Advanced Training: Recent Features, Output Analysis and Current Best Practices Part 2

Ali Akhavan
CPFD, LLC

2019 Barracuda Virtual Reactor Users Conference
June 27-28, 2019
Chicago, IL

Outline of Topics

Particle Initialization

- Specify particle mass
- Specify particle resolution

Drag Parser

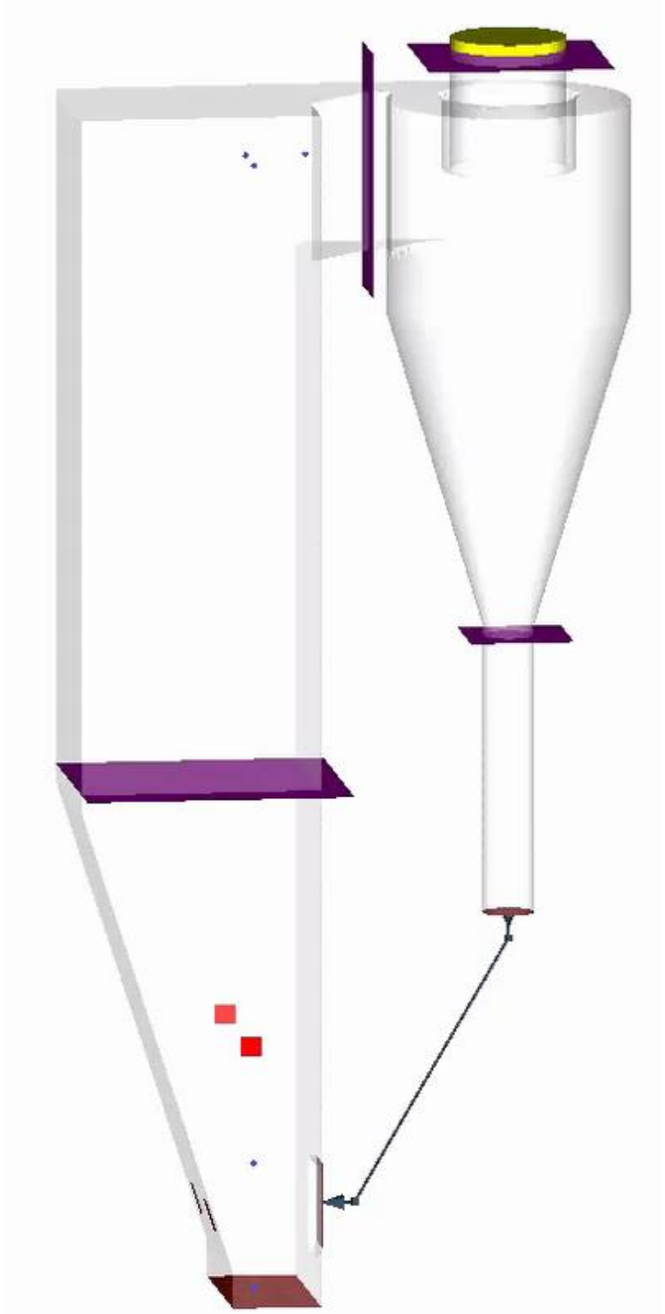
Particle Exit Flow BCs

BC Connectors

- Heating / Cooling Rate

Internal Flux Planes

- Raw particle data
- Reset particle residence time



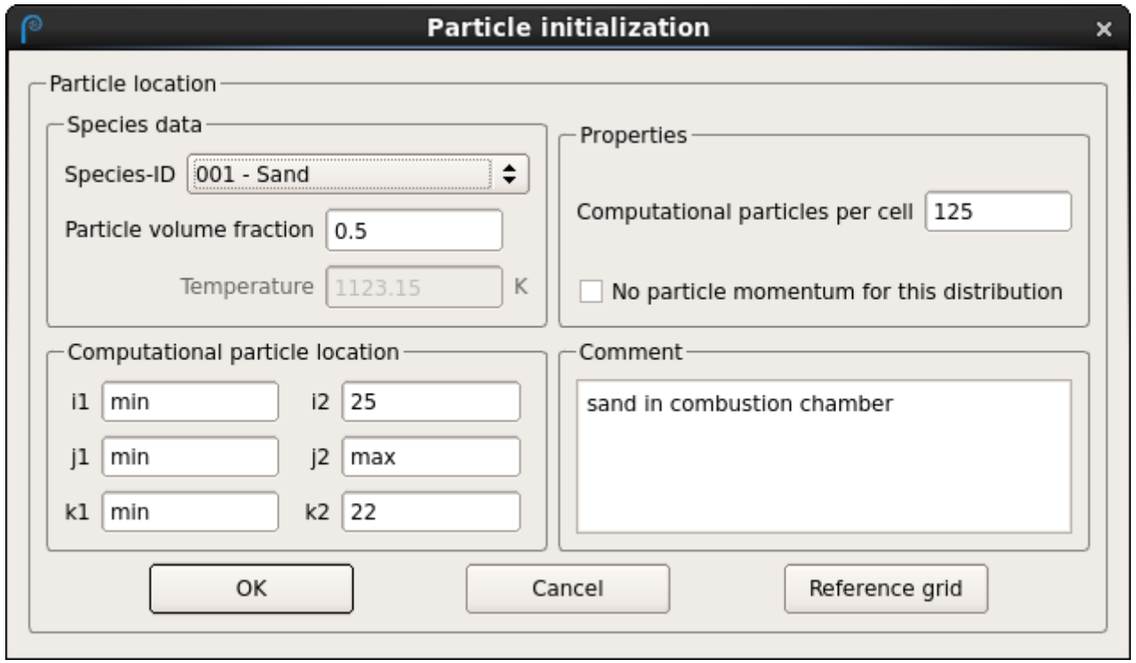
Example based on CFB Combustor from Barracuda training class

Particle Initialization: The Old Way

Volume fraction of particles in an IJK region

Iterative process

- Guess IJK and/or particle volume fraction
- Run solver 1 time-step
- Check history.log for particle mass
- Adjust IJK and/or particle volume fraction
- Run solver 1 time-step
- Check history.log for particle mass
- …



Particle Initialization: Specify Particle Mass Directly

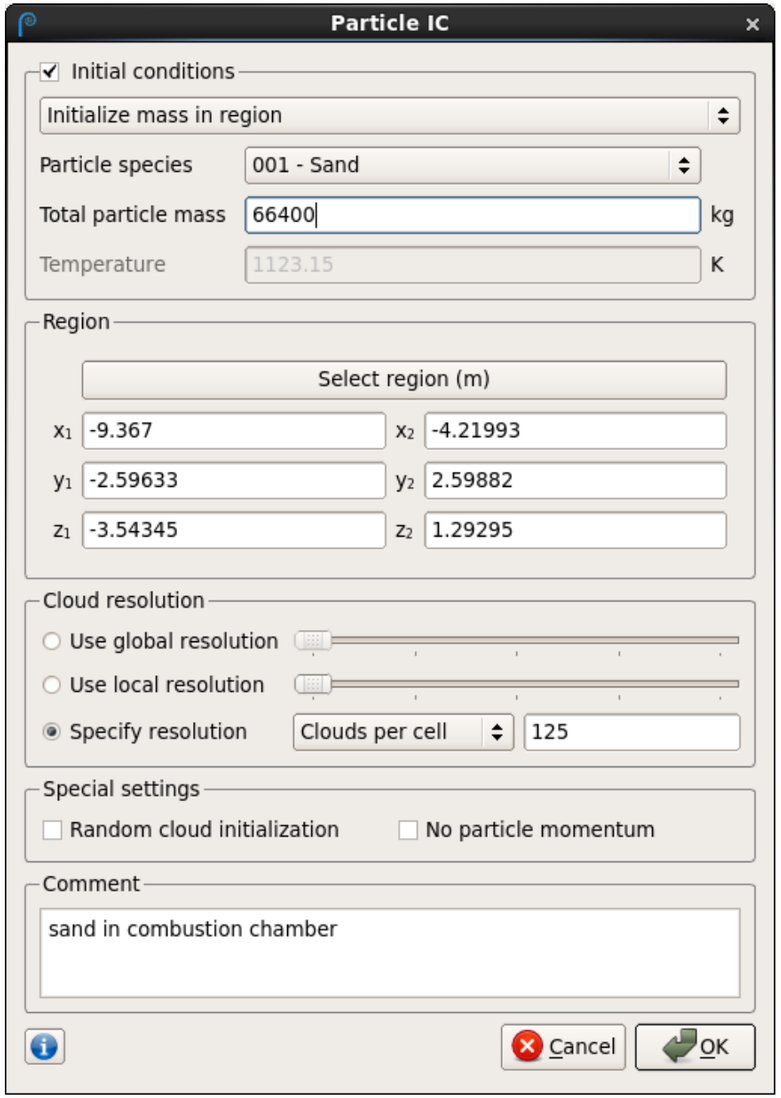
Added in release 17.3.0

In this example:

- Main combustion chamber: 66,400 kg

Specify particle mass:

- Choose “Initialize mass in region”
- Set the particle species
- Specify the desired mass (kg)
- Select the spatial region

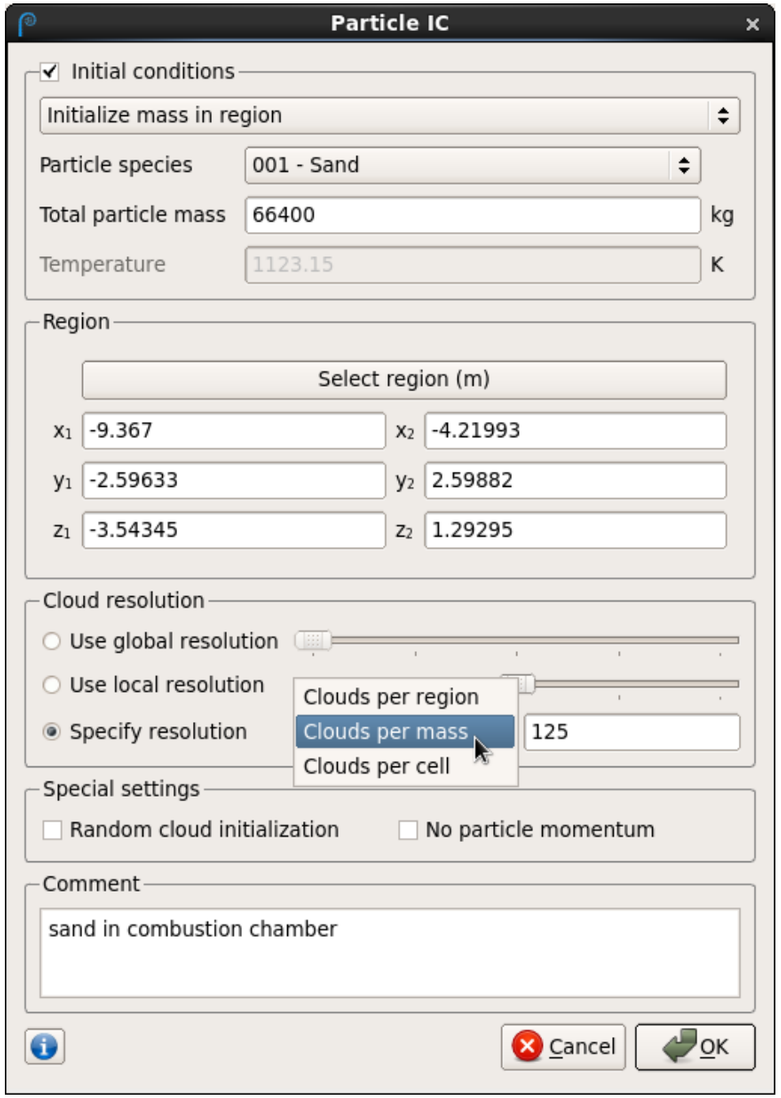


Particle Initialization: Specify Particle Resolution

Added in release 17.3.0

New options available:

- "Use local resolution": slider-bar functionality on a per-IC basis
- "Specify resolution"
  - "Clouds per cell": same as pre-17.3.0 "Manual input" value
  - "Clouds per region": specify directly the number of clouds (a.k.a. computational particles) to use for the IC
  - "Clouds per mass": specify the number of clouds to use per kg of particles in the IC



Initial Particle Resolution Example

In the CFB Erosion training problem:

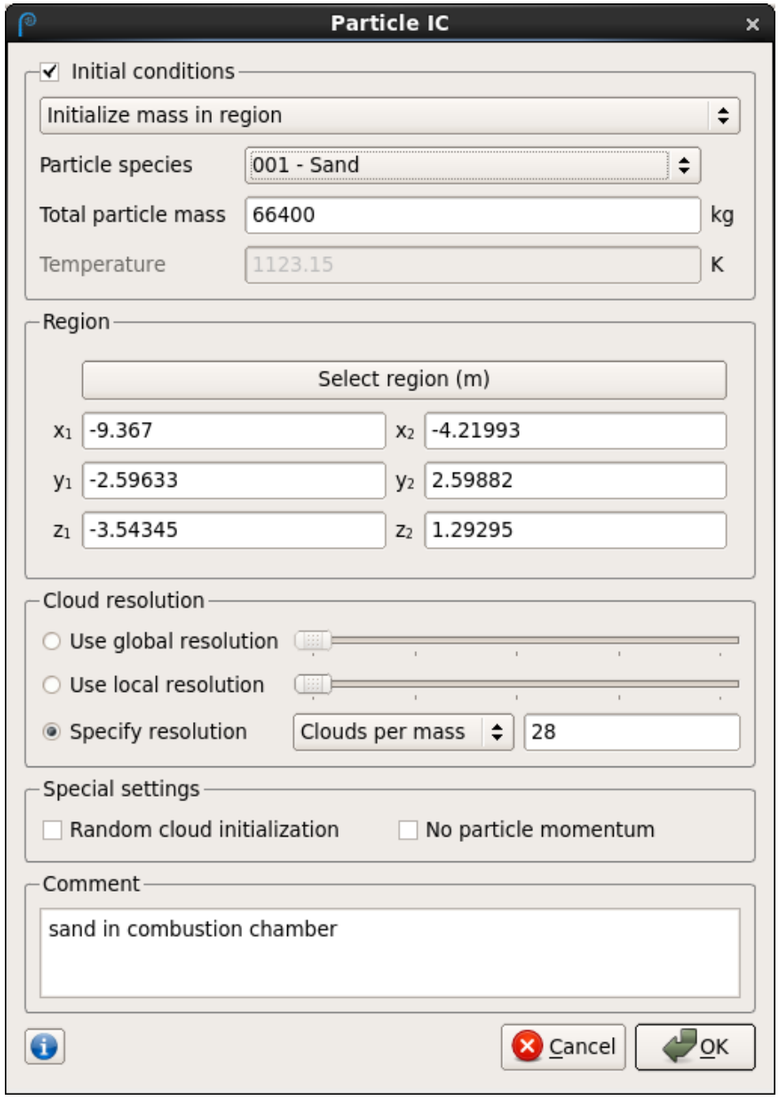
- Number of real cells = 105,061
- Initial particle mass = 71,100 kg

How much resolution do we want/need?

- 10 particles/cell → 1e+06 clouds
- **20 particles/cell → 2e+06 clouds**
- 30 particles/cell → 3e+06 clouds

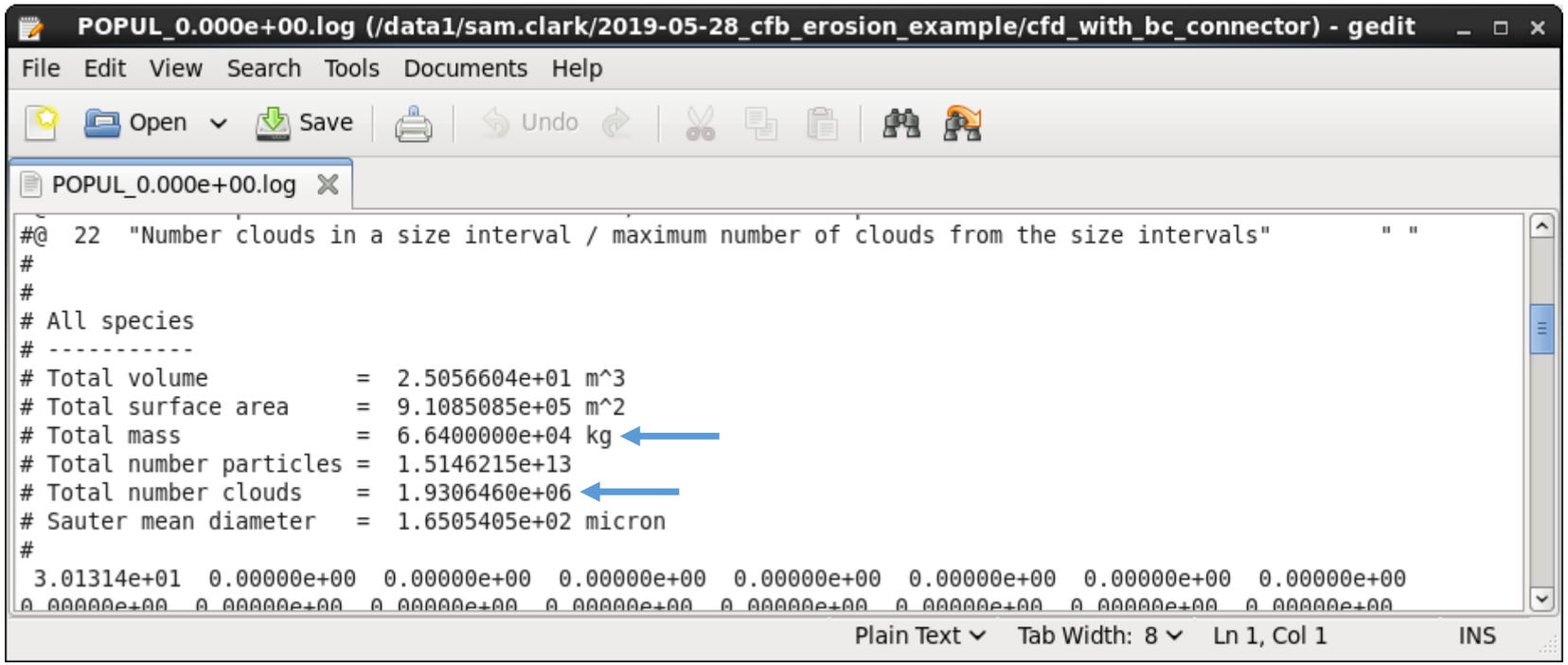
If we want 20 particles/cell:

- 2e+06 clouds / 71,100 kg = 28 clouds/kg



Confirming Particle Initialization Statistics

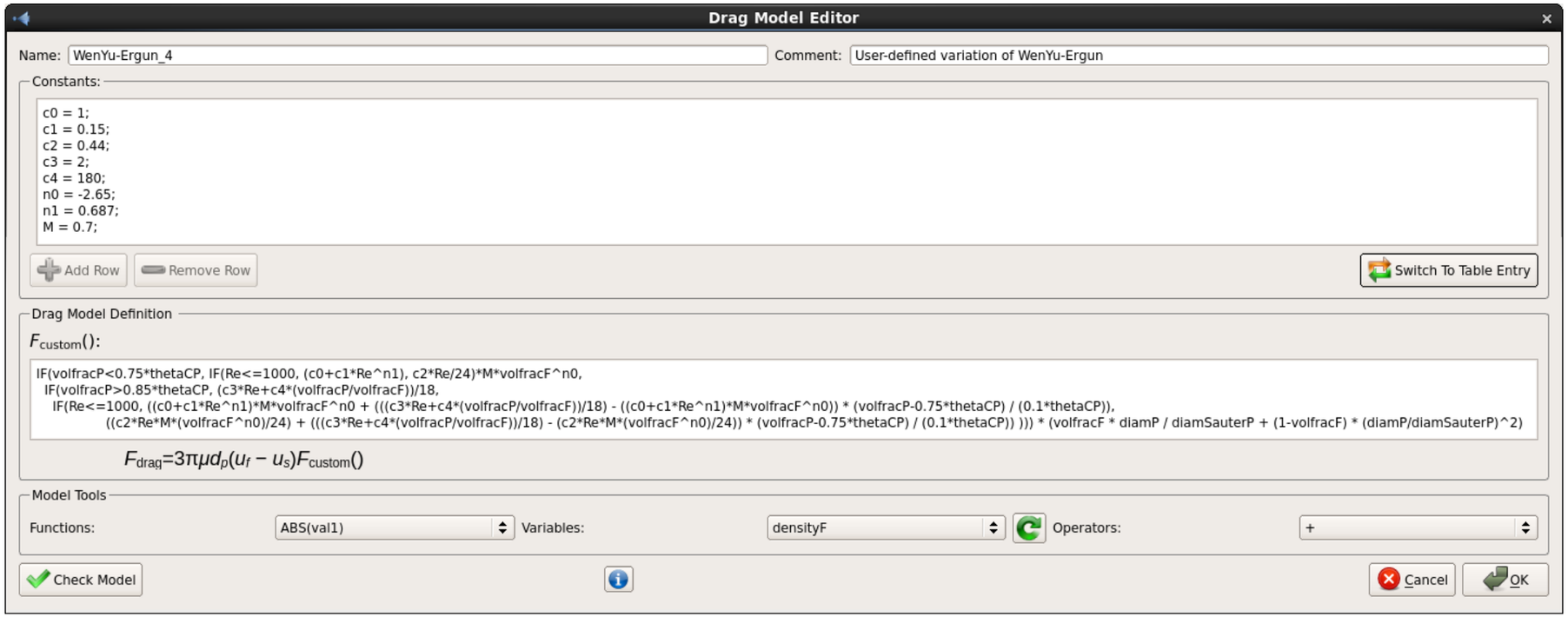
One way to confirm the initial particle mass and resolution is to look at the header section of `POPUL_0.000e+00.log`



User-Defined Drag: The Old Way

In pre-17.4.0 releases of Barracuda:

- The Drag Model Editor did not support subexpressions
- User-defined drag models often became very long and hard to understand

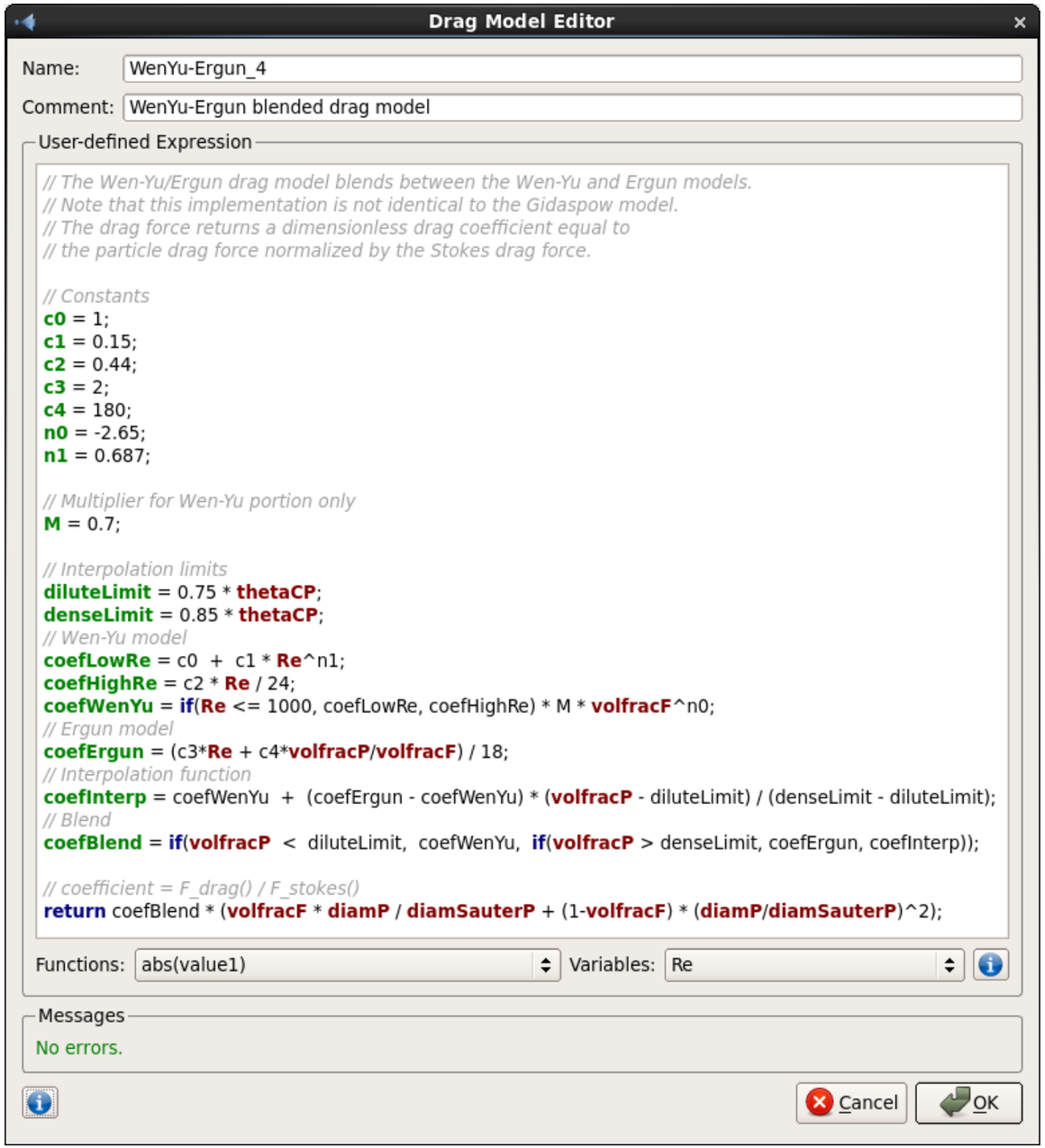


Drag Parser Improvements

Added in release 17.4.0

Drag models are much easier to define (and read) because of:

- Subexpressions
- Syntax highlighting
- Support for comments



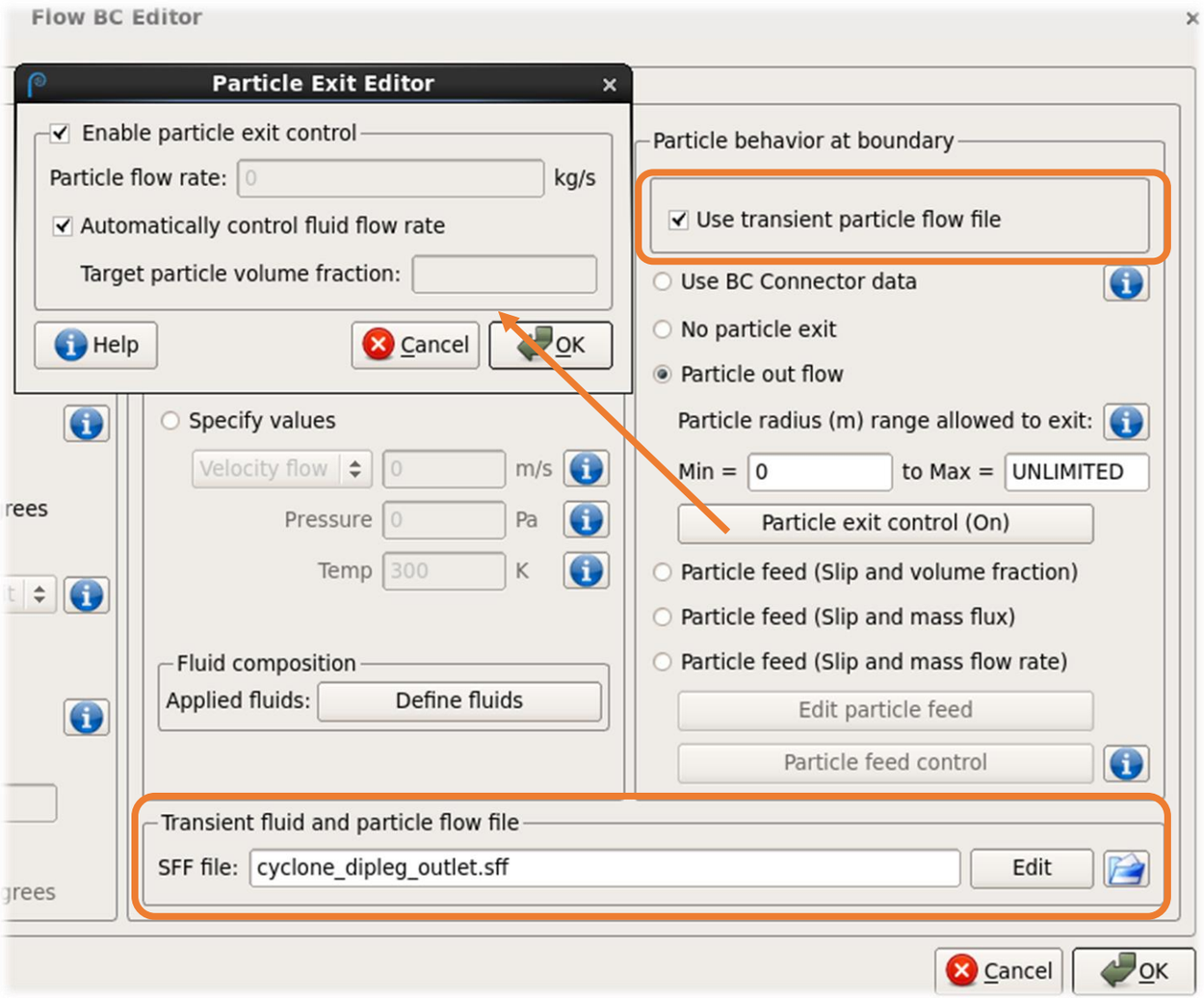
Particle Exit Flow BC

Allow up to a specified flow rate of particles to exit at a Flow BC

- Easier to control than a Pressure BC
- Specified value is a ***maximum***, and particles will exit at the BC only if they are near to and traveling toward the BC face

Specify a negative particle flow rate

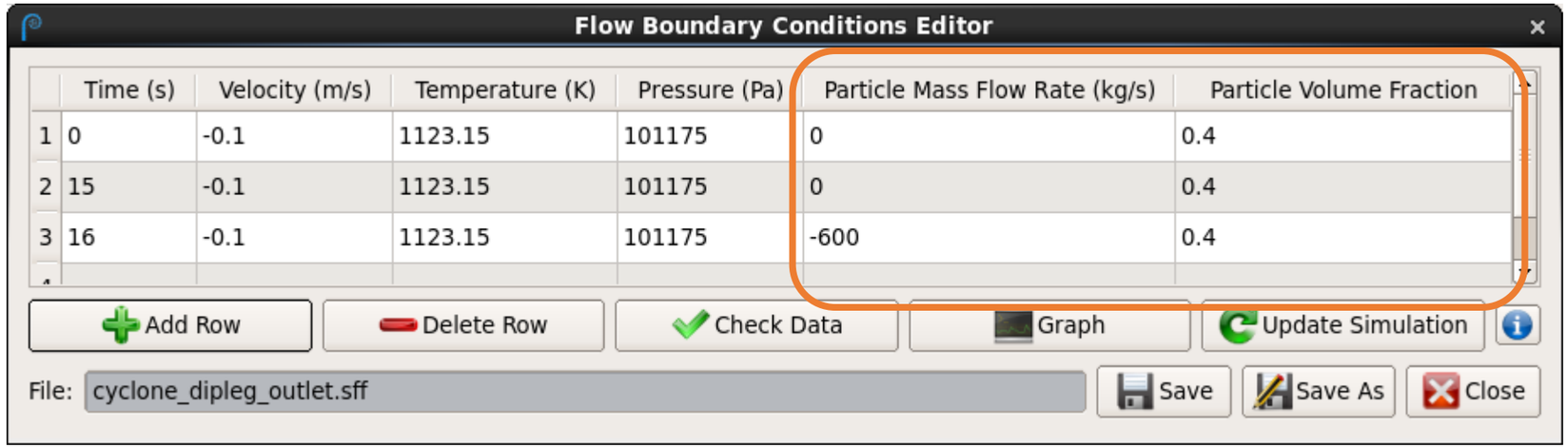
Always use a transient flow file!



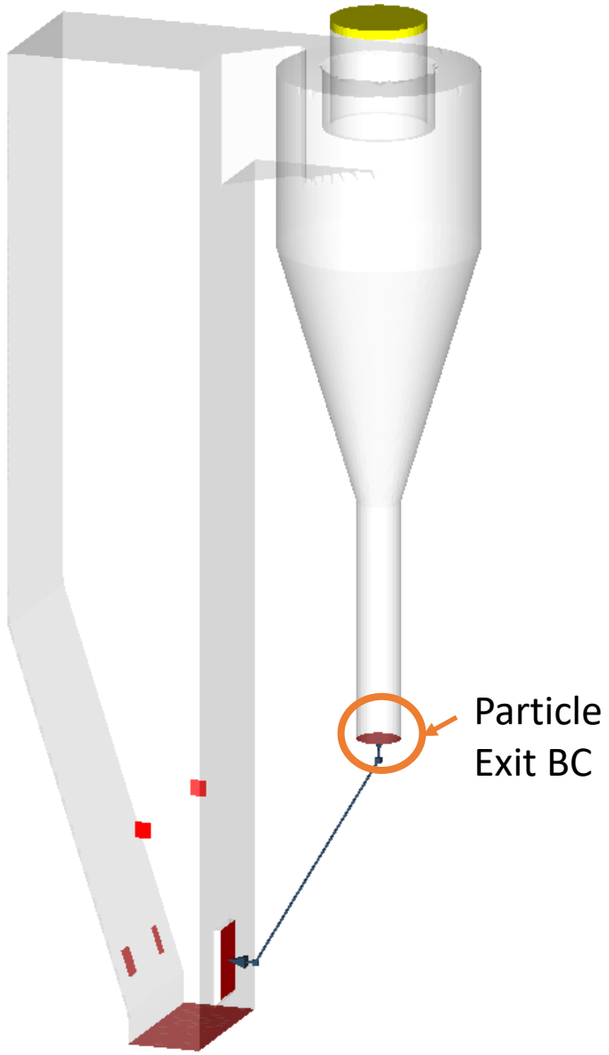
Using Particle Exit BC at Cyclone Dipleg Exit

Particle Exit BCs are useful for a number of situations:

- Standpipes in large vessels
- Bed drains
- Cyclone dipleg exits



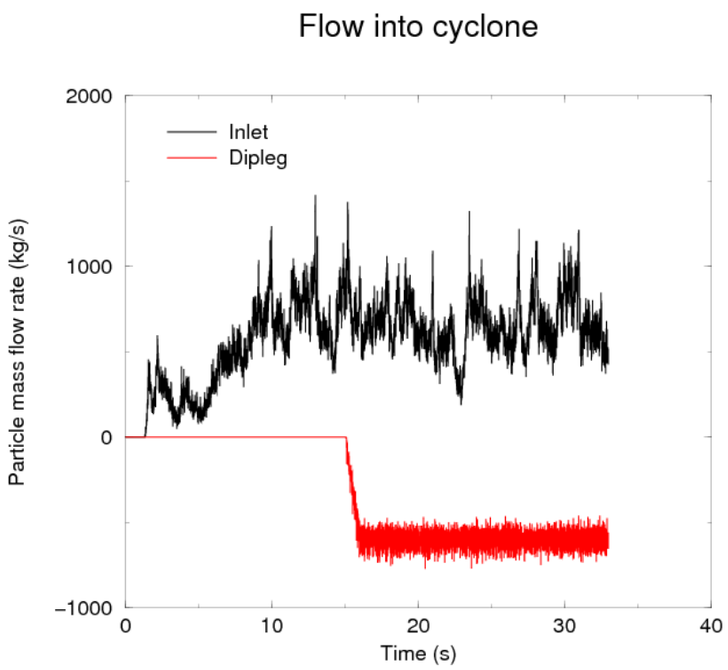
	Time (s)	Velocity (m/s)	Temperature (K)	Pressure (Pa)	Particle Mass Flow Rate (kg/s)	Particle Volume Fraction
1	0	-0.1	1123.15	101175	0	0.4
2	15	-0.1	1123.15	101175	0	0.4
3	16	-0.1	1123.15	101175	-600	0.4

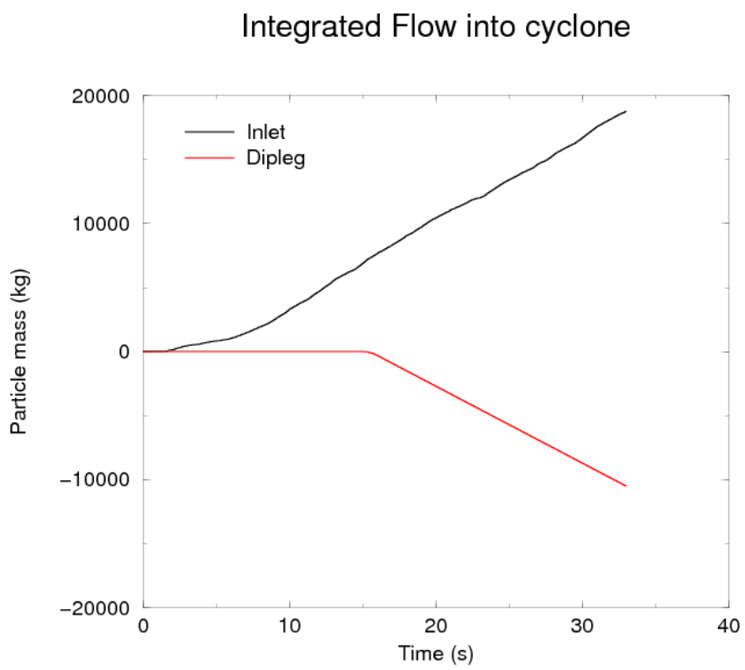


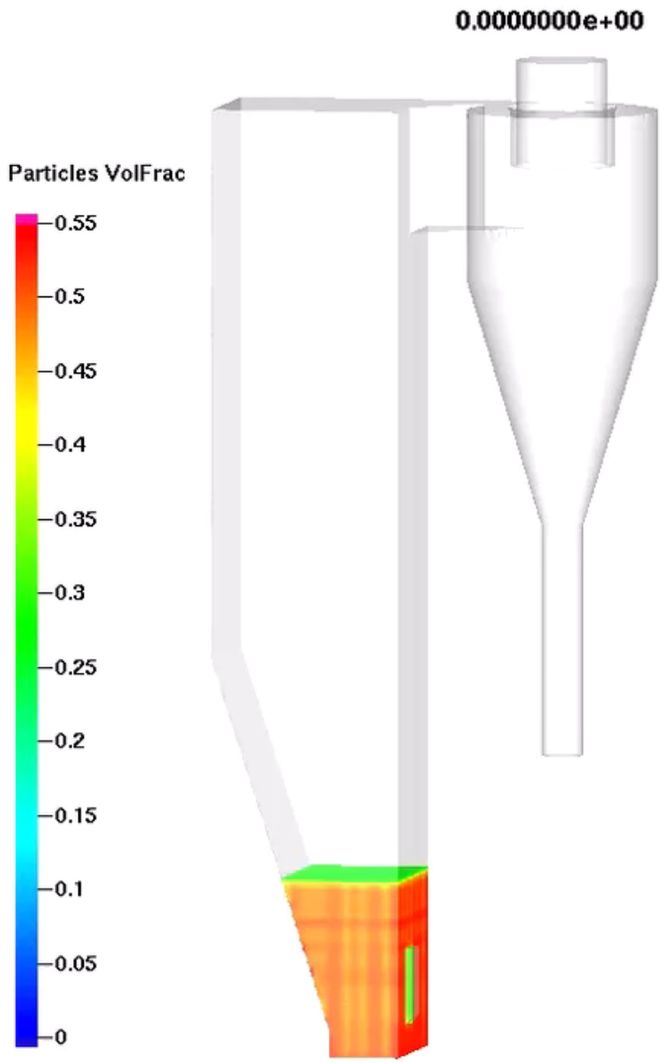
Particle Exit BC Post-Processing and Analysis

t = 0 to 15 s: no particle exit at cyclone dipleg

t = 16+ s: particles exit at 600 kg/s







BC Connectors: Heating / Cooling Rate

Added in release 17.2.0

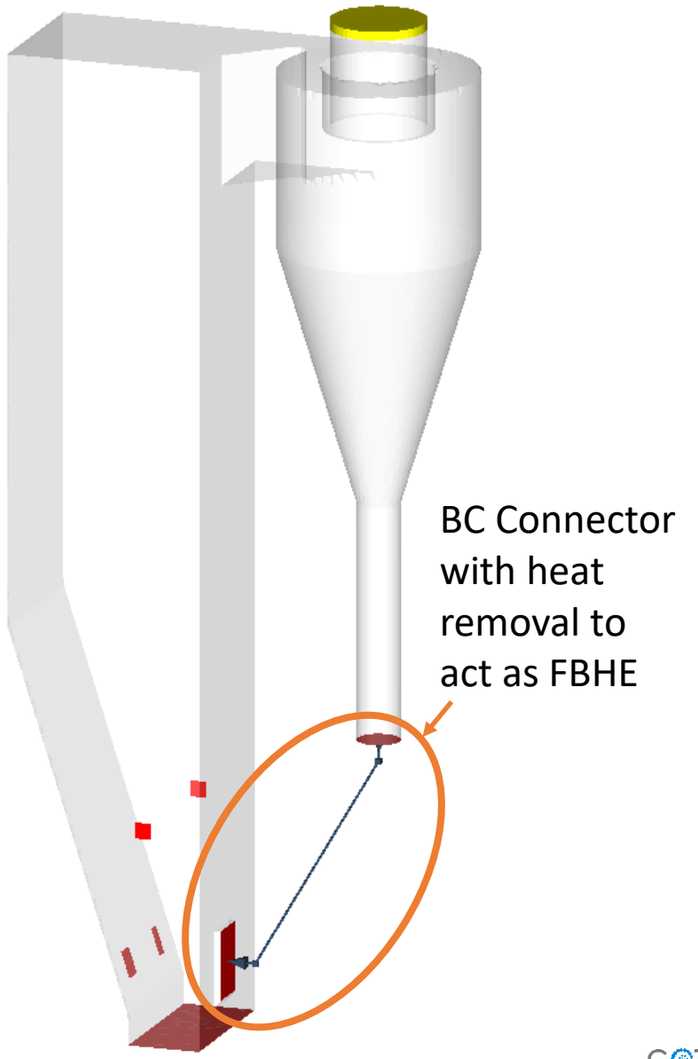
BC Connectors can add or remove heat from gas and particles

Industrial applications:

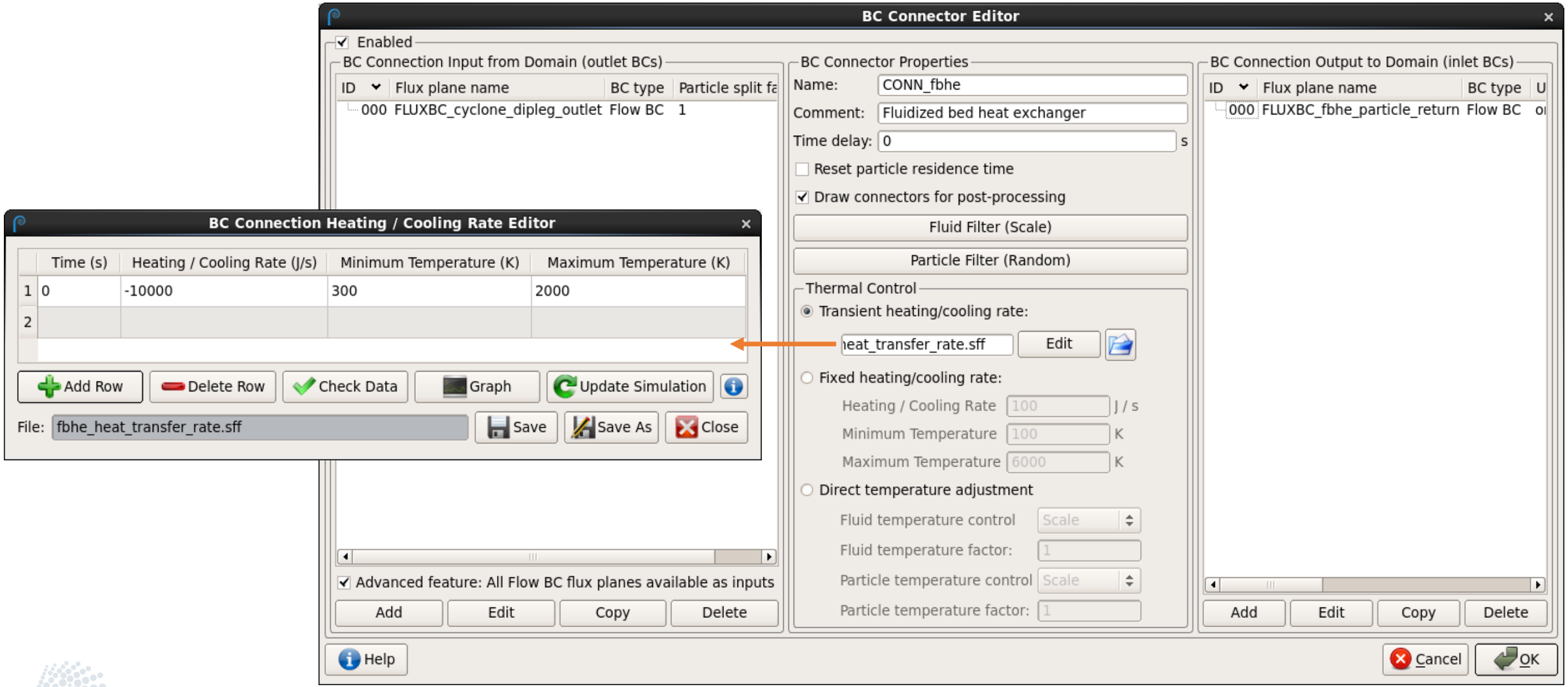
- FCCU Catalyst Coolers
- Fluidized Bed Heat Exchangers (FBHEs)

Sign convention:

- Positive = heat added to gas and particles
- Negative = heat removed from gas and particles



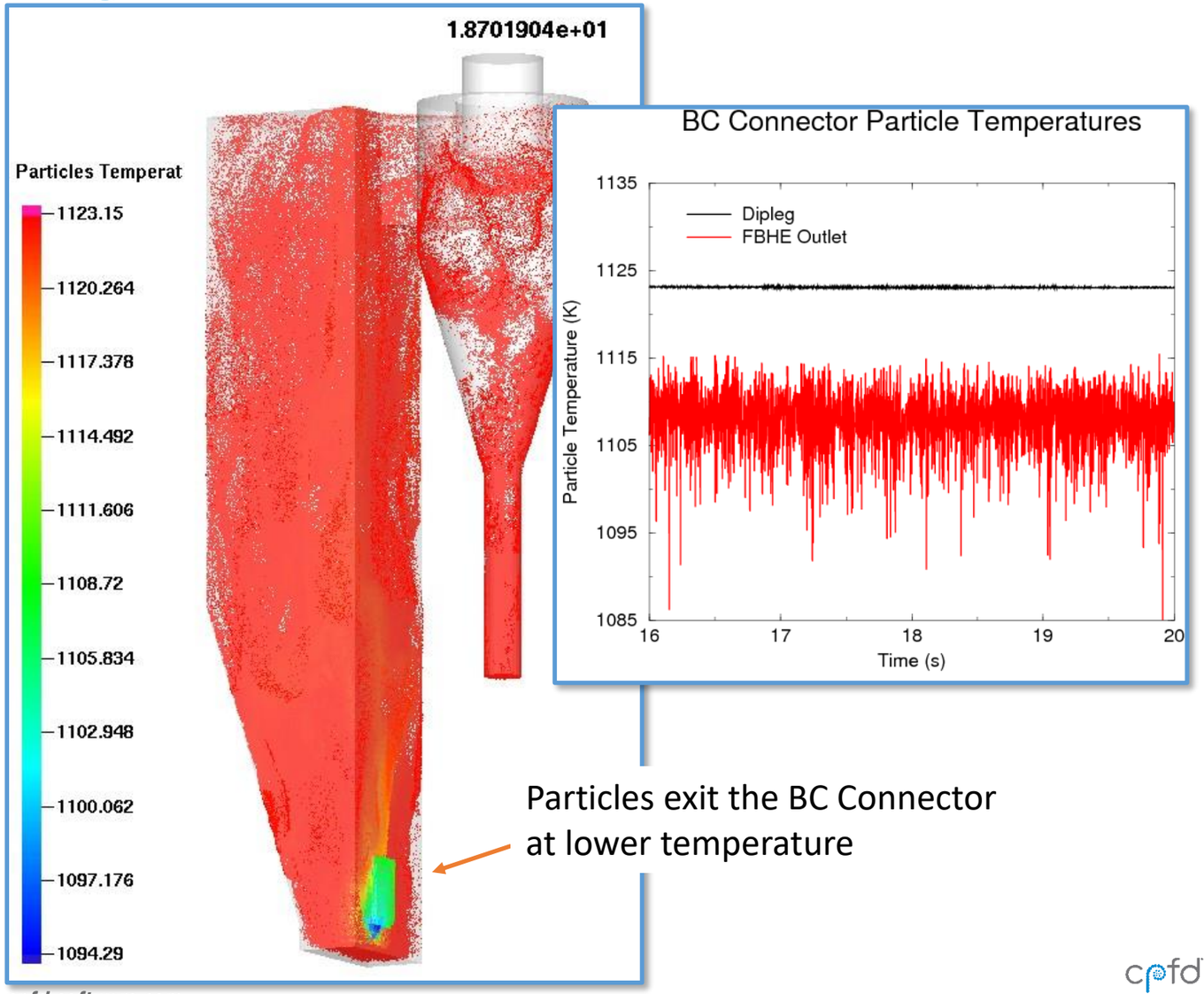
Specifying BC Connection Heating / Cooling Rate



Heat Removal Post-Processing and Analysis

Heat addition or removal from the BC Connector can be analyzed in several ways

- GMV can be used to visually confirm temperature increases or decreases
- Raw particle data from the BC flux planes contains temperature data for every particle passing through



Particles exit the BC Connector at lower temperature

New Features for Internal Flux Planes

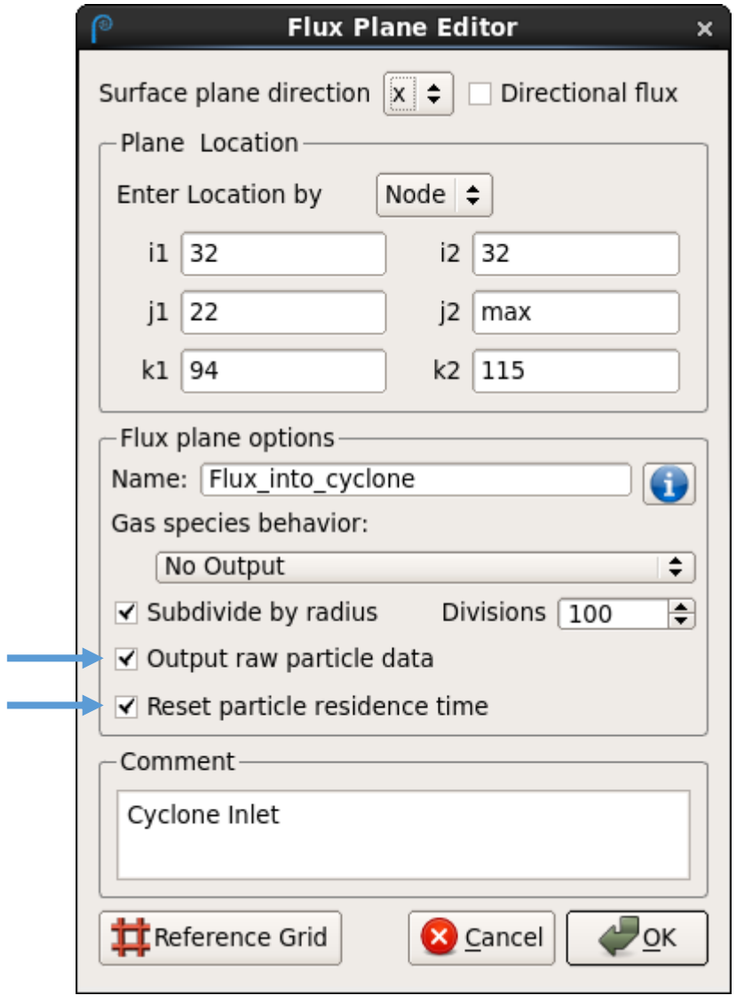
Added in release 17.4.0

Output raw particle data

- Allows for very detailed analysis of particles crossing flux plane
- Previously available only for BC flux planes

Reset particle residence time

- Useful for tracking time for particles to travel or circulate through a system



Options for Calculating PSD at Internal Flux Planes

"Subdivide by Radius"

- Can analyze data using Flux2PSD.tcl
- Must choose number of uniform bins at beginning of simulation
- Data is written every time-step, whether particles pass through flux plane or not – can result in large flux plane files

"Output raw particle data"

- Cannot analyze data using Flux2PSD.tcl
- Can choose any number of uniform or non-uniform bins when analyzing data
- Data is written for each particle that passes through the flux plane – can result in flux plane files that are either smaller or larger than "Subdivide by Radius", depending on how many particles pass through the flux plane.

Calculating PSD of Sand Particles using Python

In [1]:
```
%matplotlib inline
from __future__ import print_function
from __future__ division
import matplotlib.pyplot as plt
import numpy as np
```

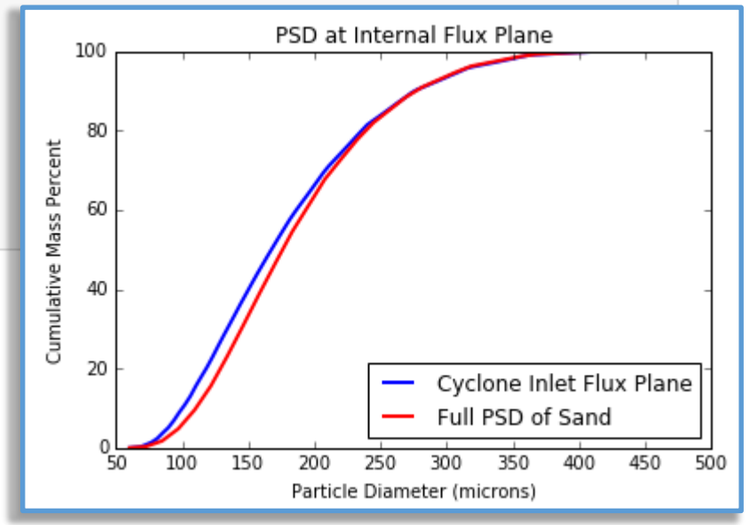
In [2]:
```
# Function to calculate PSD from Barracuda raw particle data
#
# Inputs:
#    massArray = numpy array containing particle cloud mass data
#    diameterArray = numpy array containing particle diameter data
#    numBins (optional) = integer number of bins to use for PSD calculation (default = 100)
#
# Outputs:
#    cumulativeMassPercent = numpy array containing cumulative mass percent
#    diameterEdges = edges of diameter bins, ranging from min to max diameter
#
def calculatePSD(massArray, diameterArray, numBins=100):
    massFractions, diameterEdges = np.histogram(diameterArray, bins=numBins, weights=massArray)
    massCumulative = np.cumsum(massFractions)
    massCumulative = np.insert(massCumulative, 0, 0)
    maxMass = np.amax(massCumulative)
    cumulativeMassPercent = massCumulative * 100 / maxMass

    return cumulativeMassPercent, diameterEdges
```

Read Data from the Internal Flux Plane Raw Data File

In [3]:
```
# Raw particle data at flux plane

f = 'Flux_into_cyclone_raw_particle'
#@   1  "Time"                                                    "s"
#@   2  "Unique particle ID"                                       ""
#@   3  "Particle cloud mass"                                      "kg"
#@   4  "Particle radius"                                          "micron"
#@   5  "Particle density"                                         "kg/m^3"
#@   6  "Number of particles in cloud"                             ""
#@   7  "Residence time (on)"                                      "s"
#@   8  "Species"                                                  ""
#@   ... and more lines ...

tCol = 1
pidCol = 2
massCol = 3
radiusCol = 4
resTimeCol = 7
speciesCol = 8

# Read data into arrays using the genfromtxt() function
t, pid, mass, radius, resTime, species = np.genfromtxt(f,
                                                       usecols=(tCol-1,
                                                                pidCol-1,
                                                                massCol-1,
                                                                radiusCol-1,
                                                                resTimeCol-1,
                                                                speciesCol-1),
                                                       unpack=True, skip_footer=1)

# Convert radius to diameter
diameter = radius * 2
```

Use the PSD Function and Create a Plot

```
In [4]: # Calculate and plot the PSD of sand particles passing through the internal flux plane
        # Sand is species 1 in this simulation

        # Initialize the figure and axes
        fig, ax = plt.subplots()

        # Analyze the data for species 1 from the raw particle file
        myFilter = (species == 1)
        psdMass, psdDiameter = calculatePSD(mass[myFilter], diameter[myFilter])
        ax.plot(psdDiameter, psdMass, linewidth=2, label='Cyclone Inlet Flux Plane')

        # For comparison, let's also plot the PSD from the initial sand
        radius, cumulativeMassFraction = np.genfromtxt('POPUL_01_0.000e+00.log', usecols=(0,1), unpack=True)
        ax.plot(radius * 2, cumulativeMassFraction * 100, color='red', linewidth=2, label='Full PSD of Sand')

        ax.set_title('PSD at Internal Flux Plane')
        ax.set_xlabel('Particle Diameter (microns)')
        ax.set_ylabel('Cumulative Mass Percent')
        ax.legend(loc='best')

        p = 'psd_of_sand_at_internal_flux_plane'
        fig.savefig(p + '.png', format='png')
        fig.savefig(p + '.pdf', format='pdf')
```



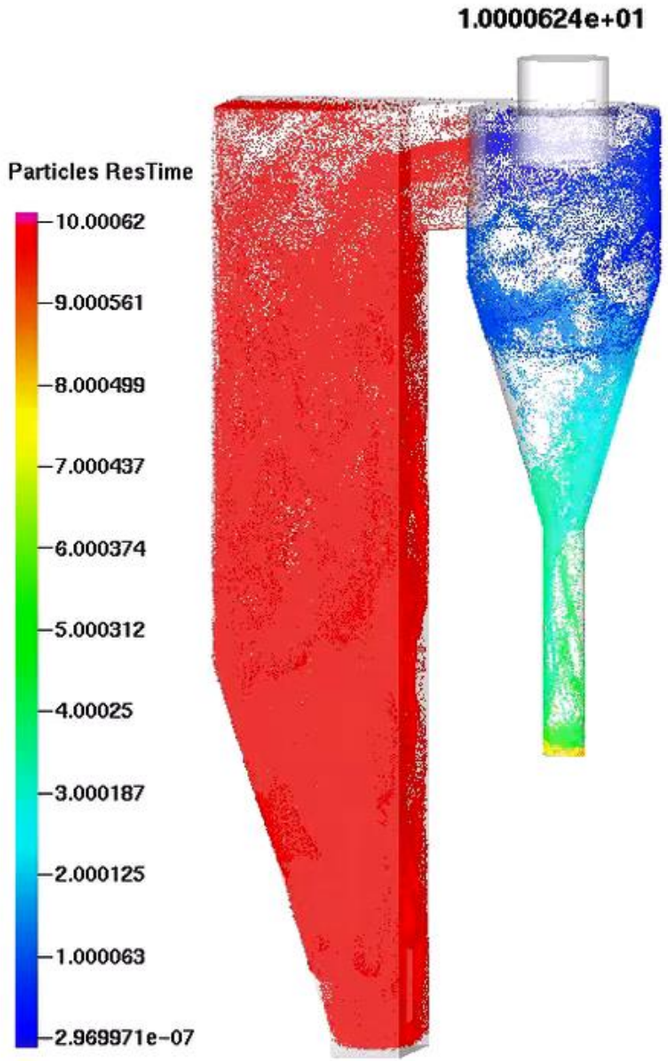
Resetting Particle Residence Time at Internal Flux Planes

Useful for calculating:

- Circulation time in looping systems
- Travel time between two specific planes of interest

Most useful when also selecting "Output raw particle data" at flux planes

Current example: particle residence time is reset to zero as particles cross cyclone inlet flux plane



Exploring Particle Circulation Time Data with print()

In [5]:
```
# Identify particles that have passed through the flux plane multiple times
u, indices, counts = np.unique(pid, return_index=True, return_counts=True)
print("Total number of entries in flux plane raw particle data:  ", len(pid))
print("Number of unique particles that have passed through plane:", len(np.unique(pid)))
print("Maximum number of times any particle has passed through:  ", np.amax(counts))
print("Number of particles that have passed through \"max\" times: ", len(u[(counts == np.amax(counts))]))
```

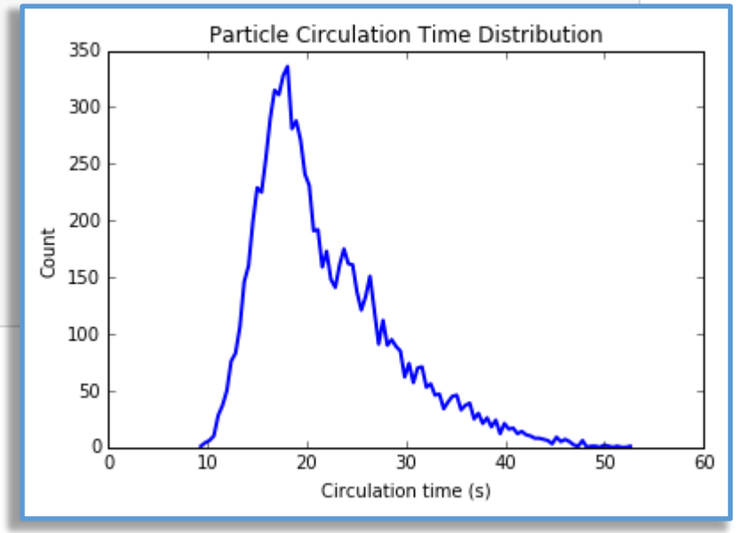
```
Total number of entries in flux plane raw particle data:   1549401
Number of unique particles that have passed through plane: 1178418
Maximum number of times any particle has passed through:   5
Number of particles that have passed through "max" times:  61
```

In [6]:
```
# Look at some detailed data for a specific particle
myPID = u[(counts == np.amax(counts))][0]
print("PID of first particle in list of those with \"max\" times:  ", myPID)
print("Times at which this particle passed through flux plane:   ", t[(pid == myPID)])
print("Circulation times between each pass through flux plane:   ", t[(pid == myPID)][1:]-t[(pid == myPID)][:-1])
print("Average circulation time for this particle (s):           ", np.mean(t[(pid == myPID)][1:]-t[(pid == myPID)][:-1]))
```

```
PID of first particle in list of those with "max" times:   20357.0
Times at which this particle passed through flux plane:    [  3.981584  25.5759    45.69334   59.48454   78.25766 ]
Circulation times between each pass through flux plane:    [ 21.594316  20.11744   13.7912    18.77312 ]
Average circulation time for this particle (s):            18.569019
```

Plotting Particle Circulation Time Distribution

```
In [7]: # Calculate and plot distribution of circulation times for particles that have passed through
        # the cyclone inlet flux plane at least four times.
        pidList = u[(counts >= 4)]
        print("Number of particles that have passed through at least 4 times:", len(pidList))

        # Create an array to hold circulation time values
        dtList = np.array([])

        for myPID in pidList:
            tList = t[(pid == myPID)]
            myDtList = tList[1:]-tList[:-1]
            dtList = np.append(dtList, myDtList)

        print("Number of dt values collected:", len(dtList))

        # Use numpy's histogram function to evaluate the distribution of circulation times
        hist, bin_edges = np.histogram(dtList, bins=100)
        bin_centers = (bin_edges[1:] + bin_edges[:-1]) / 2

        fig, ax = plt.subplots()
        ax.plot(bin_centers, hist, linewidth=2)
        ax.set_title('Particle Circulation Time Distribution')
        ax.set_xlabel('Circulation time (s)')
        ax.set_ylabel('Count')

        p = 'circulation_time_distribution'
        fig.savefig(p + '.png', format='png')
        fig.savefig(p + '.pdf', format='pdf')

Number of particles that have passed through at least 4 times: 2825
Number of dt values collected: 8536
```



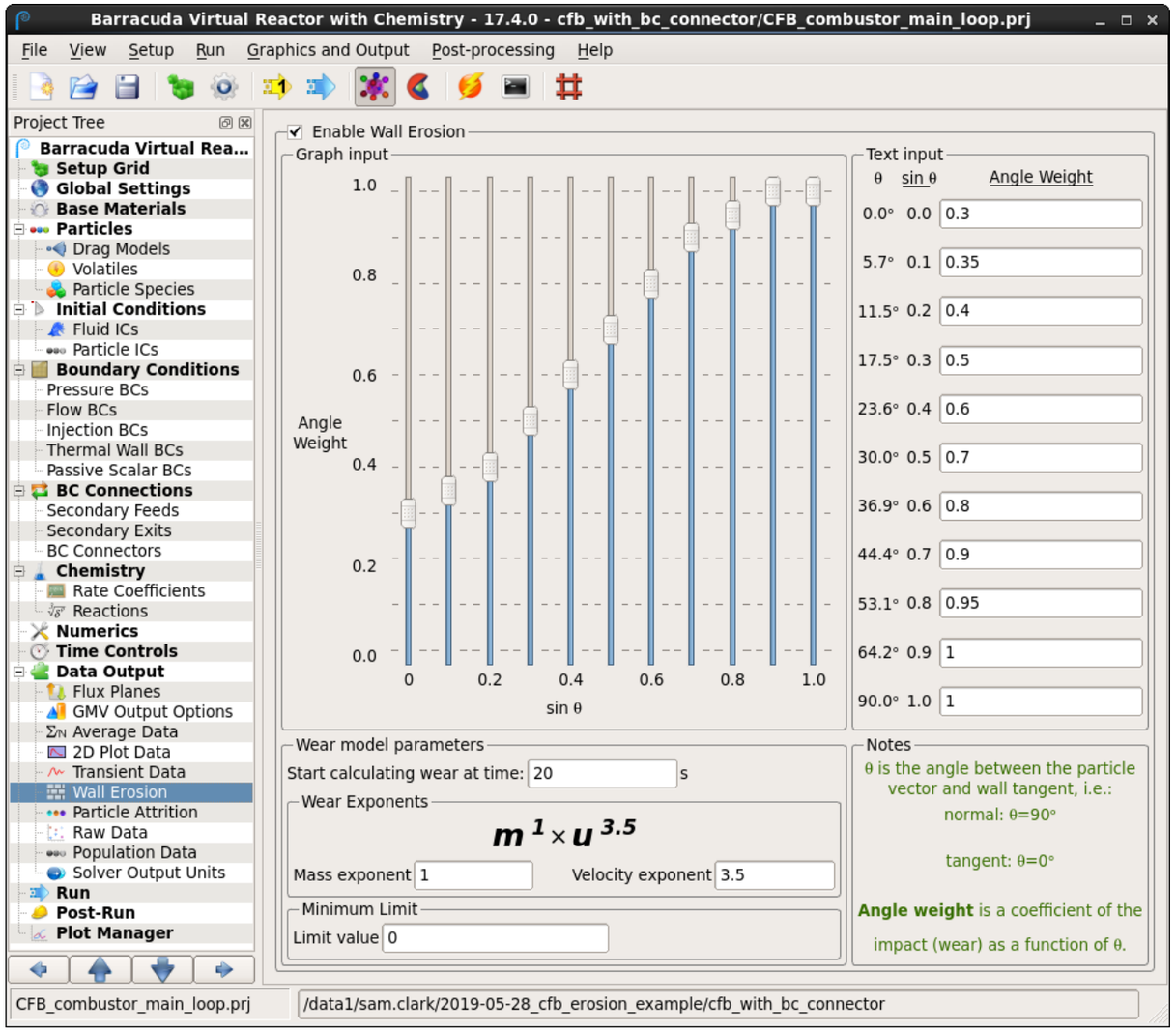
Using Isovolumes to Analyze Wear on Wall Surfaces

The Wall Erosion model must be enabled in order to get this data

Model settings:

- Angle dependence (for more info, see [How do I set wall erosion parameters for my unit?])
- Mass exponent
- Velocity exponent

The "Impact" variable in GMV indicates erosion potential on walls



Creating an Isovolume

Use a "View Particle Data" GMV shortcut

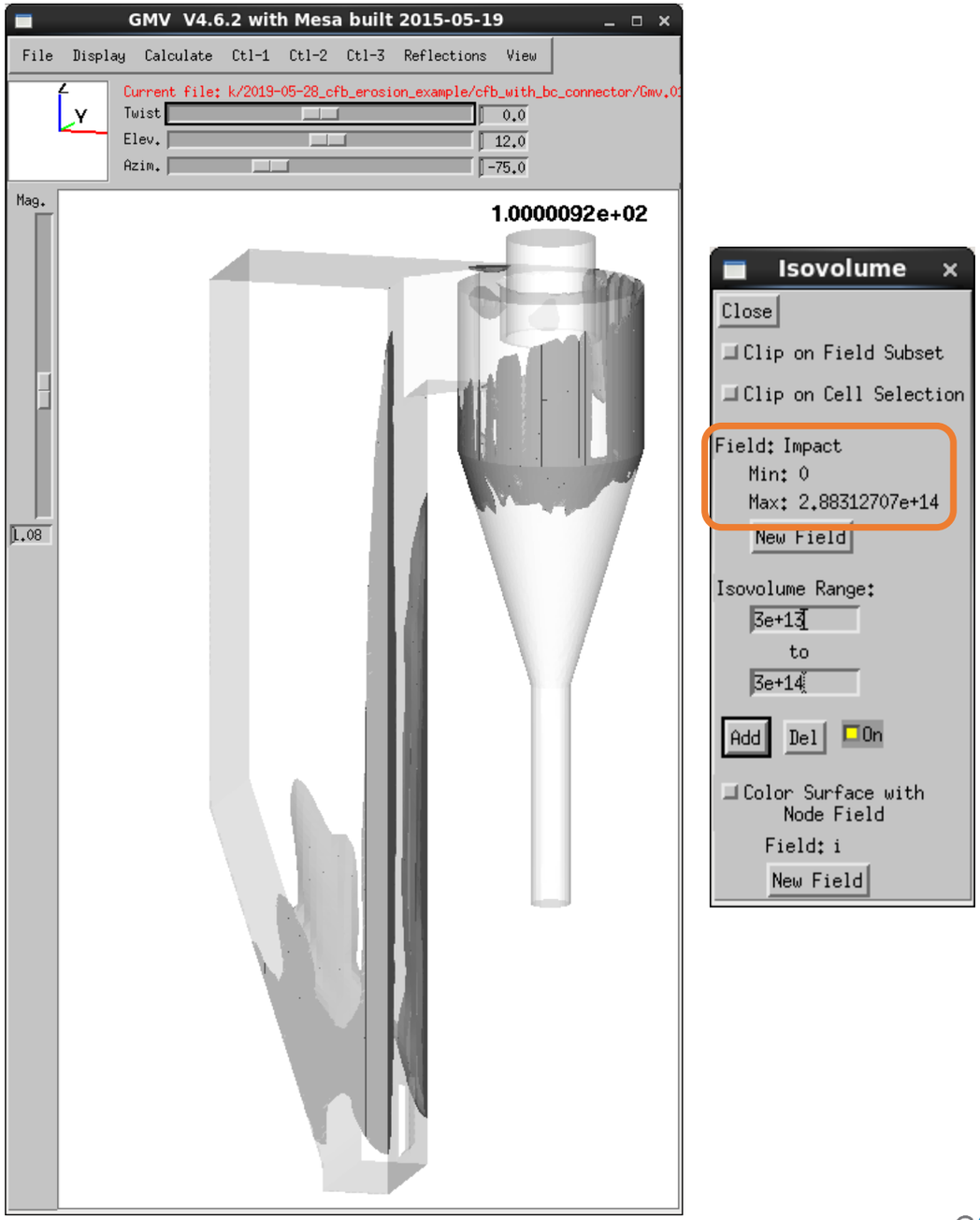
- Easy way to get transparent geometry

Turn off the particles

- Display → Particles → None → Apply

Create an isovolume

- Calculate → Isovolume
- New Field → Impact
- Set an "Isovolume Range"
  - Try Max/10 to Max as a first guess



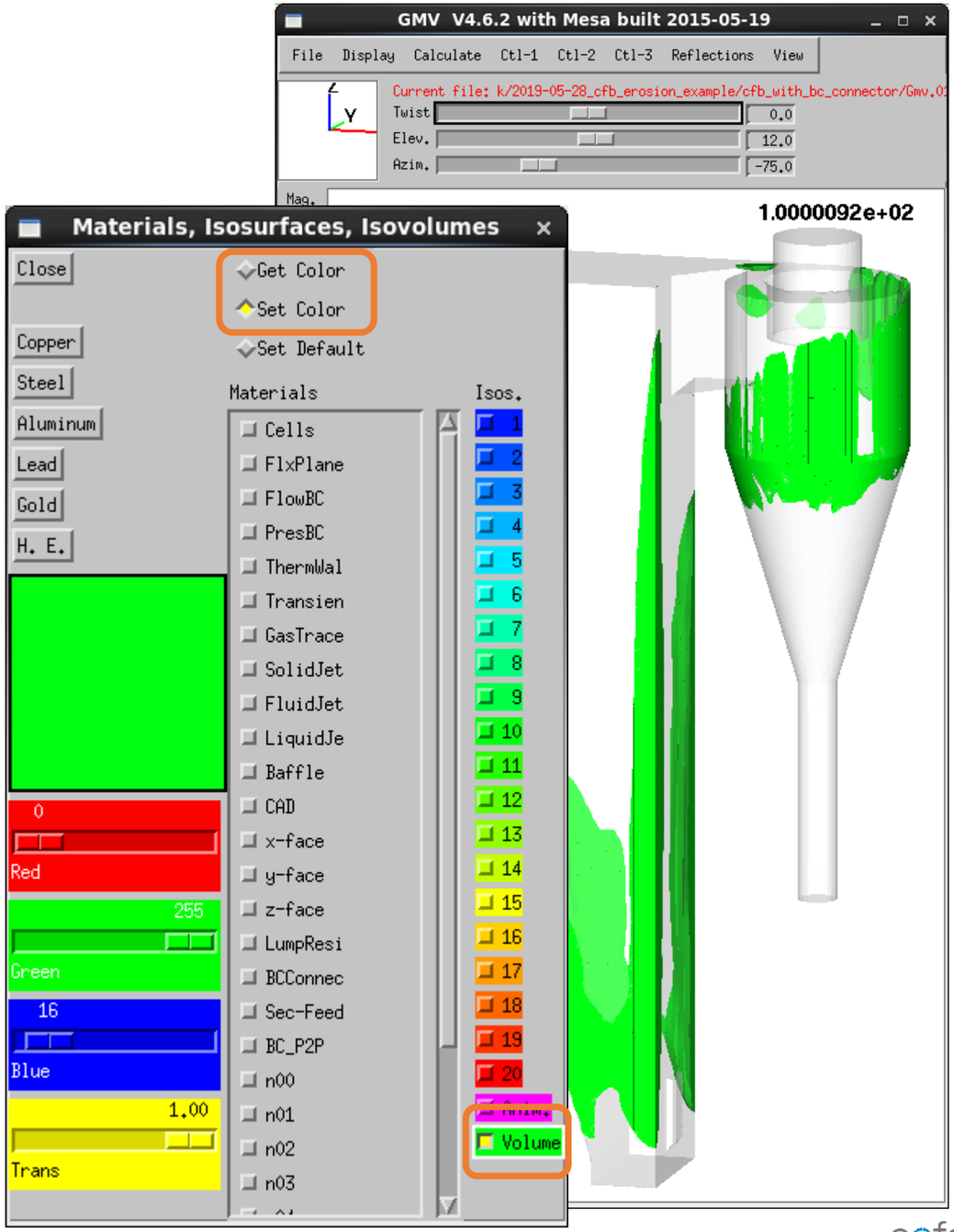
Set Isovolume Color

Isovolumes can be colored by:

- Node Field: this can sometimes be useful, but keep in mind you only see colors on the outer surface of the isovolume
- Constant color: this option removes the need for a colorbar and can be simpler to understand and explain

To set a constant color:

- Ctl1 → Coloredit → Materials…
- Get Color → Choose a color from “Isos.”
- Set Color → Click “Volume”



Explore "Min" Value Settings

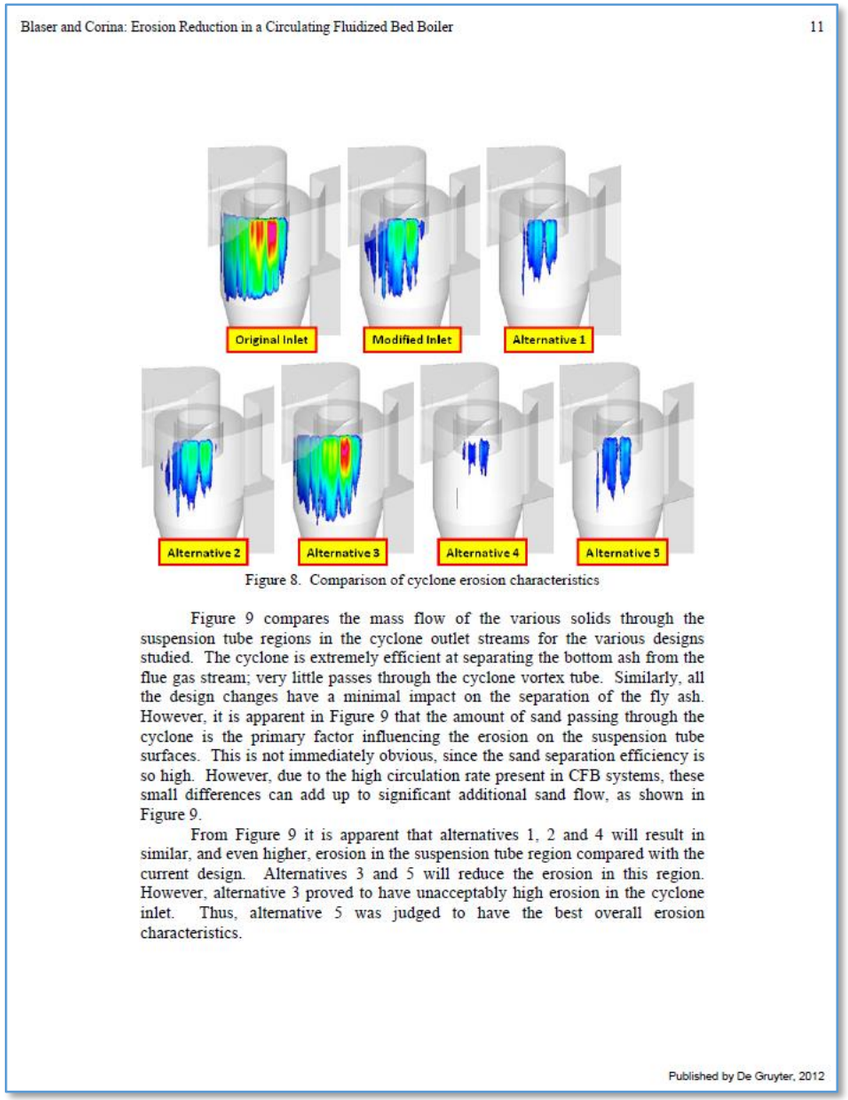
Adjust the "Min" value for the Isovolume Range

- Focus on the region of interest for wall erosion
- A low "Min" value will make the isovolume larger
- A high "Min" value will make the isovolume smaller

What range of values shows useful information?

When comparing multiple cases, open them side-by-side and explore the "Min" value across all cases simultaneously

- The wall erosion model is most useful when comparing several cases / design choices



Blaser and Corina: Erosion Reduction in a Circulating Fluidized Bed Boiler 11

Figure 8. Comparison of cyclone erosion characteristics

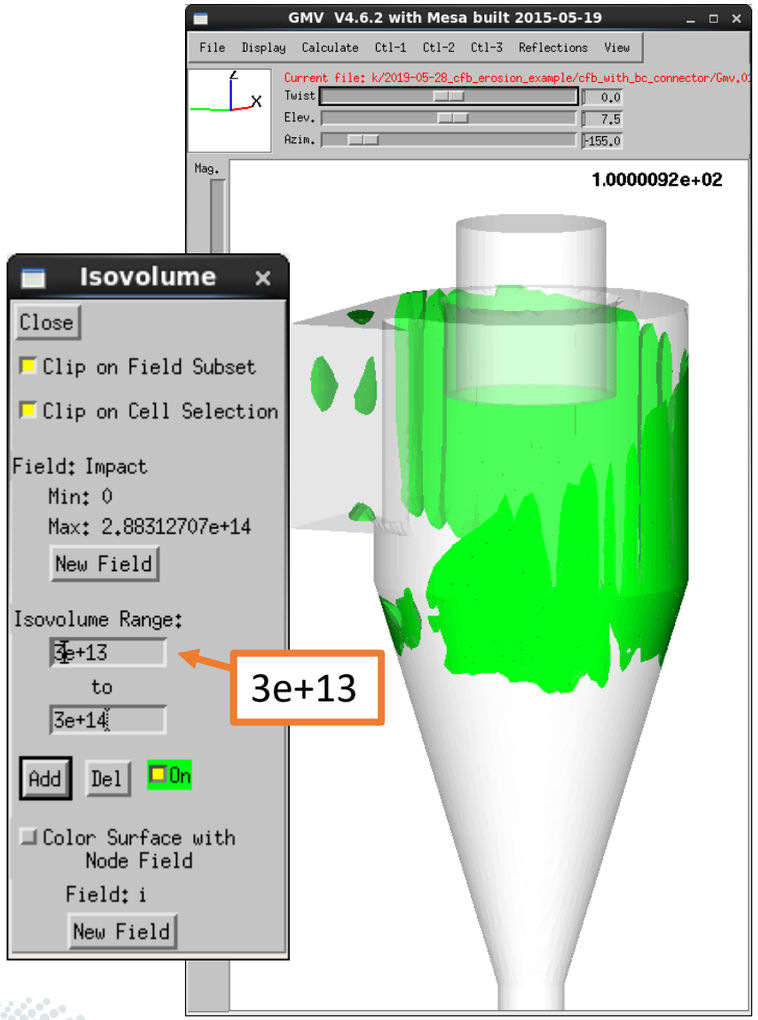
Figure 9 compares the mass flow of the various solids through the suspension tube regions in the cyclone outlet streams for the various designs studied. The cyclone is extremely efficient at separating the bottom ash from the flue gas stream; very little passes through the cyclone vortex tube. Similarly, all the design changes have a minimal impact on the separation of the fly ash. However, it is apparent in Figure 9 that the amount of sand passing through the cyclone is the primary factor influencing the erosion on the suspension tube surfaces. This is not immediately obvious, since the sand separation efficiency is so high. However, due to the high circulation rate present in CFB systems, these small differences can add up to significant additional sand flow, as shown in Figure 9.

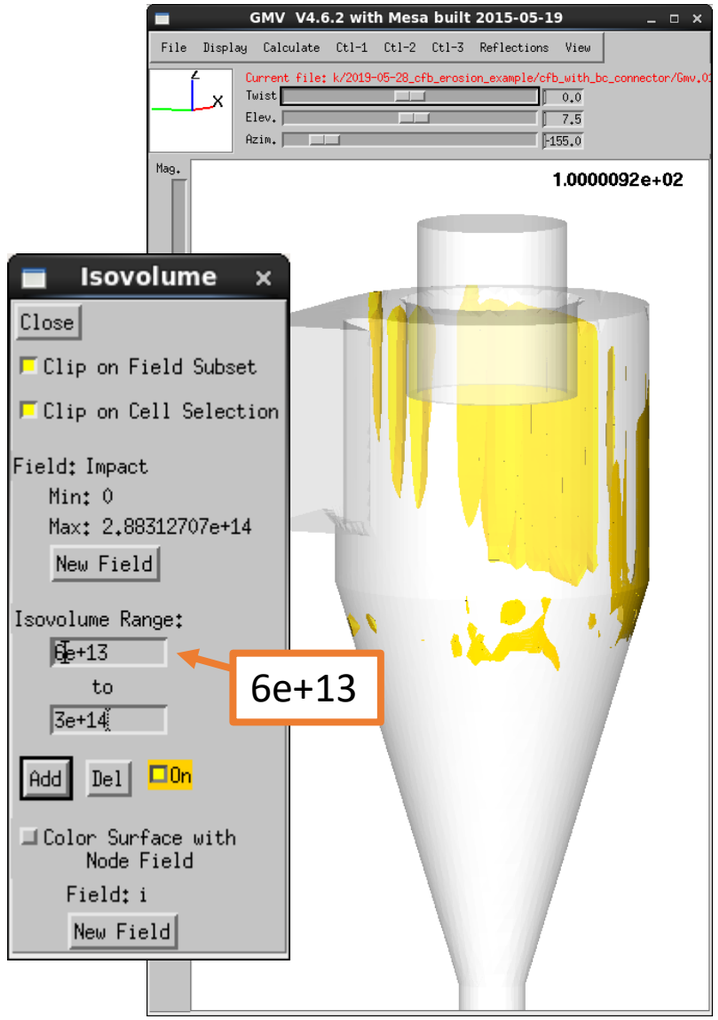
From Figure 9 it is apparent that alternatives 1, 2 and 4 will result in similar, and even higher, erosion in the suspension tube region compared with the current design. Alternatives 3 and 5 will reduce the erosion in this region. However, alternative 3 proved to have unacceptably high erosion in the cyclone inlet. Thus, alternative 5 was judged to have the best overall erosion characteristics.

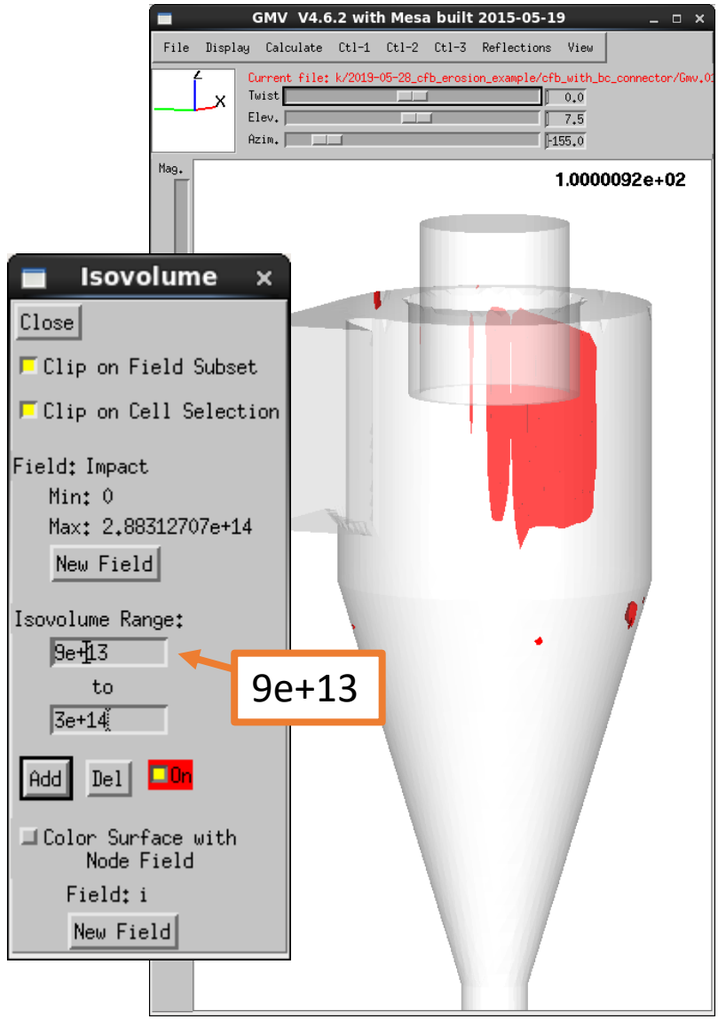
Published by De Gruyter, 2012

Blaser & Corina, IJCRE 2012 Volume 10, A51

Choose Low, Medium, and High Values for “Min”







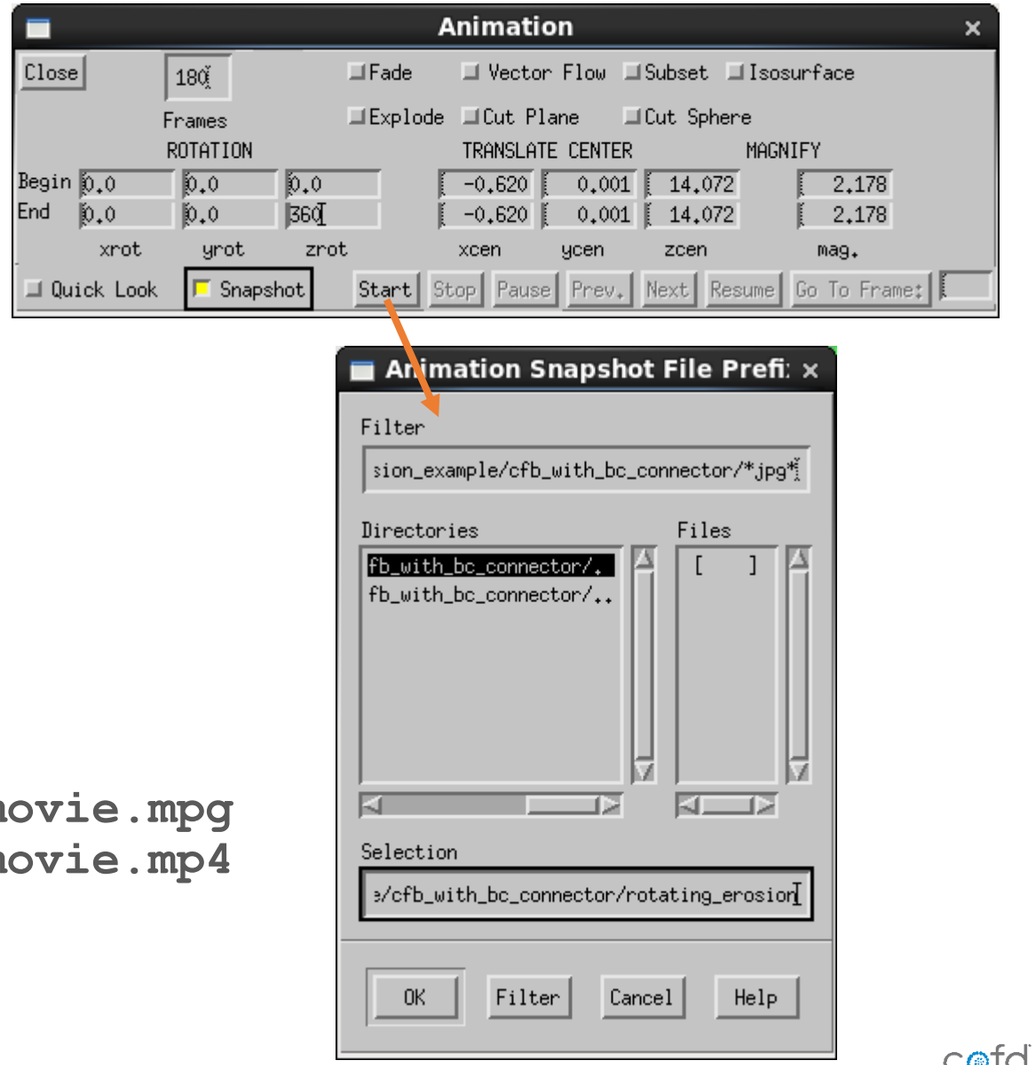
Create a Rotating View Showing Wall Erosion

Useful for showing the 3D position of erosion

Use GMV's "Animation" feature

- Ctl1 → Animation → Rotation
- Select the "Snapshot" option
- Create frames using "Start" button
- Specify the desired image prefix
- Combine frames with jpg2mpg or jpg2mp4

```
jpg2mpg rotating_erosion*jpg -o rotating_movie.mpg
jpg2mp4 rotating_erosion*jpg -o rotating_movie.mp4
```



Rotating Erosion Animation

Options to enhance erosion movie:

- MULTIFRAME.tcl with multiple erosion "Min" values (Low, Medium, High)
- ADDLOGO.tcl to stamp your company's logo onto the movie
- See Barracuda post-processing training material for the Wednesday gasifier, which discusses these topics