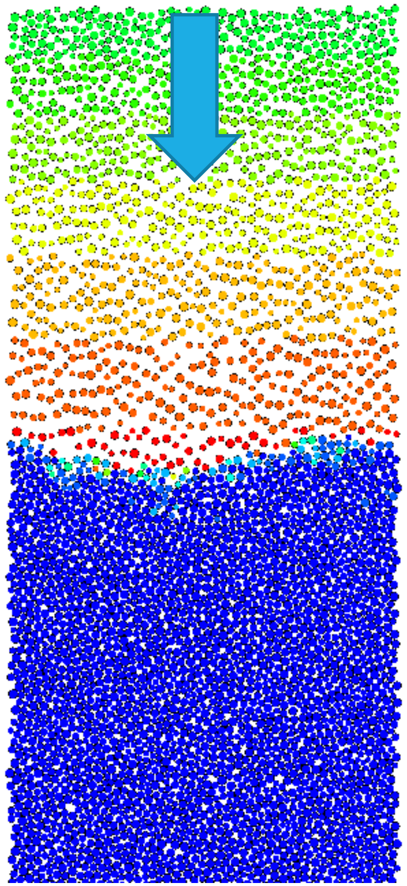
```
fish operator add_fluidforces(ball)
  local vi = 0.0
  local d1 = ball.pos.z(ball) - ball.radius(ball)
  if ball.pos.z(ball) - ball.radius(ball) >= zf_
    ; above water level
    ball.force.app(ball) = vector(0.0,0.0,0.0)
  else
    local fdrag = -6.0*math.pi*etaf_*ball.radius(ball)*ball.vel(ball)
    local vbal = 4.0*math.pi*ball.radius(ball)^3 / 3.0
    if ball.pos.z(ball) + ball.radius(ball) <= zf_ then
      ; totally immerged
      vi = 4.0*math.pi*ball.radius(ball)^3 / 3.0
    else
      ; partially immerged
      if ball.pos.z(ball) >= zf_ then
        local h = ball.radius(ball) - (ball.pos.z(ball)-zf_)
        local vcap = math.pi*h^2*(3*ball.radius(ball) - h) /3.0
        vi = vcap
      else
        h = ball.radius(ball) - (zf_ - ball.pos.z(ball))
        vcap = math.pi*h^2*(3*ball.radius(ball) - h) /3.0
        vi = vbal - vcap
      endif
    endif
    local fb = -1.0*rhof_*vi*global.gravity
    ball.force.app(ball) = fb + (vi/vbal) *fdrag
  endif
  return vi
end

fish define compute_fluidforces
  global vf = list.sum(add_fluidforces(::ball.list))
end
```

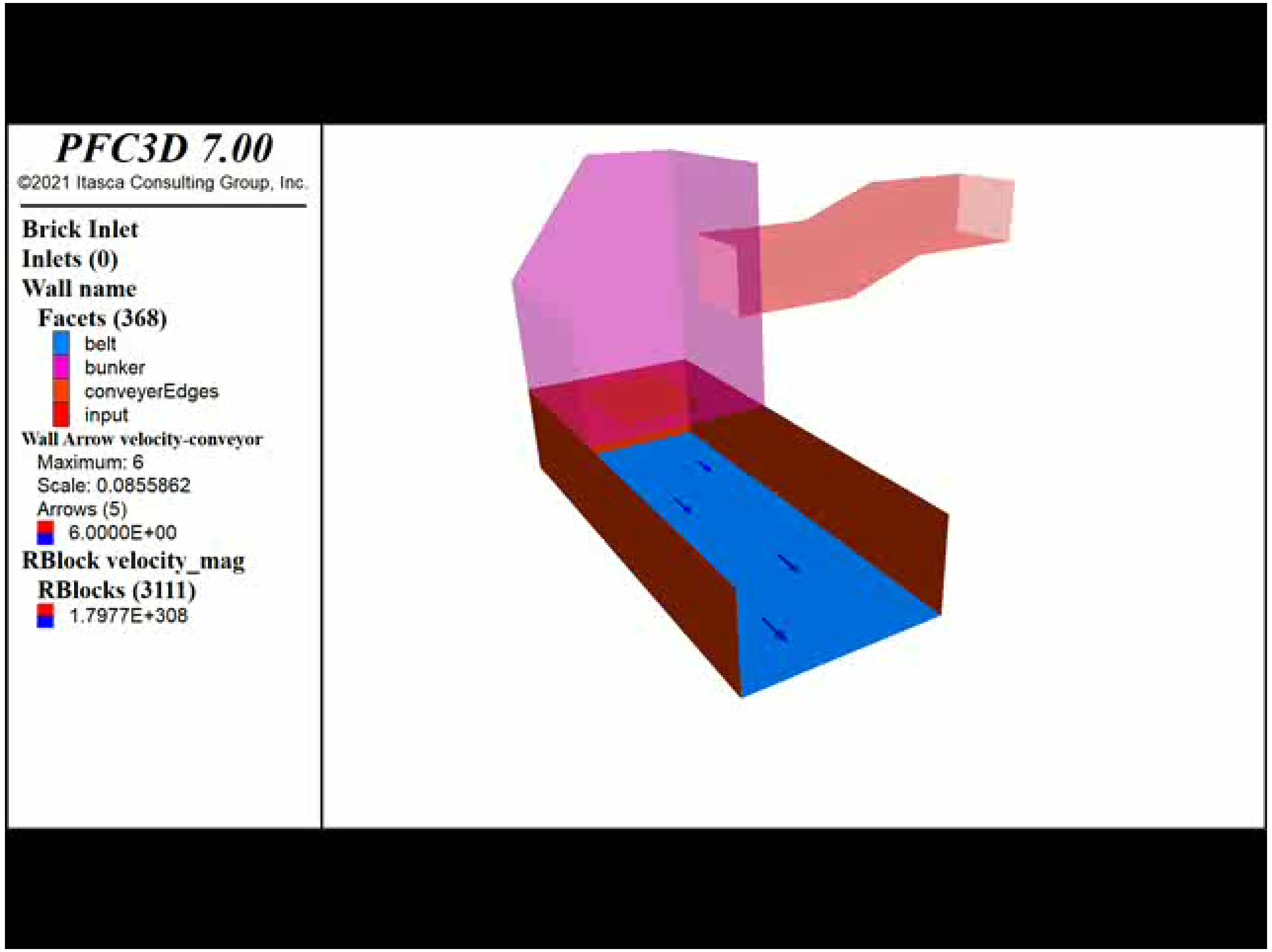
# Particle Inlets

# Particle Inlets

- Generate ("feed") particles during cycling
- Builds upon the Brick logic:
  - A brick comprising primitive particles is first generated
  - The information stored in the brick is used by the inlet to generate particles in the system at a specified flow-rate
- Compatible with balls / clumps / rigid blocks
- Inlets can be positioned / oriented as desired
- Inlets can also translate and rotate during cycling at specified velocities
- A relaxation scheme can be activated to prevent instabilities resulting from large overlaps when particles are inserted in the model



# Inlet : Conveyor example



# New Contact Models

*FISH* Model
Adhesive Models (JKR,EEPA)
Spring-Network Model

# *FISH* Contact Model

*PFC* is used frequently to introduce custom physics during cycling.

- Doing this requires a custom C++ contact model for anything other than built-in models.

Introduce a simple contact model with forces/moments and a few properties (bonding status, reference gap, stiffnesses) that is called during the force-displacement law.

The user can define stiffnesses so that the timestep is automatically computed.

The contact state information is filled and passed to the specified *FISH* function

- No need for the user to compute state information themselves.

Though ~ 4 times slower than a C++ contact model, it is substantially more efficient (by a factor of ~10) than previous *FISH* implementations.

This addition makes contact model development/debugging simpler and easier.

For efficiency, contact model properties can be accessed by integers in *FISH*

# What does this look like?

retrieve property index

Force-displacement implementation of linear model without viscous damping

- Arguments are contact state information
- Store desired properties as extra variables of the contacts
- Get/set properties via the appropriate index

```
[forceInd = contact.model.prop.index("fish","force")]
fish operator linearModel(cp,trans,ang,curv1,curv2,inertialMass,gap,canFail,activated)
    ;do the force displacement law
    local kn = contact.extra(cp,1)
    local ks = contact.extra(cp,2)
    local overlap = contact.prop(cp,rgapInd) - contact.gap(cp)
    local lin_F_old = contact.prop(cp,forceInd)
    local force = lin_F_old
    force->x = overlap * kn
    force->x = math.max(force->x,0.0)
    local sforce = vector(0,0,0)
    sforce->y = force->y - trans->y * ks
    sforce->z = force->z - trans->z * ks
    if canFail == true
        ;check for sliding
        local crit = force->x * fric;
        local sfmag = math.mag(sforce)
        if (sfmag > crit)
            local rat = crit / sfmag;
            sforce *= rat;
        endif
    endif
    force->y = sforce->y
    force->z = sforce->z

    ;set the force
    contact.prop(cp,forceInd) = force
end
```

# Catch contact create events to do stuff

- Set as the CMAT entry
- Catch when contacts are created to do things like set the stiffnesses (just like the deformability method) and to set the *FISH* operator to be executed

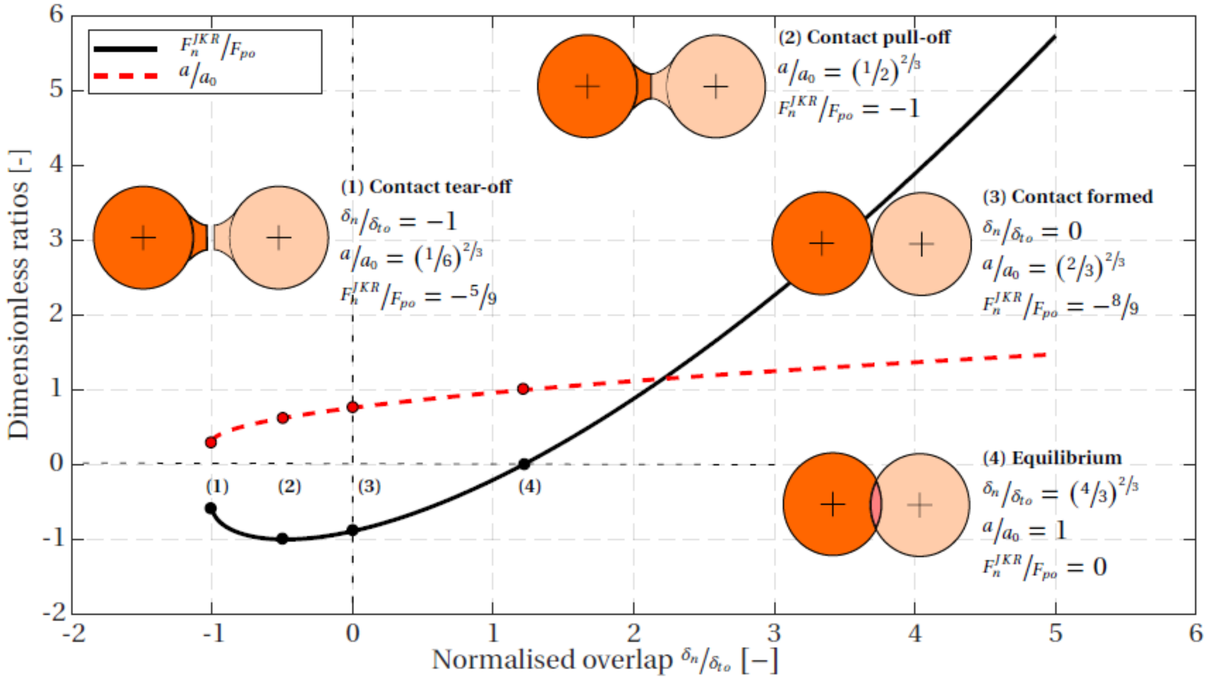
```
contact cmat default model fish
fish define catchContact(c)
    local rsum = 0.0
    local rsq = 0.0
    local rad1 = 0.0
    if type.pointer.id(c) == contact.typeid('ball-ball')
        rad1 = ball.rad(contact.end1(c))
        local rad2 = ball.rad(contact.end2(c))
        rsum = rad1 + rad2
        rsq = 1./math.min(rad1,rad2)
    endif
    if type.pointer.id(c) == contact.typeid('ball-facet')
        rad1 = ball.rad(contact.end1(c))
        rsum = rad1
        rsq = 1./rad1
    endif
    local kn = math.pi() * emod / (rsq * rsq * rsum)
    local ks = 0.0
    if krat > 0
        ks = kn / krat
    endif
    contact.prop(c,stifftInd) = vector(kn,ks)
    contact.prop(c,stiffaInd) = vector(0,0,0)
    contact.prop(c,symbolInd) = "linearModel"
    contact.extra(c,1) = kn
    contact.extra(c,2) = ks
end
fish callback add catchContact event contact_create
```

# New Adhesive Contact Models

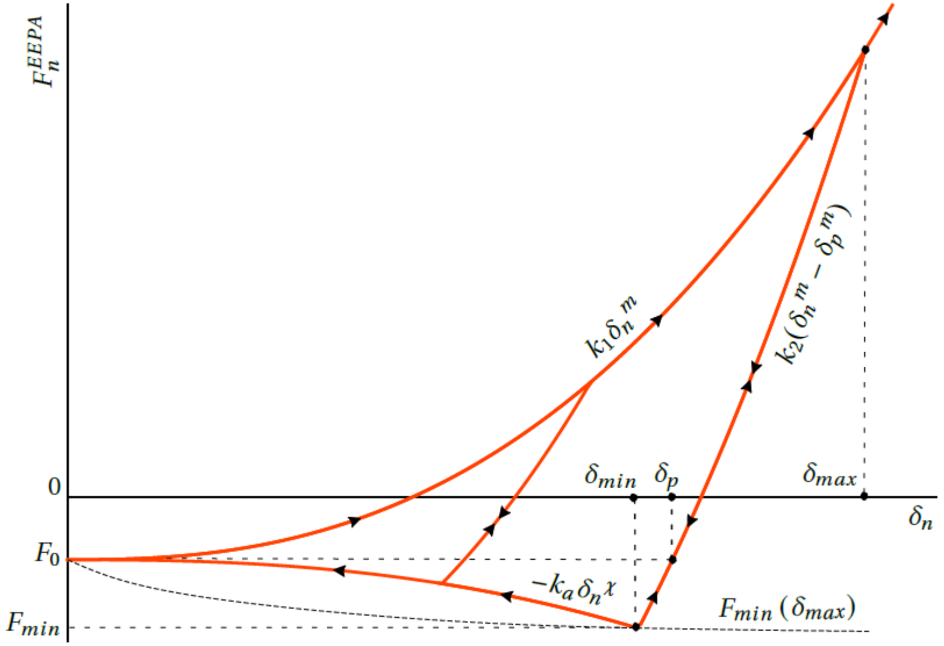
- Two new built-in contact models:
  - JKR (Johnson-Kendall-Roberts):
    - Extension of the well-known Hertz contact model proposed by (Johnson,1971)
    - Accounts for attraction forces due to van der Waals effects. Also used to model material where the adhesion is caused by capillary or liquid-bridge forces.
  - EEPA (Edinburgh Elasto-Plastic Adhesive ):
    - Extension of the linear hysteretic model by (Walton & Braun,1986). Based on (Morrisey,2013)
    - Allows tensile forces to develop, as well as a hysteretic, non-linear force-displacement behavior in compression.
  - Both models also incorporates viscous damping and rolling resistance mechanisms, similar to the Rolling Resistance Linear Model.

# New Adhesive Contact Models

## Johnson-Kendall-Roberts (JKR)



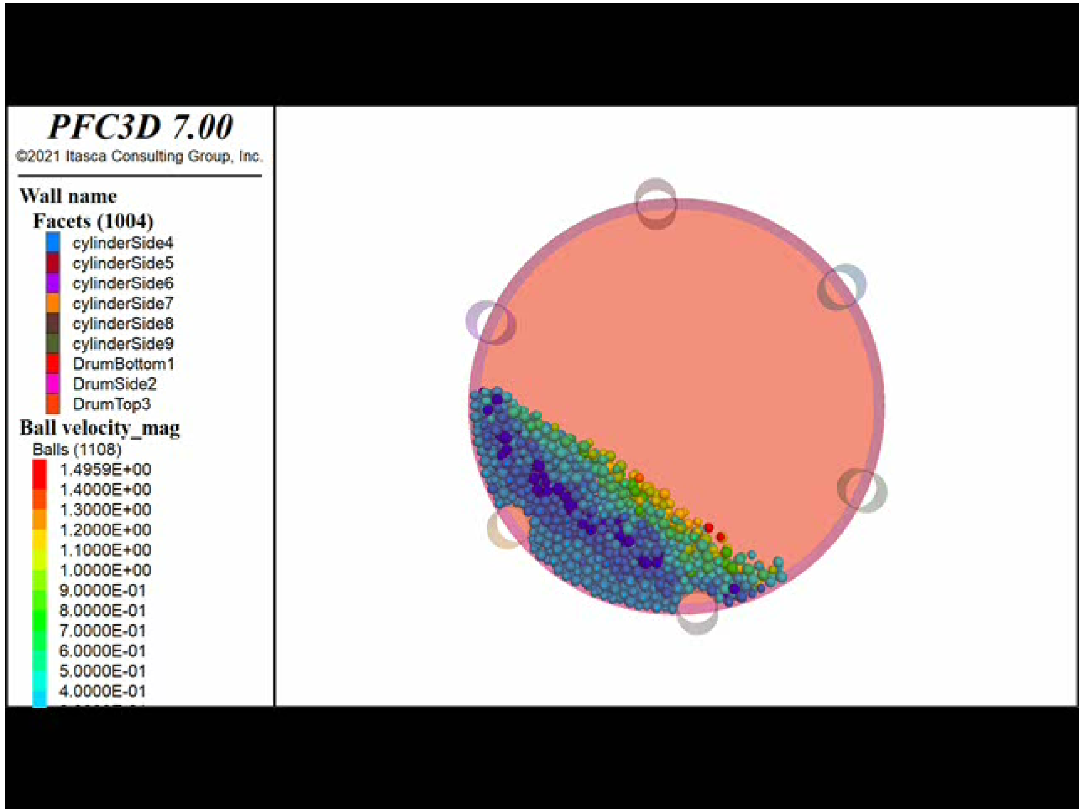

## Edinburgh Elasto-Plastic Adhesive (EEPA)



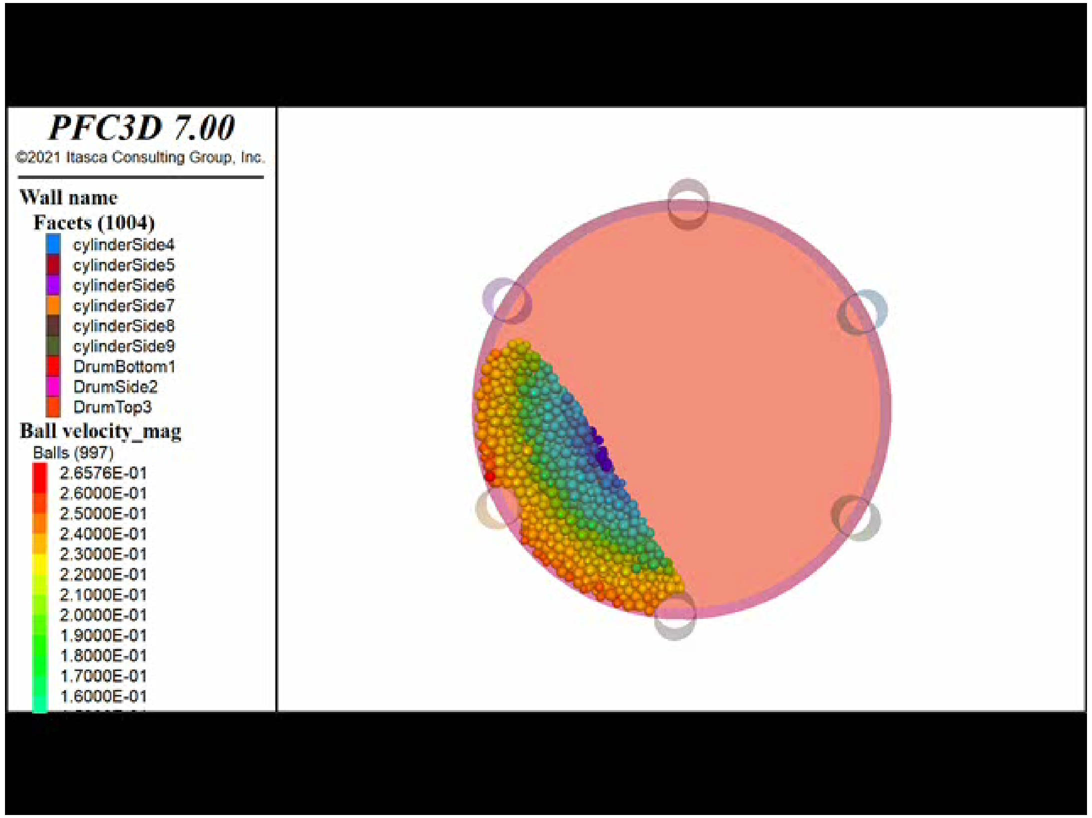

Normal force versus overlap

# New Adhesive Contact Models : Rotating Drum

Hertz model (no adhesion)

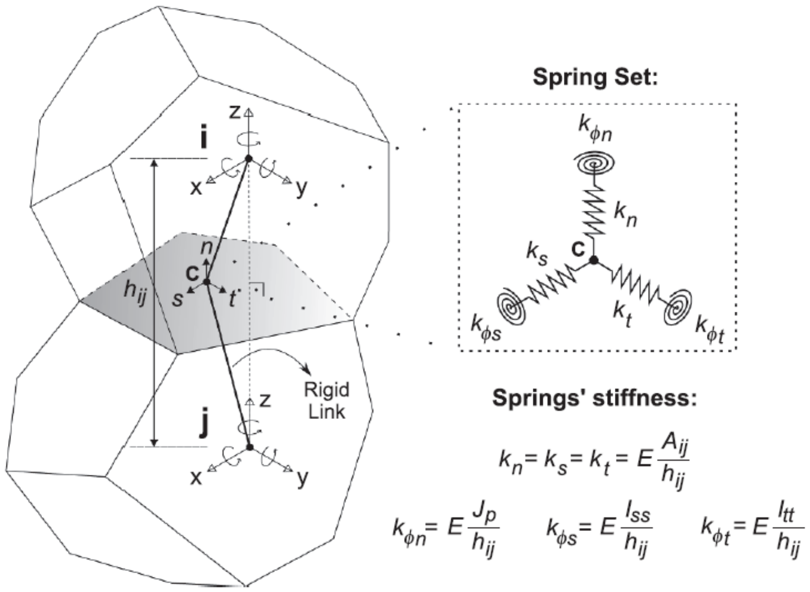



Edinburgh Elasto-Plastic Adhesive (EEPA)



# SpringNetwork for Bonded Materials – Elastic Response

- Compute translational and rotational stiffnesses based on lattice theory (zero Poisson ratio)
  - Significantly reduces heterogeneity in the elastic response at the particle scale
  - Heterogeneity can then be assigned by the user independent of microstructure
- Use continuum theory to add a fictitious force at contacts to produce the correct Poisson ratio (assuming isotropic behavior)
  - Has been extended elsewhere to anisotropic materials
- Using these methods, the elastic continuum response is matched without calibration
  - This is independent of particle type, meaning it works for balls, rigid blocks and clumps

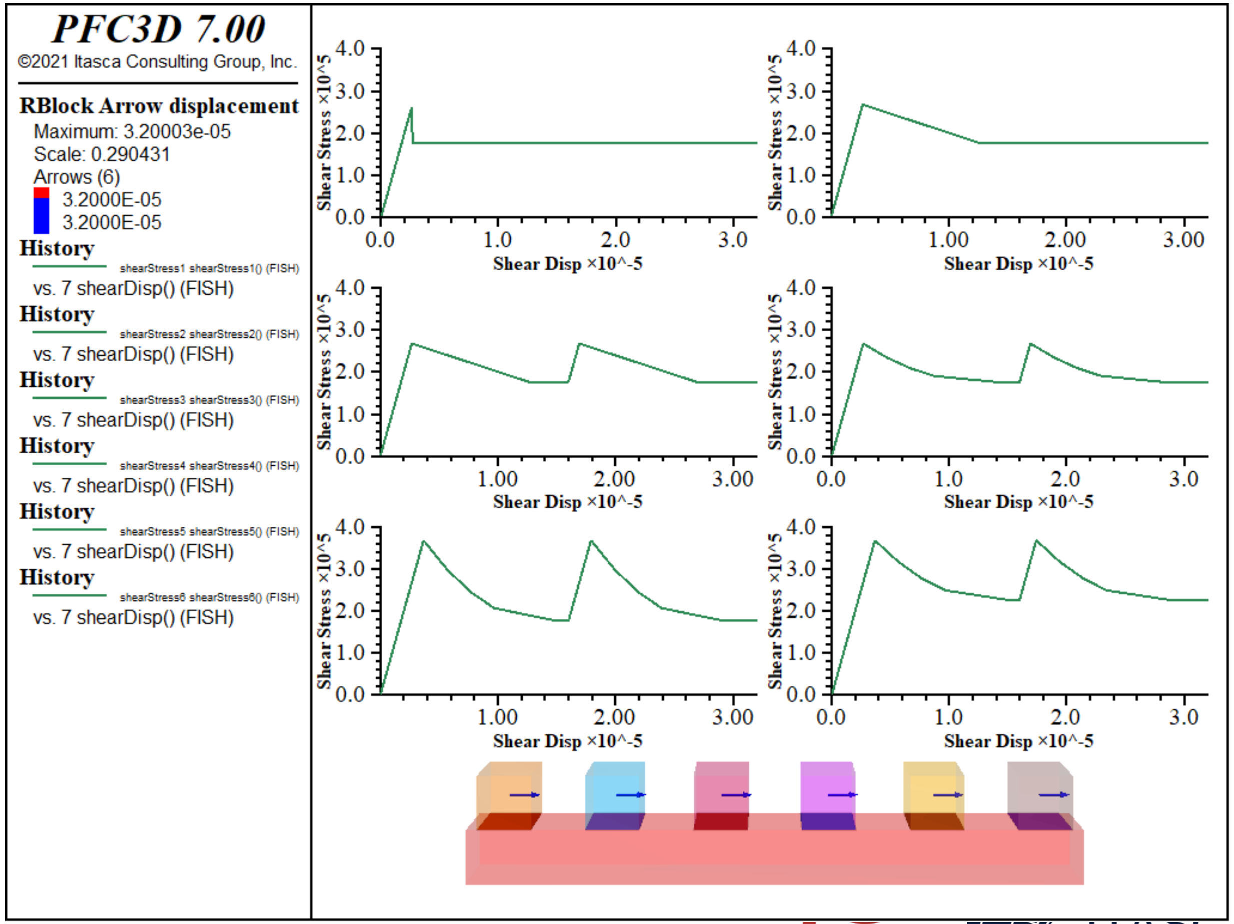



Rasmussen, L.L., 2021. Hybrid lattice/discrete element method for bonded block modeling of rocks. Computers and Geotechnics 130, 103907. https://doi.org/10.1016/j.compgeo.2020.103907

# SpringNetwork for Bonded Materials – Failure Response

- Like the parallel bond model, the maximum tensile stress at the bond periphery (including bending) is used for tensile failure
- Arbitrary tensile softening supported via a table
  - Linear interpolation of strength as a function of continued elongation
- Arbitrary slip weakening supported via a table
  - Linear interpolation of friction as a function of continued slip
- Healing supported when slip ceases
- Bending and twisting frictional resistance as in the SoftBond model
- Pore pressure included meaning effective stresses can be used for failure computations

Docs -> PFC -> Examples -> Verification Problems -> Spring Network Contact Model Capabilities

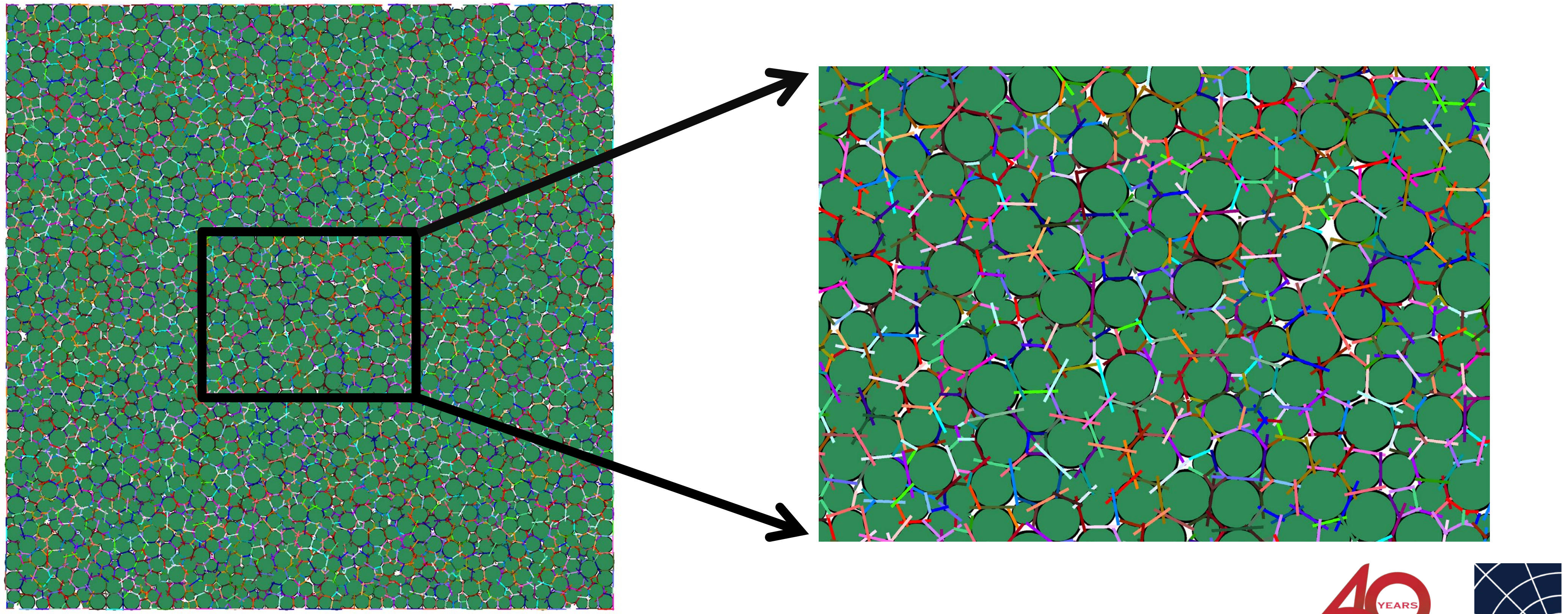


# Stress Installation

Balls and Rigid Blocks

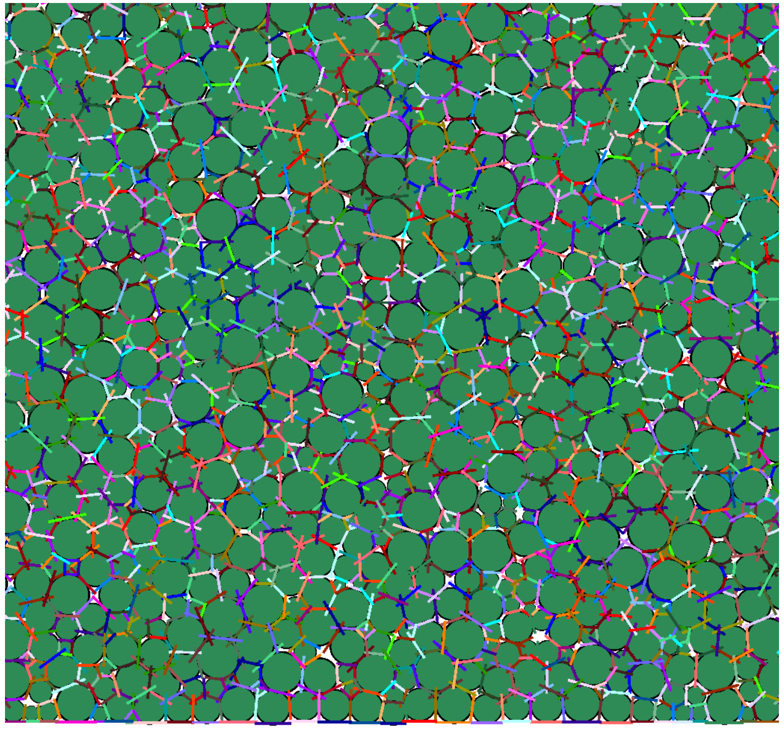
# Ball Packing Stress Installation (2D and 3D)

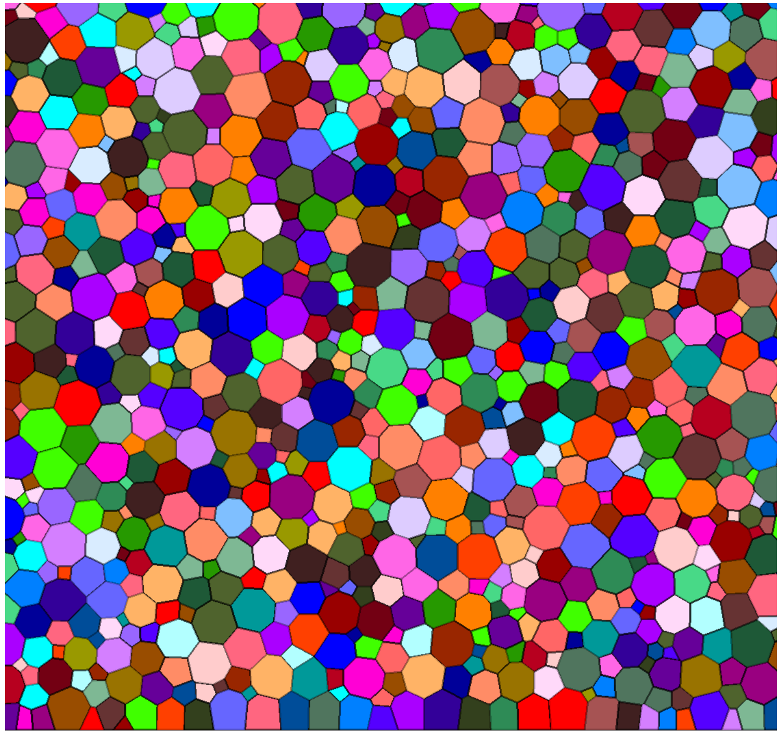
- For ball packings, the standard method of computing the contact areas is not consistent with the enclosed volumes
- This makes it very challenging to install a relatively homogenous stress state due to heterogeneity at the ball scale



# Areas/Volumes Consistent via Voronoi Tessellation

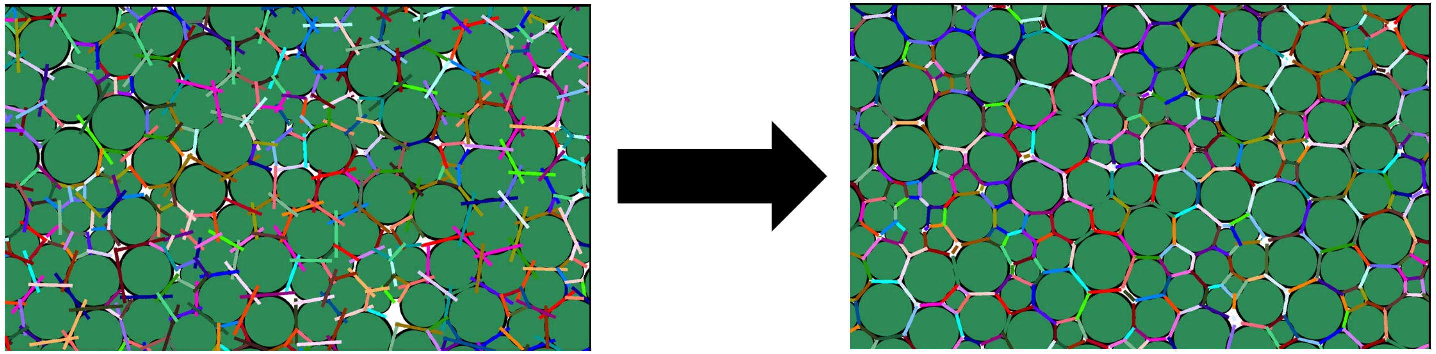
- Compute a weighted Voronoi tessellation of the ball radii/positions
- Update the ball volumes and positions to be consistent with the Voronoi cells

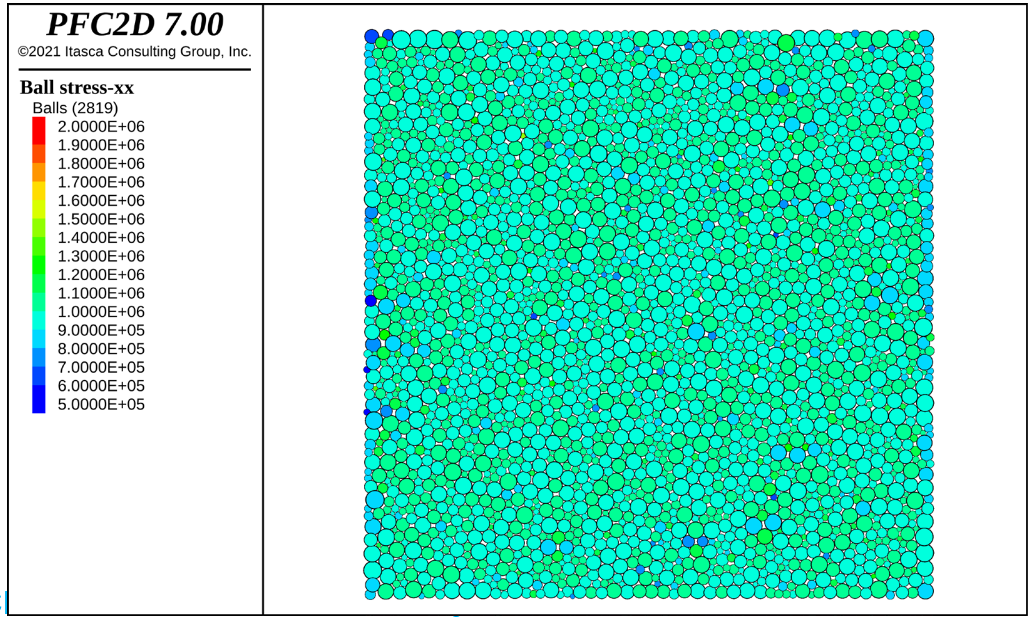


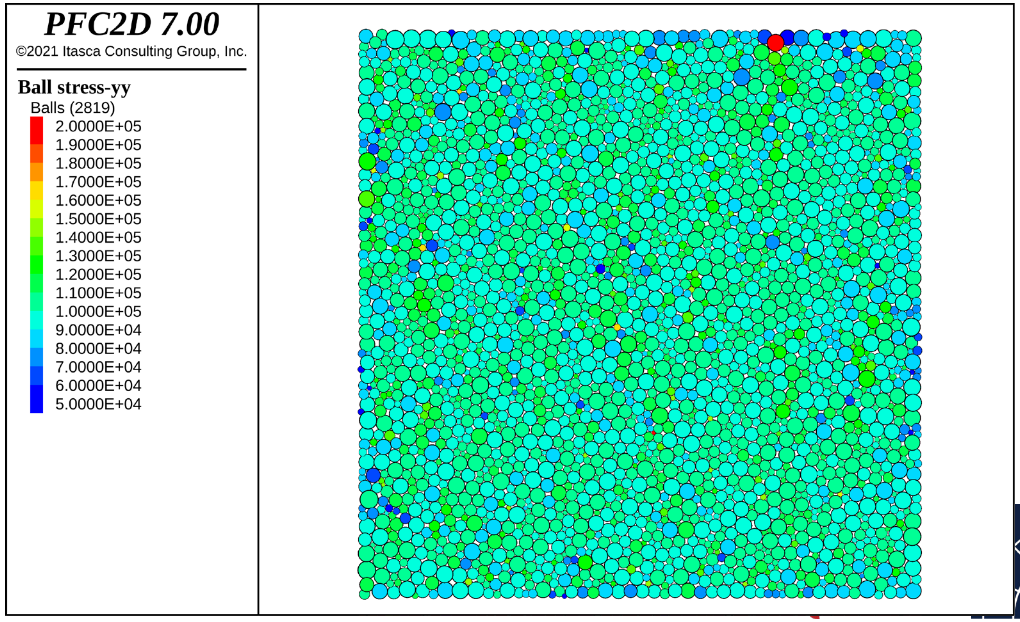


# Contact Areas, Forces, and Stresses

- Create contacts for all Voronoi faces and assign the face areas to the contact areas
- Compute tractions (surface or contact forces) between ball-ball and ball-facet contacts using these updated contacts

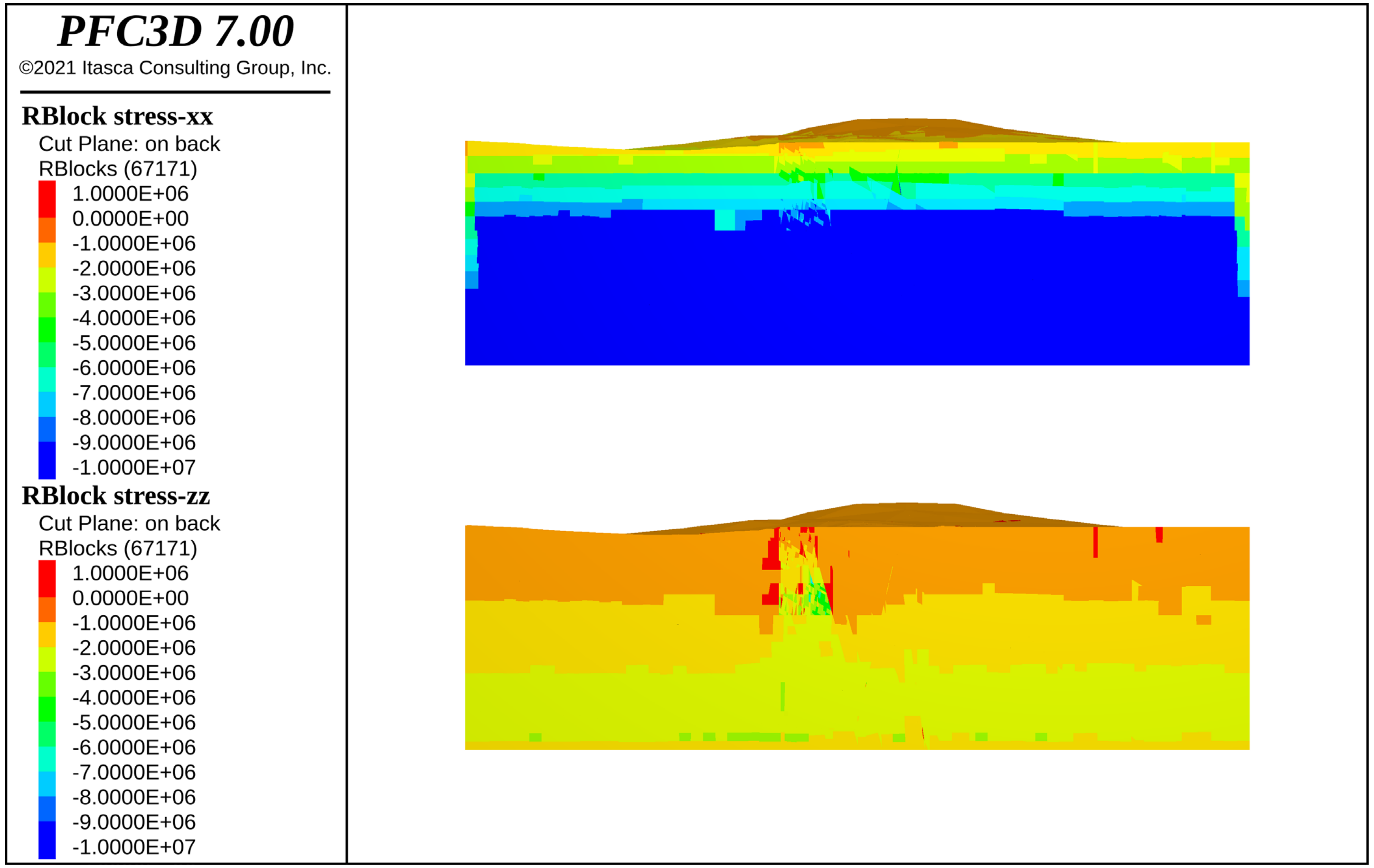




# Ball and Rigid Block Traction Commands

- Modeled on the ZONE INITIALIZE-STRESSES command
- Specify a constant stress state
- Gravitational stress with variable density layers supported
- Anisotropic stress installation
- Overburden



# Feature Enhancements

Generalized Clumping Logic

Rigid Blocks

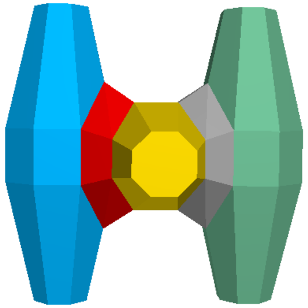
Structural Elements

# Generalized Clumping Logic

- One can clump together balls and\or rigid blocks to make a composite clump template
  - The workflow is to generate balls and\or rigid blocks, create a clump template from these pieces, and replicate the composite clump template

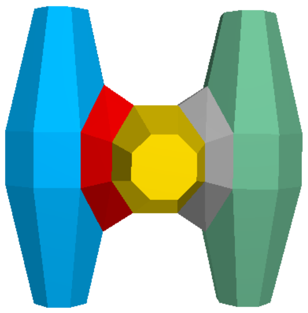
- Balls and\or rigid blocks can be clumped while cycling, retaining exterior contacts, meaning that one can freeze parts of the model to be rigid during cycling without substantial disturbance

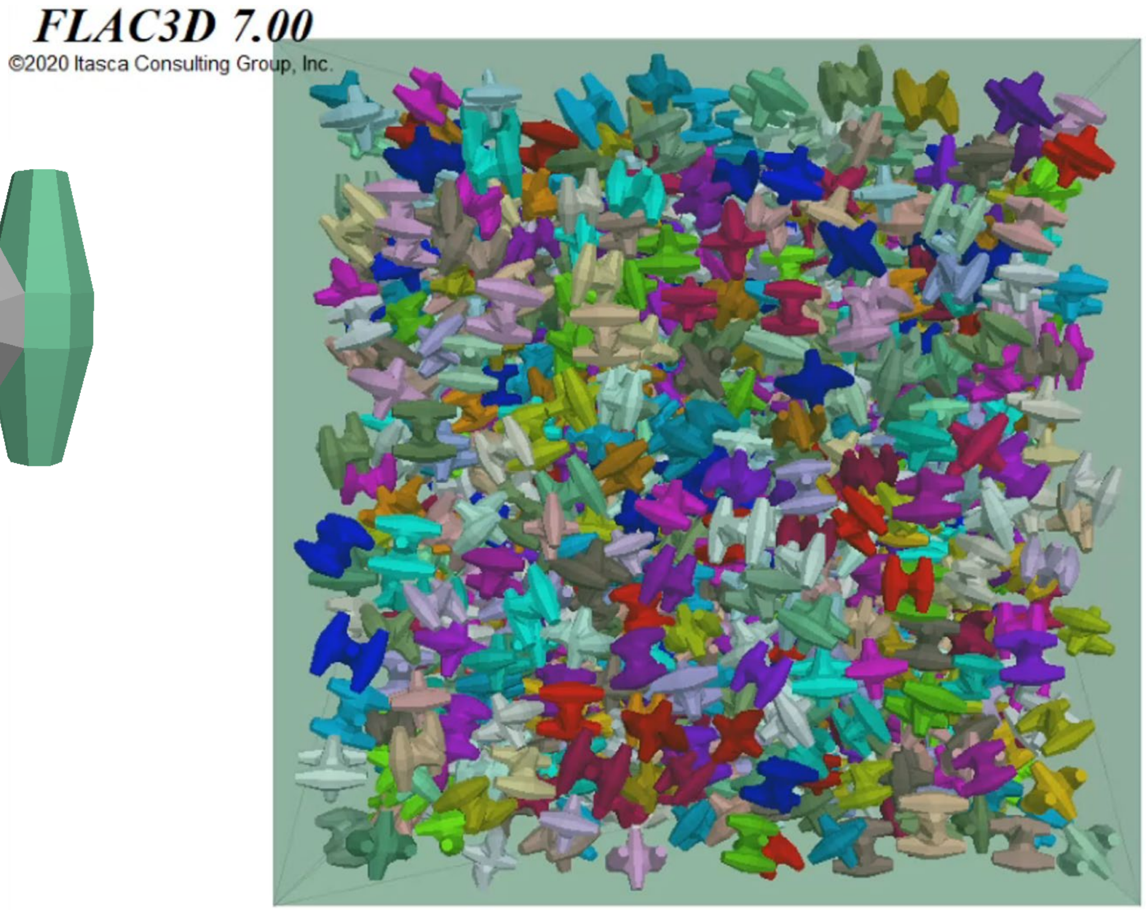
- Pebbles (balls and\or rigid blocks) of composite clumps can later be freed (some or all pebbles) without losing exterior contacts while cycling, allowing for breakage simulations



# Concave Rigid Blocks

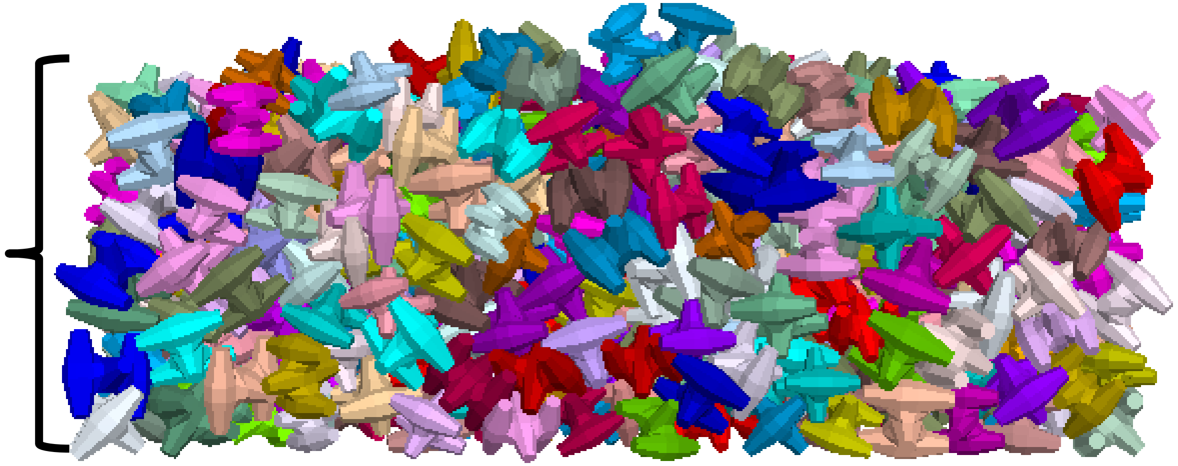
- Dolos made of 5 rigid blocks with some overlap
- About the same computational speed as a 31 pebble clump composed of spherical particles
  - Does not suffer from bumpy surfaces and normal stiffness issues
- Able to free the pebbles without losing the contacts
- Automatically compute the inertia tensor and volume accounting for overlap



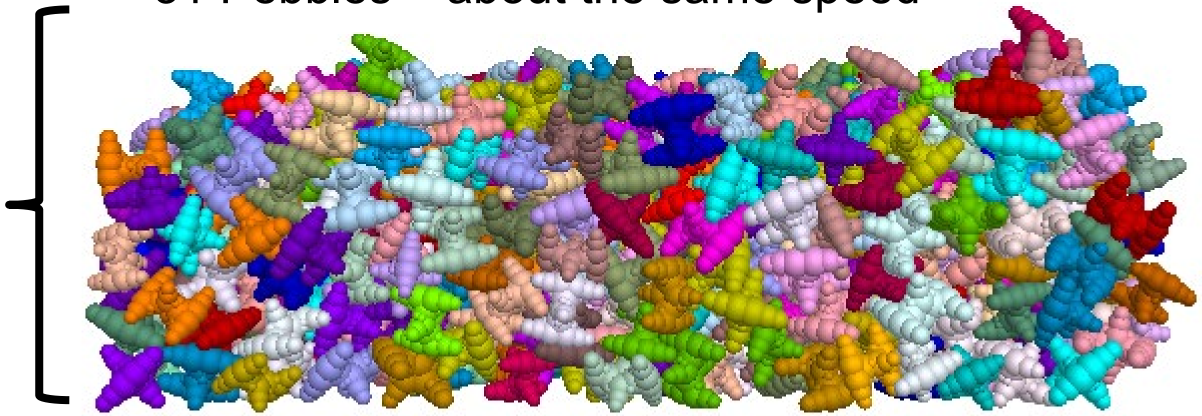


# Concave Rigid Blocks Vs. Traditional Clumps

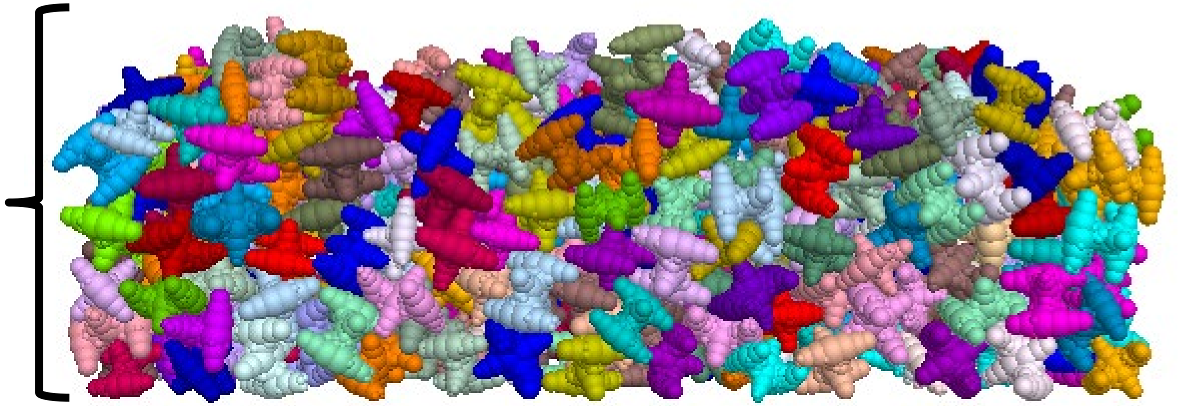
- Can easily see the difference when using spherical clumps with "reasonable" resolution



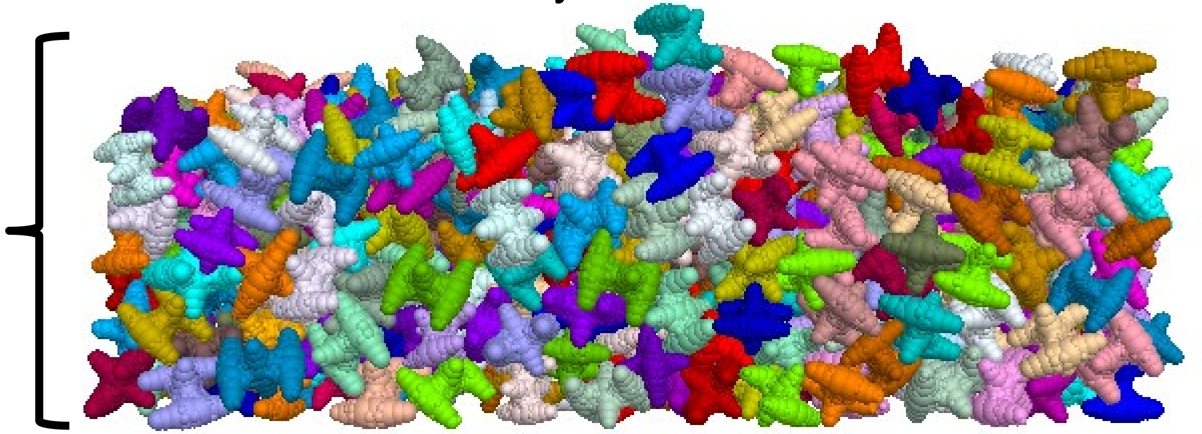
31 Pebbles – about the same speed



47 Pebbles – nearly 2 times slower



91 Pebbles – nearly 4 times slower

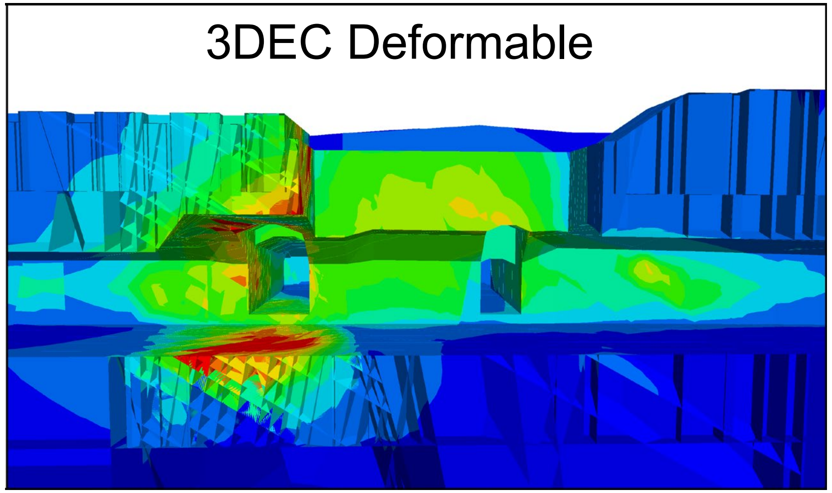


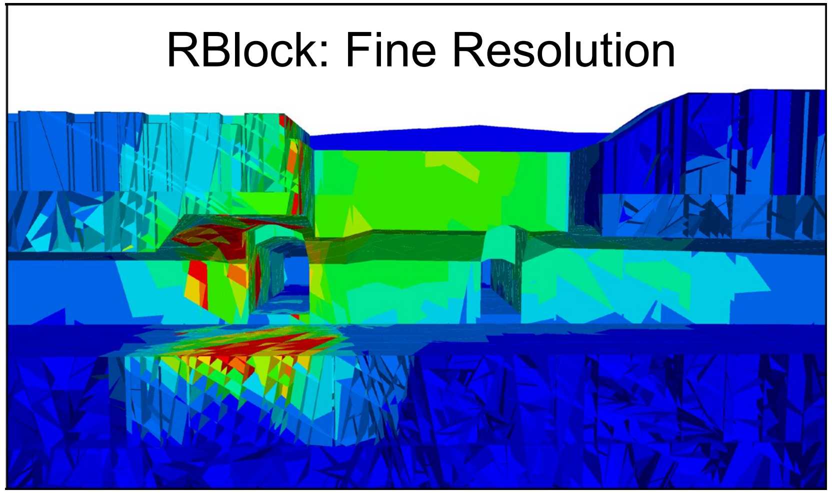
# Rigid Block Enhancements

- Facet groups allow for simple boundary condition application and easy contact assignment
  - Simple apply boundary conditions with rigid blocks
- Cut rigid blocks while cycling, retaining contacts
  - Simulate grain breakage with *FISH* criteria for breakage
- Performance enhancement for unbonded contact resolution
- Stress initialization
- Simple meshing to create rigid block assemblies and densification
  - Triangles / Voronoi cells in 2D
  - Tetrahedra / Hexahedra / Voronoi cells in 3D
  - Densification via cutting

# *3DEC* Model Comparison

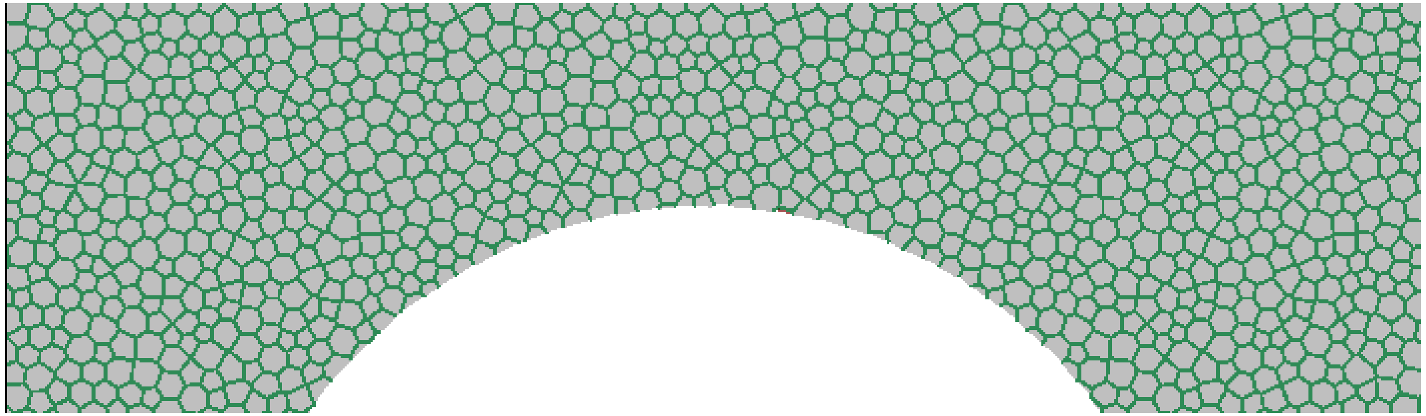
- Can now easily create models like *3DEC* models
  - No elastic calibration since using the SpringNetwork model
  - Assign facet groups to cut blocks for contact property assignment
  - DFN logic now supports joint sets (like *3DEC*)
  - Simple boundary condition assignment via facet groups
- 4+ times faster than similar *3DEC* models using rigid blocks

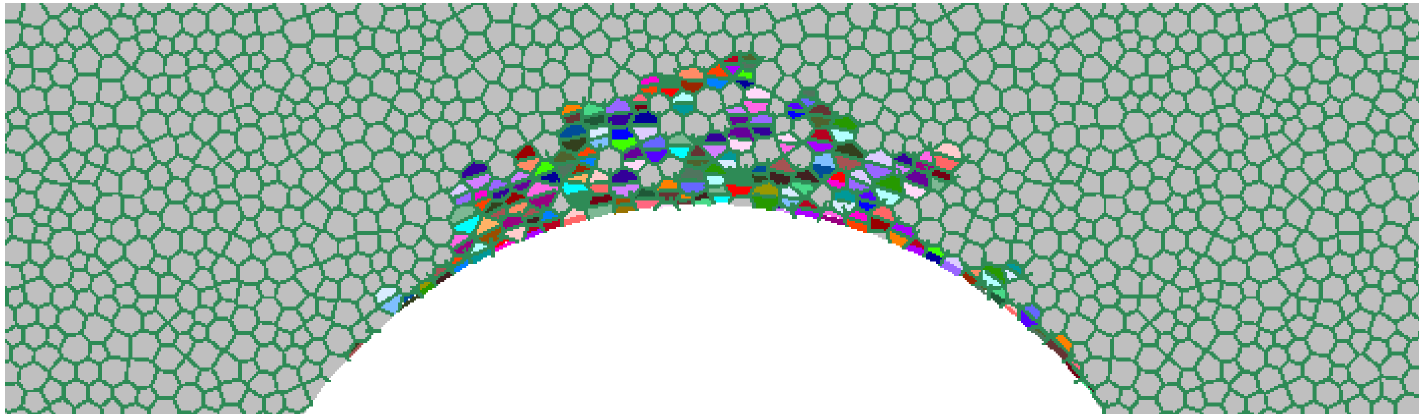


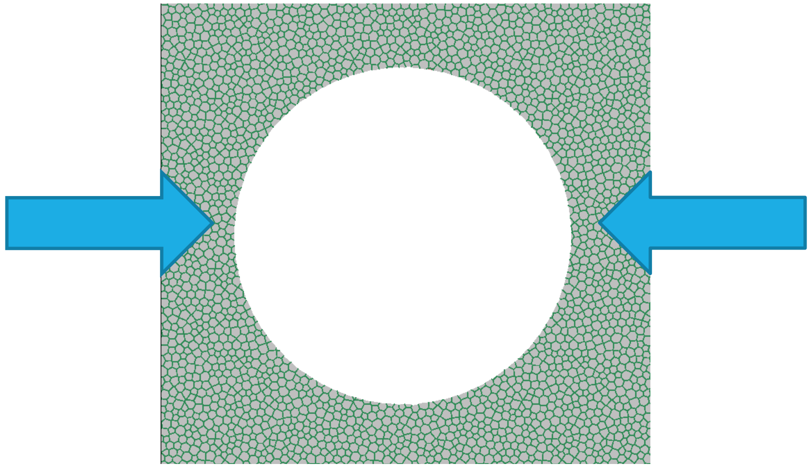



# Grain Breakage via Cutting

- Circular hole in an isotropically stressed material composed of Voronoi shaped rigid blocks
- Squeeze to twice the lateral stress
- Breakage criteria based on the minimum principal stress
- Notch formation due to stress application



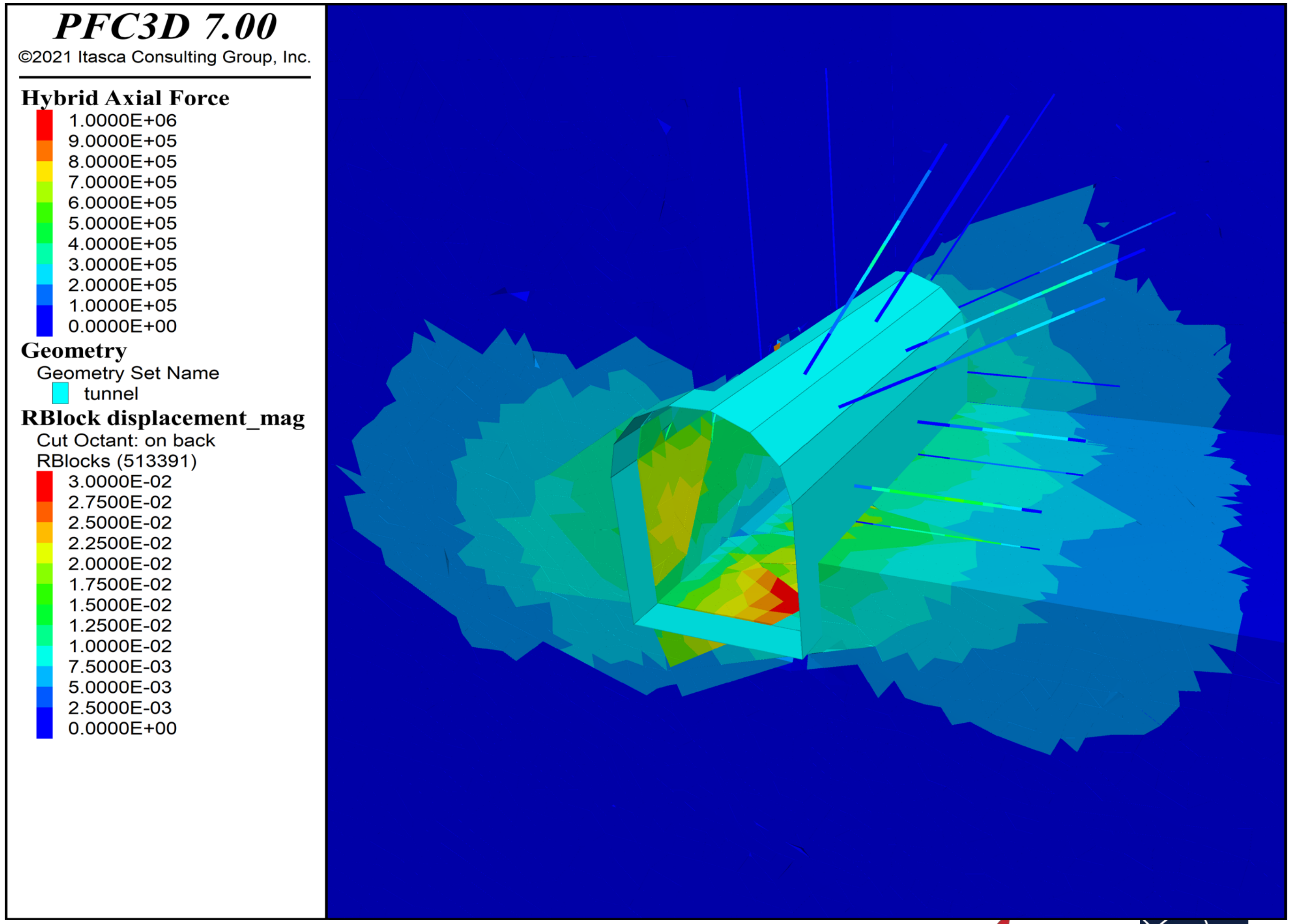




# Structural Element Support

- All *FLAC3D* structural elements can be used with balls/clumps/rigid blocks

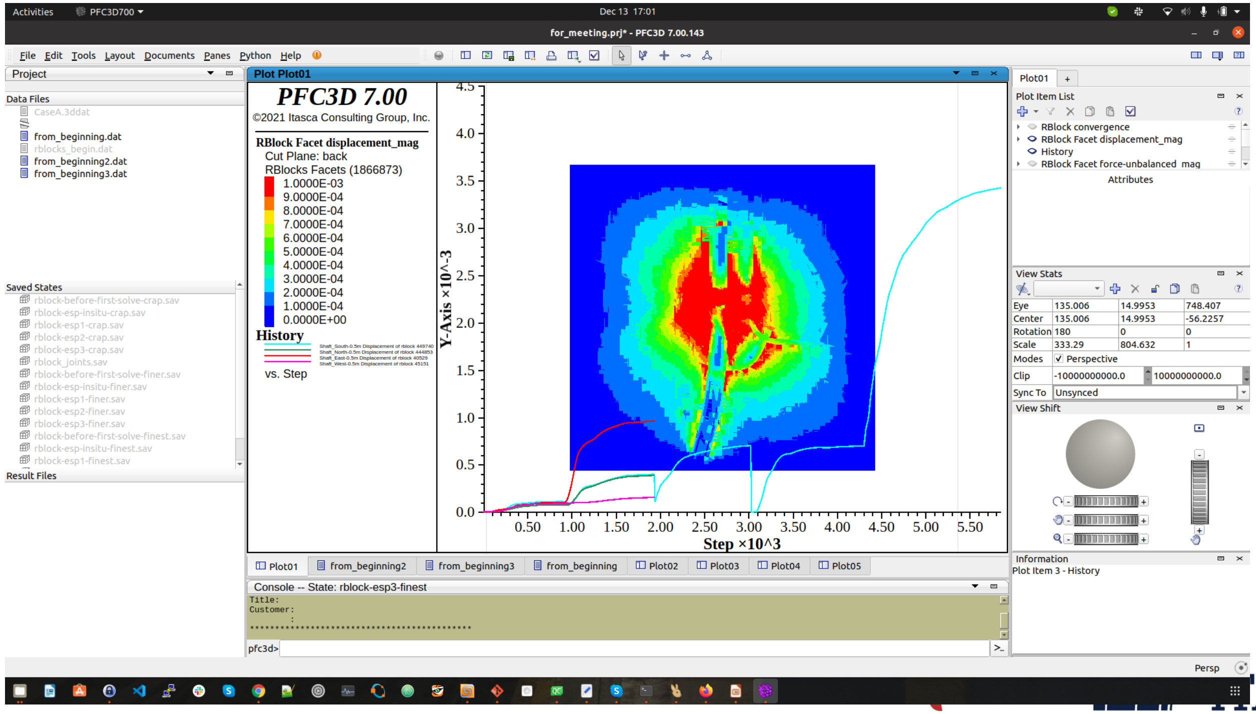
- Liners, shells, geogrids, and 1D elements (beams, cables, hybrid bolts, piles)



# Linux Version

# Linux

- https://www.itascacg.com/software/downloads/itasca-linux-software-7-0-update
- Fully functional GUI and console versions of *PFC 7.0*, *3DEC 7.0* and *FLAC3D 7.0*
- Save files compatible between OS
- Data files compatible between OS (Note: the Linux file system is case sensitive unlike Windows)
- Created for Ubuntu 20.04 LTS
- Singularity containers can be created for other Linux distributions from Ubuntu 20.04 LTS
- Web license required
- Tested on clean AWS instances
- Similar performance to Windows
- C++ contact models far easier to compile
- Documentation
- Examples/Verifications



# Conclusions

- New features open a whole new realm of possibilities:
  - Granular Flows / Manufacturing / Soil Mechanics Applications
  - Bonded-Block Modeling / Rock Mechanics
- Multithreaded *FISH* for faster calculation
- Particle fragmentation abilities (clumping or cutting) and the SpringNetwork approach are very promising
- Next steps:
  - More examples / documentation
  - More features to improve ease of use
  - Cluster/MPI computation