User's Guide

Contamination Transport Simulation Program (CTSP)

Version 1.5 (beta)

July 27th, 2020

Particle In Cell Consulting LLC

particleincell.com
info@particleincell.com

Contents

I. Introduction

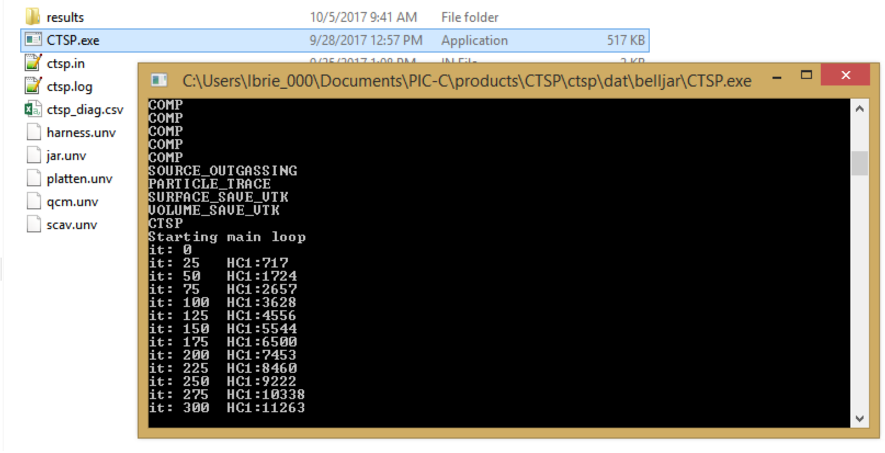
Contamination Transport Simulation Program (CTSP) is a Particle in Cell Consulting LLC (PIC-C) developed computer program for simulating transport of molecular and particulate contaminants [1]. These contaminants include hydrocarbons that diffuse out of materials exposed to vacuum as well as dust particles shaken off by vibrational loads. The code supports two simulation approaches. First, utilizing principles from the Particle in Cell (PIC) [2] and Direct Simulation Monte Carlo (DSMC) [3] techniques used by the plasma and rarefied gas community, contaminants can be represented by simulation *macroparticles,* with each macroparticle corresponding to some (typically) large number of real molecules. This scaling assures that the correct mass transport is simulated regardless of the actual number of particles in the simulation. Particle positions and velocities are advanced through small simulation time steps. Velocities change according to specified spatially and time-varying gravitational, electrostatic, or aerodynamic drag forces. On surface impact, the particles bounce off based on surface physics models, or are adsorbed to the target element surface layer. The code concurrently simulates the entire contaminant population, giving it the ability to consider inter-particle interactions (collisions) that may be of importance during events such as chamber repressurization. The concurrent simulation also allows the user to visualize densities (and other macroscopic properties) of the contaminant plume. The model is described in more detail in Section III. In addition, as of version 1.5, it is also possible to use the particles to compute black-body view factors from selected source element groups. These view factors can then be used to simulate mass transport over an extended period of time during which surface properties such as temperature or coefficient of restitution vary.

I.a) License

Please review **LICENSE.txt** included with the code. Your use of the code implies consent to the license agreement.

I.b) Getting Started

CTSP is a command line code available for Microsoft Windows and Linux (Ubuntu and CentOS) platforms. To use the code, navigate to the directory containing the simulation input file, **ctsp.in**. To run the program, launch the **ctsp** executable. For example, on Microsoft Windows:



```
COMP
COMP
COMP
COMP
COMP
SOURCE_OUTGASSING
PARTICLE_TRACE
SURFACE_SAVE_VTK
VOLUME_SAVE_VTK
CTSP
Starting main loop
it: 0
it: 25   HC1:717
it: 50   HC1:1724
it: 75   HC1:2657
it: 100  HC1:3628
it: 125  HC1:4556
it: 150  HC1:5544
it: 175  HC1:6500
it: 200  HC1:7453
it: 225  HC1:8460
it: 250  HC1:9222
it: 275  HC1:10338
it: 300  HC1:11263
```

Figure 1. CTSP running on Microsoft Windows 8.1

In the Unix shell environment, we first navigate to the folder using **"cd".** The code is launched using **"./ctsp"** assuming the executable is placed in the same directory.

CTSP requires that a valid license file, typically called **ctsp.lic**, is found. The following locations are checked:

1. Path (including the file name) specified by command line option `-l` (lowercase L)
2. Path (including the file name) specified by **options{license_file}** value.
3. "ctsp.lic" file in the current directory
4. "ctsp.lic" file in user's home directory. On Unix, this correspond to the $HOME environmental variable, and on Windows this is the combination of $HOMEDRIVE and $HOMEPATH.

Simulation results are saved in a sub-folder called **"results/"**. This directory will be created automatically if it does not exist.

I.c) A "Hello World" Example

Let's consider a simple case of computing molecular contamination due to outgassing on a generic satellite[1] as shown in Figure 2.

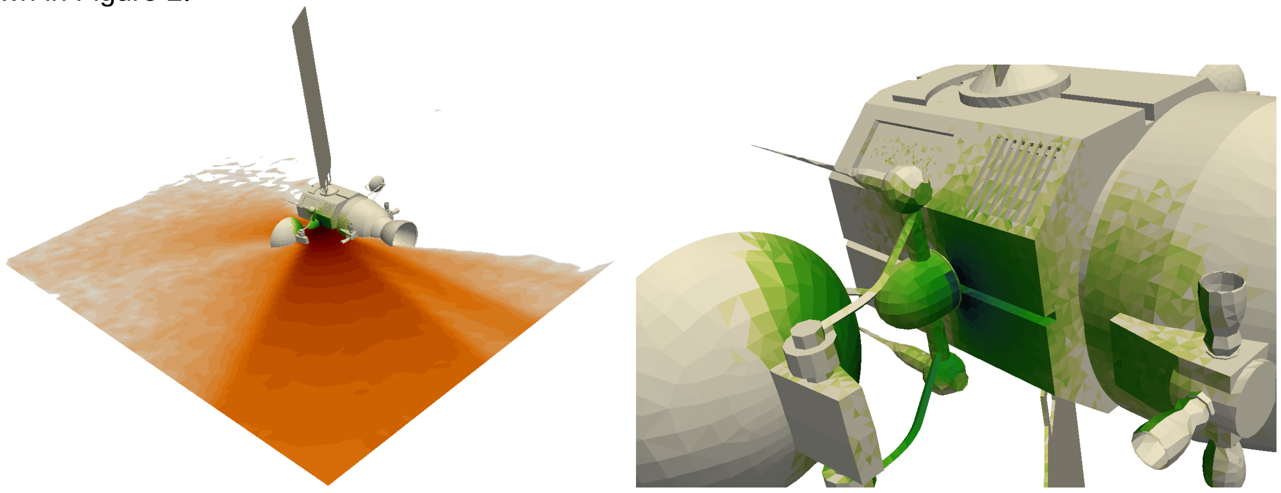


Figure 2. Contaminant plume and surface deposition on a generic satellite

Running this simulation consists of the following steps:

1. Loading the surface mesh(es). The satellite geometry is stored in a "satellite.unv" Universal file. The geometry could optionally be split into multiple files, with transformations such as translation, rotation, or scaling applied to each.
2. Specifying an optional Cartesian volume mesh. If present, the mesh is used to compute volumetric flow data such as the contaminant plume density which was used to produced the visualization in the first plot above. This mesh *is not* used during the particle push, except for bounds checking. Particles leaving the volume mesh are removed from the simulation.
3. Specifying material and material-interaction properties. The code does not contain a materials database and all solid and flying materials and their interactions need to be defined.
4. Specifying surface properties. This assignment is done on a "component" level, where components are groups of surface triangles and/or quads specified in the meshing program and stored in the mesh file. These groups typically correspond to physical objects such as radiators, solar panels, vents, and so on. We can assign material composition and time-dependent surface temperature.
5. Enabling material sources, such as outgassing. Outgassing is modeled by loading an initial concentration of molecules in the substrate and on the surface and specifying a desorption model. The code includes other source models specific to loading particulates or simulating prescribed flux vents.
6. Enabling output diagnostics, such as surface and volume plots, particle traces, and surface concentration histograms. The code supports the VTK (for Visit and Paraview) and Tecplot formats.
7. Running the simulation, and analyzing the results

[1]Pointwise, Inc. provided the mesh. CAD drawing from https://grabcad.com/library/satellite-11

An example **ctsp.in** input file is included below.

```
#load surface mesh
surface_load_unv{file_name:"satellite.unv"}

#define volume mesh for computing contaminant plume
volume_mesh{dx:0.02,dy:0.02,dz:0.02,expand:[0.5,0.5,0.5]}

#specify materials
solid_mat{name:blanketing, density: 2000}
solid_mat{name:sink, density: 2000}
molecular_mat{name:hc1, weight: 54, mpw: 1e10, r:2.776e-10}
molecular_mat{name:hc2, weight: 98, mpw: 1e9, r:4.0e-10}

#material interactions
material_interaction{source:hc1, target:*, Ea_des:12, C_pow:100}
material_interaction{source:hc2, target:*, Ea_des:10, C_pow:80}

#specify surface properties
surface_props{comps:/.*/, mats:blanketing, temp:260, c_stick:0.3}
surface_props{comps:source, temp:1000}

#enable outgassing
load_molecules{comps:source, trapped_mass:1e-10, trapped_mats:hc1, surf_mats:hc2, surf_h:1e-10}
source_outgassing{model:"power"}

#run simulation, perform screen/log file output every 5 time steps
run_sim{dt:1e-5,num_ts:500, log_skip:5}

#save results
volume_save_vtk{file_name:field,vars:[nd.hc1, nd.hc2]}
surface_save_vtk{file_name:surf, vars:[surf_height.hc1, surf_height.hc2, cv_surf_height.hc1,
temperature]}
```

I.d) Data Visualization

CTSP supports saving simulation results in VTK and Tecplot file formats. The Tecplot support is mainly retained for legacy is quite limited. Therefore, it is recommended to use the VTK format. These files can be opened in Paraview (www.paraview.org) or VisIt (visit.llnl.gov), both of which are free programs. Below we demonstrate how to perform a typical visualization in Paraview. After the simulation finishes,

```
ts: 495       HC1:128 HC2:2681
ts: 500       HC1:128 HC2:2622
Simulation of real 0.005010s took 0.139622 minutes
VOLUME_SAVE_VTK
SURFACE_SAVE_VTK
WARNING: Coefficient of variation is undefined for serial runs
Done!
```

folder called "results" will contain several files including "surf.vtp" and "field.vti". The former contains surface-based results, such as the height of the deposited contaminants. The latter file contains volumetric data such as the number density of the contaminant plume. Open these two files in Paraview. If not done so already, it is recommended to enable "Auto Apply" under Edit->Settings->General and change the background and text colors under Color Palette to white and black, respectively. You should obtain a view similar to what is shown below.

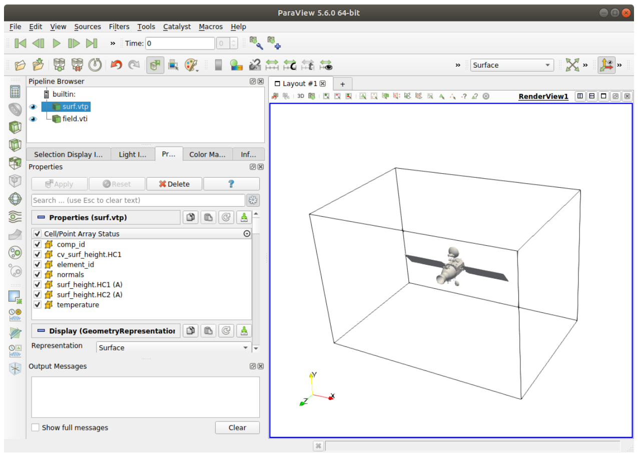


Figure 3. Initial view of results in Paraview

The Visualization Toolkit (VTK), which the Paraview is built on top of, uses the concept of a "visualization pipeline" to produce plots. The pipeline begins with the data, and is then transformed via multiple filters. The output of one filter can serve as an input to another. These outputs can also be plotted in one of several ways, such as "surface with edges". Next, with "surf.vtp" selected in the Pipeline Browser, change "Coloring" on the "Properties" tab from "Solid Color" to "surf_height.HC2". On "Color Map Editor" (enable under View menu if not visible), use the "folder with a heart" icon in the vertical toolbar to pick a different color scheme, such as "Black, Orange, White". Next click the "white/black circle" icon to invert the colormap to make white correspond to the low values and black to high values. Next use the "Rescale to Custom Range" button in this same vertical stack to set the range to 1e-4 to 0.1. Next click the "colorbar with an e" icon to adjust legend properties. Click the "gear" icon next to the Search bar, and uncheck "Add Range Labels". This will make the number consistent as by default the colorbar range is printed using different precision. Also, on the Color Map Editor tab, enable "Use log scale when mapping data". Next interact with the geometry using the mouse. Holding down the right button while moving the mouse zooms. The middle button pans, while the left button rotates. You should obtain a view as shown below.

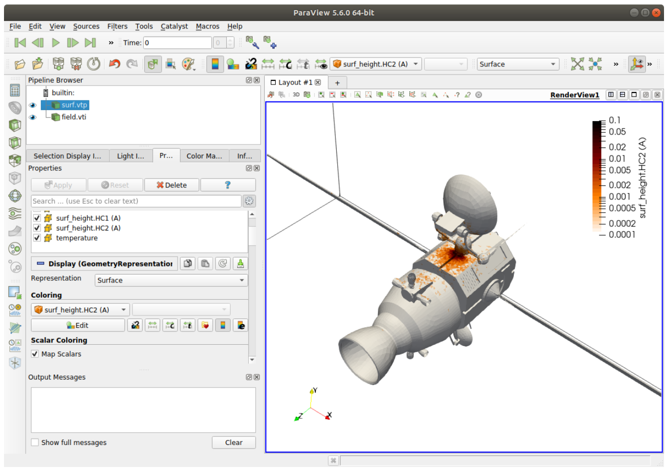
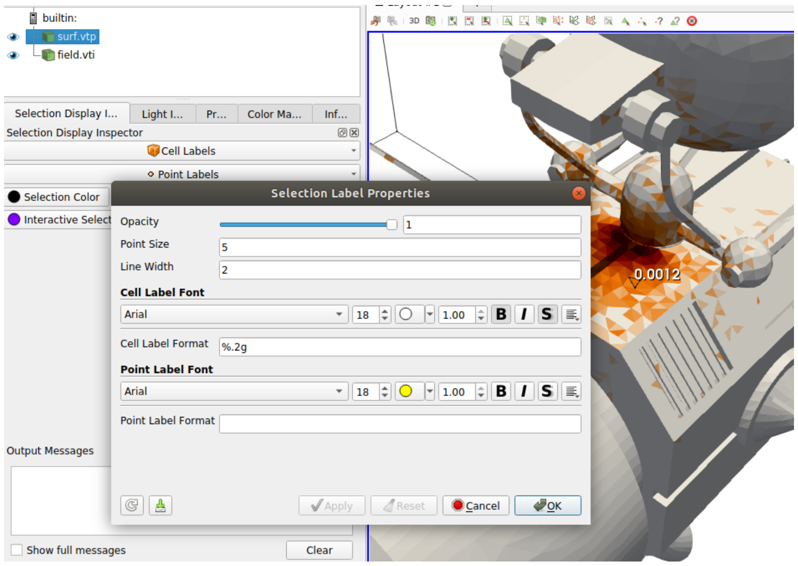


Figure 4. Visualization of surface data

Next, pressing "s" enables us to enter an interactive selection mode of cell data. Other selections can be enabled by clicking the small icons just above the blue border of the render view. Then, under "Selection Display Inspector" (enable under View menu), you can assign a specific variable for "Cell Labels". Here you can also change the color of the selection border, and the font, color, and number format (using the C printf syntax) of the displayed number (%.2g means to show a floating point number with two significant digits).



Selection Label Properties

Opacity: 1
Point Size: 5
Line Width: 2

Cell Label Font

Arial 18 1.00

Cell Label Format: %.2g

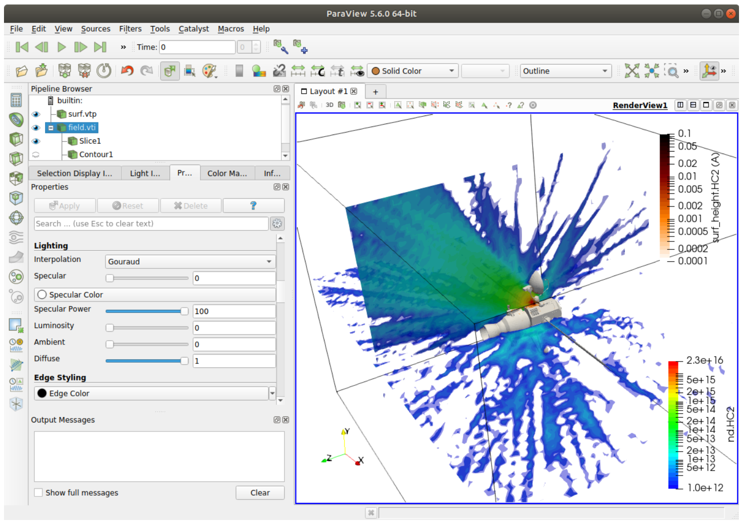
Point Label Font

Arial 18 1.00

Point Label Format:

Figure 5. Use of selection inspector to visualize deposition on a surface element

Next, we select "field.vti" in the pipeline and use the "Slice" button to add a filter to plot data on a single 2D slice. Next selecting "field.vti" again, another slice can be added, with a different orientation. We can use the Color Map Editor to further change the data range, or to "enable opacity mapping for surfaces" to make data below the minimum color map value transparent. The "Mapping Data" window can be used to interactively set the opacity transfer function. We generate a view as shown below.



Lighting

Interpolation: Gouraud
Specular: 0
Specular Color
Specular Power: 100
Luminosity: 0
Ambient: 0
Diffuse: 1

Edge Styling

Edge Color

Figure 6. Use of selection inspector to visualize deposition on a surface element

I.e) Parallel Processing

CTSP supports parallel processing via multithreading and MPI. CUDA support (for running on NVIDIA GPUs) is currently in development. CTSP implements two parallelization schemes. Bulk of the total computational effort involves advancing particle positions through time step Δt. This particle push is locally parallelized by assigning a subset of all particles to each CPU thread. The remaining parts of the code continue to run in serial. By default, if not run through MPI, the code utilizes the maximum number of concurrent threads supported by the CPU. On modern CPUs supporting hyperthreading, this will generally be twice the number of physical cores. Therefore, there may not be a noticeable speed up compared to using half this number. The number of threads to use is controlled via **options{num_threads:xxx}** command.

The code can also be run in parallel on multiple machines using "mpirun" (or mpiexec):
>mpirun -np 4 ./ctsp-ubuntu-mpi

Currently, only the Ubuntu version is shipped with MPI support enabled, but versions for other architectures can be provided on request. The MPI behavior differs from the multithreaded case described above. Instead of attempting to distribute the workload among processors, each MPI process runs the full simulation serially. The individual results are then ensemble-averaged for output. Running a parallel simulation on 10 cores is identical to running a single processor case with 10x as many numerical particles (the number of particles shown on the screen or in the log file will reflect the total particle count over the MPI domain). Parallel treatment for cases involving DSMC is in development. An example output from a parallel run is shown in Figure 4. We can clearly see that the low concentration (green) region is resolved in greater detail with more processors. Furthermore, since each processor essentially runs a serial simulation with a different initial random number seed, we can use the local results to compute the coefficient of variation (normalized standard deviation) of the results on each surface element. Visualizing this data (by prepending "cv_" to a surface variable of interest, such as surf_height) allows us to determine in which regions the results can be truested, and where we are looking mainly at numerical noise. A "cv" value of 0.1 indicates confidence within 0.1 standard deviations.

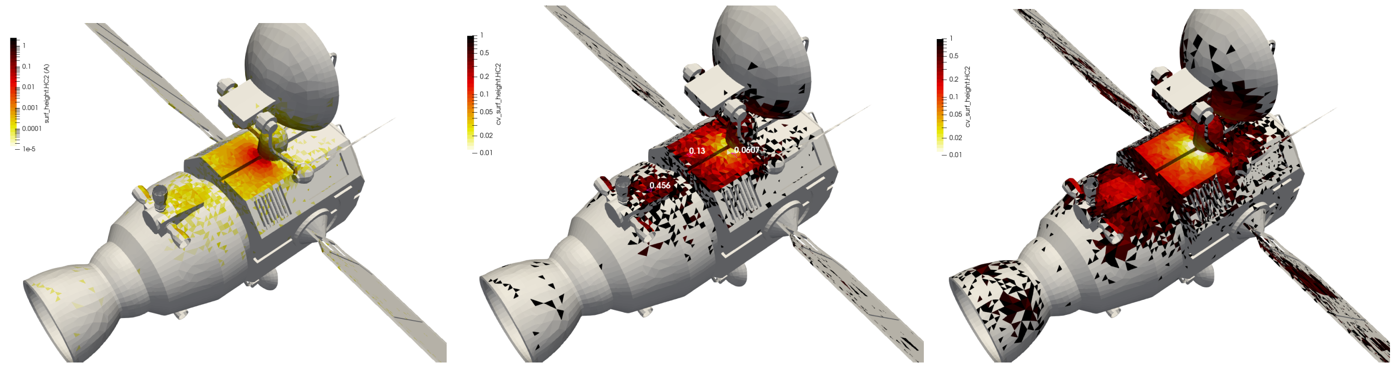
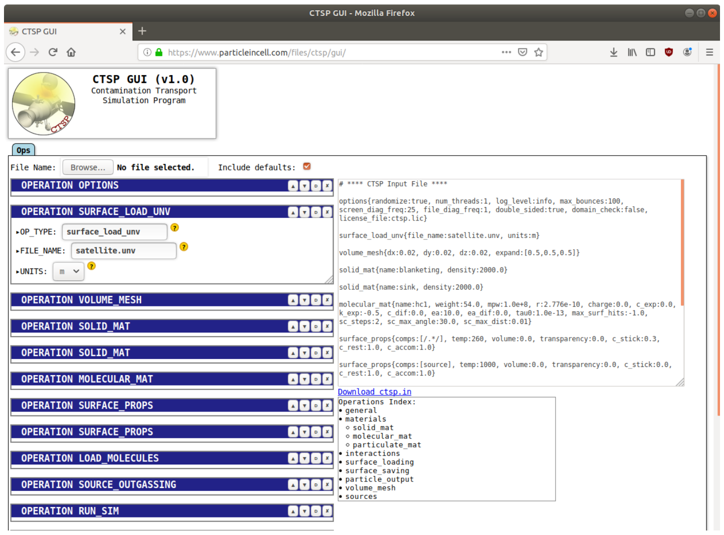


Figure 7. Plot of coefficient of variation from 6 and 48 CPU run

I.f) Online GUI

While an actual GUI is still in development, the simulation input files can be generated with the help of a browser-based interface. The GUI has been tested with Firefox and Chrome. You can run the GUI either locally or from the code website **https://www.particleincell.com/ctsp/.** Note that even when accessing the on-line version, the GUI runs locally on your machine and no information is transmitted to our server.



```
# **** CTSP Input File ****

options{randomize:true, num_threads:1, log_level:info, max_bounces:100,
screen_diag_freq:25, file_diag_freq:1, double_sided:true, domain_check:false,
license_file:ctsp.lic}

surface_load_unv{file_name:satellite.unv, units:m}

volume_mesh{dx:0.02, dy:0.02, dz:0.02, expand:[0.5,0.5,0.5]}

solid_mat{name:blanketing, density:2000.0}

solid_mat{name:sink, density:2000.0}

molecular_mat{name:hc1, weight:54.0, mpw:1.0e+8, r:2.776e-10, charge:0.0, c_exp:0.0,
k_exp:-0.5, c_dif:0.0, ea:10.0, ea_dif:0.0, tau0:1.0e-13, max_surf_hits:-1.0,
sc_steps:2, sc_max_angle:30.0, sc_max_dist:0.01}

surface_props{comps:[/.*/], temp:260, volume:0.0, transparency:0.0, c_stick:0.3,
c_rest:1.0, c_accom:1.0}

surface_props{comps:[source], temp:1000, volume:0.0, transparency:0.0, c_stick:0.0,
c_rest:1.0, c_accom:1.0}
```

Operations Index:
- general
- materials
  - solid_mat
  - molecular_mat
  - particulate_mat
- interactions
- surface_loading
- surface_saving
- particle_output
- volume_mesh
- sources

Figure 8. CTSP web-based GUI

The GUI consists of three sections: a list of operations showing all available options, a dynamic and automatically updated ctsp.in file, and an index of all available operations. Note that modifying the "OP_TYPE" field also brings up a drop down list of matching operations, see Figure 4.

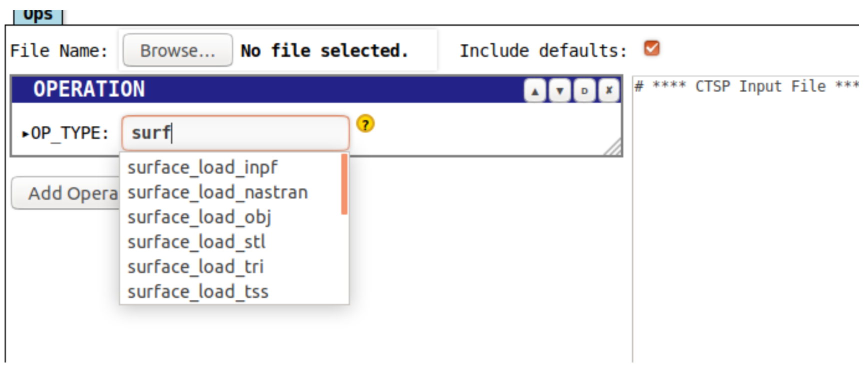


Figure 9. List of matching operations

To save the file, click the "download ctps.in" link or simply copy and paste from the output window.

I.g) Future Work and Revision History

Current work includes the following:

1. More robust particle surface adhesion model based on the DEM method
2. Adding support of particulate mat-mat interactions and time-dependent c_stick
3. GUI
4. Additional code parallelization and port to GPU
5. Implementation of ionization and triboelectric models to simulate electrostatic effects

Revision History

1.5 Major update introducing mat-mat interactions (no longer specified by surface_props) allowing time-dependent coefficients and desorption models. New support for splitting the input into multiple files, include variables, add loops, integrate results, and save element-id averaged restart data to support

	simulating systems with time-varying surface models. SurfaceSaveVTK operation now takes a list of variables to output and supports outputting coefficient of variation from parallel runs. The octree used for storing data has been revamped, although optimization is ongoing. In addition, the TSS loader has been improved to handle assembly transformations, and surface temperature data loader has been added to load SINDA results. Initial work on computing black body viewfactors.
1.00	Major rewrite of the code base to reduce reliance on pointers, split of particulate sources into loading and generation, support for tape lift data, selection of particle integrator schemes, component-level surface deposition histograms, external world force, periodic and symmetric flow data, web-based GUI, time step subcycling, better memory management
0.29	Fixes to particle surface handling, MPI, automatic creation of results directory, material density range
0.28.2	Addition of ISO-14644 source
0.28.1	Search for license file in user home directory

Please report any bugs or feature requests to info@particleincell.com.

II. Operations Reference

CTSP simulations run according to commands specified in an input file. By default, the code searches for a file called **ctsp.in** in the current directory, but an alternative path can be specified as a command line argument. For instance:

```
> ./ctsp ../simulations/case2.in
```

The input file consists of multiple lines with the following format:

```
#comment
operation{key1:value2, key2:value2, …}
```

Any line starting with a **#** is ignored. Each operation generally requires user inputs consisting of a "key" (the field name) and the corresponding values. The table below summarizes the value types.

type	description	example(s)
bool	Boolean value, can be one of **true** or **false**	key:true
int	Integer value	key:350
float	Double precision floating point value	key:1.4e-6
float2	Range of double precision values, or alternatively a single value	key:2000 or key:[2000, 4000]
string	A string value in optional quotation marks. May need to be a value from a list, or can be a regular expression between slashes	key:fairing key:/.*/
int3	List of three integers bounded by square brackets	key:[3,4,5]
i_list	Arbitrarily long list of integers	key:[50,1000,1200,5000]
float3	List of three floating point values in square brackets	key:[0.2,0,1.2]
f_list	List of one or more floating point values in square brackets	key:[0,0.2,0,0,1.2,1.2]
tupples	Arbitrarily long list of float@float pairs	key:[300@0, 300@5, 100@6]
s_list	Arbitrarily long list of strings	key:[pressure,nd.hc1,t.hc1]
s_comp	Arbitrarily long list of float*string pairs with the float* part optional. Used to specify material composition. If the fraction part is omitted, the code assumes "1*".	key:hc1 key:[0.6*hc1, 0.3*hc2, 0.1*n2]
time	Numerical value with optional d,h,m,s units included. Seconds are assumed if no units specified.	key:3d.2h

In additions, string variables can be substituted within dollar signs ($...$) and mathematical expressions can be included within a percent sign pair (%...%). The available operations are listed below. They are grouped by the operation type.

II.a) General

These operations provide support for “general” tasks such as setting program options or setting the global “world” environment. Items highlighted in yellow are new in CTPS v.1.5

II.a.1) SET_VARIABLE

Adds a new variable that can be used in subsequent parts of the code within expressions wrapped by dollar (for direct substitution) or percent (to evaluate a simple mathematical relationship) signs. Variables are evaluated during the operation parsing.

Key	Type	Default	Description
name	string		Variable name
value	string		Expression to evaluate. The code currently supports only basic arithmetic (+,-,*,/)

Example

```
set_variable{name:T_substrate, value:273}
set_variable{name:rads, value:rad_}
# set rad_1, rad_2, rad_XYZ, etc... to 298K
surface_props{comps:/$rads$.*/, temp:%T_substrate+25%}
```

II.a.2) INCLUDE

Inserts contents of the specified file at the current location. Can be used to split the simulation inputs into multiple physical files to reduce duplication.

Key	Type	Default	Description
file_name	string		File to load

Example

```
include{file_name:ctsp-master.in}
```

II.a.3) OPTIONS

Specifies various options. This op is processed first regardless of where it is located in the input file.

Key	Type	Default	Description
randomize	bool	true	Will randomize the random number generator if true. Set to false to replicate the same simulation.
num_threads	int	np-1	Maximum number of threads to use for multithreading. By default set to CPU “number of cores” minus 1.
log_level	string	info	Specifies level of screen and file output. In order of decreasing output, one of [DEBUG, INFO, WARN, ERROR].
max_bounces	int	100	Maximum number of times a particle can hit a surface in a single time step before being flagged as "stuck"
double_sided	bool	true	Specifies whether surface contains double sided elements. If set to false, particles always reflect in the surface normal direction. If true, particles will reflect in anti-normal direction if hitting the back side. This however can lead to particle leaks so this should be set to false unless thin plate elements are actually present.
domain_check	bool	false	Prints a warning message if particle leaves volume mesh. Useful for checking for particle leaks in a closed domain.
license_file	string	ctsp.lic	Path to the license file. Needs to be enclosed in quotes if “:” is

			present (in other words, if full path is specified on Windows)
surf_oct_maxdepth	int	5	Maximum number of surface octree subdivisions
Example			
`options{randomize:false,num_threads:2,log_level:info,license_file:”ctsp.lic”}`			

II.a.4) RESTART_SAVE

Saves data to a restart file.

Key	Type	Default	Description
file_name	string	restart	Prefix for the restart file. ".bin" extension is added automatically, and in parallel runs, the rank is appended to the file name
skip	int	-1	Frequency of restart saves. If non-positive, the restart file is written out right away.
group_elements	bool	false	If true, surface deposition and bulk composition will be averaged over all elements with the same element id. This is currently only applicable to TSS geometries since other surface models do not support the same id to be shared among multiple elements

Example

```
restart_save{skip:1000}
```

II.a.5) RESTART_LOAD

Loads restart data

Key	Type	Default	Description
file_name	string	restart	Prefix of the restart file name to load
load_particles	bool	true	Skips loading particles if set to false

Example

```
restart_load{file_name:restart2}
```

II.b) Simulation Control

These operations provide support for running the simulation. The simulation can be executed using either the "run_sim", which starts simulating particles, or by using "compute_viewfactors" to generate black body viewfactors which will then be used in a matrix-based gray body calculation (this functionality is still in development).

II.b.1) BLOCK_START / BLOCK_END

Defines a repeatable block of commands. The block repeats for the given number of iterations as long as the condition evaluates to a non-zero value. Multiple blocks can be nested.

Key	Type	Default	Description
repeat	int	1	Number of times the block should repeat
condition	string	1	Numerical expression to evaluate

Example

```
#approximate outgassing over a 60 minute interval using 6 steps
block_start{repeat:6, condition: $simulate_outgassing$}
run_sim{dt:1e-3, num_ts:1000}
scale_outgassing{mats:*,factor: 600}
world{add_time:10m}
block_end{}
```

II.b.2) COMPUTE_VIEWFACTORS

Uses a specified material type to compute black body viewfactors for all elements of a given zone. Specifically, a user specified number of particles is launched from each element using a cosine source. These particles are propagated until hitting a surface or leaving the domain. Particles stick on the first surface impact. Since the same algorithm is used to push particles as during a regular simulation, it is possible to include the effect of charge, drag, or even collisions. The computed viewfactor can be visualized using "vf.el_id" surface variable.

Key	Type	Default	Description
comps	s_list		List of component groups to apply the op to
mat	string		Material to use for the calculation
particle_count	int	1000	Number of particles (rays) to generate per element
max_steps	int	10000	Maximum number of time steps per element. The loop terminates early once all particles are lost.
dt	float	1e-3	Simulation time step. Vel_mag*dt should be no more than 10% of the simulation bounding box.
vel_mag	float	1	Injection velocity. Can be set arbitrarily for free molecular flow (except that dt needs to be scaled accordingly to prevent an inefficiently-large bounding box when checking for surface hits).
log_file_name	string	vf.log	Name of a file to which view factors are saved, or set to blank for no output.

Example

```
compute_viewfactors{comps:vent,mat:hc1,dt:1e-3}
```

II.b.3) RUN_SIM

Starts the main loop for a particle based simulation.

Key	Type	Default	Description
dt	float		Simulation time step
num_ts	int		Number of time steps
ts_steady_state	int	0	Time step at which gas bulk property computation starts
fields_skip	int	10	Number of time steps between computation of volume data
log_skip	int	5	Number of time steps between screen and log file output. Generating this output requires parallel sync in MPI runs and hence could lead to a possible performance penalty.

Example

```
run_sim{dt:5e-6, num_ts:10000, diag_start:100, diag_skip:10}
```

II.b.4) SCALE_OUTGASSING

Scales surface deposition and depletion by a given factor

Key	Type	Default	Description
mats	s_list		List of materials to apply the operation to
factor	float		Multiplication factor for surface deposition and depletion. Assuming source flux is the time averaged value over duration [t1,t2], factor should be set to [t2-t1]/(num_ts*dt)

Example

```
#approximate outgassing over a 60 minute interval using 6 steps
block_start{repeat:6, condition: $simulate_outgassing$}
run_sim{dt:1e-3, num_ts:1000}
scale_outgassing{mats:*,factor: 600}
world{add_time:10m}
block_end{}
```

II.b.5) STOP

Stops processing the input file. Useful for testing instead of commenting out follow on ops.

Key	Type	Default	Description

Example

```
stop{}
```

II.c) Ambient Environment

This section includes operations for setting the ambient environment

II.c.1) EFIELD_LOAD_CSV

Loads electric field components (in V/m) from a file. This operation can be used to simulate effect of the electric field on charged particulates or molecules. The code uses the inverse-distance weighing methods to interpolate field coponents, which can lead to out-of-sight points being considered near corners.

Key	Type	Default	Description
file_name	string		File to load
map_pos	int3	[0,1,2]	Column indexes for position
map_ef	int3	[3,4,5]	Column indexes for electric field components

Example

```
efield_load_csv{file_name:efn.csv}
```

II.c.2) FLOW_LOAD_CSV

Loads flow data from a CSV file containing a point cloud of positions (in meters), velocities (m/s), and densities (kg/m^3/s). Instead of density, pressures (Pa) and temperature (K) can be included. Multiple files can be loaded to correspond to different simulation times with linear interpolation used for intermediate values. The code uses the inverse-distance weighing methods to interpolate velocities, which can lead to out-of-sight points being considered near corners.

Key	Type	Default	Description
file_name	string		File to load
map_pos	int3		Column indexes for positions
map_vel	int3		Column indexes for velocities
map_rho	int	-1	Column index for density. Either density or pressure with temperature need be listed, if given. If not included, the code uses the density data per **world**.
map_p	int	-1	Column index for pressure if density not listed
map_t	int	-1	Column index for temperature if density not listed
time	float	0	Time in seconds for this data set. Multiple data sets can be loaded with the code linearly interpolating velocity data at in-between times.
end_time	float	-1	End time, in seconds. Only applicable to the last file in series.
mode	string	NORMAL	One of [NORMAL, WEDGE_X/Y/Z, or SYMMETRY_X/Y/Z]. WEDGE_Z models axisymmetric wedge rotated about the z axis, while SYMMETRY options are used with data symmetric about a single axis.
dtheta	float	0	Wedge width in degrees, zero for axisymmetry
x0	float3	0,0,0	Point on the axis for WEDGE or on the symmetric face for SYMMETRY

Example

```
flow_load_csv{file_name:"purge_1e-4.csv",map_pos:[4,5,6],map_vel:[0,1,2],
time:-1,time:0}
flow_load_csv{file_name:"purge_1e-3.csv",map_pos:[4,5,6],map_vel:[0,1,2],
time:-1,time:5}
```

II.c.3) WORLD

Specifies the ambient environment. Multiple commands can be included to simulate transient events.

Alternatively, a .csv file containing temporal data can be provided. The world command can also be used to set or advance the simulation time. This is useful for setting initial time to correspond to transient surface temperature data, or to advance the solution in conjunction with "scale_outgassing" op.

Key	Type	Default	Description
time	time	0	Simulation time corresponding to this entry
gravity	float3	[0,0,0]	Vector form of gravitational acceleration vector in m^2/s
uniform_flow	float3	[0,0,0]	Vector form of uniform flow velocity in m/s
E_field	float3	[0,0,0]	Vector form of uniform electric field in V/m
F_ext	float3	[0,0,0]	Generic external force in N
density	float	0	Ambient gas density in kg/m^3
pressure	float	0	Ambient pressure in Pascal
temperature	float	0	Ambient temperature in K, used to compute gas density if density not specified
accel_vibe	float	0	Magnitude of random vibrational acceleration, used to detach particulates
file_name	string		Optional CSV file for specifying time-dependent environment
periodic	bool	false	If true, data in the above file is assumed to be periodic, with period (time[last]-time[first]).
map_time	int		CSV file column index for time in seconds. Linear interpolation used for in-between data.
map_gravity	int3	[-1,-1,-1]	Column indexes for gravity vector, -1 indicates data not present
map_flow	int3	[-1,-1,-1]	Column indexes for flow velocities
map_ef	int3	[-1,-1,-1]	Column indexes for electric field components
map_fext	int3	[-1,-1,-1]	Column indexes for external force
map_density	int	-1	Column index for density
map_pressure	int	-1	Column index for pressure
map_temperature	int	-1	Column index for temperature
map_accel_vibe	int	-1	Column index for magnitude of random vibrational acceleration
Time Setting			
set_time	time	-1	Sets current simulation time to the provided value if non-negative
add_time	time	0	Advances current simulation time by the given interval

Example

```
world{gravity:[-9.81,0,0],uniform_flow:[0.0,0,0], density:1.2}
world{file_name:accel.csv, map_time:0, map_gravity:[1,-1,-1],
 pressure:101325, temperature:300}
world{set_time:4m5s}
```

Example accel.csv file:

```
time, gx
0,-9.81
5,-12.6
70,-16.9
```

II.d) Materials

These ops are used to define simulation materials.

II.d.1) SOLID_MAT

Specifies a general non-flying material. Solid materials are currently used only to control diffusion in the Fang model.

Key	Type	Default	Description
name	string		Material name
density	float	2000	Material density in kg/m^3. Currently unused.

Example

```
solid_mat{name:al, density: 1000}
```

II.d.2) MOLECULAR_MAT

Specifies a molecular material which simulates gas particles with negligible mass.

Key	Type	Default	Description
name	string		Material name
weight	float		Atomic weight in AMUs
mpw	float		Macroparticle weight - ratio of real to simulation molecules. This parameter controls the number of particles in the simulation. A lower value will results in more particles and less noise but results in longer run time. Typical values are around 1e15 and can be computed from: *contam_plume_number_density*domain_volume = mpw * N_sim* where a good starting point for N_sim is 10,000.
r	float	1.55e-10	Molecular radius in m
charge	float	0	Material charge in elementary charge units
max_surf_hits	int	-1	If greater than zero, limits how many times a particle can hit a surface before becoming permanently stuck
sc_steps	int	1	Number of subdivisions used for subcycling check, <2 disables s/c (disabled by default for molecules)
sc_max_angle	float	30	Max angle (deg) between sub and non-subcycled velocity and position vectors
sc_max_dist	float	0.01	Max distance (m) between final position with and without s/c

Example

```
molecular_mat{name:hc1, weight: 94, mpw: 1e13, Ea:12, C_dif:1, r:1.55e-10}
```

II.d.3) PARTICULATE_MAT

Specifies a particulate material. These are much larger than molecules and are affected by drag and gravity.

Key	Type	Default	Description
name	string		Material name
density	float2	1000	Density range in kg/m^3
charge_density	float2	0	Particulate surface charge density range in C/m^3
min_bounce_speed	float	0.001	Minimum reflected speed (in m/s) that a particulate must have not to stick to a surface
max_surf_hits	float	-1	If greater than zero, limits how many times a particle can hit a surface before becoming permanently stuck

sc_steps	int	2	Number of subdivisions used for subcycling check, <2 disables s/c (enabled by default for particulates)
sc_max_angle	float	30	Max angle (deg) between sub and non-subcycled velocity and position vectors
sc_max_dist	float	0.01	Max distance (m) between final position with and without s/c
Example			
`particulate_mat{name:flakes, density_range:[1000,3000]}`			

II.e) Interactions

This section covers operations that specify how materials interact with each other.

II.e.1) DSMC_PAIR

Turns on DSMC collisions between two flying materials.

Key	Type	Default	Description
mats	s_list		List of two materials for the collision
process	string	VHS	Collision process. Currently only VHS (Variable Hard Sphere) is supported.
sigma	string		Cross-section model, one of [const, pow, vhs]
sigma_coeffs	f_list		List of coefficients for the cross-section model, $\sigma_{const} = c_0, \sigma_{pow} = c_0 v_r^{c_1}, \sigma_{vhs} = f(c_0 : d_{ref}, c_1 : w)$
sigma_cr_max	float	1e-16	Initial value for the $(\sigma \cdot c_r)_{max}$ product

Example

```
dsmc_pair{mats:[co2,co2],process:VHS,sigma_cr_max:1e-16,sigma:VHS,
sigma_coeffs:[5.62e-10,0.93]}
```

II.e.2) MATERIAL_INTERACTION

Specifies interaction properties between material pairs. All properties can be specified as constant values or as value@time tupples to model transient behavior.

Key	Type	Default	Description
source	string		Source material name (or RegEx expression)
target	string	*	Optional. Target material (or regex). If not specified, this interaction applies to all target materials.
C_dif	tupple	0	Non-dimensional coefficient for power law model
k	tupple	-0.5	Time exponent for power law model
D_dif	tupple	0	Diffusion coefficient (m^2/s)
Ea_des	tupple	10	Activation energy for desorption (kCal/mol)
Ea_dif	tupple	10	Activation energy for diffusion (kCal/mol)
tau0	tupple	1e-13	Oscillation period of the molecule (s)
desorption_model	string	res_time	One of [RES_TIME, MURPHY_KOOP], controls surface desorption model based on residence time, or Murphy-Koop ice partial pressure model (Eq. 7 in 2005, Vol.131)

Example

```
#hc1,hc2,... interacting with al, desorption drops off with 1/t for the first 5
seconds, then switches to 1/sqrt(t)
material_interaction{source:/hc.*/,target:al,Ea_des:15,k:[-1@0,-1@5,-0.5@5.1]}
```

II.f) Surface Geometry

The following operations are used to import surfaces and to set surface component properties. Components are logical groups of surface triangles or quadrangles mapping physical regions of interest.

II.f.1) SURFACE_CLEAR

Deletes the surface mesh

Key	Type	Default	Description

Example

```
surface_clear{}
```

II.f.2) SURFACE_LOAD_INPF

Loads a surface in the INPF format generated by NX thermal solver

Key	Type	Default	Description
file_name	string		File to load

Example

```
surface_load_INPF{file_name:INPF}
```

II.f.3) SURFACE_LOAD_NASTRAN

Loads a surface mesh in NX Nastran format. Component names obtained from "$* NX mesh:comp" lines.

Key	Type	Default	Description
file_name	string		File to load
units	string	m	Dimension units, one of M,CM, MM, IN
comp_name	string	default	Default component name

Example

```
surface_load_nastran{file_name:"container.dat"}
```

II.f.4) SURFACE_LOAD_OBJ

Loads a surface mesh in the OBJ format. This format can be used to export directly from CAD programs without needing to mesh the geometry first. Component names are obtained from “g” or “o” tags.

Key	Type	Default	Description
file_name	string		File to load
units	string	m	Dimension units, one of M,CM, MM, IN

Example

```
surface_load_obj{file_name:"../part.obj",units:mm}
```

II.f.5) SURFACE_LOAD_STL

Loads a surface mesh in the STL ASCII format. This format is sometimes used to export directly from CAD. It does not support multiple components.

Key	Type	Default	Description
file_name	string		File to load
units	string	m	Dimension units, one of M,CM, MM, IN
comp_name	string	default	Component name to assign to all elements in this file.

Example

```
surface_load_stl{file_name:"part.stl",units:mm,comp_name:"part"}
```

II.f.6) SURFACE_LOAD_TRI

Loads a surface mesh in the TRI format (https://www.nas.nasa.gov/publications/software). Elements are grouped into numbered components, names for which are specified by "comp_names" property.

Key	Type	Default	Description
file_name	string		File to load
units	string	m	Dimension units, one of M,CM, MM, IN
comp_names	s_list		List of component names to assign to each comp id

Example

```
surface_load_tri{file_name:"next.tri", comp_names:[comp1,comp2,comp3,comp4]}
```

II.f.7) SURFACE_LOAD_TSS

Reads a surface file in the thermal TSS file format. This format stores geometry as an assembly of primitive analytical shapes such as cylinders, rectangles, etc. Not all TSS objects are fully implemented. Version 1.5 comprises a major update to the TSS loader.

Key	Type	Default	Description
file_name	string		File to load
group_by	string		One of [ASSEMBLY, SHAPE, OPTICS, MATERIAL, ELEMENT], controls assignment of component names to TSS objects. These groupings correspond to assembly label, shape label, shape property from ["optics", "optics_xmin", "optics_gmin"], property from "material", or the actual element id
mapping	s_list		Name of a CSV file to use for mapping. The first column (column 0) corresponds to the TSS entity specified by grouping. The column given by the second parameter (1 by default) specifies the component name.
units	string	m	Sets default units, one of [m,cm,mm,in]
tolerance	float	1e-7	Optional tolerance for removing overlapping elements if tol>0

Example

```
surface_load_TSS{file_name:"model_stowed.tssgm",  group_by:"element",  mapping:
["node_mat_mapping.csv",1]} }

#node_mat_mapping.csv
node,mat
16603,Aluminum
16604,Aluminum
32306,SLI
```

II.f.8) SURFACE_LOAD_UNV

Loads a surface mesh in the Universal format, specifically as exported by Salome. Supports datasets 2411 (nodes), 2412 (elements), and 2467 (groups).

Key	Type	Default	Description
file_name	string		File to load
units	string		Dimension units, one of M,CM, MM, IN

Example

```
surface_load_unv{file_name:"model\box.unv"}
```

II.f.9) SURFACE_LOAD_VTK

Loads a surface in a VTK .vtp or .vtu XML format. Currently only ASCII uncompressed data containing triangles or quads is supported. "Information" elements need to be manually trimmed out of the points data, if present (they are included when using Paraview Save Data feature).

Key	Type	Default	Description
file_name	s_list		File(s) to load
comp_names	s_list	default	Component names to assign to the mesh
comp_var	string		Optional CellData variable to use for assigning component names. Variable needs to contain integer values 0,1,..,num_comp_names-1. If not specified, all elements are assigned to the first component.
units	string	M	Dimension units, one of M, CM, MM, IN

Example

```
surface_load_VTK{file_name:mesh.vtp,comp_names:[chamber,pump],comp_var:zone_id}
```

II.f.10) SURFACE_LOAD_TEMPERATURE

Loads time-dependent surface temperature data. Currently supports only Fortran binary format used by SINDA with the following format:

```
NN: unsigned long number of nodes
el_id[0], el_id[1], ... el_id[NN-1]: NN unsigned long element ids
frame_time: double, time in seconds
T[0], T[1], ... T[NN-1]: temperatures for nodes
<repeat frames until EOF>
```

Key	Type	Default	Description
file_name	string		File to load
dt	float	5*3600	Only frames with time difference greater than dt are stored to reduce memory

Example

```
surface_load_temperature{file_name:"temps\element_temps.bin",dt:100}
```

II.f.11) SURFACE_PROPS

Specifies properties on one or more components (group of surface elements)

Key	Type	Default	Description
comps	s_list		Component these properties should be applied to. Can use RegEx wrapped in /../ (such as /.*/ to match all components).
mats	s_comp		String list of materials the component substrate is made of. This is currently not used but could be utilized in the future to model sputtering or other similar surface effects.
temp	tupples	300	Component temperature in K. Can specify a list of temp@time tupples such as [300@0, 100@50] for a linear ramp down from 300 to 100K over 50 seconds.
volume	float	0	Component volume used for bulk material concentration calculation. If multiple comps specified, volume for each is scaled by area ratio.
transparency	float	0	Controls fraction of particles that can pass through the component without hitting it. Useful for modeling meshes. Valid entries are 0 (opaque) to 1 (fully transparent).

c_stick	float	-1	Sets default sticking coefficient for the component group
c_rest	float	1	Sets default coefficient of restitution
c_accom	float	1	Sets default coefficient of thermal accomodation

Example

```
surface_props{comps:[/detector_.*/,wall], temp:[300@0,300@5,100@5.5]}
```

II.f.12) SURFACE_TRANSFORM

Applies transformation to surface nodes

Key	Type	Default	Description
op	string		One of [reset, scale, rotate_x, rotate_y, rotate_z, translate]. Reset op marks the starting node for the transformation. Remaining operations will operate on nodes oaded after "reset". The rotate ops perform rotation about the specified axes.
angle	float	0	Rotation angle in degrees for rotate_x/y/z
offset	float3	[0,0,0]	Vector offset for translate.
factor	float	1	Scaling factor for scale. Node positions are multiplied by this value.

Example

```
surface_transform{op:reset}
surface_load_stl{file_name:"offset.stl",units:mm}
surface_transform{op:rotate_x,angle:90}
surface_transform{op:translate,offset:[-0.6,3.223,-0.6]}
```

II.g) Sources

These ops load initial surface molecular and particulate contamination, and also include additional sources that continuously inject material into the domain.

II.g.1) DETACH_PARTICULATES

Releases surface particles generated by LOAD_FIBERS or LOAD_PARTICULATES. Supports two models: CONST and KLAVINS. With Const, each particle has some user-specified probability of being released. With the Klavins model, release probability is based on a local acceleration per paper by Klavins and Lee.

Key	Type	Default	Description
comps	s_list		Surface mesh component to associate the source with
release_model	string	Klavins	Release model, one of [Klavins, const]. The constant model releases a constant fraction of particles. The Klavins model uses the Klavins-Lee expression to compute the release fraction.
release_fraction	float	0.1	Release fraction for the constant model
accel_mag	float	0	If positive, uses this value to determine detachment probability in the Klavins and Lee model. Otherwise, uses a=F/m where F is the external force at the particle location.
release_interval	int	0	Time, in seconds, over which the particles should be released, or 0 to release all on the first time step.
detach_vel	float	0.01	Velocity with which particles pop-off the surface.

Example

```
source_particulate_1246{comp:fairing, mat:flakes, parts_per_bin:100, level:500,
C:0.926, model:klavins, accel_mag: 73.6, release_interval:90}
```

II.g.2) LOAD_FIBERS

Generates fibers at a prescribed surface density. Fiber sizes are generated by sampling a random length and a random aspect ratio in the given limits.

Key	Type	Default	Description
comps	s_list		Surface mesh components to associate the source with
mat	string		Material to inject
size_range	float2	[1e-3, 2.5e-3]	Minimum and maximum fiber length
ar_range	float2	[20, 120]	Minimum and maximum fiber aspect ratio
vel_range	float2	[0.001, 0.01]	Initial velocity magnitude, m/s
surf_den	float	2000	Surface density in #/m^2 of physical fibers to generate
particle_count	int	1000	Number of simulation particles to generate

Example

```
load_fibers{comps:[top,bottom],mat:dust,ar_range:[50,200], surf_den:2000}
```

II.g.3) LOAD_MOLECULES

Adds molecular contamination to specified components. Contamination is set by specifying composition of volatile species trapped within the bulk material and the composition of a surface film. The actual outgassing is handled by SOURCE_OUTGASSING.

Key	Type	Default	Description
comps	s_list		The component these properties should be applied to. Can use RegEx wrapped in /../ (such as /.*/ to match all components).

trapped_mats	s_comp		String list of materials trapped in the component with optional fractions, such [0.9*hc, 0.1*n2] or just "hc" for homogeneous composition.
trapped_mass	float	0	Mass in kg of outgassing material trapped in the component
surf_mats	s_comp		String list of materials making up the surface layer
surf_h	float	0	Height, in meters, of a surface layer

Example

```
comp_props{name:/.*/, mat:al, trapped_mass:0, temp:300, mass_flux:0}
comp_props{name:harness, mat:al, trapped_mass:1e-4, trapped_mats:[0.6*hc1,
0.3*hc2, 0.1*n2], surf_h:0, surf_mat:hc1, temp:350}
comp{name:scav, mat:al, trapped_mass:0, temp:[300@0,300@5,100@5.5]}
```

II.g.4) LOAD_PARTICULATES_1246

Generates particles according to the IEST-STD-1246D model. Particles are created by sampling size from uniform distribution in [1,max_size]. Macroparticle weight for each particle is then set such that the analytical PAC is recovered. Particles are then randomly redistributed to surface elements according to area ratio. This approach is more noisy than the prior per-element versions but decouples the surface mesh resolution from the particle count.

Key	Type	Default	Description
comps	s_list		Surface mesh component to associate the source with
mat	string		Material of the particulates
particle_count	int	10000	Number of simulation particles to generate across all comps
level	float	500	IEST-STD-1246 level (largest particle per 0.1m^2)
slope	float	0.926	IEST-STD-1246 distribution slope, 0.926 per standard, 0.3-0.4 typical for surfaces exposed for a prolonged period
max_size	float	-1	If specified, controls the maximum particle size in micron to sample. Otherwise, defaults to level*1.2

Example

```
load_particulate_1246{comps:fairing, mat:flakes, particle_count:20000,
level:500, C:0.4}
```

II.g.5) LOAD_PARTICULATES_TAPELIFT

Generates particles according to tape lift data. The input consists of two vectors. The first one is the particle size in micrometer, and the second is the number of particles in the bin.

Key	Type	Default	Description
comps	s_list		Surface mesh components to associate the source with
mat	string		Material to inject
particle_count	int	10000	Number of simulation particles to generate
bins	f_list		Particle bin bounds
counts	f_list		Number of particulates in each bin. The length must be one less than the length of bins.

Example

```
load_particulates_tapelift{comp:fairing, mat:flakes, bins:[10,25,100,250,500],
counts:[1000,800,100,10]}
```

II.g.6) SOURCE_COSINE

Samples uniform speed particles with velocity direction following the cosine law

Key	Type	Default	Description
comps	s_list		Surface mesh components to associate the source with
mats	s_comp		List of materials to inject. Can specify a single material, or a list with relative fractions.
Q	float	0	Volumetric flow rate in m^3/s at 1atm and 293.15K
mass_flux	float	0	Mass flux in kg/m^2/s
mdot	float	0	Mass flow rate in kg/s. Either Q, mass_flux, or mdot need to be specified.
v_drift	float		Source drift velocity
start_ts	int	0	Time step to start the particle injection
stop_ts	int	-1	Time step to end the particle injection if >0

Example

```
source_cosine{comps:bottom, mats:hc1, mass_flux:1e-6, v_drift:100, stop_ts:100}
```

II.g.7) SOURCE_DROPLET

This source attempts to approximate a liquid droplet jet injected to vacuum. It is a very simplistic model and based on visual observations of such jets. It injects particles with uniform velocity but after some distance from the source, the velocities are perturbed by a small offset to model the jet break up.

Key	Type	Default	Description
comps	s_list		Surface mesh injection component
mats	s_comp		Particulate materials to inject
diam	float2	[50e-6,250e-6]	Range of droplet radii in m
mdot	float	1e-6	Injection mass flow rate in kg/s
vel	float	10	Initial injection velocity, m/s
temp	float	293	Droplet temperature in K
dispersion_r	float	0.1	Distance in m at which velocities will be perturbed
dispersion_vel	float	0.1*vel	Magnitude of velocity perturbation
start_ts	int	0	Starting time step for source injection
stop_ts	int	-1	Ending time step for source injection, or -1 to run forever

Example

```
source_droplet{comp:inlet, mdot:2.5e-4, mats:[co2], temp:253, vel:20, diam:
[50e-6,250e-6], dispersion_r:0.08, dispersion_vel:0.5,stop_ts:3000}
```

II.g.8) SOURCE_EVAPORATION

This source can be coupled with the above droplet source to convert liquid droplets to gaseous molecules. It utilizes the Hertz Knudsen model to determine the evaporation rate at a given droplet temperature. It can be optionally coupled with a Shin model to model droplet cooling [4].

Key	Type	Default	Description
source_mat	string		Particulate material of the droplets
target_mat	string		Molecular material into which gaseous molecules are added
alpha	float	0.65	Coefficient of Hertz Knudsen model
shin_model	bool	false	Controls whether the model of Shin should be used to model droplet

			cooling.
antoine_coeffs	float3		A,B,C for Antoinne vapor pressure equation, Pv=10^(A-B/(T+C))
Example			
`source_evaporation{mats:n2h4,alpha:0.65,shin_model:true}`			

II.g.9) SOURCE_MAXWELLIAN

Injects particles with velocity sampled from the Maxwellian velocity distribution function

Key	Type	Default	Description
comps	s_list		Surface mesh components to associate the source with
mats	s_comp		List of materials to inject. Can specify a single material, or a list with relative fractions.
Q	float	0	Volumetric flow rate in m^3/s at 1atm and 293.15K
mass_flux	float	0	Mass flux in kg/m^2/s
mdot	float	0	Mass flow rate in kg/s. Either Q, mass_flux, or mdot need to be specified.
temp	float		Maxwellian source temperature in K
v_drift	float	0	Source drift velocity
start_ts	int	0	Time step to start the particle injection
stop_ts	int	-1	Time step to end the particle injection if >0

Example

```
source_maxwellian{comps:spot, mats:n2i+, mdot:1.0e-9, temp:1000.0, v_drift:1,
start_ts: 2000}
```

II.g.10) SOURCE_OUTGASSING

Turns on material outgassing. The actual amount generated is controlled by surface molecular loading specified by LOAD_MOLECULES and material coefficients given by MATERIAL_INTERACTION.

Key	Type	Default	Description
comps	s_list	*	Surface components to apply the source to
model	string	POWER	Material desorption model controlling mass generation. The options include [POWER, FANG, NONE]. These models are described in the Governing Equations section. The specified model has no impact on surface desorption. Setting model to NONE is analogous to running with POWER model and having no trapped materials.
exact	bool	false	Controls whether particles with fractional material specific weight should be created. **True** implies that mass will be conserved exactly at each time step but the simulation may end up with a large number of particles as at least 1 will be created from each outgassing surface element per time step. **False** results in particles created using a stochastic method such that mass is conserved only on average.
start_ts	int	-1	Starting iteration for mass diffusion if >=0
stop_end	int	-1	Ending iteration for mass diffusion if >=0

Example

```
source_outgassing{exact:false}
```

II.g.11) SOURCE_PARTICULATE_14644

This source models particulates embedded in a gas flow with following the ISO-14644-1 clean room standard. The maximum concentration of particles (#/m^3) is given by $C_N = 10^N (0.1/D)^{2.08}$ where N is the "ISO class".

Key	Type	Default	Description
comps	s_list		Surface component from where particles are injected
mats	s_comp		List of materials to inject. Can specify a single material, or a list with relative fractions.
flow_rate	float		Volumetric flow rate of the background gas, m^3/s
flow_speed	float		Flow velocity, controls the drift speed of the injected particles. m/s
v_sigma	float	0	Standard deviation for each velocity direction used to add a random Gaussian velocity component, m/s
iso_class	float		ISO class N, valid range is [0.1:9]
exponent	float	2.08	Exponent used in the C_N equation
particle_sizes	f_list	[0.1,0.2,0.3,0.5,1,5]	Particle sizes to simulate in μm.
macroparticle_count	int	5*size(particle_bins)	Number of simulation particles to generate per time step. This number is divided evenly across all particle sizes. Higher number results in a decreased simulation noise but longer simulation times.
start_ts	int	-1	Time step to start the source
stop_ts	int	-1	Time step to end the source

Example

```
source_particulate_14644{comp:inlet, mats:dust, iso_class:6, flow_rate:10,
flow_speed:1,v_sigma:0.1,stop_it:500}
```

II.g.12) SOURCE_PARTICULATE_PPM

This source models particulates embedded in a background flow, as may be the case with cleanrooms or ECS flow. Particulates are born with random size sampled from a distribution, and initial drift velocity in the surface normal direction. Velocity can also have optional random component sampled from the Gaussian distribution. Injection mass flow rate is obtained from $\dot{m} = Q_{flow} \rho_{flow} (\text{ppm})_m \cdot 10^{-6}$. The source samples the specified number of simulation particles per time step per component (the particles are assigned simulation specific weight to satisfy the injection mass).

Key	Type	Default	Description
comps	s_list		Surface component from where particles are injected
mats	s_comp		List of materials to inject. Can specify a single material, or a list with relative fractions.
ppm_mass	float		Parts per million, per mass.
flow_density	float		Mass density of the background gas, kg/m^3.
flow_rate	float		Volumetric flow rate of the background gas, m^3/s
v_drift	float		Drift velocity of the injected particles (physically should be the same as gas flow velocity), m/s
v_sigma	float	0	Standard deviation for each velocity direction used to add a random Gaussian velocity component, m/s

size_range	float2		[min,max] range for particulate sizes, μm
size_dist	string	UNIFORM	Distribution method for size sampling, one of [UNIFORM,GAUSSIAN]
particle_count	int		Number of simulation particles to create per time step, per component
start_ts	int	0	Time step to start the source
stop_is	int	-1	Time step to end the source
Example			
`source_particulate_ppm{comp:inlet, mats:dust, ppm_mass:1, v_drift:2, v_sigma:0.1, size_range:[1,10], size_dist:gaussian,start_it:0, stop_it: 500, flow_rate:10,flow_density:1.18, particle_count: 5}`			

II.g.13) SOURCE_PLUME

Projects analytical plume per the model of Woronowicz [5]. This source computes surface flux to target components and also sets volume mesh density for visualization. No particles are generated. The source does not consider re-emission and only direct line of sight is considered.

Key	Type	Default	Description
comps	s_list		Source component, the plume will be centered at component centroid and directed in the mean normal direction.
x0	float3		Alternatively, can provide position and direction for the plume
dir	float3		Plume direction if component not specified
temp	float		Plume temperature in K
ue	float		Plume mean velocity in m/s
mdot	float		Flow rate in kg/s
start_ts	int	0	Time step to start the source
stop_ts	int	-1	Time step to end the source
Example			
`source_plume{comp:inlet, mdot:0.5e-4, mats:[co2], temp:273, ue:20, start_ts:0, stop_ts:3000}`			

II.g.14) SOURCES_CLEAR

Deletes all sources

Key	Type	Default	Description
Example			
`sources_clear{}`			

II.h) Surface Output

The following operations are used to write out the surface mesh. Some formats support only the shape information (STL, UNV, etc.) and hence are primarily useful for file conversion. For instance, a TSS-imported mesh can be exported in the UNV format for additional post-processing in Salome. Other writers (Tecplot, VTK) also include mesh-based simulation results. The Tecplot writer supports only the legacy format and is not as actively updated as the VTK writer. The VTK writer is recommended, with files then visualized in Paraview or VisIt. The table below lists the variables supported by the VTK writer. The Mat column indicates whether the variable is material-specific. Variables listed as M or P can be used to write out material data for molecular or particulate species. If the mat is omitted, the cumulative sum is saved. For example, "surf_height.hc1" is the height of molecular film of material "hc1", while "surf_height" includes all molecular species.

Name	Mat	Units	Description
area	no	m^2	Element area
normals	no	-	Surface element normal vectors (included by default)
node_id	no	-	Mesh node id
element_id	no	-	Mesh element id (included by default)
comp_id	no	-	Component group index (included by default)
temperature	no	K	Surface element temperature
cv_*	-	-	(prefix), outputs coefficient of variation (standard deviation scaled by mean) for parallel runs
surf_height	M	A	Height of a surface molecular film for a species or all cumulative
surf_mass	M	kg	Mass of the surface molecular film
surf_height_rate	M	m/s	Surface layer height accumulation rate
surf_mass_rate	M	kg/s	Surface layer mass accumulation rate
surf_mass_flux	M	kg/ (m^2/s)	Surf_mass_rate divided by element area
trapped_mass	M	kg	Mass of the trapped materials in the material substrate layer
trapped_mass	M	kg/s	Rate of mass change in the material substrate layer
PAC	P	%	Percent area coverage in % (0.1 is 0.1% not 10%)
level	P	-	Non-dimensional particulate level computed from PAC assuming C=0.926.

II.h.1) SURFACE_SAVE_STL

Writes the surface mesh (no data) in the STL format. Quads are broken down to two triangles.

Key	Type	Default	Description
file_name	string		Saves in "results/"+file_name
skip	int	-1	Frequency of saves

Example

```
surface_save_stl{file_name:"surf.stl"}
```

II.h.2) SURFACE_SAVE_TECPLOT

Saves the surface mesh and associated data in legacy Tecplot ASCII format. Outputs molecular height, and particulate levels, PAC.

Key	Type	Default	Description

file_name	string		Saves surface file in “results/”+file_name
skip	int	-1	Frequency of surface saves.
Example			
`surface_save_tecplot{file_name:"surf.dat"}`			

II.h.3) SURFACE_SAVE_UNV

Writes the surface mesh only (no data) in the UNV format

Key	Type	Default	Description
file_name	string		Saves in “results/”+file_name
skip	int	-1	Frequency of saves

Example

```
surface_save_stl{file_name:"surf.unv"}
```

II.h.4) SURFACE_SAVE_VTK

Saves the surface mesh and corresponding surface properties in a VTK format

Key	Type	Default	Description
file_name	string		File name prefix, will save in “results/”+file_name+”.vtp”
skip	int	-1	Frequency of surface saves.
format	string	binary	One of [ASCII, BINARY]. Binary data is base64-encoded.
vars	s_list		List of variables to output. See top of the section for the available list. Only geometry is saved if vars empty

Example

```
#save multiple files 100 time steps apart
surface_save_vtk{skip:100,file_name:"surf",vars:[surf_h.hc1,trapped_mass]}
run_sim{}
#save the final results
surface_save_vtk{file_name:"surf_final",vars:[surf_h.hc1,trapped_mass],
format:ascii}
```

II.i) Particle Output

These operations are used for saving particle traces and random samples for post processing.

II.i.1) PARTICLE_SAVE_TECPLOT

Saves particles in Tecplot format

Key	Type	Default	Description
file_name	string		Output file name
start_ts	int	0	First time step for output
end_ts	int	-1	Last time step for output
skip	int	100	Number of time steps between saves

Example

```
particle_save_tecplot{file_name:"particles.dat"}
```

II.i.2) PARTICLE_SAVE_VTK

Saves a random subset of particles in the VTK format

Key	Type	Default	Description
file_name	string		Saves in “results/”+file_name+”.vtp”
mat	string		Gaseous or particulate material to output
num_particles	int	1000	Number of particles to output
max_id	int	-1	Maximum particle id for picking the random particles, or -1 to use all available particles.

Example

```
particle_save_vtk{file_name:"particles",mat:flakes,skip:10,num_particles:2000}
```

II.i.3) PARTICLE_SAVE_HISTOGRAM

Saves a histogram of particles on the specified surface components. Data is stored in .csv format, with the y[i] value corresponding to the number of real particles in the x[i]-x[i+1] bin. Optionally, a species-level breakdown is possible.

Key	Type	Default	Description
file_name	string		Saves in “results/”+file_name+”.csv”
comps	s_list	*	List of component names, supports regex within slashes
mat	string		Optional material to limit output to
bins	f_list	`[1,10,25,50,100,150,250,500,750,1000,2500]`	Histogram bin bounds
start_ts	int	0	First time step for file output
end_ts	int	-1	Last time step for file output
skip	int	100	Number of time steps between saves

Example

```
surface_save_histogram{file_name:hist,comps:[fairing,detector],skip:1000}
```

II.i.4) PARTICLE_TRACE

Saves position and other data for one or more particle at each time step. Saves in VTK format.

Key	Type	Default	Description

file_name	string		Saves in in "results/"+file_name+".vtp"
mat	string		Gaseous or particulate material to output
num_traces	int		Number of particles to trace. Alternative is to specify ids directly.
max_id	int	-1	Maximum id for generating random trace ids if num_traces is specified
ids	i_list		List of particle ids to trace
skip	int	-1	Frequency in time steps of file saves. -1 saves only at end.

Example

```
particle_trace{file_name:trace,mat:flakes,num_traces:100,max_id:28736}
particle_trace{file_name:trace,mat:hc1,ids:[250,1050,1007]}
```

II.j) Volume Mesh

These operations control the creation and export of an optional volume Cartesian mesh. The following variables are currently supported. Any species variable needs to include the species name separated by a dot, for example, "nd.hc1" for hc1 number density.

Name	Sp	Units	Description
nd	yes	#/m^3	Number density (molecules per cubic meter)
u	yes	m/s	Species mean velocity vectors
t	yes	K	Species temperature from Maxwellian distribution
pressure	no	Pa	Total pressure from partial pressure sums per p=nkT
flow_vel	no	m/s	Loaded flow velocity vectors
flow_rho	no	kg/m^3	Loaded flow number density as given or computed from ideal gas law

II.j.1) VOLUME_MESH

Generates an optional volume Cartesian mesh. CTSP will then compute bulk gas properties such as density and pressure. These can useful for visualization of contaminant plumes. Volume mesh also specifies a bounding box for removing particles.

Key	Type	Default	Description
dx	float		Approximate cell spacing in x direction
dy	float		Approximate cell spacing in y direction
dz	float		Approximate cell spacing in z direction
xmin	float3		Optional xmin coordinate of mesh bounding box. If not specified, surface mesh bounding box is used.
xmax	float3		Optional xmax coordinate of mesh bounding box. If not specified, surface mesh bounding box is used.
expand	f_list	0	By default, the volume mesh bounds are identical to the surface bounding box. This option allows for the bounds to be expanded. There are three options: a) single value results in uniform offset added to all 6 faces, b) three values add the same offset to +/- X,Y,Z faces, and c) six values provide specific offset for each [-X,+X,-Y,+Y,-Z,+Z] dimension.

Example

```
volume_mesh{dx:0.01,dy:0.02,dz:0.01,expand:0.01}
volume_mesh{dx:0.1,dy:0.1,dz:0.1,expand:[0.0,0.0,0.001]}
```

II.j.2) VOLUME_SAVE_TECPLOT

Saves the volume mesh and corresponding data in a Tecplot format. Data is averaged between saves.

Key	Type	Default	Description
file_name	string		File name, results saved in "results/"+file_name
vars	s_list		List of variables to output. Some options include "nd.fm, u.fm, t.fm, pressure", where "fm" is name of a flying material.
skip	int	-1	Frequency of file saves

Example

```
volume_save_tecplot{skip:2000,file_name:"field.dat",vars:[nd.hc1]}
```

II.j.3) VOLUME_SAVE_VTK

Saves the volume mesh and corresponding data in a VTK format. Data is averaged between saves. When generating animations, it is possible to get less noisy data without using more particles by saving less frequently.

Key	Type	Default	Description
file_name	string		File name, results saved in "results/"+file_name+".vti"
vars	s_list		List of variables to output. Some options include "nd.fm, u.fm, t.fm, pressure", where "fm" is name of a flying material.
skip	int	-1	Frequency of file saves
format	string	binary	One of [ASCII, BINARY]. Binary data is base64-encoded.

Example

```
volume_save_vtk{skip:2000,file_name:"field",vars:[nd.hc1,pressure,t.hc1]}
```

III. Governing Equations

III.a) Molecular Contamination

Molecular outgassing arises from volatile gases trapped inside the material diffusing to the surface and desorbing into the gas phase. The diffusion rate between the bulk and the substrate, and the desorption to the gas phase is a function of concentration gradients and surface temperature. The model in CTSP attempts to captures this basic physical process. As plotted in Figure 10, all surface objects are assumed to consist of a native solid substrate containing an arbitrary heterogeneous mixture of trapped gases (region I). At the surface is a thin layer composed of an arbitrary combination of molecular species (region II). The surface layer can also contain particulates. Molecules and particulates can leave the surface layer and enter the gas phase (region III). Material in the gas cavity eventually encounters other geometry components (unless they leave the computational domain through open boundaries) and possibly adsorbs to that component surface layer. Molecules can also migrate from the surface back to the substrate. For generality, we also allow the surface to generate additional material according to some prescribed flux, as is the case with openings venting an internal cavity. This prescribed flux is also useful for representing objects for which a QCM-measured outgassing rate is available.

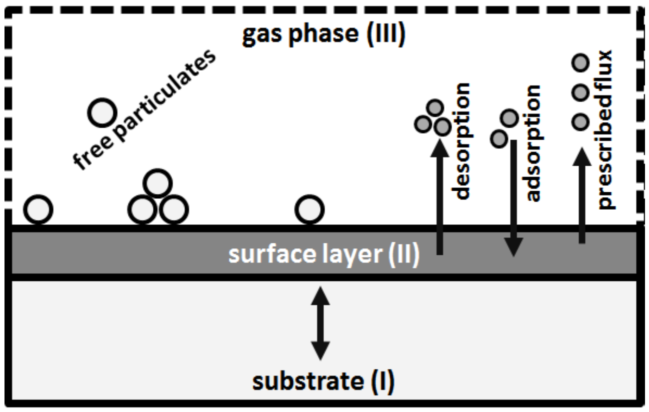


Figure 10. Overview of the surface model used by CTSP. All components are assumed to consist of a solid substrate containingsome trapped contaminants and a surface layer in contact with the gas phase.

The initial composition of the substrate and surface regions is specified in the input file for each geometry component as illustrated below:

```
surface_props{comp:ebox, mats:ti, temp:[250@0,300@2,350@5]}
load_molecules{comp:ebox, trapped_mass: 1e-4, trapped_mats: [0.8*hc1,0.1*water,0.1*n2],
surf_h:5e-10, surf_mats:water}
```

The first command specifies the composition of the surface substrate (100% Titanium). It also specifies a time-varying component temperature. The next command load molecular contaminants into the substrate and the surface layer. Here the amount of trapped material is specified by providing the total weight of the contaminant population to be distributed across all surface elements of the listed component. The surface contamination is specified by listing the height of the surface layer. Properties for materials such as ``hc1" and ``water" need to be specified prior to this step. Migration of mass from the substrate to the surface layer is governed by molecular diffusion. CTSP implements two models. First, the code supports a simple power law commonly used by the contamination community [7],

$$\frac{dN}{dt} = C_{pow} N \exp\left(-\frac{(E_a)_{dif}}{RT}\right) t^k$$

where C_{pow} is a reaction constant, N is the number of molecules left in the substrate, $(E_a)_{dif}$ is the activation energy for the diffusion process (in kCal/mol), R is the gas constant (in kCal/mol/K), and T is the component temperature (K). The coefficient for the time term t is taken to be $k=-0.5$ for a diffusion-limited process. The reaction constant and activation energy can be determined experimentally from the industry standard ASTM-E1559 test [8]. This exponential model does not support mass diffusion from the surface layer back to the solid material.

CTSP also implements a detailed model based on the work of Fang [9]. In their work, the authors put forth an analytical model for outgassing and contamination transport in a domain also consisting of a source solid material, a source layer, a cavity, and a target surface. An advection-diffusion model was used to treat the mass transport in the cavity and analytical model was utilized to capture the adsorption of gas material to the surface. These two processes are handled by the kinetic particles in CTSP. The concentration gradient in the solid region I is given by the diffusion equation

$$\frac{\partial n_c}{\partial t} = \nabla\cdot(D\nabla n_c) + f_1$$

Here n_c is the contaminant number density (m^{-3}) inside the region I and D is the diffusion coefficient (m^2/s). It is important to realize that computing the concentration gradient requires knowing the *volume* of the component, which is not available from the surface mesh data. Therefore, it is critical that component volumes are set appropriately using the SURFACE_PROPS command. f_1 is a source term allowing for additional contaminant generation. In our formulation, we set $f_1 = 0$. Integrating through a volume of the solid region, we have $dN/dt = (\partial n_c/\partial n)A$. Here $N$is the total number of contaminant molecules inside the solid region I, $N = \int_V n_c dV$and A is the area of interface between regions I and II. The divergence theorem was used here, and we assumed that $(\partial C/\partial n)$ is spatially uniform across the interface. All other boundaries are impermeable. This equation describes the number of molecules lost by the solid region. To satisfy mass conservation, it also governs the number of molecules gained by the surface layer. The surface layer is also incremented by adsorption of molecules from the gas phase and is similarly depleted by desorption of surface film into the gas phase. We label these two terms Γ_a and Γ_d. The time evolution of the surface number density θ(m^{-2}) is thus given by

$$\frac{\partial\theta}{\partial t} = -D\left(\frac{\partial n_c}{\partial n}\right) + \Gamma_a - \Gamma_d$$

We next assume that the concentration gradient $-(\partial n_c/\partial n)$ is proportional to the amount of material inside and on the surface of the object. We define a sorption function $H(n_c,\theta)$ such as

$$-\frac{\partial n_c}{\partial n} = H(n_c,\theta) = n_c - \gamma\theta$$

Here γ (m^{-1}) is an ``equilibrium partition coefficient'' such that at equilibrium $n_c = \gamma\theta$. We thus have $d\theta/dt = D(n_c - \gamma\theta) + \Gamma_a - \Gamma_d$. From this expression we see that at equilibrium we also require $\Gamma_a = \Gamma_d$. Satisfying this requirement is demonstrated in Example 1. The desorption flux is given by $\Gamma_d = \theta_1/\tau_r$ where

$$\tau_r = \tau_0 \exp\left(\frac{(E_a)_{des}}{RT}\right)$$

is the molecular residence time. The parameter τ_0 is the vibrational period of the molecule with typical values around 10^{13} s [7] and $(E_a)_{des}$ is the activation energy for the desorption process. The parameter $\theta_1 = \min(\theta, \eta/(\pi r^2))$, where $\eta_0/(\pi r^2)$ is the maximum number of molecules per unit area. Here r is the molecular radius and η_0 is a scaling ``packing'' factor. Both of these values are user parameters. This formulation limits the desorption rate to that corresponding to a fully occupied monolayer if surface film thickness exceeds a single monolayer. The adsorption flux Γ_a term is described in a subsequent section on particle impact.

The above algorithm is implemented numerically as follows. We loop through all surface elements. On each, we first use the exponential or the detailed model to compute the total number of molecules diffusing to (or from) the surface layer. Here the code takes into account the relative molar composition of materials in the substrate and the surface layer. Next, the appropriate number of molecules is transferred to (or from) the surface layer. Note, with the "power" model, there is no transfer from the surface to the bulk. We next compute the number of real molecules to desorb, $N_d = (N_{surf}/\tau_r)\Delta t$. The corresponding number of simulation particles is $M = N_d/w_{mp}$. Generally, N_d will not be evenly divisible by the macroparticle weight w_{mp}.The code supports two injection schemes. In the exact scheme, as many particles as possible will be created with the default w_{mp} and then an additional particle will be created with some fractional weight. The second stochastic approach does not create fractional weight particles but instead uses random numbers to create full weight particles with probability N_d/w_{mp}. This second approach is mass conserving only on average at steady state but avoids the excessive number of simulation particles that may result with the first exact model.

III.b) Particulates

Just as with molecular contaminants, simulating particulate redistribution requires models for generation, transport, and deposition. Particulate contamination is traditionally divided into two categories: ``standard'' particulates and fibers. Fibers are particulates with aspect ratio $AR \equiv l/d > 10$ [10]. Fibers can be characterized by specifying their count per unit area along with the observed size ranges. CTSP implements a source for fibers which generates the user specified surface concentration with lengths and aspect ratios sampled from the uniform distribution within the user specified limits. The model for the standard particulates is more involved, and is described next.

The contamination control community often uses the IEST-STD-1246D standard to describe the size variation of particulates on a surface [11]. This standard provides a cumulative distribution function for particle counts given by

$$\log_{10}(N_{cum}) = C\big(\log_{10}^2(L) - \log_{10}^2(l)\big)$$

N_{cum} is the total number of particulates with sizes greater or equal to l per 0.1 m^2 (prior versions of this standard used the same model but the count was per ft^2). The particulate length l is given in micrometers . The parameter L is the surface cleanliness level and is the ``slope'' of the distribution. The standard assumes $C = 0.926$ for freshly cleaned surfaces, however real-world tape lifts indicate values closer to 0.4 [7]. Both C and L are user inputs.

We are generally more interested in the actual number of particulates of some size. This value can be approximated by subtracting two cumulative counts offset by 1 μm,

$$N = 10^{C\left(\log_{10}^2(L)-\log_{10}^2(l)\right)} - 10^{C\left(\log_{10}^2(L)-\log_{10}^2(l+1)\right)}$$

The distribution given by the above equation indicates that the concentration of particulates increases exponentially as the particle size decreases. However, small particles are also less likely to come off the surface due to an increased ratio of adhesion to detachment forces. Particles unable to detach do not contribute to contaminant redistribution. Determining what fraction of particles of some size l detaches remains a significant uncertainty in our model. In 1987, Klavins and Lee studied the problem of surface adhesion by applying static loads to a test sample placed in a centrifuge [12]. These measurements were performed at loads up to 10^5g and showed large variation in the detachment probability between individual tests. This variation is expected, since probability that a particle detaches is strongly influenced by the local surface roughness, particle shape, and orientation. Environmental effects such as humidity or electrostatic charge also play a role. Hence, at best, only a simple macroscopic estimates of detachment probability Φ can be made. The authors found it to follow

$$\phi = \left[1 + \left(\log\left(\frac{a_d}{a_m}\right) / \left(\sqrt{2}\sigma_0\right)\right)\right]/2$$

where a_d is the applied acceleration and $\sigma = 1.45$ is the standard deviation. The parameter a_m is the mean acceleration for 50% removal, and is given by $(85.07/L)^{4.08}$ for particles smaller than 42 μm and $(52.37/L)^{13.6}$ otherwise[2]. The distribution of particles released from the surface can then be obtained by multiplying the initial size distribution with the release probability, $N\Phi$. These expressions are visualized in Figure 11 for L=400, C=0.926, and a=5 g. As can be seen, the release model predicts all particles larger than 100 *µm* detach given the 5 g acceleration. On the other hand, the detachment probability is less than 30% for particles smaller than 20*µm*. The light gray dashed line corresponds to the distribution that needs to be generated by the particulate source.

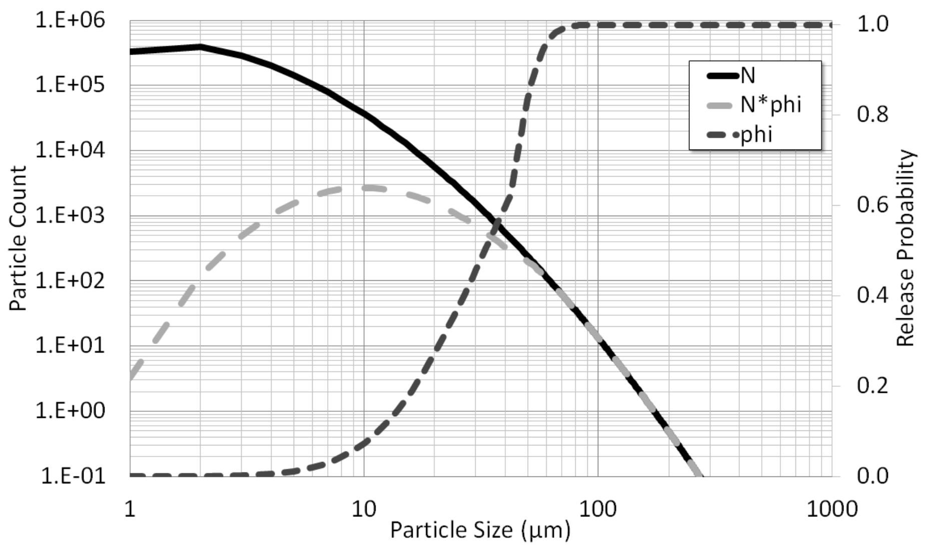


Figure 11.Particle counts, release probability, and the number of released particles for $L=400$, $=0.926$ and $a=5$ g.

For generality, we prefer to evaluate Φ with a_d obtained from the sum of forces acting at the particle injection location. We can generate particles by sampling sizes from the IEST-STD-1246D distribution and for each compute the release probability. Without changes, this approach is however not practical. Considering L=400 and C=0.926, we see that for every 200 micron particle, there are 13,079 20-*µm* particles. We thus need to sample, on average, at least 13,079 particles per surface element to sample a single 200 *µm* particle. The dynamics of the large and small particles are sufficiently different making it important that all sizes are represented. We thus instead generate a constant number of simulation particles in each of the following bins: [1,10), [10,25), [25,50), [50,100), [100,250), [250, 500), [500,750), [750,1000) and use the macroparticle weight w_{mp} to recover the original distribution function. In each bin, particle sizes are sampled from the uniform distribution. This approach assures that the simulation contains a statistically significant number of particles off all sizes. Simulations presented in this report used 100 particles per bin for a total of 800 particles per surface element. All particle in a single bin share the same macroparticle weight. The weight is set such that the total percent area (PAC) coverage represented by the particles in the bins equals to the PAC given by the original distribution. In each $[l_1, l_2)$ bin, we first sample N_p random sizes, and then compute

[2]The exponents on these two expressions are switched in the reference paper due to an apparent type. The formula shown here is consistent with graphs in the paper.

$$W\sum_{p}^{N_p} A_p = A_{ele}\sum_{l=l_1}^{l_2-1} N(l)A(l)$$

Here A_p is the cross-sectional area of p-th particle and A_{ele} is the surface area of the element. $N(l)$ is given by the IEST-STD-1246D distribution equation and $A(l)$ is the cross-sectional area of a particle of size l . For consistency, it is imperative that the same expression is used to compute A_p and the total distribution-based area. The aspect ratio $AR = l/d$ varies with particle size. As summarized by Perry [13] ,

Particle size (microns)	Aspect Ratio
1-69	$AR = L^{0.1088}$
70-175	$AR = L^{0.8804}/26.53$
176-346	$AR = L^{2.589}/181500$
347+	$AR = L^{0.8964}/9.138$

The cross-sectional area is then $A = d(l-d) + \pi d^2/4$. The volume becomes $V = (l-d)\pi d^2/4 + \pi d^3/6$. The particle volume is used to set the mass from the material density, $m = \rho V$. Note that L corresponds to the cross-sectional area parallel to the particle long axis. For aerodynamic drag computations, we assume that the particle aligns with the flow so that $A_d = (\pi d^2)/4$.

Particles generated by the above algorithms are initially attached to the surface. The code next iterates over all particles and for each computes the detachment probability Φ from the Klavins and Lee model or from a constant value specified by the user. The particle is detached if $\Phi \geq R$, where R is a random number. Otherwise, the particle cross-sectional area is used to update the source element percent area coverage. The model of Klavins and Lee does not offer any insight into detachment rate. It is not clear whether particles detach instantly or whether the detachment takes place over an extended period of time. The source model thus implements two schemes. In the first one, all particles able to leave do so at the first time step. The second approach models uniform detachment rate over a finite period of time. In this approach, Φ/n_{it} is compared to R, with n_{it} being the number of time steps over which detachment is considered. The second approach is useful in simulations with time-varying gravitational or aerodynamic environments such as in the payload fairing during spacecraft launch.

III.c) Particle Motion and Surface Impact

Once particles are generated by the respective sources, their positions are updated by numerically integrating $d\vec{x}/dt = \vec{v}$. Velocity is integrated from $d(m\vec{v})/dt = \sum \vec{F}$ using either the second order Leapfrog method or the fourth-order Runge Kutta method. $\sum \vec{F}$ is the sum of all forces acting on the particle and m is the particle mass.The total force acting on particles is $\sum \vec{F} = \vec{F}_g + \vec{F}_d + \vec{F}_e + \vec{F}_r + \vec{F}_x$ where the terms correspond to gravitational, aerodynamic drag, electrostatic, solar radiation forces, and a general user defined external force. Orbital motion of the parent body can also be defined to model particulate return on orbit crossings. The following expressions are used to evaluate the forces:

$$\vec{F}_g = m\vec{g}$$

$$\vec{F}_d = \frac{1}{2}\rho_g C_d A_d \frac{(\vec{v}_a - \vec{v}_p)^2}{|\vec{v}_a - \vec{v}_p|}$$

$$\vec{F}_e = q\vec{E}$$

$$\vec{F}_r = \frac{A_d E_f}{c}\frac{\vec{r}_s}{|\vec{r}_s|}$$

$$\vec{F}_x = \vec{F}_x(t)$$

The drag force is computed using a drag coefficient for a flow past a sphere from the model of White [14],

$$C_d = \frac{24}{R_e} + \frac{6}{1+\sqrt{R_e}} + 0.4 \qquad R_e < 2 \times 10^5$$

The Reynolds number is computed from $R_e = \rho_g ul/\mu$, where ρ_g is the gas mass density, u is the magnitude of relative velocity between the particle and the ambient gas, l is the particle major length, and μ is the dynamic viscosity of the gas. A_d is the cross-sectional particle area for drag computations. As mentioned previously, CTSP assumes that particles can be modeled as cylinders with spherical end caps, and thus $A_d = \pi b^2/4$, with b being the minor length (cross-section diameter). Additional work is needed to quantify the error introduced by using the sphere drag coefficient for particulate motion.

In the electrostatic Lorentz force, q is the particle charge and $\vec{E}$ is the electric field. For molecular materials, charge is set directly in the input file. For particulates, we instead specify the surface charge density, and charge is then computed by multiplying through the particle total surface area, $q = \sigma A_{tot}$. The radiation pressure expression assumes the particle is uniformly spinning so the incidence angle can be averaged out and $\vec{r}_s$ is a vector from the source to the particle. To improve performance, the code applies forces only if appropriate data are specified. For instance, gravitational force is considered only if the world acceleration vector $\vec{g}$is nonzero. Aerodynamic drag is computed only if $\rho_g > 0$. Intermolecular collisions in regimes outside the free molecular flow regime can also be modeled using the DSMC No Time Counter (NTC) scheme of Bird [3].

During each particle push the code checks for surface interaction. If the particle strikes a surface, an impact handler is used to determine whether the particle sticks to the surface, and if not, to set its post-impact velocity. Surface impact is the only part of the code where a different algorithm is applied to particulate and molecular contaminants. When a molecule impacts a surface, CTSP first determines the probability of the molecule striking the native material instead of another adsorbed molecule. This probability is given by $P = N_2 \pi r^2 (\eta_0 A_{ele})$, where N_2 is the number of molecules adsorbed to the surface. CTSP then computes residence time using the previously noted equation at the temperature of the impacted surface element and appropriate activation energy. The molecule is re-emitted if $\tau_r/\Delta t \geq R$ where R is a random number. In this sense, $\tau_r/\Delta t$ is analogous to the sticking coefficient α_{sc} commonly utilized in other mass transport codes. For generality, CTSP allows the user to define a fixed α_{sc}, in which case the prescribed value will be used instead of the temperature-based model. For rebounding particles we first sample the new velocity magnitude v_t from the Maxwellian speed distribution function at the surface temperature. The particle's new speed is set to $v + \alpha_a (v_t - v)$where α_a is the thermal accommodation coefficient. The new velocity direction follows the cosine law sampled according to a model of [3]. On the other hand, if $\tau_r/\Delta t < R$, the particle is adsorbed to the surface layer. Computationally, this involves removing the particle from the simulation and incrementing the surface count of particle's $N\Phi$ by the particle specific weight w_{mp}. This term correlates to the adsorption flux, $\Delta N_2 = \Gamma_a A_{ele} \Delta t$. For post-processing, the surface layer molecular count can be converted to a thin film height by assuming spherical molecules,

$$h = N_2 \frac{(4/3)\pi r^3}{A_{ele}}$$

Different model is used for particulates. Particulate post-impact velocity is given by a user-defined coefficient of restitution, $\alpha_r = v_2/v_1$. This parameter controls the ``bounciness'' of the particle. The particulate is allowed to leave until the post-impact velocity falls below some user defined threshold. The default value used in most of our simulations is 0.001 m/s. The mass of the particle is assumed to remain constant (i.e., the particle does not break up upon impact). Particles without sufficient post-impact velocity are deleted from the simulation and their cross-sectional area is used to update the target surface element percent area coverage,

$$PAC = 100\frac{\sum A_p}{A_{ele}}$$

PAC can then be converted to the corresponding cleanliness level given some slope C . Perry [13] provides a simple to use equation to map PAC to level. However, in experimenting with that model, we found the results to deviate by up to 5% for some values of C . Therefore, CTSP uses a lookup table to map PAC to level. The codes pre-computes the PAC for a range of levels and then uses linear interpolation to map the PAC to the level.

III.d) Subcycling

Since version 1.0, the code supports particle push subcycling. In PIC or DSMC simulations, the time step is typically set based on volume cell size such that particles do not travel more than a cell per time step. This rule is not applicable to CTSP due to the mesh-free nature of the particle push. Therefore, the code attempts to find the correct time step automatically. The user provided time step is used to integrate position and velocity through a "full step". The code then performs the integration using a specified number of steps. With two steps, the integration is performed through two half time steps. The resulting position and velocity vectors are compared to each other and if within a given tolerance (30 degrees by default), the code uses the selected time step. Otherwise, the process repeats with 0.5*dt. The process repeats until an optimal time step has been found. This is illustrated in Figure 5. The colored trace corresponds to a simulation run with dt=0.1 s, while the black trace is dt=1e-4s. Without subcycling, the 0.1s time step would produce an incorrect trace. With subcycling, both simulations produce an identical trace. Subcycling is turned on automatically for particulates but is disabled by default for molecules.

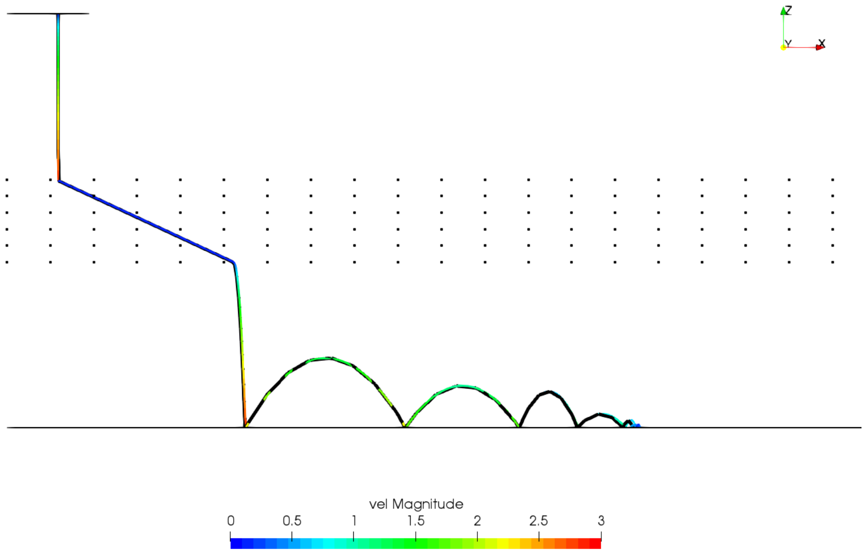


Figure 12. Identical particle trace with 0.1 and 1e-4 s time step

IV. Examples

The code is shipped with a several examples, three of which are described in detail below. These examples are also discussed in [1]. The first one verifies the molecular outgassing model by computing the steady-state distribution of a contaminant inside a closed vessel. The second example simulates a commonly encountered engineering task: using a quartz crystal microbalance (QCM) to characterize the outgassing rate of a test article exposed to vacuum. The final example demonstrates the use of gaseous purge to reduce infiltration of particulate contaminants.

IV.a) Vacuum Bakeout

We start by considering a common task encountered by contamination engineers: using a QCM to determine the outgassing rate of some test article during a vacuum bakeout. The files for this example are located in **dat/belljar.** The geometry consists of a bell jar containing a platen supporting a harness (the test article), a scavenger plate, and a QCM. The bell jar is pumped by a cryopump connected by a long duct. The CAD drawing and the corresponding surface mesh are shown in Figure 13.

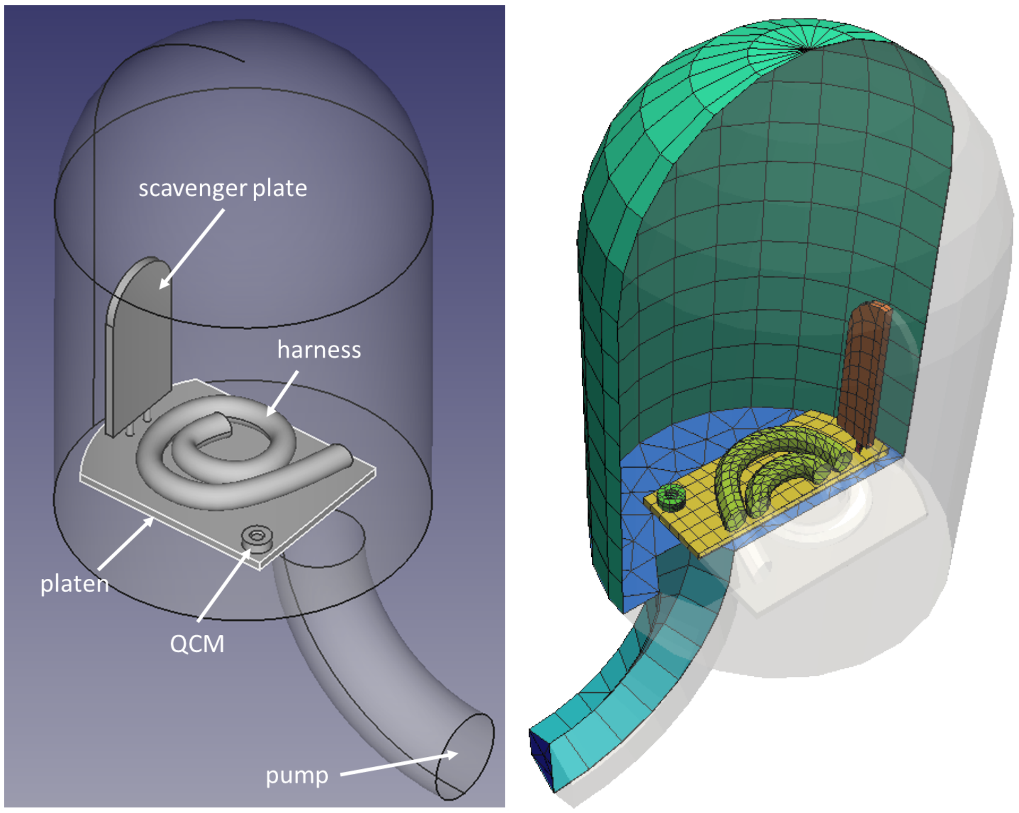


Figure 13. Geometry and the corresponding mesh for the belljar example

From mass conservation, we know that $\Gamma_h A_h - \Gamma_q A_q - \Gamma_p A_p = 0$, where Γ is flux and A is area, and the subscripts correspond to the harness, QCM, and the pump. For equally cold sinks, $\Gamma_q = \Gamma_p$. The mass balance equation can then be rewritten as $\Gamma_h A_h k_{h\to q} = \Gamma_q A_q$ where

$$k_{h\to q} = \frac{A_q}{A_q + A_p}$$

is the view factor. In other words, knowing the view factor, we can relate the deposition rate on the QCM to the outgassing rate of the hardware (the harness). The difficulty arises in that we do not have a value for the effective pump area A_p. This value will always be smaller than the cross-sectional area of the physical pump since pump effectiveness will be reduced by the duct conductance, pumping speed inefficiencies, and so on. One way we can obtain this value experimentally is by introducing a secondary cold surface, such as a scavenger plate [8]. Figure 14 compares the sources and sinks in a vacuum chamber without and with a scavenger plate.

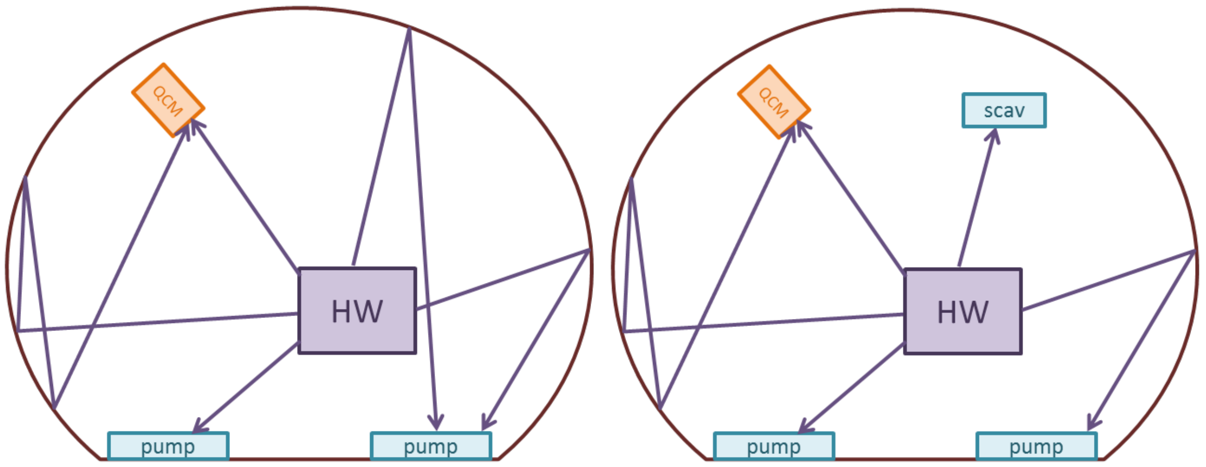


Figure 14. Chamber schematic without and with a scavenger plate

Again writing the mass conservation relationship for these two cases, we have

$$\Gamma_h A_h \left(\frac{A_q}{A_q + A_p}\right) = (\Gamma_q)_1 A_q \qquad \text{(Ex-1)}$$

$$\Gamma_h A_h \left(\frac{A_q}{A_q + A_p + A_s}\right) = (\Gamma_q)_2 A_q \qquad \text{(Ex-2)}$$

Where $(\Gamma_q)_1$ and $(\Gamma_q)_2$ are QCM fluxes without and with the scavenger plate. We now have two equations for two unknowns. Combining them, we obtain an expression for the effective pump area.

$$A_p = \frac{A_q\left[(\Gamma_q)_1 - (\Gamma_q)_2\right] + A_s(\Gamma_q)_2}{\left[(\Gamma_q)_1 - (\Gamma_q)_2\right]} \qquad \text{(Ex-3)}$$

Just as in an experiment, we start by collecting steady-state QCM flux with the scavenger at room temperature, $(\Gamma_q)_2$. We then numerically flood the scavenger by dropping its temperature to a cryogenic value cold enough to collect the contaminant. This will give us a value for $(\Gamma_q)_2$. Using combined areas of the QCM and scavenger mesh elements, we can then compute the effective pump area. Finally using this value we can numerically correlate Γ_q to Γ_h.

The simulation input file, **ctsp.in**, is listed below

```
#set some option
options{randomize:true, num_threads:4, log_level:info}

#load surface
surface_load_unv{file_name:"jar.unv"}
surface_load_unv{file_name:"harness.unv"}
surface_load_unv{file_name:"platen.unv"}
surface_load_unv{file_name:"scav-both.unv"}
```

```
#flush QCM with the platen otherwise particles get stuck in a 0.1mm gap
surface_transform{op:reset}
surface_load_unv{file_name:"qcm.unv"}
surface_transform{op:translate, offset:[0,0,-0.000095]}
surface_save_vtk{file_name:geom,format:ascii}

#specify volume mesh
volume_mesh{dx:0.01,dy:0.01,dz:0.01,expand:0.01}

#specify material properties
solid_mat{name:al, weight: 100}
molecular_mat{name:hc1, weight: 94, mpw: 2e10, r:1.55e-10}
molecular_mat{name:n2, weight: 28, mpw: 5e14, r:1.55e-10}

#set material interactions
material_interaction{source:hc1, Ea_des:12, C_pow:1}
material_interaction{source:n2, Ea_des:3.5, C_pow:0.01}

#specify component data, scavenger transition to 100K at t=10s
surface_props{comps:/.*/, mat:al, temp:350, c_outgas:0.0}
surface_props{comps:scav, mat:al, temp:[350@0,350@1,0@1.0001]}
surface_props{comps:pump, mat:al, temp:0}
surface_props{comps:qcm, mat:al, temp:0}
surface_props{comps:harness, mat:al, temp:350}

#example of setting initial molecular loading, although we use cosine source instead
#load_molecules{comps:harness, trapped_mats:[1.0*hc1], trapped_mass:0, surf_mats:hc1,
surf_height:0}
#source_outgassing{model:pow}

#set contaminant injection settings
source_cosine{comps:harness, mass_flux:1e-8, mats:[1.0*hc1], v_drift: 100}

#save animation data
particle_trace{file_name:trace,mat:hc1,ids:3}
surface_save_vtk{skip:200,file_name:"surf",vars:[surf_height.hc1, temperature]}
volume_save_vtk{skip:200,file_name:"field",vars:[pressure,nd.hc1,t.hc1,u.hc1]}

#save restart data every 5000 time steps
restart_save{skip:5000}

#run simulation of 2s of real time, show/save log every 5 time steps
run_sim{dt:2e-4,num_ts:10000, log_skip:5}
```

For this particular example, the surface mesh was split into multiple files: *jar.unv, harness.unv*, *platen.unv*, *scav.unv*, and *qcm.unv*. We load these surfaces using **surface_load_unv**. While loading the QCM geometry, we also translate the QCM slightly in the –z direction using **surface_transform{op:reset}** and **surface_transform{op:translate}**. This translation is needed since the QCM, as loaded, is not perfectly flushed with the platen but instead there is a tiny (<0.1mm) gap separating the two. During the simulation, some particles would eventually reach this gap. These particles continue bouncing between the platen and the bottom of the QCM, noticeably slowing down the simulation as many surface hits need to be checked during a single time step. Next a volume mesh is created using **volume_mesh**. We will use the mesh to compute bulk gas properties such as density and pressure. Materials are specified next using **solid_mat** and **gas_mat**. The latter command generates some hypothetical 94 AMU hydrocarbon with activation energy 12 kJ/mol used on impact with all materials. This value of E_a will prevent the molecule from sticking at room temperature surfaces. The specific weight is set to 5e13 real molecules per simulation particle. Higher value will result in the simulation running faster at the expense of noisier results.

Base materials and surface temperatures are then assigned to surface components using **surface_props**. The first command uses a regular expression to assign default values to all surfaces. Specifically of interest is that surface temperature is set to 300K. The pump and QCM crystal are assigned cryogenic temperatures. The pump is set to 15K while the QCM is set to 100K. This difference has no impact on this particular setup since our example hydrocarbon sticks equally well to both surfaces. The difference may be important if we were interested in including lighter gases like nitrogen, in which case the higher QCM crystal temperature could be used to selectively avoid deposition of that gas. A list of tupples is used to set temperature on the scavenger. The property *temp:[300@0,300@10,100@10.5]* will set the scavenger temperature to 300K at t=0s. CTSP linearly interpolates between the provided data, and hence temperature will remain 300K until t=10s. It will then rapidly drop to 100K at t=10.5s. The scavenger will then remain at this final temperature until the simulation finishes.

All components by default contain zero trapped mass and zero mass flux. We could next assign some trapped mass to the *harness*, but instead of doing that, we instead prescribe a constant 1e-8 kg/m^2/s constant mass flux out of all surface elements on this component with the help of **source_cosine**. The outgassed material will consist of 100% "hc1" molecules. We next enable periodic saving of surface and volume files using **surface_save_vtk** and **volume_save_vtk**. For now, it is not necessary to specify list of variables to save on the surface as the code saves all available data. Variables need to be specified for the volume mesh, however. We specify to output total pressure, and number density and temperature of the "hc1" gas material. Data will be saved every 10000 time steps, with data averaged between the saves.

This simulation may take about an hour to complete. Figure 15 shows the progression of surface contaminant thickness and the hydrocarbon gas number density visualized in Paraview. The first plot corresponds to a time shortly after the simulation starts. We can notice that all surfaces are colored white, indicating no hydrocarbon deposition, with the exception of the QCM crystal and the pump. The second plot shows the chamber just prior to the scavenger activation. The contaminant number density has increased and so has the deposition thickness on the QCM and the pump. All other surfaces still show no deposition. The third plot shows the state as the scavenger starts transitioning cold. We can immediately note the decrease in the contaminant number density (which corresponds to a decreased chamber pressure) as well as the presence of the contaminant on the scavenger plate. The final picture shows the chamber at an even latter time. The contaminant thickness on the scavenger has increased and the chamber pressure has dropped even further.

At a 5 time step interval, CTSP saves time dependent global data to **ctsp_diag.csv**. A snippet is below.

```
it,time,np.HC1,real.HC1,trapped_mass.JAR,surf_h.JAR,trapped_mass.JAR_BASE,surf_h.JAR_BAS
E,trapped_mass.JAR_HOSE,surf_h.JAR_HOSE,...
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
5,6.94444e-007,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0
```

The fields include the current time step, current time, the number of simulation and real molecules, as well as the total mass of a molecular contaminant on each component. We can then plot the "trapped_mass.QCM" column against the time step "it" to generate a plot as shown in Figure 16. We can clearly see three different regimes. First, between time step 0 and 20000, the deposited mass grows in a non-linear fashion. Subsequently, growth reaches constant rate. This second region corresponds to the steady state with the scavenger turned off. The slope, $\dot{m}_q$, can be converted to flux using $(\Gamma_q)_1 = (\dot{m}_q)_1/A_q$. We obtain A_q, the total surface area of QCM crystal mesh elements by reviewing **ctsp.log**:

```
      QCM (8), area:    1.107917e-004
```

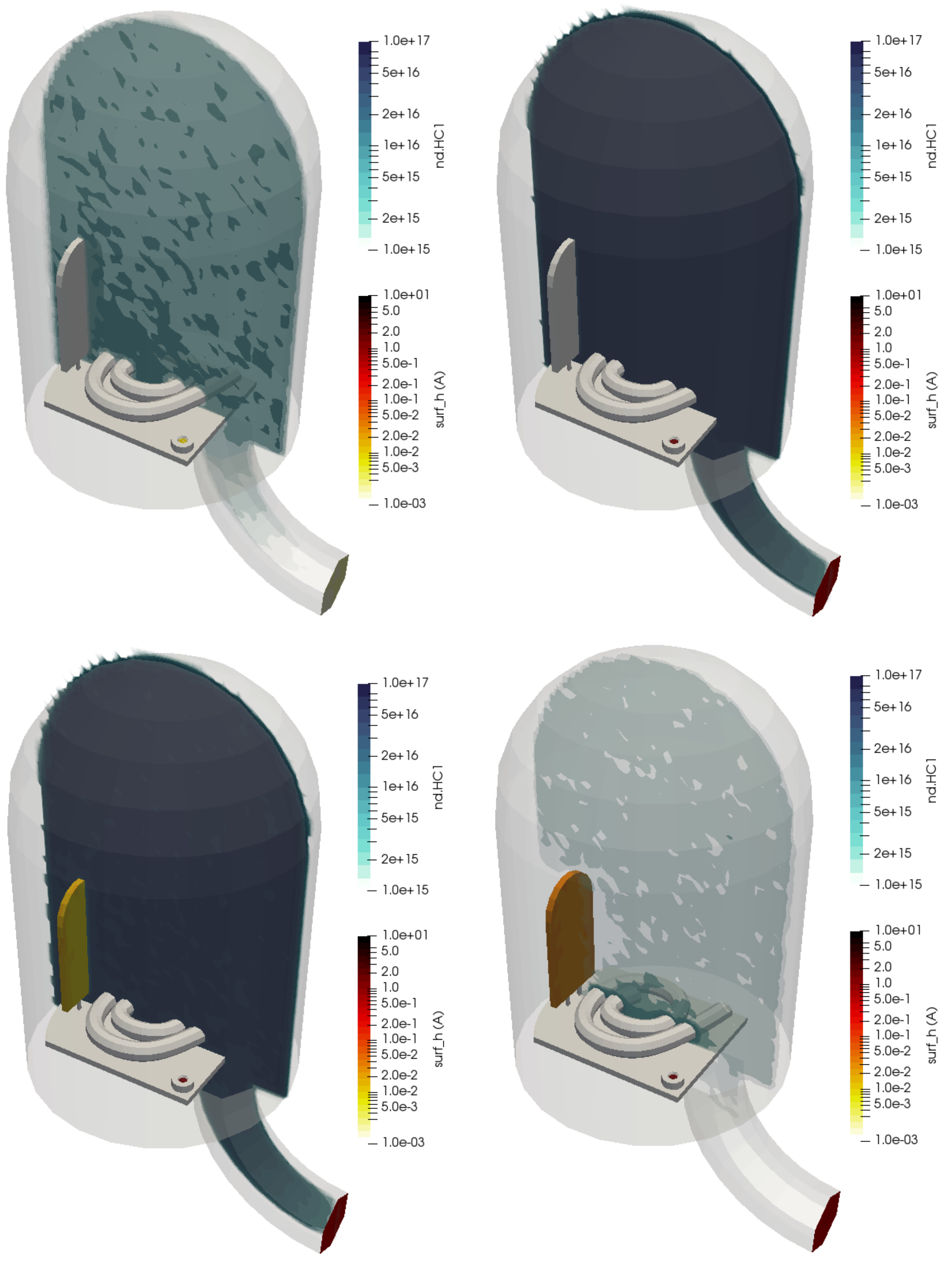


Figure 15. Hydrocarbon density and surface layer height shortly after simulation starts, right before scavenger activation, right after scavenger activation, and at a later time

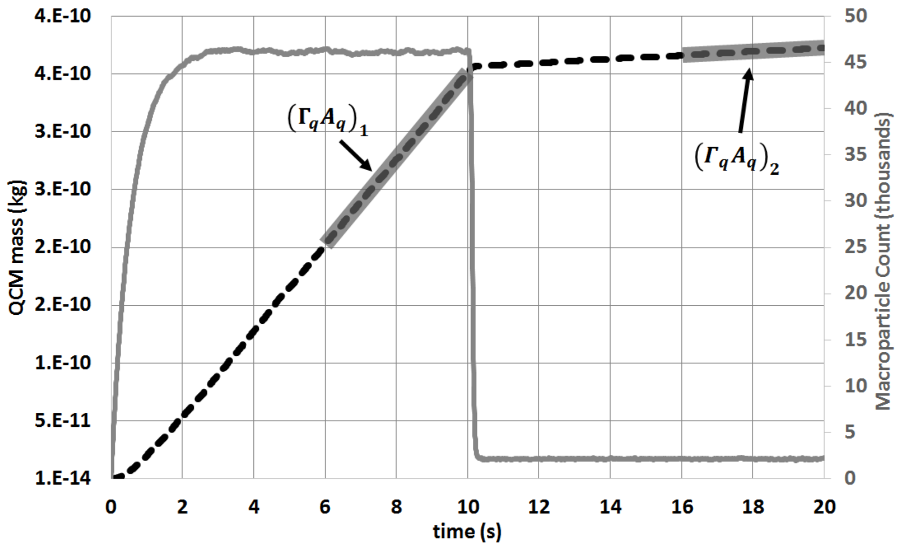


Figure 16. QCM deposited mass versus time step

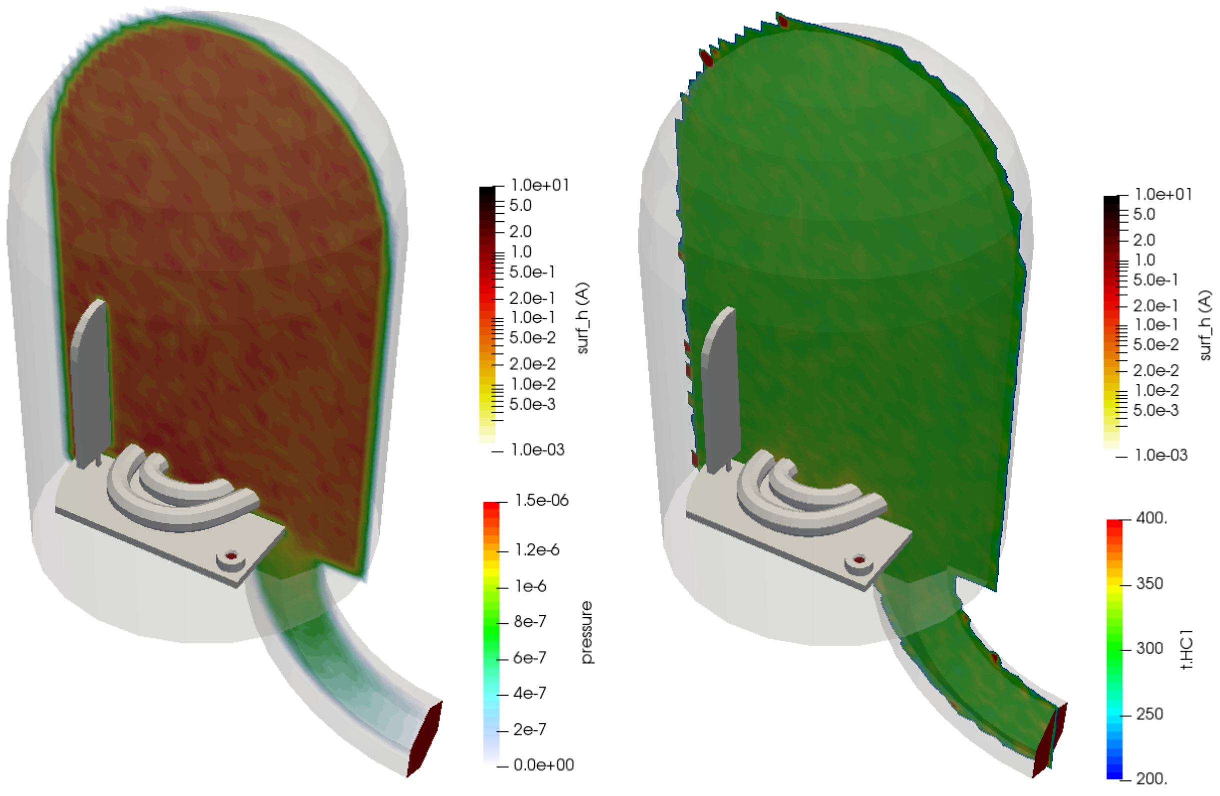


Figure 17. Pressure and temperature just prior to scavenger activation

By comparing the slopes with and without the scavenger, the effective pump area is found to be over 4x smaller than the geometrical pump cross-section due to conductance loss in the duct. We find the harness outgassing rate to be 1.03e-8 kg/s, about 2.6% higher than the expected value. Although ideally we would expect this error to be zero, this agreement is still excellent. Figure 17 visualizes the other saved volume properties: pressure and temperature. Pressure is obtained from $P = nkT$. As expected, gas temperature is uniform and equal to the surface 300K temperature.

IV.b) Molecular Equilibrium

Consider a solid sphere placed inside a larger, hollow sphere. This closed system can be used to test various aspects of the molecular transport model. We let the inner sphere initially contain some molecular contaminant trapped within it but the rest of the system is initially contaminant free. Clearly, this system is not at equilibrium. The molecules present inside the sphere (region I) are expected to diffuse to the surface (region II) and then desorb into the gas phase (region III). These molecules will then impact the outer sphere and some fraction will deposit onto the outer sphere surface (region IV). These molecules then diffuse into the outer sphere bulk material (region V). At steady-state, concentration gradients between these five regions must vanish. For instance, we have

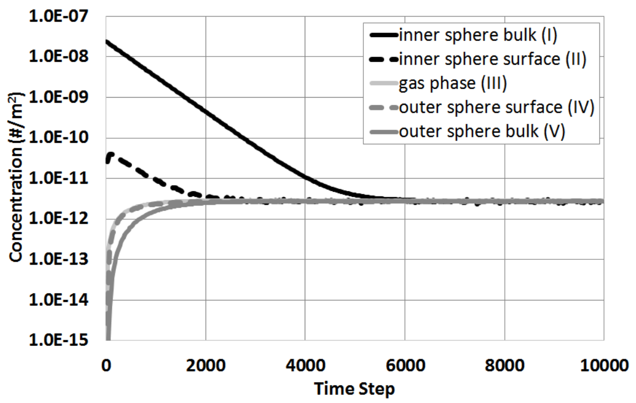
$$-\frac{dn_c}{dt} \equiv H(n_c, \theta) = n_c - \gamma\theta$$

so that at steady state $N_c = \gamma\theta$. We define $n_c = N_1/V_i$, where N_1 is the number of molecules inside the inner sphere with volume V_i. Similarly,$\theta = N_2/A_i$, with N_2 being the number of molecules in the surface layer and A_i is the surface area of the inner sphere. The equilibrium partition coefficient is set to $\gamma = 1$ m^{-1}. Furthermore the adsorption and desorption fluxes must also be equal, $\Gamma_a = \Gamma_d$. From kinetic theory, the adsorption flux is given by $\Gamma_a = (N_g\bar{u})/(4V_g)$ where is the number of molecules in the gas phase and the mean velocity $\bar{u} = \sqrt{8kT/\pi m}$. The volume occupied by the gas is given by $V_g = (4/3)\pi(r_2^3 - r_1^3)$. The desorption flux is $\Gamma_d = \theta/\tau_r$. Here we ignore any multilayer effects. At steady state we thus require

$$\theta = \left(\frac{\bar{u}\tau_r}{4V_g}\right) N_g \equiv k_g N_g$$

We run this model for two temperature configurations. The first assumes that the entire system is isothermal with both spheres at T=250 K. The radius of the inner sphere is $r_1 = 10$ cm. The outer sphere has inner radius $r_2 = 1$ m and is assumed to be 1 cm thick. The inner sphere contains 10^{-10} kg of some hypothetical contaminant with $m_a = 94$ amu, $(E_a)_{dif,des} = 12$ kcal/mol, and $r = 1.5 \times 10^{-10}$ m. The diffusion coefficient is D = 20 m^2/s. The exact mass generation model generating particles with fractional weight is used. The simulation was run for 10,000 $\Delta t = 10^{-4}$ s time steps. Figure 18 shows the evolution of $n_c, \theta, k_g N_g, \theta_2$ and $n_{c,2}$ versus the simulation time step. The system reaches equilibrium around time step 6,000 with all concentration gradients vanishing. This figure also shows the corresponding combined mass of molecules in each of the five zones. We can see that the total mass remains constant. Figure 19 plots the typical contaminant partial pressure in the cavity between the spheres. This is an example of a macroscopic data that can be computed due to CTSP concurrently pushing multiple particles. Pressure was computed by scattering particle positions to a Cartesian mesh to compute number density n. Temperature is also computed from particle velocities. Pressure can then be obtained from the ideal gas law. This figure also plots the surface concentration in the number of particles per unit area. The average of these element values is represented by the dashed lines in Figure 18.The increased noise on the inner sphere is due to the smaller surface elements.

We next consider a case with the outer sphere temperature reduced to 200K. This result is plotted in Figure 20. Now the bulk and surface layer regions form an internal balance but there is no longer an equilibrium between the small and the large sphere. This is expected. Since the larger sphere is colder, we expect it to act as a better sink for the contaminants. The molecular concentration should thus be higher on the outer sphere, which is indeed confirmed by the simulation. The gas phase concentration now also tracks the larger 200 K sphere.



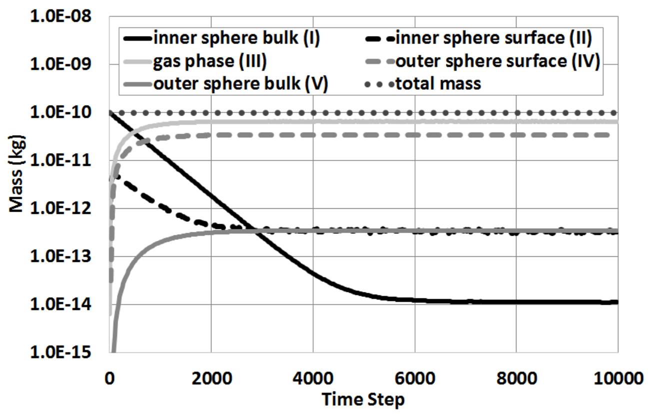


Figure 18.Molecular concentration and molecular mass for the isothermal spheres case

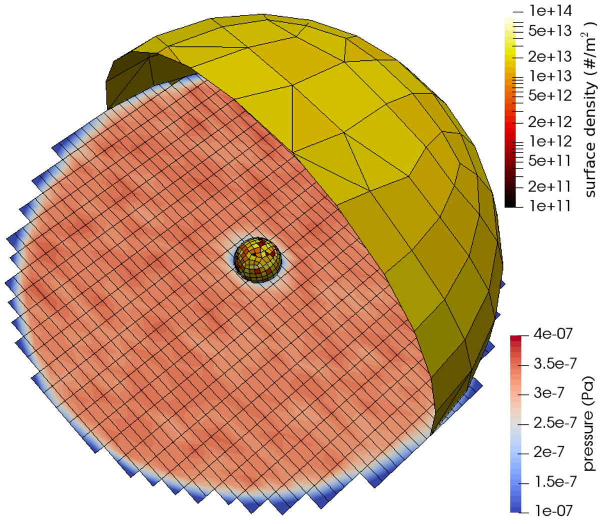


Figure 19.Typical pressure and surface density for the isothermal case

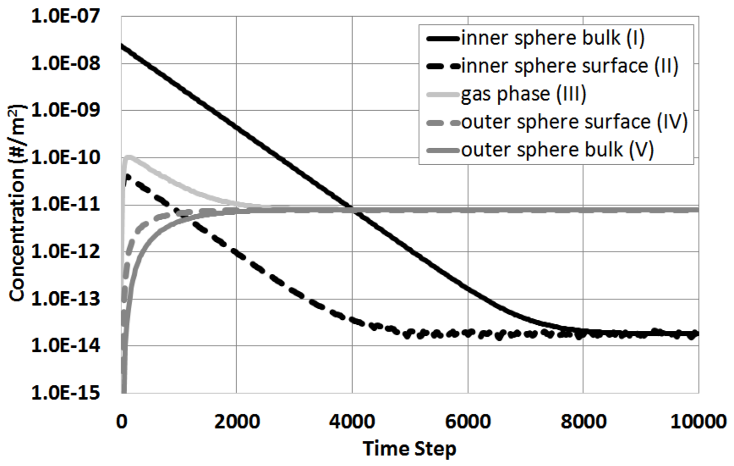


Figure 20. Evolution of molecular concentration with 200K outer sphere. Gas phase concentration computed using T = 200 K.

The input file for this case (in `dat/spheres`) is listed below. The input geometry file, `spheres.unv`, contains the two spheres with normal vectors facing into the gas cavity, and zone names set as “inner” and “outer”.

```
#set options
options {log_level:info, domain_check:true, randomize:false, num_threads:1,
screen_diag_freq:25, file_diag_freq:2}

#load surface
surface_load_unv{file_name:"spheres.unv"}
surface_save_vtk{file_name:"surf-all"}

#define volume mesh
volume_mesh{dx:0.04,dy:0.08,dz:0.08, expand:0.02}

#define materials and material interactions
solid_mat{name:metal, weight: 100}
molecular_mat{name:hc1, weight: 94, mpw: 1e10, r:1.55e-10}
material_interaction{source:hc1, target:*, Ea_des:12, C_dif:2.5e11}

#define inner and outer sphere properties
surface_props{comps:inner, mats:metal, volume: 4.189e-3, temp:250 }
surface_props{comps:outer, mats:metal, volume: 1.269e-1, temp:250}

#enable outgassing
load_molecules{comps:inner, trapped_mass:1e-10, trapped_mats:[1.0*hc1]}
source_outgassing{model:fang, stop_it:-1, exact:true}

#set output
particle_trace{file_name:trace,mat:hc1,num_traces:20, skip:100}
volume_save_vtk{skip:1000,file_name:"field",vars:[nd.hc1,t.hc1]}
surface_save_vtk{skip:1000,file_name:"surf",vars:[surf_h.hc1,dep_rate.hc1,T]}

#run simulation
run_sim{dt:1e-4,num_ts:2000,diag_start:0,diag_skip:5}
```

IV.c) Particulate Transport

In this example we consider the effect of nitrogen purge on particulate contamination. Suppose that some instrument containing two detectors is stored on a lab bench. The instrument is connected to a nitrogen purge which vents out around the perimeter of the larger of the two detectors. An impact to the bench dislodges particles from a shelf located above the detector. We are interested in numerically studying the effectives of the purge on preventing contamination of the detector. The entire set up can be seen in Figure 21. The mesh shown in the second image correspond to five different zones. Note that only the bottom surface of the top shelf is modeled. The normal vectors are oriented to point away from the solid surface into the simulation volume. Files for this example are in **dat/purge**.

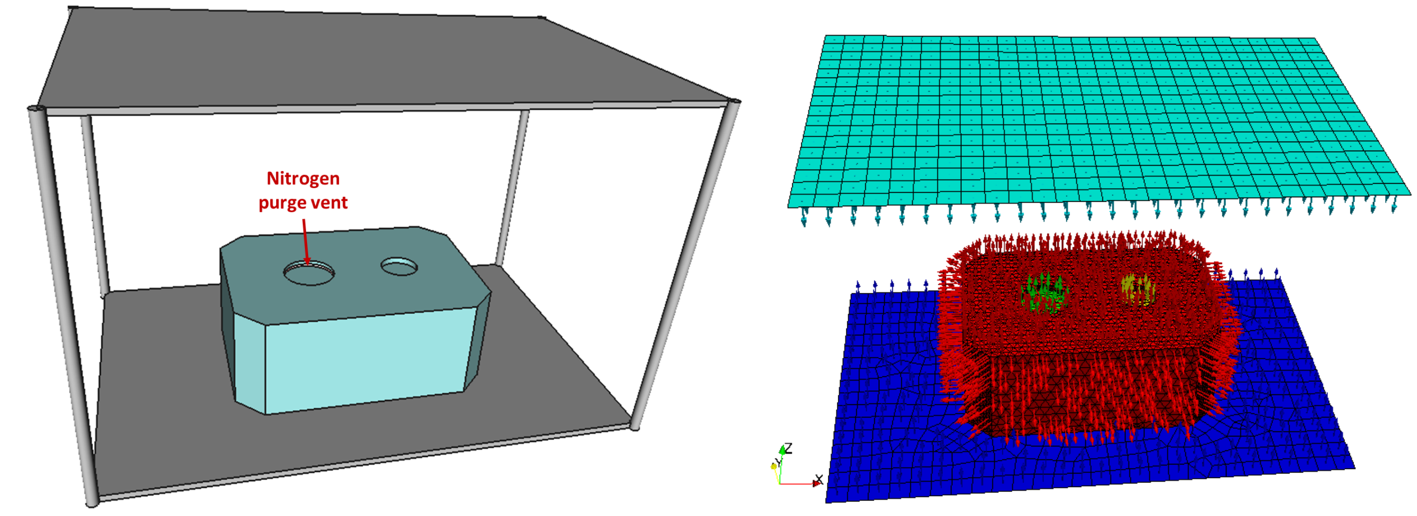


Figure 21. Geometry model and the corresponding mesh for the purge example.

The simulation input file, ctsp.in is listed below:

```
options{randomize:false, num_threads:1}

#set ambient environment
world{gravity:[0,0,-9.81], pressure:101325, temperature:300}

#load surface
surface_load_unv{file_name:"surf-mesh.unv",units:m}

#volume mesh added to remove particles leaving the domain
volume_mesh{dx:0.1,dy:0.1,dz:0.1}

#component information, setting coefficient of restitution to 0.5
surface_props{comps:/.*/, mat:al, temp:0,c_rest:0.5}

#load cfd data, comment out to use still air at constant pressure per world
flow_load_csv{file_name:"purge_1e-3.csv",  map_pos:[4,5,6],  map_vel:[0,1,2],  time:-1,
save_view: false}

#specify materials
solid_mat{name:al, weight: 100}
particulate_mat{name:flakes, density:2700, integrator:subcycle}

#load  level  600,  slope=0.926  particulates,  model  detachment  with  Klavins-Lee  4g
acceleration
load_particulates_1246{comps:top, mat:flakes, particle_count:200000, level:600, C:0.926}
```

```
detach_particulates{model:klavins, klavins_accel_mag: 39.24, release_interval:-1}

#setup particle trace and periodic particle saving for animations
particle_trace{file_name:trace,mat:flakes,num_traces:100}
particle_save_vtk{file_name:"particles",                          mat:flakes,skip:25,
num_particles:2000,max_id:300000}

#save animation of surface composition
surface_save_vtk{skip:25,file_name:"surf",vars:[level,pac]}

#enable histogram output
surface_save_histogram{file_name:"hist",skip:100,                                bins:
[1,10,20,30,40,50,60,70,80,90,100,200,300,400,500,600,700,800,900,1000]}

#run simulation
run_sim{dt:0.001,num_ts:4000,diag_start:1,diag_skip:100}

#save final surface data
surface_save_vtk{file_name:"surf-final",vars:[level,pac]}
```

We first set ambient environment using **world**. Specifically, we let gravity act in the –z direction, and prescribe 1 atm background pressure. Next, we use **surface_load_unv** to load the surface mesh from the surf-mesh.unv file. We specify that dimensions are in meters, which is also the default. Next a volume mesh is created using **volume_mesh**. This mesh is being used delete particles leaving the volume mesh bounding box. This is important for computational reason since there are no surfaces in the +/- X and Y directions to stop particles. As will be apparent later, as the purge flow is increased, flow with non-negligible tangential velocity develops along the bottom of the top shelf. This flow acts to push particles laterally, and without the volume mesh, these particles would then fly out of the computational domain. Eventually gravity will make them travel in the –Z direction, but by then there will not be any floor present to stop them and these particles would continue traveling forever. Next we specify the coefficient of restitution using **surface_props**. The same coefficient is assigned to all surfaces.

Next we load the flow data from a file using **flow_load_csv.** This command can be commented out to simulate particles settling out in a still air. The CFD solutions for this example were generated using OpenFoam. Simulation results were save in VTK format and were then saved as CSV in Paraview. The structure of the file is as follows:

```
"U:0","U:1","U:2","p","Points:0","Points:1","Points:2"
4.846505e-003,0.000000e+000,1.140431e-004,-1.259318e-004,-2.100000e-001,0.0e+00,0.0e+00
3.844443e-003,0.000000e+000,1.380693e-004,-3.659827e-004,-2.024971e-001,0.0e+00,0.0e+00
3.828275e-003,0.000000e+000,1.411461e-004,-7.318797e-004,-1.949940e-001,0.0e+00,0.0e+00
```

This file contains a point cloud of flow velocities and pressures. As can be seen, the three velocity components are stored in columns [0,1,2], while positions are in [4,5,6]. Relative pressure is in column 3. Since the deviations in pressure are so small, we ignore this column (there is no **map_pressure**) specified, and instead the code will utilize the constant value from **world**. The *time:-1* field tells CTSP to use this file for the entirety of the simulation.

The simulation next specifies material properties using **solid_mat** and **particulate_mat**. We use the latter command to specify density (which is used to compute mass) of the particulate “flakes” coming off the top shelf. The actual distribution of these is specified by the **load_particulates_1246** command. We tell the code that we would like to generate level 600 and slope 0.926 distribution of particulates on component “top”. We use the Klavins and Lee release model with release acceleration equal to 4g via **detach_particulates**. The particles should be all released at once at the first time step, as noted by *release_interval:-1*.

Next we enable tracing of particles with **particle_trace**. We tell the code to generate traces for random 100 “flakes” particles in the id range of [0,100000]. We also let the code save the positions of 2000 particles every 10 time steps with **particle_save_vtk**. This command will save positions of the same particles (as opposed to sampling random particles at each save) so it can be used to generate animations. We also enable a surface composition histogram output with **surface_save_histogram**. We are then ready start the simulation using the **run_sim** command. We tell the code to run for 10000 time steps. Volume diagnostics are computed once every 100 steps. Since we are not interested in the any of these data, this interval could be set even higher. Once the simulation finishes, surface results are saved with **surface_save_vtk**. Note that here we use a *vars: [level,pac]* which is currently ignored but may be added in the future. For now, this command saves all available surface properties.

Figure 22 shows typical results from the simulation for static air (no flow), and Figure 23 shows results for purge flow at 1e-4, 2e-4, 5e-4, and 1e-3 m^3/s, respectively, visualized using Paraview. These results were obtained by running the CTSP 5 times with different file specified for the **flow_load_csv** command. The no flow case was obtained by commenting out this command entirely. Since only the bottom surface of the top shelf was included in the mesh, the coloring we see in the plot actually corresponds to the PAC on the downward facing surface. As can be noted, the magnitude of PAC is approximately constant for the entire source surface. This is indeed the value that we obtain. Similarly an approximately constant value is obtained on all upward facing surfaces, despite differences in mesh density. This indicates that the result is not mesh dependent. We can numerically compute (see drag.py) the expected PAC on the top shelf after particle detachment to find that it should equal to 0.351427. The small variations in value from cell to cell are due to the statistical nature of the simulation. Rerunning the simulation will result in a slightly different variation, which could be averaged as part of post processing to further reduce the noise. The level of noise also decreases with an increase in the number of particles per bin given by the *parts_per_bin* parameter.

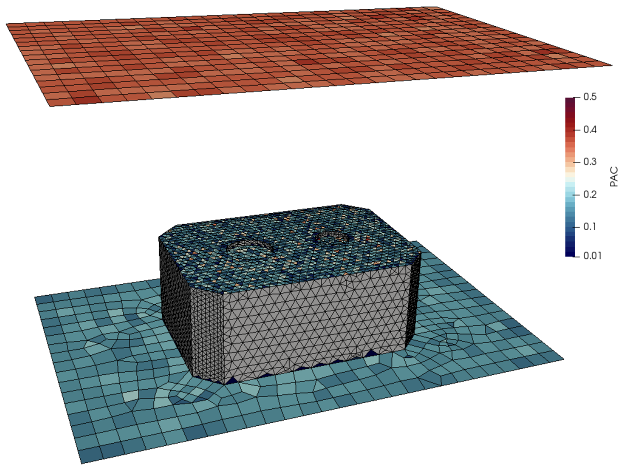


Figure 22. Simulation results showing the surface particle area coverage for no flow.

Figure 24 shows the top down view of the instrument. As the purge flow rate is increased the PAC on the detector is reduced. At 1e-4 m^3/s, the reduction is mainly limited to a circumferential strip around the vent inlet. This region grows with increasing flow rate. At 1e-3 m^3/s, the purge flow rate is sufficiently strong to generate an “umbrella” over a large portion of the instrument, effectively shielding not only the larger detector, but also the smaller one that does not have an associated flow.

To further analyze the effect of purge flow, we may be interested in determining the average PAC on the detector for each flow rate. This could be done in Paraview by selecting mesh elements on the detector and using the Integrate filter. However, CTSP already prints this information to the **ctsp.log** file. Below is the example output. For each zone, the particulate PAC as well as particulate level, assuming C=0.926, is printed.

```
FLOOR:      0.000e+000(A, surf_h)   1.396e-001(%, PAC)      4.607e+002(level)
TOP:  0.000e+000(A, surf_h)   3.514e-001(%, PAC)      5.562e+002(level)
DET1: 0.000e+000(A, surf_h)   7.120e-003(%, PAC)      2.063e+002(level)
DET2: 0.000e+000(A, surf_h)   1.576e-001(%, PAC)      4.670e+002(level)
UNNASIGNED: 0.000e+000(A, surf_h)   3.427e-002(%, PAC)      1.466e+002(level)
```

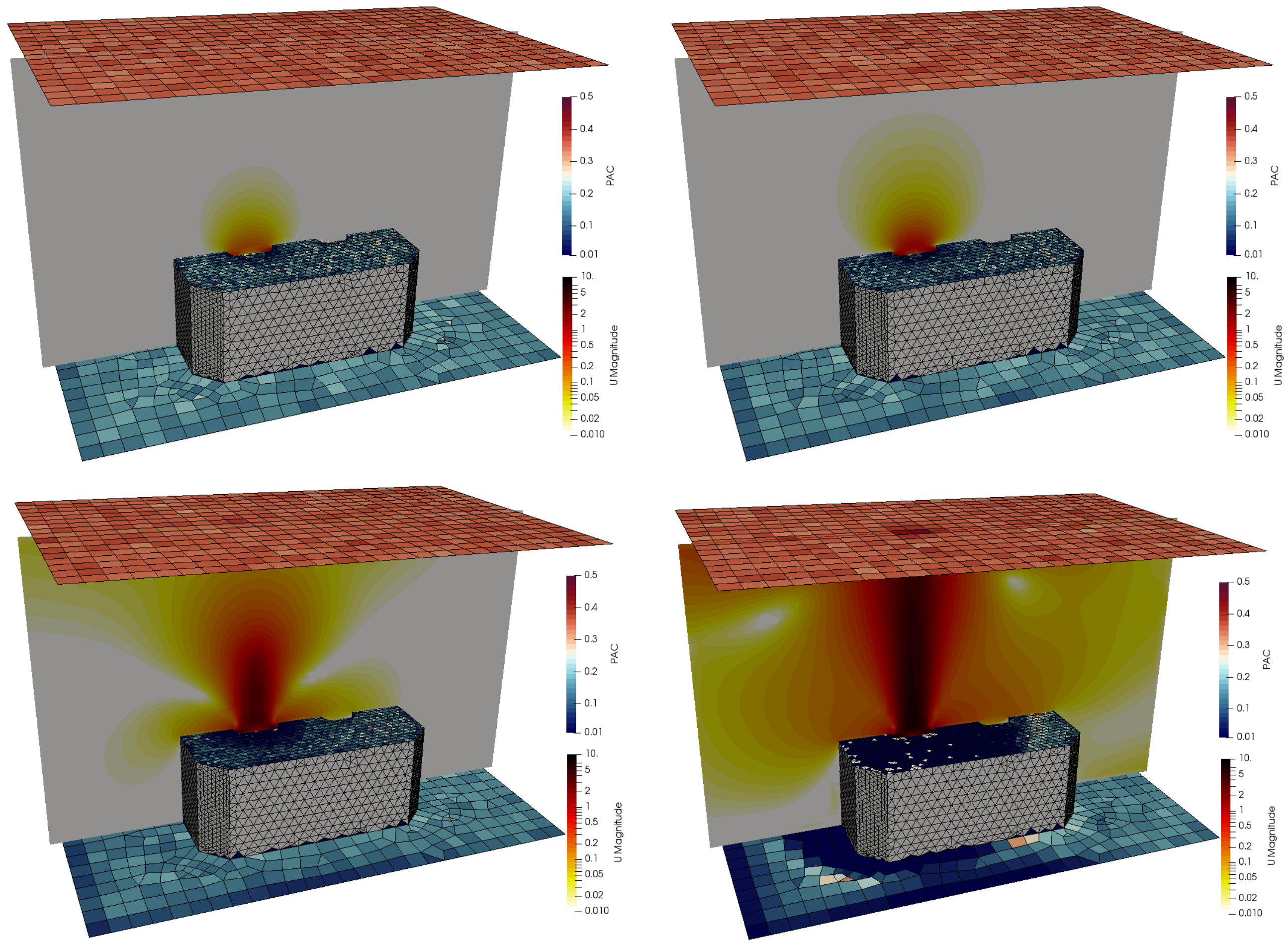


Figure 23. Simulation results showing the surface particle area coverage as well as the corresponding velocity field for 1e-4, 2e-4, 5e-4, and 1e-3 m^3/s, respectively.

By capturing this data for the 5 cases, we can generate a plot shown in Figure 25. The solid line is the averaged simulation PAC on the larger detector as a function of purge flow rate. We can use this data for validation. The data shown by the dashed line was obtained by running a Python script **drag.py**. This script computes the number of particles in each 1Cm bin in [1,1000] and determines the fraction that is released according to the Klavins and Lee model at a prescribed *a_mag*. It then computes particle areas and masses using the same algorithm utilized within CTSP, based on aspect ratios per a paper of Perry. In each bin, the code then balances the gravitation force versus drag force assuming a stationary particle and some prescribed uniform vertical velocity. If the drag force dominates, the particle is assumed not to be able to reach the detector. The cross-sectional areas of the remaining particles is summed up to obtain the effective PAC on the target. We can see a generally good agreement.

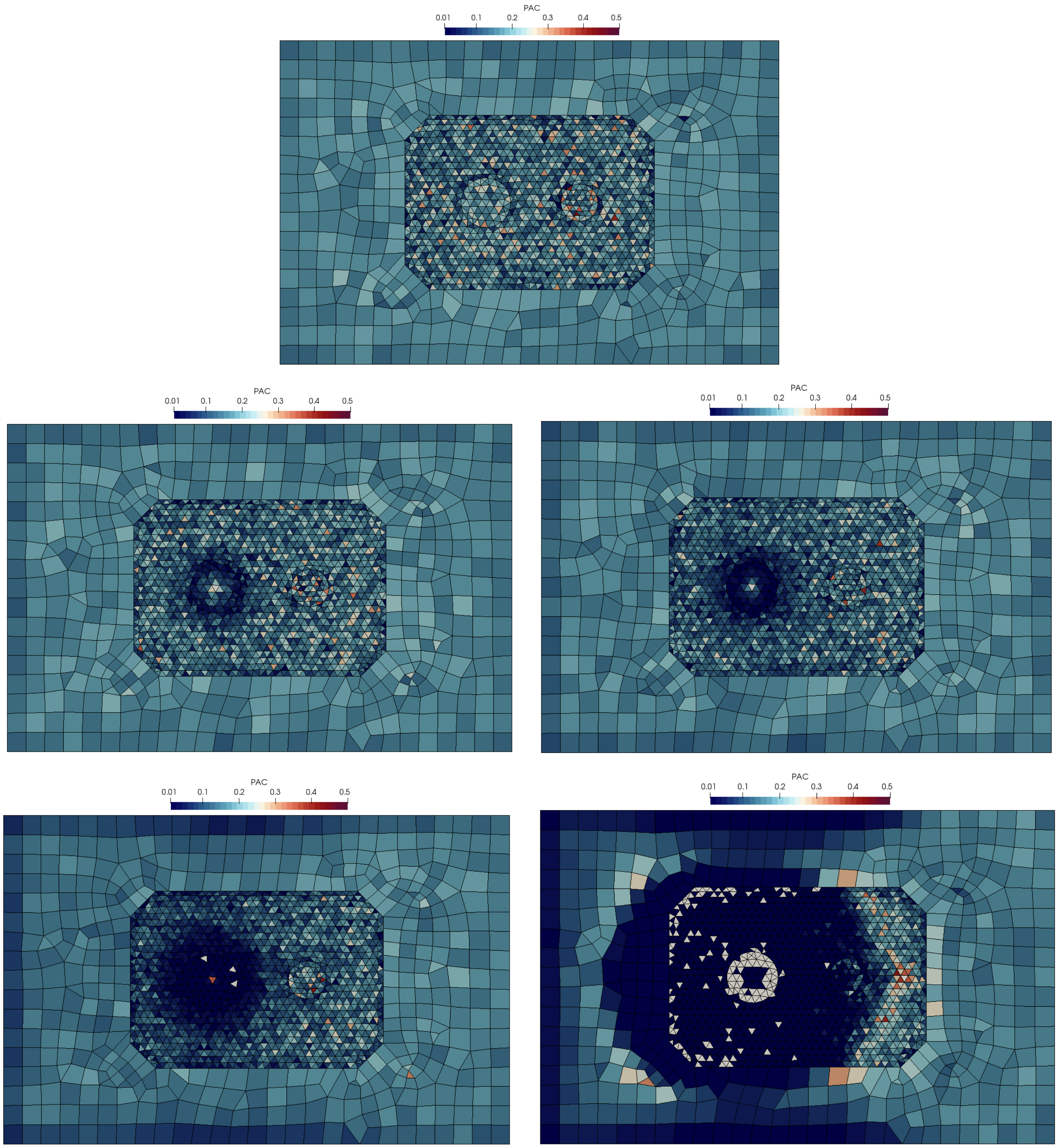


Figure 24. Top down view of the instrument for 0, 1e-4, 2e-4, 5e-4, and 1e-3 m^3/s flow rate

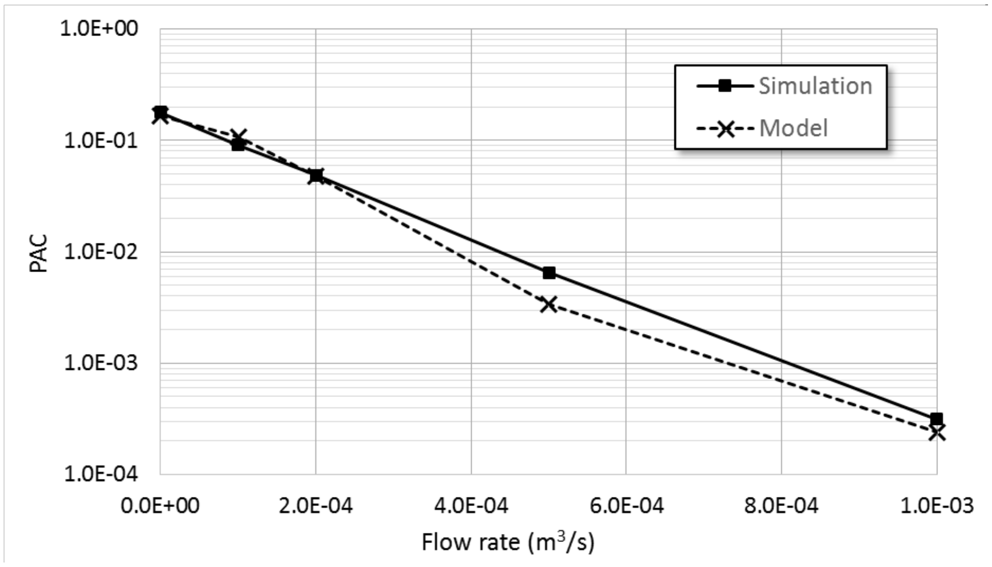


Figure 25. Comparison of simulation to model PAC

A major source of error in our simple Python model is that it assumes uniform flow velocity. But as can be seen from plots in Figure 23, the velocity field is highly non-uniform. The approach used to obtain representative velocity for the model was to select a large number of points near the detector, and use Paraview's histogram feature to obtain the average velocity magnitude (this approach was used since it appears that Paraview lacks a built in filter to average point data). The same data points were used across the 4 input files to maintain consistency.

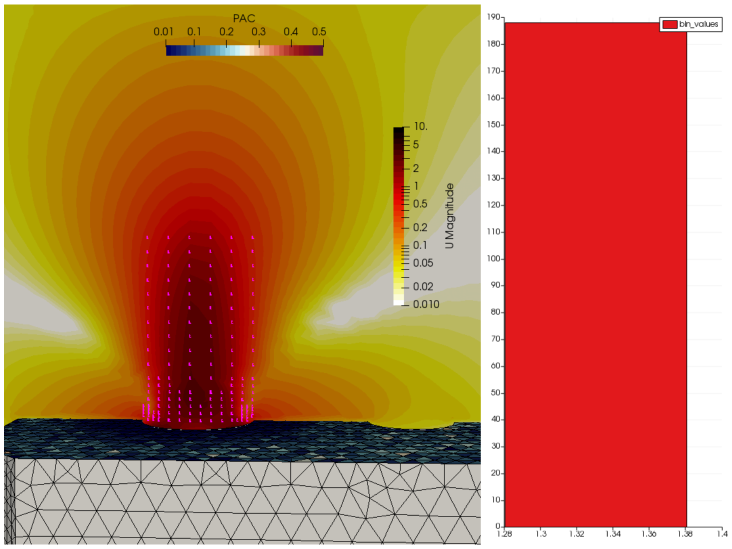


Figure 26. Data points used to sample flow velocity

Upon a closer inspection of plots in Figure 24, we may notice that the center of the detector appears to have higher particle deposition values than the surrounding areas. We can take a look at the generated traces in **trace.vtp** to better understand this issue. Figure 27 shows a typical result for the 5e-4 m^3/s flow rate. Note that the CFD solution indicates that the highest velocity magnitude occurs some short distance above the detector and that there is actually a local minimum along the centroid. This seems to imply presence of a recirculating region. As shown by the trace, at the default coefficient of restitution of 0.5 particles bounce around quite a bit.

As such it appears that some of these bounces bring particles onto the detector where they are constrained by this near-surface flow. The particles will naturally settle out at the local minimum. The result shown in the second image was obtained by reducing to 0.3. Particles are now less bouncy and the peak value on the detector is reduced (the triangle went from red to light blue). However outside this region the two solutions comparable.

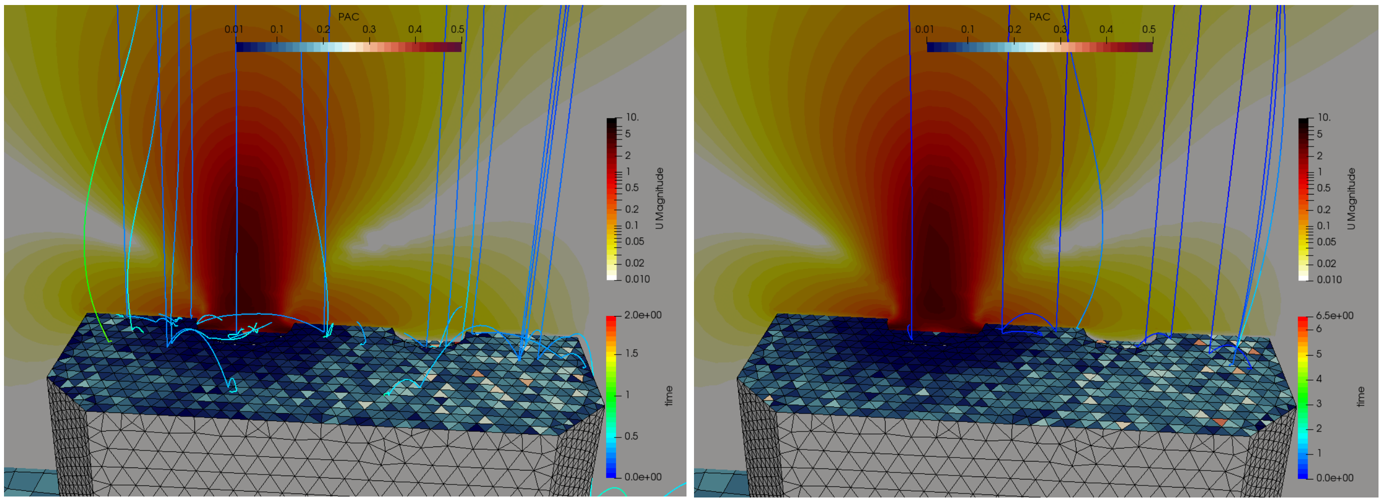


Figure 27. Particle traces with $\alpha_{COR}=0.5$ *on left and* $\alpha_{COR}=0.3$ *on right*

Finally, Figure 28 illustrates the ability to generate animations by visualizing data saved by **particle_save_vtk**. The particles_*.vtp files were loaded into Paraview and visualized using the glyph filter. Particles are colored according to their mass. We can see that the heavier (dark red) particles settle out first. The lighter (white) particles tend to remain suspended for a long time after the heavier particles have settled out.

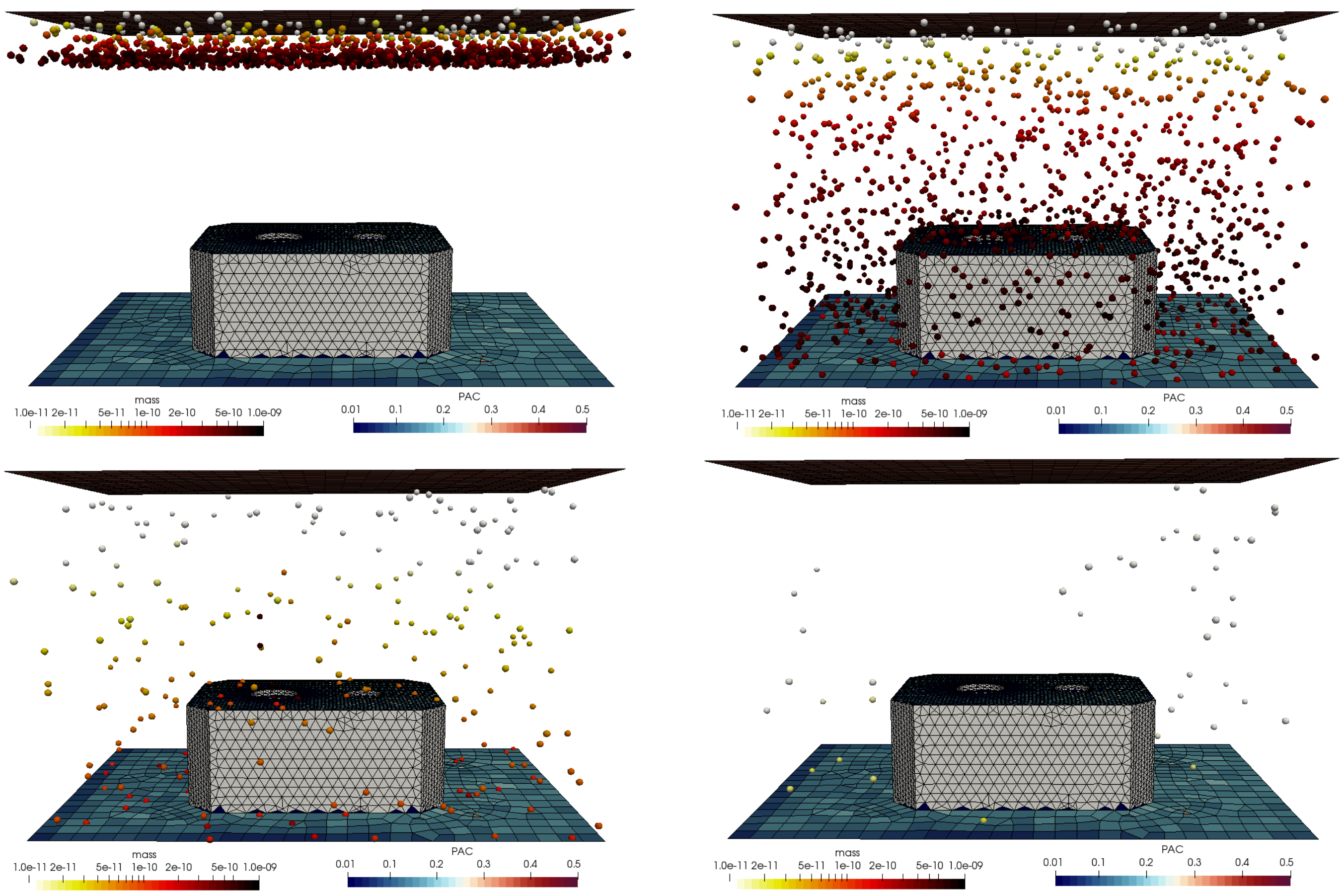

mass
1.0e-11 2e-11 5e-11 1e-10 2e-10 5e-10 1.0e-09
PAC
0.01 0.1 0.2 0.3 0.4 0.5
mass
1.0e-11 2e-11 5e-11 1e-10 2e-10 5e-10 1.0e-09
PAC
0.01 0.1 0.2 0.3 0.4 0.5
mass
1.0e-11 2e-11 5e-11 1e-10 2e-10 5e-10 1.0e-09
PAC
0.01 0.1 0.2 0.3 0.4 0.5
mass
1.0e-11 2e-11 5e-11 1e-10 2e-10 5e-10 1.0e-09
PAC
0.01 0.1 0.2 0.3 0.4 0.5

Figure 28. Heavier particles (red) settle out much faster than lighter ones (white).

IV.d) Additional Examples

The "dat" folder contains input files for several additional examples. The **DSMC** folder contains examples of setting up a DSMC simulation. The "nozzle" subdirectory contains inputs for simulating flow through a converging nozzle, while the "astronaut" case is an example simulating gas flow past an astronaut model.

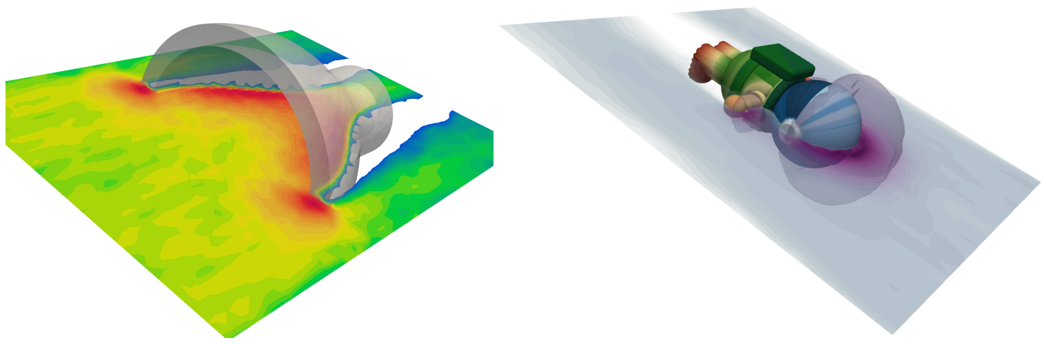


Figure 29. DSMC examples

The **cleanroom** folder demonstrates how to simulate a cleanroom environment. Here, the source_particulate_14644 is used to simulate a Class 6 clean room inflow.

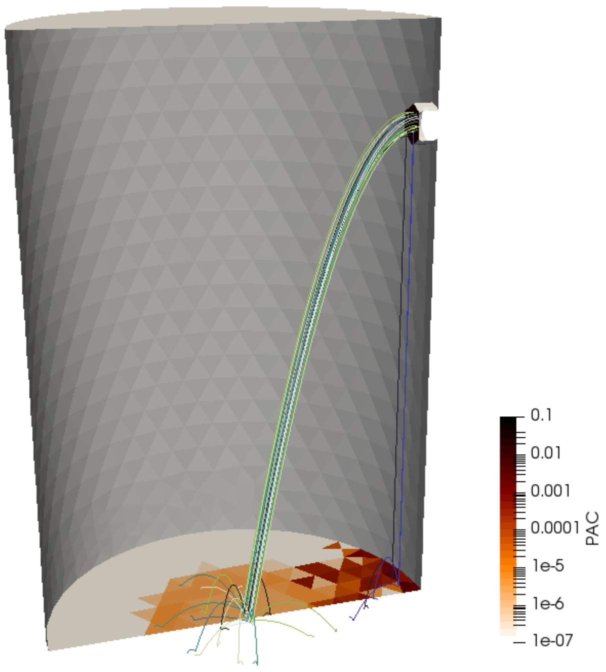


Figure 30. Example of using a particulate and a cleanroom source to inject particulates

Finally, the **tube** folder contains inputs for simulating conductance through a cylindrical tube. This was the first-ever test of CTSP, but has not been verified in a while, so it may not be working properly anymore.

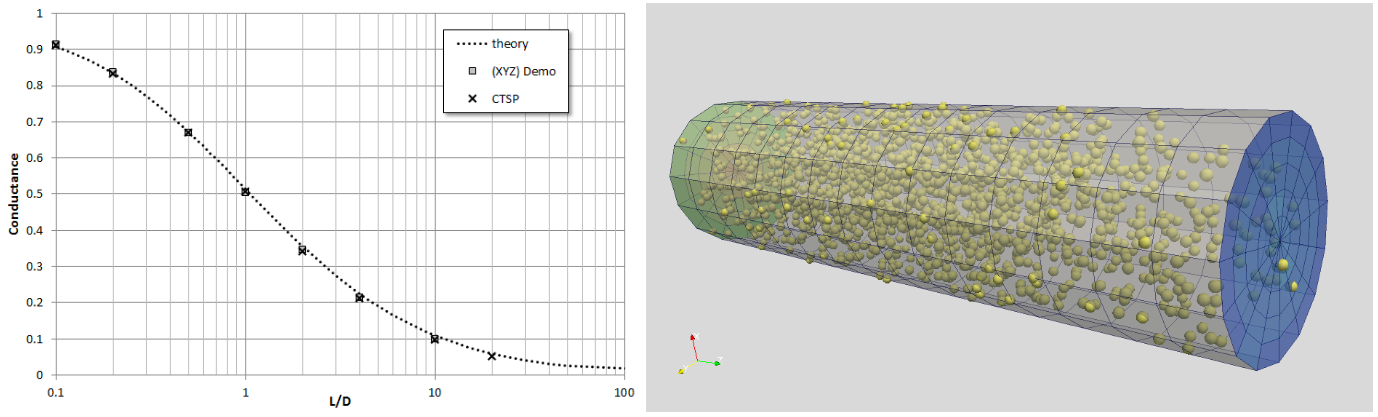


Figure 31. Simulation of conductance through a cylindrical tube

V.References

[1] L. Brieda, "Numerical Model for Molecular and Particulate Contamination Transport," *Journal of Spacecraft and Rockets*, Vol. 56, No. 2, 2019

[2] C. Birdsall and A. Langdon, Plasma physics via Computer Simulations, Institute of Physics Publishing, 2000.

[3] G. Bird, Molecular Gas Dynamics and the Direct Simulation of Gas Flows, Oxford Science Publications, 1994.

[4] H. Shin, Y. Lee and J. Jurng, "Spherical-shaped ice particle production of spraying water in a vacuum chamber," *Applied Thermal ENgineering,* pp. 439-454, 2000.

[5] M. S. Woronowicz, "Highlights of Transient Plume Impingement Model Validation and Applications," in *42nd AIAA Thermophysics Conference*, 2011.

[6] J. Koo, Hybrid PIC-MCC computational modeling of Hall Thrusters, Aerospace Engineering and Scientific Computing, University of Michigan, 2005.

[7] A. Tribble, Fundamentals of Contamination Control, SPIE, 2000.

[8] ASTM International, "Standard Test Method for Contamination Outgassing Characteristics of Spacecraft Materials (ASTM-E1559)," 2009.

[9] W. Fang, M. Shillor, E. Stahel, E. Epstein, C. Ly, M. J. and E. Zaron, "A Mathematical Model for Outgassing And Contamination," *SIAM Journal of Applied Math,* vol. 51, no. 5, 1991.

[10] A. C. Tribble, B. Boyadjian, J. Davis, J. Haffner and E. and McCullough, "Contamination control engineering design guidelines," NASA Contractor Report 4740, 1996.

[11] IEST, "Product Cleanliness Levels Applications, Requirements, adn Determinations (IEST-STD-CC1246D)," 2002.

[12] A. Klavins and A. L. Lee, "Spacecraft particulate contaminant redistribution," *Optical system contamination effects, measurement, control,* vol. 777, pp. 236-244, 1987.

[13] R. Perry, "A Numerical Evaluation of the Correlation of Surface Cleanliness Level and Percent Area Coverage," in *SPIE Optics and Photonics*, 2006.

[14] F. White, Viscous Fluid FLow, New York: McGraw-Hill, 1991.