

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE 30 Jun 2021	2. REPORT TYPE	3. DATES COVERED	
		START DATE 01 Jan 2021	END DATE 30 May 2021
4. TITLE AND SUBTITLE Tutorial for Thermophysics Universal Research Framework – TURF-IR 2021			
5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER	
5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER Q2DH	
6. AUTHOR(S) Samuel J. Araki (Jacobs), David Bilyeu, Robert Martin			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory In-Space Propulsion Branch 1 Ara Drive Edwards AFB, CA 93524			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Rocket Propulsion Division Edwards AFB, CA 93524		10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RQRS	11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RQ-ED-OT- 2021-154
12. DISTRIBUTION/AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.			
13. SUPPLEMENTARY NOTES PA Clearance Number: AFRL-2021-1856 Clearance Date: 15 June 2021			
14. ABSTRACT TURF development has been ongoing at AFRL/RQRS since late 2011 and has reached a level of development and testing where the core data-structures, interface, and select operators are ready to be used by external collaborators familiar with plasma physics and C++. To facilitate bringing these collaborators up to speed in writing modules for TURF, the TURF Infrastructure Release (TURF-IR) has the capabilities in place to demonstrate kinetic simulations, including several example input files to provide convenient starting points for the development of additional physics capabilities. Though the framework has been designed in part to facilitate the creation of modules that replicate the functionality of Coliseum/HPHall, it must be emphasized that the TURF-IR is intended to stimulate academic collaboration and does not provide geometric models or physical models calibrated against operational devices. As such, this software by itself cannot be used to design or analyze real systems.			
15. SUBJECT TERMS			
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	SAR
19a. NAME OF RESPONSIBLE PERSON David Bilyeu			18. NUMBER OF PAGES 156
19b. PHONE NUMBER (Include area code)			

TUTORIAL FOR THERMOPHYSICS UNIVERSAL RESEARCH FRAMEWORK – TURF-IR 2021 –

June 2021

Samuel J. Araki¹, David Bilyeu², Robert Martin²,

¹JACOBS

²AFRL/RQRS



1 TURF User Guide	1
1.1 Introduction	2
1.2 Core Structure	3
1.2.1 Key Data-structures	3
1.2.2 Basic Functionality	4
1.3 World.list file	5
1.3.1 Keywords	5
1.3.2 Logical Domain Definition	7
1.4 Tutorials	9
1.5 Heatbath with Multi-Species Particle Data	10
1.5.1 Introduction	10
1.5.2 Setting up a simulation	10
1.5.3 Operations	13
1.5.4 Useful Tools	23
1.6 Grounded Box: 3D Electrostatic PIC with Multi-Species Particle Data	26
1.6.1 Included Files	26
1.6.2 Introduction	26
1.6.3 world.list	26
1.6.4 operations.list	27
1.7 One Dimensional Normal Shock: DSMC with Multi-Species Particle Data	34
1.7.1 Included Files	34
1.7.2 Introduction	35
1.7.3 Description of the Example Problem	35
1.7.4 Setting up the DSMC Example	36
1.7.5 Operations	37
1.7.6 Comparison of Shock Profiles with Bird's DSMC Code	41
1.8 Sphere Immersed in a Plasma: Charging with Hybrid PIC/Boltzmann Fluid Model	43
1.8.1 Introduction	43
1.8.2 Defining the World	44
1.8.3 Additional Files	45
1.8.4 TURF Operations	45
1.8.5 Plotting Floating Potential Result	54
1.8.6 Analytical Solution	55
1.9 Adaptive Mesh Refinement	57
1.9.1 Introduction	57
1.9.2 TURF AMR Basics	57
1.9.3 Example 1: Using Coordinates and Field Data in Mesh Refinement Criteria	59
1.9.4 Example 2: Extending Example 1 to Multi-Domain	61
1.9.5 Example 3: Using Sugarcube Data in Mesh Refinement Criteria	62

1.10 EP Plume Simulation 1: Baseline	65
1.10.1 Introduction	65
1.10.2 Defining the World	65
1.10.3 Geometry Components	66
1.10.4 Particle-Surface Interaction	66
1.10.5 TURF Operations	67
1.10.6 Restarting the Simulation	82
1.10.7 Extension to Multi-Domain Simulation	83
1.11 EP Plume Simulation 2: Fractional DSMC Collision	84
1.11.1 Introduction	84
1.11.2 Interspecies Fractional DSMC	85
1.11.3 Using Different Collision Multipliers for Fast and Slow Ions	86
1.11.4 Separating Primary and CEX Ions	87
1.12 EP Plume Simulation 3: Spacecraft Charging	89
1.12.1 Introduction	89
1.12.2 Changes in the World File	89
1.12.3 Operations for Spacecraft Charging	89
1.13 EP Plume Simulation 4: AMR and Dynamic Load Balancing	94
1.13.1 Introduction	94
1.13.2 Converting the Uniform Cartesian Mesh to the Cartesian Mesh with AMR	95
1.13.3 Reordering Operations to Update Field Data after Mesh Refinement	97
1.13.4 Dynamic Load Balancing	99
1.14 EP Plume Simulation: Cubit	101
1.14.1 Introduction	101
1.14.2 General Geometry	101
1.14.3 Mesh	105
1.14.4 Blocks	105
1.14.5 Export	105
1.14.6 Advanced Examples	106
1.14.7 References	107
1.15 EP Plume Simulation: ParaView	108
1.15.1 Introduction	108
1.15.2 ParaView	108
2 List of Operators in TURF	113
3 Module Documentation	121
3.1 Charging	122
3.1.1 Surface Charging	122
3.2 DSMC	126

3.2.1 The DSMC Method	126
3.3 Field	127
3.3.1 Arithmetic Operations	127
3.4 Geometry	132
3.4.1 Introduction	132
3.4.2 Structured Mesh	132
3.4.3 Unstructured Mesh	132
3.5 Particle	134
3.5.1 Introduction	134
3.5.2 Theory	134
3.5.3 Current Capabilities	134
3.6 Plotting	136
3.7 Probe	137
3.8 Surface	138
3.8.1 Introduction	138
3.8.2 Particle-Surface Interaction	138
3.8.3 Virtual	139
3.8.4 Sputtering	139
3.8.5 Implementation	145
Bibliography	148

Chapter 1

TURF User Guide

1.1 Introduction

TURF development has been ongoing at AFRL/RQRS since late 2011 and has reached a level of development and testing where the core data-structures, interface, and select operators are ready to be used by external collaborators familiar with plasma physics and C++.

To facilitate bringing these collaborators up to speed in writing modules for TURF, the TURF Infrastructure Release (TURF-IR) has the capabilities in place to demonstrate kinetic simulations, including several example input files to provide convenient starting points for the development of additional physics capabilities. Though the framework has been designed in part to facilitate the creation of modules that replicate the functionality of Coliseum/HPHall, it must be emphasized that the TURF-IR is intended to stimulate academic collaboration and does not provide geometric models or physical models calibrated against operational devices. As such, this software by itself cannot be used to design or analyze real systems.

1.2 Core Structure

TURF is designed around a basic tree hierarchical object structure. TURF objects are built around this core "General Service Object" or "GSOBJect" that facilitates construction of object trees and allows branches of the tree to be recursively copied across disparate memory spaces such as from the CPU to GPU or across the message passing interface (MPI) without losing their structural integrity. Because TURF objects are all derived from this basic type, in addition to the core functionality required for this recursive data movement, the GSOBJect also facilitates recursive auto-documentation of runtime object structure.

The objects of TURF can be divided between data objects and operation objects. The number of data objects are intentionally restricted to provide a basic common skeleton of data storage on which a broad set of operations can be applied. These basic data objects with minimal independent functionality facilitate code reuse by providing a common basis through which operations interact. They also help simplify parallel communication by minimizing the set of disparate objects that must be transferred. The operation objects define the key application programming interface through which developers are encouraged to interact with the framework. They are intended to represent compact mathematical operations and consist of 3 key components. Every operation includes an "init" function that is passed a map of parameter-key values. It is the operation's responsibility to parse this map to initialize all the local information, create data-structures as needed, and query the existing object hierarchy for references to additional required objects needed to function. For some operations, such as initial conditions, this initialization is all that is required. For operations that evolve the problem solution, an "apply" function is defined. This function is called as the framework iterates through a sequence of operations to perform a specific function on the data. Whenever possible, this functionality should be further broken down into one or more "core" functions that represent the functionality in a data independent parallel form that is designed to be agnostic to whether the core is applied sequentially on a single process, in parallel through threads, or as implemented by OpenMP, or on an accelerator such as the GPU. The use of OpenMP and GPU are currently considered experimental and are not widely implemented or tested.

Though additional data-structures may be created and inherited from those already included in the TURF-IR, this practice is discouraged in favor of using existing data-structures to the greatest extent possible so that operations apply in the largest context possible. Both the organizational layout of the data as well as the sequence of operations performed on the data is then constructed at runtime facilitating rapid modular algorithm design. As a result of this highly modular design, library modules of operations can be included or omitted without impacting the functionality of other modules. This "plug-in" model implies that, if a particular release of TURF is missing a module (e.g., a Collisional Radiative physics module); there are no hooks to indicate that a particular module is missing. In this way, reverse engineering the functionality of a module from the generic interfaces defined by the framework is impossible. For an authorized developer, however, it is critical that the interfaces between their modules and the framework are well defined. Actually, having a copy of TURF-IR permits them to test the compatibility of their modules within the framework and helps ensure data-structure compatibility and adherence to the module interface for delivered code.

1.2.1 Key Data-structures

Creating modules that interface efficiently with TURF requires a detailed understanding of the layout of data in memory (i.e. the data-structures). These underlying data-structures are tied to the data being stored (e.g., particle or field data) along with the type of mesh (mesh-free/structured mesh/unstructured mesh) it's stored on but are all based on a custom multi-dimensional matrix object called gkMatrix, a GSOBJect encapsulation and extension of the matrix objects used extensively in AFRL/RQRS's prior research codes ([19] [20], [23], [15], [24]). As a fundamental data object, it is generic templated container class. The mesh classes of SMesh (structured) and UMesh (unstructured) are compound objects of header information and gMatrix objects based on generic mesh classes used also extensively by AFRL/RQRS over the last five years in numerous PhD and Master's Thesis. In addition to mesh objects, the TURF-IR includes examples of basic particle and field data objects with similar header/gMatrix compound structure.

1.2.2 Basic Functionality

For this version of the TURF-IR, the computational domain is centered on a constant spaced global Cartesian coordinate system. Though this "LogicalWorld" object is derived from a more general "World" object, the current version of the framework assumes the existence of the global real to logical coordinate system to facilitate domain patching automation while varying resolution. Active regions of the global coordinate system are defined by "LogicalDomain" objects. These are essentially intended to be discrete axis aligned blocks for the sake of domain decomposition. Computation on every domain should be able to run independent of the others except for discrete synchronization points at which time patches between domains are guaranteed to have completed. These synchronization points are the end of computational stages.

Once the world and domains have been created, various "Operations" are applied. These operations can create additional data objects to attach to the domain, set initial and boundary conditions, solve sets of equations, write output, or any number of other manipulations of the data within the framework so long as they may be applied for one domain at a time independently. Communication between domains is restricted to stage boundaries to preserve this independence. The intent of this is two pronged. The first goal is to encourage as much fine grain parallelization as possible. Extremely broad definition of an operation is also intended to avoiding locking the framework to a specific set of applications due to more rigidly defined interfaces and phases of computation. The function of the framework is then an extremely generic statement of, "There exists a set of data on which a sequence of parallel operations may be applied that can be broken down into discrete stages between which communication between the datasets may be performed". In general, the world typically advances in computational time looping through the operations, but this functionality is only included to assist in timestep synchronization between domains, as this is a commonly required functionality not easily obtainable in a domain independent manner. The form of the time advancement, whether implicit, explicit, or iteration towards steady state, must be defined as built into the sequence of operations rather than a pre-selected as a trait of the framework

1.3 World.list file

The world.list file shown below represents all of the possible input variables. The world.list file is broken up into two different parts. The first sets the WORLD parameters while the second sets up the structured mesh. *NOTE* that field variables and the domain creation may be moved to the operations.list file in future versions.

```

DEFINE WORLD
  NAME = plume_example
  OP_FILE = operations.list
  MATERIAL_LIST = defaults/materials.list
  COORDINATES = cartesian
  ORIGIN = (0.000000, 0.000000, 0.000000)
  DELTA = (0.250000, 0.250000, 0.250000)
  END_TIME = 0.0040001
  START_DT = 0.000002
  FIELDS = pflag, n_Fe, V_Xe, Enx, Eny, Enz, NC, V_Xe+,
  MATERIALS = [Xe@g, Xe+@g, Al@g, Fe@g, Al@s, Fe@s, Cu@s]
  STAGES = [ INITIALIZE, FIELD, MOVE, POSTMOVE ]

## Optional arguments
  PRINT_STATS_SKIP = 0
  PRINT_PROFILE_INFO = True
  SAVE_PROFILE_INFO = True
#   PROFILE_FILE =
  GROW_DT = 1.000000
  START_TIME = 0.000000
  START_ITERATION = 0

## Advanced arguments
  INIT_ONLY = False
  ON_GPU = False
  PLOT_FILE =
  FORCE_VISIT = False
  VELOCITY_COORDINATES = cartesian
  VELOCITY_DELTA = (0.000000, 0.000000, 0.000000)
  VELOCITY_ORIGIN = (0.000000, 0.000000, 0.000000)
  WORLD_RNG_METHOD = RAND_NR1
  DOMAIN_RNG_METHOD = RAND_NR1
  WORLD_RNG_SEED = 0.000000
  DOMAIN_RNG_SEED = 0.000000
  SUPPRESS_OP_PARSER_WARNINGS = FALSE
  SAVE_TIME_INFO = FALSE
  CONFIGURATION_TEST_FILE = FALSE [Deprecated]
  OUTPUT_DIR = ./ [EXPERIMENTAL]
  RESTART_OUTPUT_DIR = "None" [EXPERIMENTAL]
  RESTART_INPUT_DIR = "None" [EXPERIMENTAL]
END WORLD

#####
## Domain Geometry
#####
DEFINE DOMAIN DOM
  BOUND_LO = (-2.500000, -2.500000, -2.500000)
  BOUND_HI = (5.000000, 2.500000, 5.500000)
  SUB_DOMAINS = (2, 2, 2)
  GHOST_CELLS = (1, 1, 1)
END DOMAIN

```

1.3.1 Keywords

The world parameters can be broken up into three groups, required, optional, and advanced.

Required keywords have a large effect on the simulation and need to be set by the user for each simulation.

Table 1.1 Required World.list Keywords

Key	Description	Notes
NAME	The name of the simulation.	Atring
OP_FILE	Name of operation.list file	File name, relative to execution directory.
MATERIAL_LIST	Name of the materials.list file	File name, relative to the execution directory.

COORDINATES	Coordinates type.	Atring, currently only supports Cartesian.
ORIGIN	location of origin	3-Tuple, center of computational domain, is typically (0.0, 0.0, 0.0)
DELTA	Grid spacing.	3-Tuple, grid size of structured mesh.
END_TIME	Final time of the simulation.	Float, typically a good idea to set it to be slightly greater than desired end time.
START_DT	The initial time step.	Float, advanced keyword, GROW_DT sets how much it can change each time step.
FIELDS	Structured mesh field variables	Comma separated list of strings.
MATERIALS	Materials to be included in simulation.	Comma separated list of strings. Materials is a generic term that includes both surface materials and species.
STAGES	Stages present in the simulation.	Comma separated list of strings, should be capitalized.

Optional keywords do not have a large effect on the simulation or are rarely used. All optional keywords have a default value.

Table 1.2 Optional World.list Keywords

Key	Description	Notes
PRINT_STATS_SKIP	Number of iteration between generic output to terminal.	int, default 1.
PRINT_PROFILE_INFO	Print operator level profiling to screen.	boolean, default true.
SAVE_PROFILE_INFO	Save operator level profiling to file.	boolean, default false.
PROFILE_FILE	Name of output file for op level profiling.	file name, default OperationTiming.json. Setting this value sets SAVE_PROFILE_INFO to true.
GROW_DT	How much to grow the time step.	float, default 1.0. Operator dependent.
START_TIME	Initial simulation time.	float, default 0.0. Typically only used for restarts.
START_ITERATION	Initial iteration count.	int, default 0. Typically only used for restarts.

Advanced options are rarely used for normal simulations. Values such as the velocity keywords are only used by Vlasov simulations and can be modified by the CreateVlasovVariableOp operator. The RNG keywords control the random number methods and seeds. For a list of available random number generators please see, Random.

Table 1.3 Advanced World.list Keywords

Key	Description	Notes
CONFIGURATION_TEST_FILE	Runs configuration test from TURF.	Boolean, default False. [DEPRECATED]
FORCE_VISIT	same as running turf with -v.	Boolean name, default False. [DEPRECATED]
INIT_ONLY	Only runs the init function for each operator in the operations.list file.	Boolean, default False.
OUTPUT_DIR	Specifies base output directory	Directory, default none. [EXPERIMENTAL]

ON_GPU	Use the GPU for computation.	Boolean, default False. Requires CUDA and is not currently supported.
PLOT_FILE	Name of a separate plot file info.	File name, default none. [DEPRECATED]
RESTART_INPUT_DIR	Specifies base directory for restart files to be read from.	Directory, default none. [EXPERIMENTAL]
RESTART_OUTPUT_DIR	Specifies base directory to save restart files to	Directory, default none. [EXPERIMENTAL]
SAVE_TIME_INFO	Saves MPI timing information to a file.	Boolean, default False.
SUPPRESS_OPARSER_↔ WARNINGS	Does not show warnings from reading operator.list file.	boolean, default False.
VELOCITY_COORDINATES	Velocity coordinates type.	String, currently only supports cartesian.
VELOCITY_DELTA	Grid spacing for velocity coordinates.	3-Tuple, default (0.0, 0.0, 0.0) Vlasov only.
VELOCITY_ORIGIN	Origin for velocity coordinates.	3-Tuple, default (0.0, 0.0, 0.0) Vlasov only.
WORLD_RNG_METHOD	World random number generation method.	String, default RAND_NR1
DOMAIN_RNG_METHOD	Domain specific random number generation method.	String, default RAND_NR1
WORLD_RNG_SEED	World random number seed value.	Float, default 0.0. Each MPI rank adds its rank number to this value. This prevents MPI ranks from having the same random number seed.
DOMAIN_RNG_SEED	Domain random number seed value.	Float, default 0.0. Each domain adds its domain number to this value. This prevents domains from having the same random number seed.

1.3.2 Logical Domain Definition

The final section of the world.list file concerns the logical domain creation. A logical domain is a six sided box defined by setting the low and high bounds. *Note* that the low and high bounds need to be an integer multiple of the grid spacing in that direction. In the case that the bounds are not an integer multiple of the grid spacing the bounds will be enlarged and snapped to the nearest grid point. This happens because the grid spacing is a world attribute and is enforced on all logical domains.

```

DEFINE DOMAIN <DOMAIN_NAME>
  BOUND_LO = (Xmin, Ymin, Zmin)
  BOUND_HI = (Xmax, Ymax, Zmax)
  SUB_DOMAINS = (Nx, Ny, Nz)
  GHOST_CELLS = (Gx, Gy, Gz)
END DOMAIN

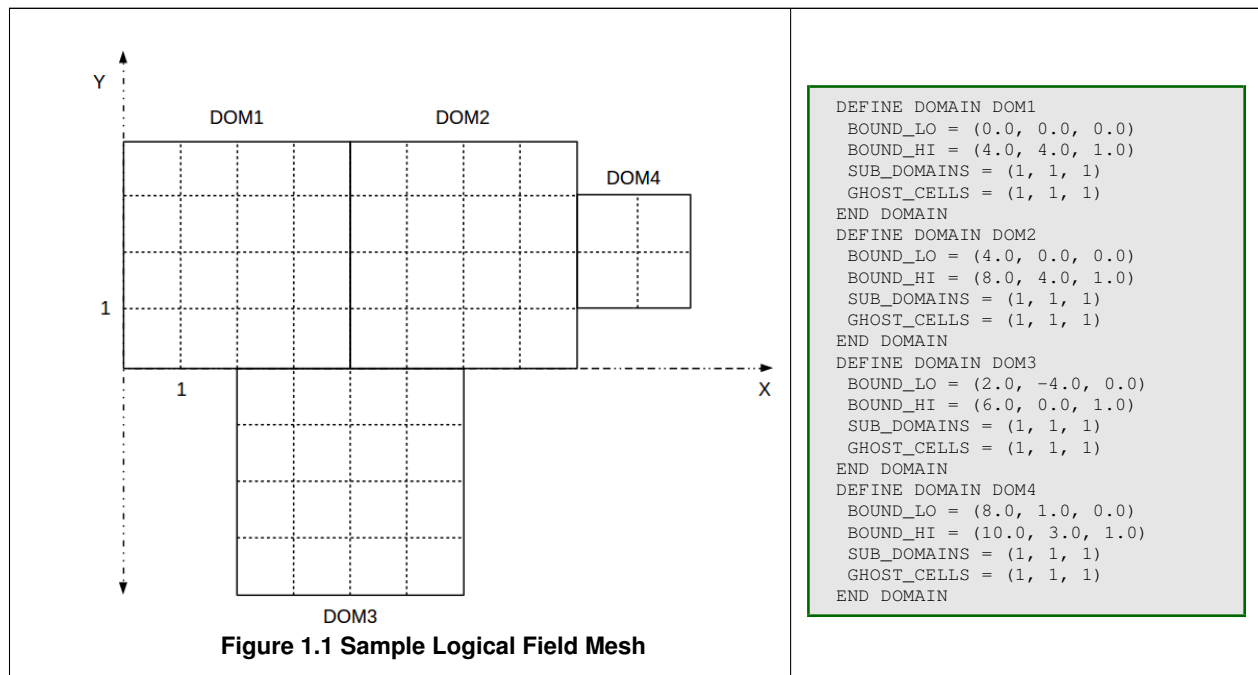
```

Table 1.4 Optional World.list Keywords

Key	Description	Notes
DOMAIN_NAME	The name of the domain needs to be unique.	String, Required. Typically something like DOM
BOUND_LO	The lower bound of the domain.	3-Tuple, Required.
BOUND_HI	The upper bound of the domain.	3-Tuple, Required.
SUB_DOMAINS	Number of subdomains.	3-Tuple, Optional, default to (1,1,1)
GHOST_CELLS	Number of ghost cells. Currently sets to max value provided.	3-Tuple, Optional default to (3,3,3)

If the SUB_DOMAINS option is used a domain index is appended to the end e.g., DOM becomes DOM_X_Y_Z, where the X, Y, and Z are integer subdomains. A subdomain value of (2,1,1) would split the x-direction in two but the y- and z-directions would be left alone.

Consider the sample logical mesh shown in the figure below. In this case, the grid spacing is 1 in each direction and, for simplicity, assumes that the mesh is only one unit thick in the z direction.



1.4 Tutorials

This section has multiple tutorials. New users should start with the Heatbath tutorial before moving onto the Grounded-box tutorial.

Table 1.6 Tutorials

Folder	Description	Solver	Title
Heatbath	Free molecular flow in a specular box	Particle	Heatbath with Multi-Species Particle Data
GroundedBox	Plasma in a grounded box	PIC	Grounded Box: 3D Electrostatic PIC with Multi-Species Particle Data
1DShock	One-dimensional normal shock wave	DSMC	One Dimensional Normal Shock: DSMC with Multi-Species Particle Data
SphereInPlasma	Sphere immersed in a plasma	Hybrid PIC/Fluid	Sphere Immersed in a Plasma: Charging with Hybrid PIC/Boltzmann Fluid Model
AMR	Adaptive Mesh Refinement	AMR	Adaptive Mesh Refinement
Plume/ Plume_1.Base	Baseline plume simulation	Hybrid PIC-MCC/Fluid	EP Plume Simulation 1 : Baseline
Plume/ Plume_2.DSMC	Plume with DSMC Collisions	Hybrid PIC-DSMC/Fluid	EP Plume Simulation 2 : Fractional DSMC Collision
Plume/ Plume_3.Charging	Plume with Spacecraft Charging	Hybrid PIC-DSMC/Fluid + Charging	EP Plume Simulation 3 : Spacecraft Charging
Plume/ Plume_4.AMR	Plume with AMR and dynamic load balancing	Hybrid PIC-DSMC/Fluid + Charging + AMR	EP Plume Simulation 4: AMR and Dynamic Load Balancing
Plume/ CubitExample	Electric propulsion plume simulation	Cubit	EP Plume Simulation: Cubit
Plume	Electric propulsion plume simulation	Paraview	EP Plume Simulation: ParaView

1.5 Heatbath with Multi-Species Particle Data

Author

Samuel J. Araki, Jonathan Tran, and David Bilyeu

1.5.1 Introduction

This tutorial is the simplest among all the other examples in the Thermophysics Universal Research Framework (TURF) and should be the first one to be covered. This tutorial replaces the single species heatbath example used in prior versions of TURF and only uses operators that utilize the MSPDist class object. In the directory `tutorial/Heatbath←_MS/Particle`, there are seven files ending with `.list` extension. These files are listed in Table 1.7. This tutorial contains two `world.list` files, `world.heatbath.list` & `world.heatbathx2.list`, that are used for the single and multiple domain cases respectively. The `world.list` file includes the global parameters that define the entire simulation such as the domain size, the number of stages, time step, etc. To select the desired example you need to comment/uncomment the required `OP_FILE` keyword in the `world.list` file. A full description of the keyword in a `world.list` file can be found [here](#).

In this tutorial, the complexity of the problem is incremented in stages; this helps readers understand how to build a simulation in TURF. In particular, this tutorial demonstrates how to create, move, remove, output, and apply boundary condition to particles. With all the operators turned on, it will simulate free molecular flow in a specular box. Initially, thermal particles are uniformly distributed within a box smaller than the simulation domain. As the simulation progresses the particles become evenly distributed throughout the domain. This tutorial also demonstrates how to extend the simulation to use multi-domain simulations.

Table 1.7 List of Input files

File Name	Description of the Simulation	Example
<code>world.heatbath.list</code>	world.list file for all single domain simulations	A-F
<code>world.heatbathx2.list</code>	world.list file for all multi domain simulations	G
<code>operations.nextstage.list</code>	Time advances but code does nothing	A
<code>operations.addparticles.list</code>	Add particles in a box	B
<code>operations.writeoutput.list</code>	Write vtk field output files periodically	C
<code>operations.push-untrimmed.list</code>	Push particles forward in Time	D
<code>operations.push.list</code>	Trim escaped particles from distribution	E
<code>operations.heatbath.list</code>	Add Reflecting Boundary Conditions at Domain Bound	F
<code>operations.heatbathx2.list</code>	Particle patching between sub-domains	G
<code>operations.heatbathx2_particlepatch.list</code>	Improved Particle patching between sub-domains	H

1.5.2 Setting up a simulation

Running the Simulation

The `world.list` file contains inputs for global information about the simulation as well as the definition of the domain size and how it's partitioned for multi-domain simulations. Each MPI process possesses one World object, and each World object possesses one or more sub-domains. The script file, `world.heatbath.list`, is shown below. For simplicity, we use `world.list` and `operations.list` when speaking in general and will cite a specific `world.list` or `operations.list` file as necessary.

When a simulation starts, TURF always looks for a file named as `world.list`, while this tutorial has two world scripts with slightly different names, `world.heatbath.list` and `world.heatbathx2.list`. Without `world.↵list`, TURF terminates immediately with an error message. There are two options: the first, and preferred method, is to run TURF with the `-input` option,

```
/path/to/TURF-o -input world.heatbath.list
```

The second is to create a symbolic link between `world.list` and one of two file can be made, such that `world.↵list` is routed to the proper world file. A symbolic link in the Linux terminal can be created by

```
tutorial/Heatbath_MS/Particle> ln -s world.*.list world.list
```

where `*` can be “heatbath” or “heatbathx2” for this example. When `world.list` already exists in the directory, `-sf` option can be used to force replacing the existing link.

Once the proper world script is linked to `world.list`, TURF can be run from the directory. For a single domain/single MPI process simulation, simply typing the binary name with its absolute or relative path from the directory will start the simulation. For multiple MPI processes simulation, `mpiexec` command can be used before the binary. Assuming that the binary, `TURF-o`, is created in `bin` directory under `TURF` folder, the command to run TURF is,

```
tutorial/Heatbath_MS/Particl> mpiexec -n N /path/to/TURF-o
```

where `N` is the number of cores to run.

In order to run a case, an input to `OP_FILE` needs to be changed to coincide with one of the `operations.list` file listed in 1.7.

```
DEFINE WORLD
  NAME = Heatbath-Example
  OP_FILE = operations.list # Replace the file name with the one to run
  COORDINATES = cartesian
  ORIGIN = (0.0,0.0,0.0)
  DELTA = (100.0e-6,100.0e-6,100.0e-6)
  END_TIME = 250.1e-9
  START_DT = 1.0e-9
  FIELDS = [ NC NP N ]
  MATERIALS = [ He@g ]
  STAGES = [INITIALIZE, MOVE]
  PRINT_PROFILE_INFO = true
  SAVE_PROFILE_INFO = true
  PRINT_STATS_SKIP = 10
END WORLD
#####
## domain Geometry
#####
DEFINE DOMAIN DOM000
  BOUND_LO = (0.0,0.0,0.0)
  BOUND_HI = (3.2001e-3,3.2001e-3,3.2001e-3)
END DOMAIN
```

Simulation Time and Defining the Grid

In this script, simulation time-step (`START_DT`), end time (`END_TIME`), origin of the coordinate system (`ORIGIN`), and volume mesh cell size (`DELTA`) are defined. With the given time-step and end time, the total number of iterations is 250.

In TURF, the number of grid cells is determined by finding how many `DELTAS` fit in the user-defined domain bounds (`BOUND_HI-BOUND_LO`), and the domain bounds are redefined such that the coordinates of the lowest corner of the domain box are placed at `ORIGIN`. Therefore, the domain box may not necessarily be bounded by the user-defined `BOUND_HI-BOUND_LO`. In this example, the grid contains 32 interior cells in each directions. On top of this, layers of ghost cells are added around the interior mesh. Here, `BOUND_HI` is set to slightly larger numbers than intended; this ensures that the domain size is not reduced by one cell due to the rounding down of the number of cells.

Species and Field Variables

Only helium gas (He@g) is used in this simulation, and the field parameters to be computed are N , NC , and NP as defined in 1.8. For multi-species simulations, field data for each species can be obtained by adding species name followed by an underscore. For example, if there are two species A@g and B@g, partial densities for species A@g and B@g and total density are named as $N_A@g$, $N_B@g$, and N_SUM , respectively.

Table 1.8 Volume mesh fields computed in this heatbath example.

Field Name	Description
NC	Number of simulation particles.
NP	Number of physical particles.
N	Density

Simulation Stages

This example uses two stages (INITIALIZE and MOVE). Stages in TURF serve two important functions:

- MPI communication and synchronization is performed at the end of a stage,
- Sub-cycling, is when a stage repeats itself before advancing to the next stage. This is useful for iterative solvers or for tracking particles as they bounce off of surfaces.

Adding additional stages can help the user to break the simulation into pieces and will not cause a change in the results but it does add a small overhead due to the MPI communication. This particular tutorial does not use the sub-cycling feature but examples of this can be found in the [Grounded Box](#) tutorial and the Plume simulations.

Note that the names for fields and stages are only labels and do not refer to any existing information in the code. On the other hand, the material name must correspond to the name or composition defined in the material database file provided in `src-TURF/src/Materials/database/materials.list`. However, it is important to reference the same name if they are used elsewhere in the code.

Units

TURF is written to assume all units are in MKS and Kelvin for temperature. Many inputs for temperature and energy allow for electron volts to be used as well, when this is allowed a different key word with `_EV` at the end is used, e.g. `ION_TEMPERATURE` vs. `ION_TEMPERATURE_EV`.

Table 1.9 Operation list files and added operations from the last.

Stage	Operation	Description
INITIALIZE	MSPDistInitOp	Create MSPDist objects (P-DST, P-GST, and P-EXC)
	MSPDistBoxICOp	Fill a box with particles and add to P-DST
	MSPDistCombineOp	Combine P-DST and P-EXC to complete patching
	MSPDistCellIDOp	Find cell-id associated with particle location. Particles outside the sub-domain are marked, but there should not be any here in this example.
	MSPDistRemovePartOp	Remove particles outside of entire simulation domain
	NextStageOp	Go to next stage

MOVE	MSPDistWriteVTKOp	Write P-DST particle data to a VTK file
	MSPDistSampleOp	Sample P-DST particles and compute volume mesh field data
	LogicalFieldWriteVTKOp	Output 3D field data to a VTK file
	MSPDistMoveOp	Advancement of P-DST particles
	MSPDistBCSpecOp	Apply specular boundary condition
	MSPDistCellIDOp	Find cell-id associated with particle location. Particles outside the sub-domain are marked.
	MSPDistSplitOp	Split the P-DST particles outside of sub-domain to P-EXC
	GSOPatchOp	Pass P-EXC across sub-domains via MPI communication
	MSPDistParticleCountOp	Output particle count
	NextStageOp	Go to next stage

1.5.3 Operations

Table 1.8 lists all the operations used in this tutorial for the complete heatbath tutorial. In this tutorial, a combination of operations is added in stages in the order listed in 1.7. Text file comparison utilities such as Meld¹ are very useful in finding the new operations added between the examples. Output files from TURF are typically saved in the Visualization ToolKit (VTK) format. Therefore, it is recommended that users have access to scientific visualization software such as ParaView and VisIt.

(A) Go to Next Stage

The first example, `operations.nextstage.list`, demonstrates a case with minimal number of operations. A minimal case contains a single operation per stage that is responsible for progressing to the next stage. Without this operation, TURF will not proceed to the next stage. There are multiple operations that can progress to the next stage but this one uses the `NextStageOp` operation. This one will prevent the simulation from progressing until all of the domains complete the stage. Since this is a single domain case, the simulation will progress right away. Alternatively, the `CriteriaStageOp` can be used to make the stage cycle until a particular criteria is met.

Despite being named `INITIALIZE`, this stage is called every iteration.

```
##
DEFINE STAGE INITIALIZE
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
    TYPE = NextStageOp
END OPERATION
#
END STAGE INITIALIZE
DEFINE STAGE MOVE
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
    TYPE = NextStageOp
END OPERATION
#
END STAGE MOVE
```

¹<http://meldmerge.org>

(B) Add Particles

The second example, `operations.addparticles.list`, creates a `MSPDist` object that holds particle information (`MSPDistInitOp`) and adds particles uniformly within a user-specified box (`MSPDistBoxICOp`). When calling `MSPDistInitOp`, its name (`MSPDIST_DATA_NAME`), the buffer size (`MAX_NP`), and species held by the `MSPDist` object (`SPECIES_NAMES`) are required. It is important to ensure that the number of particles does not exceed the buffer size, otherwise TURF crashes with an error message in the course of the simulation. The second operation, `MSPDistBoxICOp`, defines the bounds of a box (`BOUND_LO` and `BOUND_HI`) and distributes particles sampled from a Maxwellian distribution with the provided temperature (`TEMPERATURE`) and drift velocity (`VEL`) inside it such that it fulfills the number density (`NUMBER_DENSITY`). When the `INIT_ONLY` option is on, particles are created only during the first iteration. The `MSPDistBoxICOp`, by default, truncates the user-defined box such that it fits within the simulation domain, which can be turned off by enabling the `BOX_OUTSIDE_DOMAIN` option.

```
#####
## Create Particle Distribution ##
#####
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-DST
  MAX_NP = 1280000
  SPECIES_NAMES = He@g
END OPERATION
#####
## Fill a Box with Particles ##
#####
DEFINE OPERATION
  TYPE = MSPDistBoxICOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = He@g
  BOUND_LO = (0.10e-3,0.10e-3,0.10e-3)
  BOUND_HI = (2.32e-3,2.32e-3,2.32e-3)
  TEMPERATURE = 11604.5059
  NUMBER_DENSITY = 1.0e14
  REAL_TO_COMPUTATIONAL = 10.0
  VEL = (0.0,0.0,0.0)
  INIT_ONLY = TRUE
  BOX_OUTSIDE_DOMAIN = FALSE
END OPERATION
```

(C) Output Field Data and Particles

The third example, `operations.writeoutput.list`, builds upon the previous example by adding the capability to save particle and field variables to output files. The plotting is accomplished through the `LogicalFieldWriteVTKOp`, `MSPDistWriteVTKOp`, and `MSPDistParticleCountOp` that saves the field variables, particle location and velocity, and the total number of particles per domain respectively. A fourth operation, `MSPDistSampleOp`, is used to sample mass, momentum, and kinetic energy from a particle distribution and computes volume mesh field data from these sampled quantities. The top left figure in 1.11 shows the initial particle distribution.

- `LogicalFieldWriteVTKOp`
 - `FIELD_DATA_NAME` specifies which field data set contains the variables. By default, the field variables defined in the `world.list` file belongs to the `FieldData` object.
 - `FIELD_NAMES` lists the variables to save.
 - `FILE_HEAD` specifies the output file names. In this case, multiple files are saved so only the head is saved; the iteration number and proper suffix will be added.
 - `PLOT_GHOST` is a boolean flag that toggles saving data from the ghost cell regions.
 - `SKIP` specifies how many iterations pass in between saves.
 - `FORMAT` allows saving the data in either ASCII or Binary format. The binary format uses a base64 encoding and is platform independent.

- COMPRESS_DATA will compress the binary format; it has no affect for ASCII format.
- MSPDistWriteVTKOp
 - FILE_HEAD, SKIP, FORMAT, and COMPRESS_DATA are the same as above.
 - MSPDIST_DATA_NAME is the name of the MSPDist that the variables are saved in. For this case it is P-DST
 - PARTICLE_SKIP can use used to skip particles during printing, useful for reducing the size of the output files.
 - SPECIES requires a list of species from MSPDist to save; alternatively, you can specify ALL to save all of the species in the list.
 - OVERWRITE, if set to true, it will only keep one output file and overwrite it each time this is run, this is useful for restart files.
 - BACKUP_ON_OVERWRITE, when used in conjunction with the OVERWRITE it will make a backup of the previous output file before writing the current file. Only one backup file is kept.
 - Note, a MSPDist file can be read into a simulation through the MSPDistReadVTKOp operator.
- MSPDistParticleCountOp
 - MSPDIST_DATA_NAME same as above
 - LOG_FILE is the name of the output file
 - GLOBAL provides a global count of the total number of particles across all domains.

The fourth operator, MSPDistSampleOp, is a little more complex.

- MSPDistSampleOp
 - FIELD_DATA_NAME and MSPDIST_DATA_NAME are the same as above.
 - FIELD_NAMES specifies which fields to sample, options are
 - * NC_ number of computational particles (#)
 - * NP_ number of physical particles (#)
 - * N_ Number density ($\#/m^3$)
 - * n_ Number density ($\#/m^3$)
 - * VX_, VY_, VZ_ x, y, and z components of velocity (m/s)
 - * V_ velocity magnitude (m/s)
 - * TX_, TY_, TZ_ x, y, and z components of temperature (K)
 - * T_ scalar temperature (K)
 - * Either the species name or SUM is required after the underscore.
 - MIX_START_TIME, time to start time-averaging the fields, before this time the sampling is instantaneous.
 - STEPS_PER_SAMPLING, frequency to sample particles to compute moments, e.g. number of particles, momentum, and kinetic energy.
 - STEPS_PER_FIELD_CALCULATION, frequency to calculate field quantities, e.g. density, velocity, and temperature.

MSPDistSampleOp is one of the more complicated and numerically expensive ops used in this simulation. What follows are some notes and best practices when using this op.

- The main purpose of MIX_START_TIME is to reduce the statistical noise by averaging the field values over time. The downside is that it will suppress transient features in the flow.

- `MIX_START_TIME` should be set to a time after the simulation has reached steady-state.
- Higher `STEPS_PER_SAMPLING` values will provide less accurate but faster results. It is recommended to split this Op into multiple ones;
 - Field variables that are used by other ops, e.g. density for MCC calculations, should have a `STEPS_PER_SAMPLING` and `STEPS_PER_FIELD_CALCULATION` close to unity,
 - Field variables that are only used for plotting can have a `STEPS_PER_SAMPLING` large enough to reduce statistical noise and a `STEPS_PER_FIELD_CALCULATION` that corresponds to plotting frequency `SKIP` in `LogicalFieldWriteVTKOp`.
- `MSPDistSampleOp` will not sample ghost cells so if `PLOT_GHOST` is set to `True` in `LogicalFieldWriteVTKOp` those cells will have a value of zero.
- If `STEPS_PER_FIELD_CALCULATION` is set to 1 there may be numerical artifacts caused by a fixed macro-particle weight particle injection scheme. E.g. if 10.3 particles needs to be injected only 10 macro particles will be injected and the remainder, .3, is injected during the next time step.

```
#####
## Sum to Fields for Output ##
#####
DEFINE OPERATION
  TYPE = MSPDistSampleOp
  FIELD_DATA_NAME = FieldData
  MSPDIST_DATA_NAME = P-DST
  FIELD_NAMES = NC NP N
  MIX_START_TIME = 1.0e-2
  STEPS_PER_SAMPLING = 1
  STEPS_PER_FIELD_CALCULATION = 5
END OPERATION
#####
## Write Field Data ##
#####
DEFINE OPERATION
  TYPE = LogicalFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = plot3D/plt_
  FIELD_NAMES = NC NP N
  PLOT_GHOST = FALSE
  SKIP = 5
END OPERATION
#####
## Output Particles ##
#####
DEFINE OPERATION
  TYPE = MSPDistWriteVTKOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = ALL
  FILE_HEAD = particle/MSPDist_
  SKIP = 5
  PARTICLE_SKIP = 1
  OVERWRITE = FALSE
END OPERATION
#####
## Output particle count ##
#####
DEFINE OPERATION
  TYPE = MSPDistParticleCountOp
  MSPDIST_DATA_NAME = P-DST
  LOG_FILE = ParticleCount.csv
  VERBOSE = FALSE
  GLOBAL = TRUE
END OPERATION
```

(D) Move Particles

The fourth example, `operations.push-untrimmed.list`, advances particles to the time corresponding to the next iteration. In this example, the particle movement is controlled through the `MSPDistMoveOp` operator. This

Op uses the explicit first-order Euler method and does not take into account any external forces such as electric and magnetic fields. If charged particles are to be advanced in an electric field, `MSPDistESPushOp` should be used instead. This example does not include boundary conditions so the particles will quickly leave the computational domain will not be removed from the simulation. The top right figure in 1.11 shows the particle distribution at the final time.

- `MSPDistMoveOp`
 - `MSPDIST_DATA_NAME`, same as previous.
 - `SPECIES_NAMES`, particles species in `MSPDIST_DATA_NAME` to be moved.
 - `USE_CFL_TIME_STEP`, limits how far a particle will move during a single push. This requires that the `Stage` has a `CriteriaStageOp` instead of a `NextStageOp` so that it will loop until all particles move a full time step.
 - `CFL_NUMBER`, only used if `USE_CFL_TIME_STEP` is true. The target CFL number.

```
#####
## Advance Particle Positions ##
#####
DEFINE OPERATION
  TYPE = MSPDistMoveOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_NAMES = He@g
  USE_CFL_TIME_STEP = False
END OPERATION
```

(E) Trim Particle List

The fifth example, `operations.push.list`, improves upon the previous one by removing particles that have left the computational domain. This is accomplished through the use of the following three operators: `MSPDistCellIDOp`, `MSPDistRemovePartOp`, and a new Multi-Species particle list via `MSPDistInitOp`. The operator `MSPDistCellIDOp` determines if the particle is inside the computational domain, if it is then it determines the volume mesh cell that the particle resides in, in an (i, j, k) format then determines the cell's unique cell-id. When the particle is outside of the computational domain it is assigned a cell-id that is larger than the greatest valid cell-id. Then, `MSPDistRemovePartOp` can remove these particles from the `MSPDist`. Note that `MSPDistRemovePartOp` will only remove particles if outside of the domain if `REMOVE_CELLID_GT_MAX` is set to True. The particles can also be removed based on their weights by turning `REMOVE_ZERO_WEIGHT` option on. To remove particles, the particle list is first copied from `P-DST` to `P-GST`, and then only the ones to be kept are copied back to `P-DST`. In this way, the operation can be executed in parallel with OpenMP. The deletion of particles can also be done with `MSPDistSortOp`, but this operation sorts particles based on their cell-ids in addition to removing particles². The bottom left figure in 1.11 shows the particle distribution at the final time.

Here is a brief list of the key words for the new operators:

- `MSPDistInitOp`
 - Same as previous
- `MSPDistCellIDOp`
 - `MSPDIST_DATA_NAME`, name of particle distribution.
- `MSPDistRemovePartOp`

²The Direct Simulation Monte Carlo collision calculation can be more efficient when particles are sorted in advance. For this case, 'MSPDistSortOp' is used in place of 'MSPDistRemovePartOp'

- MSPDIST_SRC_NAME, name of particle distribution to remove particles from
- MSPDIST_DST_NAME, name of particle distribution to send particle to
- REMOVE_ZERO_WEIGHT, remove particles that have zero weight. Some operators assign a particle a macro-particle weight of zero if they should be removed.
- REMOVE_CELLID_GT_MAX, remove particles that have a cell-id greater than the maximum one. Since the background mesh is Cartesian there is an analytical expression to go from an i,j,k index to a single cell-id value that is unique.
- DISCARD_REMOVED_PARTICLE, if False removed particles will be copied back into MSPDIST_DST_NAME.

The resulting simulation from `operations.push.list` and `operations.push-untrimmed.list` should produce the same results but the untrimmed version will typically be less computational efficient. For this particular example, the untrimmed example took about 20% longer to complete. This may not be the best comparison since over 90% of the simulation time is spent in plotting. If `LogicalFieldWriteVTKOp` and `MSPDistWriteVTKOp` were reduced to removed completely then the trimmed simulation would be nearly 60% slower than untrimmed version.

```
#####
## New Particle Distribution ##
#####
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-GST
  MAX_NP = 1280000
  SPECIES_NAMES = He@g
END OPERATION
#####
## Setting cell-id==Max for the Ones out of Domain ##
#####
DEFINE OPERATION
  TYPE = MSPDistCellIDOp
  MSPDIST_DATA_NAME = P-DST
END OPERATION
#####
## Remove Particles If Outside the Domain ##
#####
DEFINE OPERATION
  TYPE = MSPDistRemovePartOp
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-GST
  REMOVE_ZERO_WEIGHT = FALSE
  REMOVE_CELLID_GT_MAX = TRUE
  DISCARD_REMOVED_PARTICLE = TRUE
  VERBOSE = FALSE
END OPERATION
```

(F) Apply Boundary Condition

The sixth example, `operations.heatbath.list`, demonstrates the use of the specular particle boundary condition, `MSPDistBCSpecOp`, to keep particles inside of the computational domain.

- `MSPDistBCSpecOp`
 - `MSPDIST_DATA_NAME` and `SPECIES_NAMES` are the same as previous.
 - `DIRECTION`, specifies which direction the particles will reflect and are specified as `xm`, `xp`, `ym`, `yp`, `zm`, and `zp`, and the 'm' and 'p' in these directions refer to "minus" (from negative to positive direction) and "plus" (vise versa), respectively.
 - `BOUND_LO` & `BOUND_HI` specifies a box that the particles will bounce off of.
- Notes

- In the heatbath example, particles are reflected off six different surfaces such that `MSPDistBCSpecOp` is required to be defined six times, each with different inputs for `DIRECTION`, `BOUND_LO`, and `BOUND_HI`.
- It is important to note that particles may escape the domain if they never land within these boxes defined for `MSPDistBCSpecOp`. In order to prevent this situation, the box has to be sufficiently large and/or the particle time-step has to be sufficiently small.
- The box should be placed such that it slightly overlaps the simulation domain. This is to prevent particles landing exactly on the domain edge and lost.

Another way to apply boundary conditions is through the `MSPDistSurfaceInteractionOp` operator, but this operation requires a surface mesh and is more computationally expensive. The `MSPDistSurfaceInteractionOp` will be addressed in future tutorials. A graphical representation on how the `MSPDistBCSpecOp` is applied is shown in 1.10. In this figure:

- Sub-figure A is the boundary setup where the solid-black line with a gray gradient is the boundary, the red box is the Box along the right boundary, and the blue Box is the Box along the top boundary.
- Sub-figure B shows a simple one bounce specular reflection. The solid arrow shows what the "physical" bounce would look like while the dashed lines show how it is applied by the algorithm. In this case, the particle starts at x_0 and ends its step at x'_1 ; the `MSPDistBCSpecOp` Op takes the particle and reflects it along the dashed red line to the point x_1 . The dashed black line shows the path that the particle takes in the ghost cell.
- Sub-figure C shows a more complex two bounce specular reflection. In this case, it takes two boundary conditions to ops to move the particle back into the domain. The first from x'_1 to x''_1 reflects along the x boundary and the second reflects across the y boundary from x''_1 to x_1 .

The bottom right figure in 1.11 shows the particle distribution at the final time. By introducing the boundary condition, particles are distributed nearly uniformly at the end of simulation.

```
#####
## Boundary Conditions ##
#####
# Specular boundary for +x to make the problem 1D
DEFINE OPERATION
  TYPE = MSPDistBCSpecOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_NAMES = He@g
  DIRECTION = xm
  BOUND_LO = (-10.0e-4, -10.0e-4, -10.0e-4)
  BOUND_HI = (0.00001e-4, 42.0e-4, 42.0e-4)
END OPERATION
# Specular boundary for -x to make the problem 1D
DEFINE OPERATION
  TYPE = MSPDistBCSpecOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_NAMES = He@g
  DIRECTION = xp
  BOUND_LO = (31.9999e-4, -10.0e-4, -10.0e-4)
  BOUND_HI = (42.0e-4, 42.0e-4, 42.0e-4)
END OPERATION
```

Table 1.10 Boundary Conditions

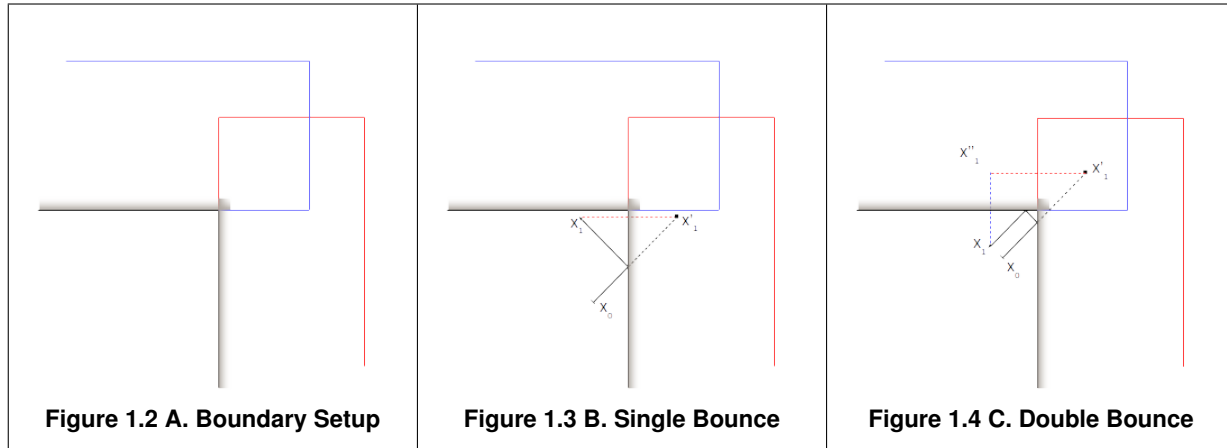
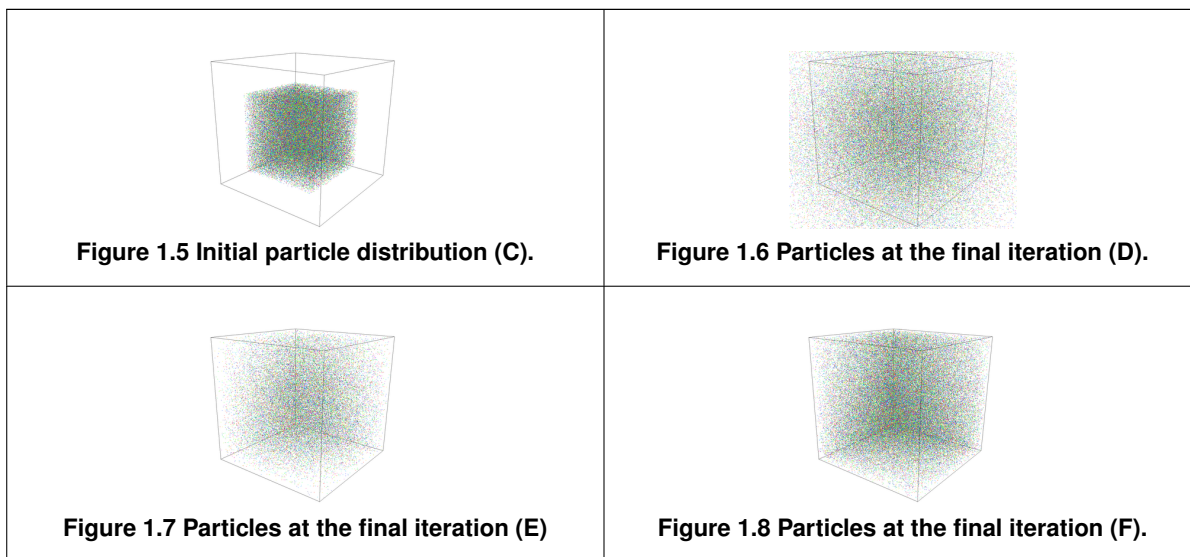


Table 1.11 Initial and final particle positions from different examples. These outputs are from ParaView.



(G) Particle Patching with MPI Communication

The final example shows how to modify the simulation so that it runs in parallel. There are two aspects of the simulation that need to be modified, the first is to tell TURF how to partition the domain into sub-domains and the second involves modifying the `operations.list` file so that information can be passed between domains at the appropriate time.

In order to run this example, either `world.heatbathx2.list` should be linked to `world.list` or TURF should be run with the `-input world.heatbathx2.list` option. In `world.heatbathx2.list`, the simulation domain is set up as follows.

```

DEFINE DOMAIN DOM
  BOUND_LO = (0.0,0.0,0.0)
  BOUND_HI = (3.2001e-3,3.2001e-3,3.2001e-3)
  SUB_DOMAINS = (2,1,1)
END DOMAIN

```

The SUB_DOMAINS keyword divides the domain evenly into the number of sub domains specified in each direction. In this case, a value of (2, 1, 1) divides the domain once in the x-direction creating two domains. The same domain decomposition can be achieved by manually defining each domain.

```

DEFINE DOMAIN DOM000
  BOUND_LO = (0.0,0.0,0.0)
  BOUND_HI = (1.6e-3,3.2e-3,3.2e-3)
END DOMAIN
DEFINE DOMAIN DOM001
  BOUND_LO = (1.6e-3,0.0,0.0)
  BOUND_HI = (3.2001e-3,3.2001e-3,3.2001e-3)
END DOMAIN

```

In addition to the updated domains, a new STAGE (POSTMOVE) has been added; this stage is not used in this example but it will be used in the next example.

In this world.list file there are two operations.list files to select from, the first operations.heatbathx2.list is the original version and uses the generic GSOPatchOp to transfer the particle distribution between processes while the second one operations.heatbathx2_particlepatch.list uses the more specialized MSPDistPatchOp to transfer the particle list between processes. GSOPatchOp is a generic patch op that can be used to send any GSObject from one domain to its neighbors, the downside of this operator is that it is not computationally efficient.

When running with multiple sub-domains, additional operations are required to properly exchange particle information between the sub-domains. In TURF, the sub-domains do not overlap with each other. However, the ghost cell layers around a sub-domain will overlap with the interior cells of its neighboring sub-domains. TURF passes the information stored in the ghost layers to a sub-domain through MPI communications. Since MPI communication is performed between stages, particle patching is performed across at least two stages. This also requires a new particle distribution to be created to store the particles that need to be transferred. The following is a brief outline of the steps required for

- Create a new particle distribution P-EXC to hold the particles to transfer. The buffer size for P-EXC should be sufficient but small for efficient communication between processes.
- Use MSPDistCellIDOp to determine which particles need to be transferred between domains.
- Use the MSPDistSplitOp operator to move particles from the primary distribution (P-DST) to the exchange distribution P-EXC.
- GSOPatchOp tells TURF to pass them to a process that owns the sub-domain.
- Finally, these particles are combined with the main particle distribution, P-DST, through MSPDistCombineOp.
- Notes:
 - MSPDistInitOp has two options to set the size of the particle distribution, MAX_NP and GLOBAL_MAX_NP. MAX_NP determines how many particles each domain/process will create while GLOBAL_MAX_NP will specify how many particles to have in the entire simulation. Since MAX_NP was used, each domain will create 1.28M particles, so with two domains a total of 2.56M can exist in the simulation. If GLOBAL_MAX_NP was used instead, each domain would only allocate enough space for 640,000 particles.
 - In the rare case that a particle moves too far during a single time step and leaves the box defined in the MSPDistBCSpecOp operators, MSPDistRemovePartOp is called afterward to remove particles that escaped from the entire domain.

- MSPDistCombineOp can be placed in any stage after the GSOPatchOp and before the MSPDistMoveOp
- MSPDistCombineOp will delete the contents of the MSPDIST_SRC_NAME after its contents are copied to MSPDIST_DST_NAME.
- GSOPatchOp sends the entire particle distribution, not just the parts that have been set. Since P-EXC was allocated to hold 10,000 particles it will always send 10,000 worth of data to each of its neighbors.

```
#####
## Setting cell-id==Max for the ones out of domain ##
#####
DEFINE OPERATION
  TYPE = MSPDistCellIDOp
  MSPDIST_DATA_NAME = P-DST
END OPERATION
#####
## Move cell-id==Max particles to P-EXC ##
#####
DEFINE OPERATION
  TYPE = MSPDistSplitOp
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-EXC
END OPERATION
#####
## Patch MSPDist ##
#####
DEFINE OPERATION
  TYPE = GSOPatchOp
  SRC_NAME = P-EXC
  DST_NAME = P-EXC
END OPERATION
#####
## Combine MSPDist after patching ##
#####
DEFINE OPERATION
  TYPE = MSPDistCombineOp
  MSPDIST_SRC_NAME = P-EXC
  MSPDIST_DST_NAME = P-DST
  VERBOSE = FALSE
END OPERATION
```

(H) Improved Particle Patching with MPI Communication

The example improves upon the parallel simulation presented in the previous section by drastically reducing the amount of time spent sending data between processes. To accomplish this we use the MSPDistPatchOp operator instead of the generic GSOPatchOp. Without going into too much detail, the MSPDistPatchOp is more efficient due to better selection rules about which particles to send where and reduces the size of the arrays to send via MPI. For example, if P-EXC was allocated to hold 10,000 particles but only 10 particles need to be sent it will only send the data associated with the 10 particles. To implement the new patching method requires the following changes from operations.heatbathx2.list:

- The MSPDist used to hold the exchange particles need to have the GLOBAL_DIST flag set to True in MSPDistInitOp.
- The MSPDistCombineOp is replaced by MSPDistCopyOp. MSPDistCombineOp deleted the source distribution but since a single distribution is used to hold the particle distribution for other domains as well this will delete the data before it's read by other processes.
- MSPDistSplitOp and GSOPatchOp are no longer needed and are removed.
- A new stage POSTMOVE is added
- MSPDistPatchOp is added to the POSTMOVE stage along with the required NextStageOp operator.

Summary of new and updated operators

- MSPDistCopyOp
 - MSPDIST_SRC_NAME, name of source distribution
 - GLOBAL_SRC_DIST, boolean if True then TURF will look for MSPDIST_SRC_NAME in the world instead of the domain.
 - MSPDIST_DST_NAME, name of destination distribution.
 - GLOBAL_DST_DIST, same as GLOBAL_SRC_DIST but for MSPDIST_DST_NAME.
 - ONLY_COPY, if False then the destination MSPDIST_DST_NAME will be reset before the copy and MSPDIST_SRC_NAME will be reset after the copy. For this simulation, it must be set to true.
- MSPDistPatchOp
 - MSPDIST_SRC_NAME and GLOBAL_SRC_DIST are same as above
 - GLOBAL_MSPDIST_NAME name of global distribution used to transfer particles. When this MSPDist is initialized in MSPDistInitOp, it must have the GLOBAL_DIST set to true.

```
#####
## Create Particle Distribution ##
#####
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-EXC
  MAX_NP = 10000
  SPECIES_NAMES = He@g
  GLOBAL_DIST = TRUE
END OPERATION
#####
## Combine MSPDist after patching ##
#####
DEFINE OPERATION
  TYPE = MSPDistCopyOp
  MSPDIST_SRC_NAME = P-EXC
  GLOBAL_SRC_DIST = TRUE
  MSPDIST_DST_NAME = P-DST
  GLOBAL_DST_DIST = FALSE
  ONLY_COPY = TRUE
END OPERATION
#####
## Perform Particle Patch ##
#####
DEFINE OPERATION
  TYPE = MSPDistPatchOp
  MSPDIST_SRC_NAME = P-DST
  GLOBAL_SRC_DIST = FALSE
  GLOBAL_MSPDIST_NAME = P-EXC
  VERBOSE = FALSE
END OPERATION
```

1.5.4 Useful Tools

Heatbath-Example. The branches include `GXObject` named `GSMemberVector` (which have the functionality of a vector and can be used by the GPU), a material database, a logical domain, and the coordinate system defined by the `world.list` file. It is possible to expand the hierarchy to investigate any underlying databases or arrays that are automatically generated.

Another useful feature is the visualization of the simulation environment as shown in [World-Rank](#). The green cubic box represents the simulation domain, and the surrounding gray region is a layer of three ghost cells that are automatically generated when the domain is formed. By selecting the visualization and pressing the 'm' key, the user can cycle through volume view, line view, and point view. Also shows the region where particles are distributed during the first iteration (Example B). Furthermore, [World-Rank](#) shows the boxes used for the specular reflection of particles (Example F).

Graphical User Interface (In Development)

As TURF runs through the "initialization" of class objects, it gathers the input parameters available for each object from the source code. TURF also parses actual inputs from `world.list` and `operations.list` files. With these data, an html file named `index.html` is created, which is essentially a Graphical User Interface (GUI) to generate/modify TURF input files, `world.list` and `operations.list`. TURF has gone through a large number of changes since the first release of the GUI. For that reason, the GUI is currently broken and is subjected to be fixed in the near future.

1.6 Grounded Box: 3D Electrostatic PIC with Multi-Species Particle Data

Author

Samuel Araki, Robert Martin, and David Bilyeu

1.6.1 Included Files

The input files for this tutorial are located in the `tutorials/GroundedBox_MS/ES-PIC` directory and contains the following files.

- `world.list`
- `operations.list`

1.6.2 Introduction

This tutorial demonstrates a fully kinetic 3D electrostatic particle in cell (PIC) case in the Thermophysics Universal Research Framework (TURF). This tutorial assumes familiarity with the [heatbath tutorial](#). New users are referred to the tutorial for further explanation.

The grounded box test case was developed to verify TURF's PIC algorithms with respect to AFRL/RD's ICEPIC particle in cell code running in electrostatic PIC mode [10]. The initial conditions are a uniform unit meter cube of zero velocity protons at a density of 10^{10} m^{-3} . In one octant of the cube, the proton charge is neutralized with 10^{10} m^{-3} electrons initially at stationary. The walls of the cube are set to a fixed 0 Volt potential. The electrons are then accelerated by the field due to the charge of the non-neutralized protons in the remaining 7 octants of the box. The field evolves as the electrons accelerate such that the cloud oscillates and evolves within the box. Particles that hit the edge of the box are assumed to be neutralized and removed from the simulation.

This tutorial takes a long time to reach equilibrium and can be shortened by changing the simulation time in the `world.list` file.

1.6.3 world.list

Running the TURF executable in the working directory will have TURF search for the default script file, `world.list`, and parse it automatically. The first block that defines the `WORLD` is shown below. The options defined in this file should look familiar after completing the [heatbath tutorial](#). In this example, the world is named `ICEPIC-Bench` to denote that it was originally intended to serve as a benchmark verification run against the ICEPIC code. The example uses the `operations.list` operations file to define the simulation algorithm that will be discussed below. The remainder of the world definition sets a global Cartesian coordinate system with 2 cm cells along with 2.5 ns time steps up to a final simulation time of 10 μs . The next line defines 10 field variables for charge (`Q`), charge density (`rho`), 3 node-centered electric field components (`En`), the electrostatic potential (`phi`), an auxiliary variable for calculating the residual of the potential during the field solve (`residual_phi`), and proton and electron physical and computational particle counts in cells (`NP_H+`, `NP_e-`, `NC_H+`, and `NC_e-`).

The example run is broken into four stages named `INITIALIZE`, `FIELD`, `MOVE`, and `PLOT`. The two additional stages compared to the [heatbath](#) example are to accommodate an iterative electrostatic potential solve stage (`FIELD`) and to ensure synchronization prior to the plotting operation stage (`PLOT`), though the latter is not strictly necessary because this simulation runs in serial. In this tutorial, the operations file will be discussed in stages in the [Operations](#) section. A list of operations defined in `operations.list` is given in [Table 1.13](#). The last section of `world.list` defines the active domain for the simulation. This example is simply a 1m unit cube starting from the coordinate origin. Using the global mesh spacing of 2 cm from the `WORLD` definition results in a $50 \times 50 \times 50$ active cell cube with the default 3 ghost-cells added to the high and low side in each direction for application of boundary conditions.

```

DEFINE WORLD
  NAME = ICEPIC-Bench
  OP_FILE = operations.list
  COORDINATES = cartesian
  ORIGIN = (0.0,0.0,0.0)
  DELTA = (0.02,0.02,0.02)
  START_DT = 2.50e-9
  END_TIME = 10.0e-6
  FIELDS = [Q, rho, Enx, Eny, Enz, phi, residual_phi, NC_e-, NC_H+, NP_e-, NP_H+ ]
  MATERIALS = [e-, H+@g]
  STAGES = [INITIALIZE, FIELD, MOVE, PLOT]
  START_ITERATION = 0 # Number of Poisson Iteration Before Start
  PRINT_PROFILE_INFO = true
  SAVE_PROFILE_INFO = true
  PRINT_STATS_SKIP = 100
END WORLD
#####
## Domain Geometry
#####
DEFINE DOMAIN DOM000
  BOUND_LO = (0.0,0.0,0.0)
  BOUND_HI = (1.0,1.0,1.0)
END DOMAIN

```

World.list file for the GroundedBox simulation.

Table 1.13 Summary of operations listed in operations .list

Stage	Operation	Description
INITIALIZE	MSPDistInitOp	Create particle distribution
	MSPDistBoxICOp	Initialize particle distribution within the domain
	MSPDistCellIDOp	Flag cell in which particle resides
	MSPDistSortOp	Sort particles in cells by Cell-id
	MSPDistSampleOp	Sum particles to cells
	FieldSetOp	Set field values to constant
	FieldArithmeticCoeffOp	Compute total charge
	NextStageOp	Continue to next stage
FIELD	LogicalBCOp	Set value of cell centers in box every iteration
	LogicalPoissonStripOp	Red/Black line relaxing Poisson solve
	LogicalResidualOp	Calculate residual of Poisson solve
	LogicalNormOp	Calculate L^p -norm of field variable
	CriteriaStageOp	Continue to next stage if quantity below criteria
MOVE	LogicalNodeGradientOp	Calculate node-centered gradient of cell center field
	MSPDistESPushOp	Electrostatic particle push using node electric field
PLOT	VolumeRenderOp	Single cubic domain real-time volume rendering
	LogicalFieldWriteVTKOp	Write to output files for 3D plots.

1.6.4 operations.list

Stage: Initialize

This stage creates the MSPDist objects that hold electron and proton macro-particles, distributes these particles uniformly within a box, and removes particles that are outside of the simulation domain. The MSPDistInitOp and MSPDistBoxICOp operations should be familiar from the heatbath example. "Ghost" particle distributions are also created, again using MSPDistInitOp. These are empty buffers where particles that have escaped the domain get copied later on in the sort.

```

DEFINE STAGE INITIALIZE
#####
## Initial Particle Distributions and Ghost/Exchange Distributions ##
#####
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-DST
  MAX_NP = 4000000
  SPECIES_NAMES = e- H+@g
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-GST
  MAX_NP = 4000000
  SPECIES_NAMES = e- H+@g
END OPERATION
#####
### Distribute Particles in a Box Uniformly ###
#####
DEFINE OPERATION
  TYPE = MSPDistBoxICOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = e-
  BOUND_LO = (0.0,0.0,0.0)
  BOUND_HI = (0.5,0.5,0.5)
  TEMPERATURE = 0.0
  NUMBER_DENSITY = 1.0e10
  REAL_TO_COMPUTATIONAL = 1.25e3
  VEL = (0.0,0.0,0.0)
  INIT_ONLY = True
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistBoxICOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = H+@g
  BOUND_LO = (0.0,0.0,0.0)
  BOUND_HI = (1.0,1.0,1.0)
  TEMPERATURE = 0.0
  NUMBER_DENSITY = 1.0e10
  REAL_TO_COMPUTATIONAL = 8.0e4
  VEL = (0.0,0.0,0.0)
  INIT_ONLY = True
END OPERATION

```

The next set of operations sort the particles by the cell in which they reside. Though still part of the INITIALIZE stage, they will run every time the simulation loops back through the INITIALIZE stage. The MSPDistCellIDOp operation identifies which cell the particle resides in and saves it to the CellID variable within the particle distribution. The MSPDistSortOp operation sorts the particles by their Cell-ids and any particle that has escaped the domain gets separated into the ghost distribution, P-GST.

```

#####
## Remove Particles Leaving the Domain ##
#####
DEFINE OPERATION
  TYPE = MSPDistCellIDOp
  MSPDIST_DATA_NAME = P-DST
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistSortOp
  NAME = Sort_P-DST
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-GST
END OPERATION

```

The next section accumulates particle quantities into the cell field variables. The accumulation of field data is performed by MSPDistSampleOp. It sets the diagnostic fields for number of computational (NC_x) and real (NP_x) particles per cell. It is worth noting that these numbers are both raw sums. Since this is a transient, i.e. a time accurate simulation, and there are sufficient particles to suppress noise, the MIX_START_TIME variable is set to a large value so that the field values are not averaged over time. Setting both STEPS_PER_SAMPLING and STEPS_PER_FIELD_↵ CALCULATION to one have a similar effect but setting MIX_START_TIME high prevents that particular routine from running and saves computational time.

Before the data can be accumulated, the field variable Q (charge) must be cleared using `FieldSetOp`, and then `FieldArithmeticCoeffOp` calculates the total charge by multiplying the real particle counts, `NP_x`, by their respective charges and summing the values.

```
#####
## Sum to Fields ##
#####
DEFINE OPERATION
  TYPE = MSPDistSampleOp
  FIELD_DATA_NAME = FieldData
  MSPDIST_DATA_NAME = P-DST
  FIELD_NAMES = NC_e- NC_H+ NP_e- NP_H+
  MIX_START_TIME = 1.0 # Don't mix
  STEPS_PER_SAMPLING = 1
  STEPS_PER_FIELD_CALCULATION = 1
END OPERATION
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = Q
  VALUE = 0.0
  OPERATION = SET
END OPERATION
DEFINE OPERATION
  TYPE = FieldArithmeticCoeffOp
  FIELD_DST_NAME = Q
  FIELD_SRCB_NAME = NP_e-
  FIELD_SRCC_NAME = NP_H+
  COEFFICIENT_B = -1.602189200e-19
  COEFFICIENT_C = 1.602189200e-19
  OPERATION = ADD
END OPERATION
```

Finally, the `INITIALIZE` stage is completed with the `NextStageOp` operation to proceed to the `FIELD` stage.

```
# Default Criteria to Proceed to the Next Stage
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE INITIALIZE
```

Stage: Field

This stage iterates on solving for the electrostatic potential until the residual is small enough to proceed. The first step of the iterative field solve is to set the boundary condition potential to 0 on all six faces of the box. This is done using the `LogicalBCOp` operation. The operation is relatively straightforward.

- The field variables identified through the `FIELD_NAMES` keyword and is set to the value defined by the keyword `VALUE`.
 - In this example `FIELD_NAMES = phi` and `VALUE = 0.0`.
- The region where these values are set is defined by a combination of the boxes `DST` and `SRC`.
 - The smaller box, defined by `SRC_LO` and `SRC_HI`, encloses the interior cells.
 - The larger box, defined by `DST_LO` and `DST_HI`, encloses both the interior and ghost cells.
 - The region enclosed by `DST` but not enclosed by `SRC` is set to the desired value.

More specifically, the potential is set to 0 for every cell that has a cell center inside the physically defined region. This may lead to errors on the order of Δx on the location of the application of the boundary condition, but with the boundary conditions defined physically, the solution should converge to the exact solution with grid refinement without manual reconfiguration of the operations. The same approach is used when creating the domains that snap to the nearest approximation of cells based on the physical constraints independent of the underlying mesh resolution. Boundary condition boxes are chosen to be large enough to contain at a minimum the first few layers of cell centers even at the coarsest resolutions run. In regions where the physical boundary conditions overlap, the value will be set repeatedly.

```

DEFINE STAGE FIELD
  DEFINE OPERATION
    TYPE = LogicalBCOp
    FIELD_DATA_NAME = FieldData
    FIELD_NAMES = phi
    SRC_LO = ( 0.0, 0.0, 0.0)
    SRC_HI = ( 1.0, 1.0, 1.0)
    DST_LO = (-0.1,-0.1,-0.1)
    DST_HI = ( 1.1, 1.1, 1.1)
    VALUE = 0.0
  END OPERATION

```

After the boundary conditions have been set, the potential solver is performed. In this example, `LogicalPoissonStripOp` is used to solve Gauss's Law (the integral form of Poisson's equation).

$$\oint_s \mathbf{E} \cdot d\mathbf{n} = \frac{Q}{\epsilon_0}$$

and

$$\mathbf{E} = \nabla \phi$$

where $\mathbf{E}$ is the electric field, ϕ is the electric potential, $\mathbf{n}$ is the normal vector, Q is the charge, and ϵ_0 is the free space permittivity. The above equation is applied to every cell, and the right hand side of the above equation is approximated with finite difference equations. Currently, the set of elliptic solvers in TURF is relatively minimal and includes only red-black Gauss-Seidel and tri-diagonal ADI-type solvers. There is also a degenerate 1D version of the solver that can be used in fundamentally 1D problems or as an accelerated initial guess for solutions that are primarily one-dimensional. `LogicalPoissonStripOp` also has an option to sub-cycle the calculation multiple times before continuing (`SUBCYCLE`). The operation is applied in a red-black checkerboard in the iterative directions so that the solution is independent of the order in which the line relaxation sweeps are performed. Note that this operation requires the charge `Q` instead of the charge density `rho` as it solves the Gauss's equation instead of the Poisson equation.

```

DEFINE OPERATION
  TYPE = LogicalPoissonStripOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = phi
  SOURCE_NAME = Q
  SUBCYCLE = 3
END OPERATION

```

The last part of the `FIELD` stage is to define the criteria to exit the stage. Since the Poisson solution is iterative, we cannot exit the stage until its error is reduced to an acceptable level. The Ops required to complete solver is shown in the code snippet and can be summarized in the following steps:

- Calculate the residual of `phi` through the `LogicalResidualOp`.
- Calculate the L^2 -norm of the residual through the `LogicalNormOp`.
- Use the `CriteriaStageOp` to check if the L2 norm is less than the criteria.

In a multi-domain simulation, each domain applies these operations independently. At the end of the stage, every process collects one vote from every domain as to whether or not to proceed to the next stage or to loop to iterate on the stage. These votes are broadcast across all processes and evaluated by the world when determining whether or not to proceed.

```

DEFINE OPERATION
  TYPE = LogicalResidualOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = phi
  SOURCE_NAME = Q
  FIELD_RESIDUAL_NAME = residual_phi
END OPERATION
DEFINE OPERATION
  TYPE = LogicalNormOp
  FIELD_DATA_NAME = FieldData

```

```

FIELD_NAME = phi
FIELD_RESIDUAL_NAME = residual_phi
NORM = 2.0
NORM_NAME = SUMresidual_phi_L2
VERBOSE = FALSE
END OPERATION
DEFINE OPERATION
  TYPE = CriteriaStageOp
  QUANTITY_NAME = SUMresidual_phi_L2
  CRITERIA = 5.0e-4
END OPERATION
END STAGE FIELD
#####

```

Stage: Move

In this stage, the particle positions are updated using the potential obtained in the previous step. To do this, the node-centered electric field, E_n , is evaluated first using the `LogicalNodeGradientOp` operator. Because the field is the negative gradient of the potential, the `FIELD_MULTIPLY_CONSTANT` of -1.0 is included. The `FIELD_←GRADIENT_DIRECTIONS` are suffixes attached to the root name E_n that the operator uses to construct the three components of the field names needed to store the result.

```

#####DEFINE STAGE MOVE
DEFINE OPERATION
  TYPE = LogicalNodeGradientOp
  FIELD_DATA_NAME = FieldData
  FIELD_POTENTIAL_NAME = phi
  FIELD_GRADIENT_PREFIX = En
  FIELD_MULTIPLY_CONSTANT = -1.0
  FIELD_GRADIENT_DIRECTIONS = [x, y, z]
END OPERATION

```

The next operation uses the electric field to advance the electron and proton positions. The inputs are similar to the basic linear push described in the heatbath tutorials, but with extra field options so that the operator knows which field data to use for the acceleration. This push does not actually test boundary intersections during the push, which is a fast method for simple boundary conditions. After the particle push, the last operation in the stage, `NextStageOp`, tells TURF to proceed to next stage.

```

DEFINE OPERATION
  TYPE = MSPDistESPushOp
  FIELD_DATA_NAME = FieldData
  FIELD_EN_PREFIX = En
  FIELD_EN_DIRECTIONS = [x, y, z]
  MSPDIST_DATA_NAME = P-DST
  SPECIES_NAMES = e- H+@g
END OPERATION
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE MOVE

```

Stage: Plot

The `PLOT` stage consists of three operators and the `NextStageOp`. The first Op `FieldArithmeticOp` calculates the charge density ρ by dividing the charge Q by the volume. The division by volume is triggered by setting `USE←_VOLUME` to true. The second Op writes the 3D field variables in the VTK file format; this is accomplished through the `LogicalFieldWriteVTKOp` Op. The desired field variables are set with the `FIELD_NAMES` keyword and the frequency of the saves is set using the `SKIP` keyword. As an added option, the values of the ghost cells are not saved to the VTK file. The last of the printing ops is `MSPDistParticleCountOp` that saves the number of macro-particles in each domain and the total number in the entire simulation. This information is useful when looking at load balancing and how close the simulation is to steady state (if applicable).

```
#####
DEFINE STAGE PLOT
#####
## Get Charge Density ##
#####
DEFINE OPERATION
  TYPE = FieldArithmeticOp
  FIELD_DATA_NAME = FieldData
  FIELD_DST_NAME = rho
  FIELD_SRC_NAME = Q
  OPERATION = DIV
  USE_VOLUME = TRUE
END OPERATION
#####
## VTK Output of Field Data ##
#####
DEFINE OPERATION
  TYPE = LogicalFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = Plot3D/plt_
  FIELD_NAMES =NP_e- NP_H+ Q phi rho
  SKIP = 5
  PLOT_GHOST = FALSE
END OPERATION
#####
## Output particle Count ##
#####
DEFINE OPERATION
  TYPE = MSPDistParticleCountOp
  MSPDIST_DATA_NAME = P-DST
  LOG_FILE = ParticleCount.csv
  VERBOSE = FALSE
  GLOBAL = TRUE
END OPERATION
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE PLOT
```

Results

The example in this tutorial was originally developed to verify TURF functionality with respect to the ICEPIC code. Using parameters as similar as possible between the two codes, the example was run and visualized in ParaView as shown in the [Comparison Figure](#). The setup was nearly identical to what was outlined above except more particles were used to provide a smoother output. In particular, particle specific weights of 1.25×10^3 and 1.0×10^4 are used for electron and proton, respectively, to ensure a similar number of particles were used in TURF as in ICEPIC.

For the real-time visualization, the low proton numbers make little difference in the electron density visualization but they make charge density plots, like those used to compare the code, noisier.

The agreement between the two codes was very reasonable considering all the particle trajectories are coupled to the field solution and vice versa.

A major difference is the appearance of more charge neutrality on the surface of the ICEPIC result, but this is essentially a difference due to node-centered versus cell-centered output between the two codes. The background in ICEPIC is slightly noisier as well, because the real to computational weights of particles in TURF are modified to ensure the intended cell densities rather than randomly inserting equal weight particles throughout the domain.

On longer timescales after the protons have had the opportunity to move further, the noise level in TURF would appear more similar to that in the ICEPIC result.

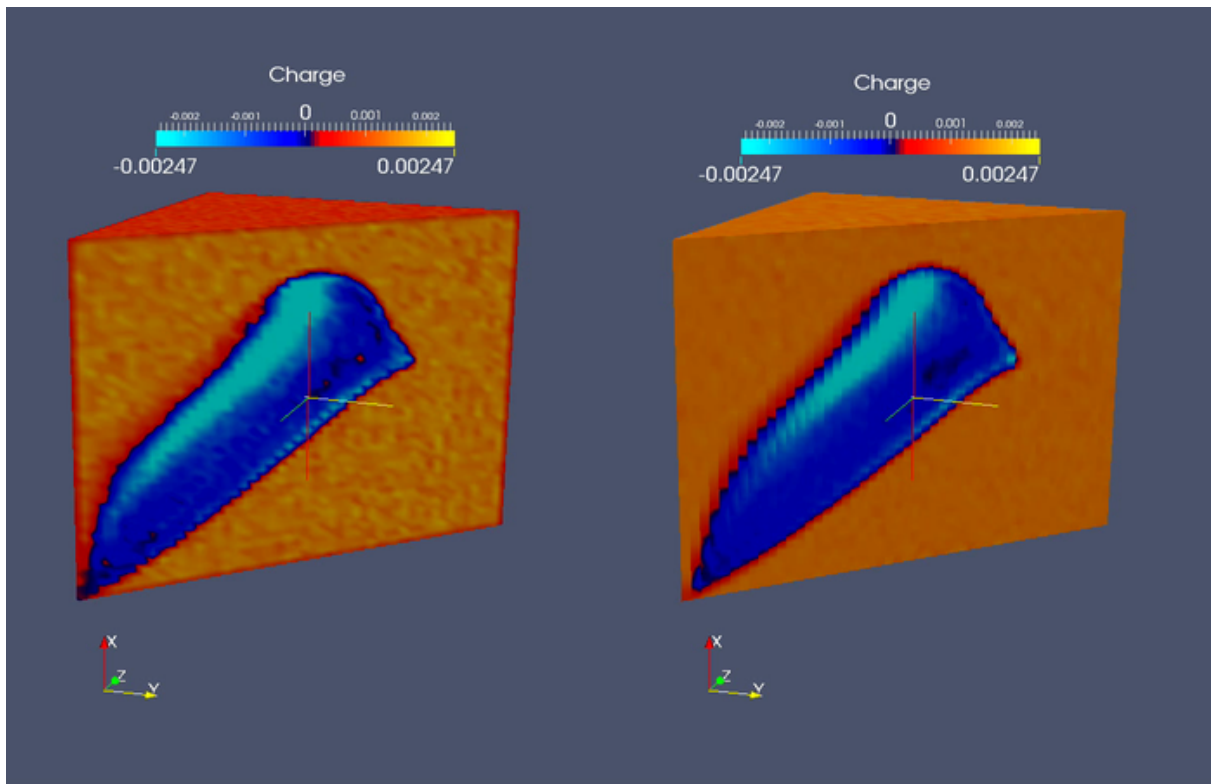


Figure 1.12 Comparison of ICEPIC (left) and TURF (right) grounded box results

1.7 One Dimensional Normal Shock: DSMC with Multi-Species Particle Data

Author

Samuel J. Araki

1.7.1 Included Files

The input files for this tutorial are located in the tutorials/1DShock/DSMC directory.

- DS1V
 - DS1_gfort.f90
 - DS1_original.f90
 - M1.2
 - * ds1vd.dat
 - * DS1VD.TXT
 - * PARAMETERS.DAT
 - M1.4
 - * ds1vd.dat
 - * DS1VD.TXT
 - * PARAMETERS.DAT
 - M2.0
 - * ds1vd.dat
 - * DS1VD.TXT
 - * PARAMETERS.DAT
 - M8.0
 - * ds1vd.dat
 - * DS1VD.TXT
 - * PARAMETERS.DAT
 - makefile
- DSMC_Compare.py
- DSMC_Plot.py
- TURF
 - operations.M12.list
 - operations.M14.list
 - operations.M20.list
 - operations.M80.list
 - world.list

The DS1V directory contains Bird's DS1V program [11] and input files for four different inflow conditions corresponding to upstream Mach number (1.2, 1.4, 2.0, & 8.0). Explanation about the code can be found in the reference and the FORTRAN source code.

The TURF directory contains the input files for TURF. The four operation.list files correspond to the upstream Mach numbers (1.2, 1.4, 2.0, & 8.0). Comment/uncomment the OF_FILE lines in the world.list file to select the desired case.

The DSMC_Compare.py python script compares the results from DS1V and TURF.

The DSMC_Plot.py python script plots the results from TURF.

1.7.2 Introduction

This tutorial demonstrates how to setup TURF to solve a one-dimensional normal shock problem using the DSMC module. Also included in this tutorial is Bird's DS1V code that is used to provide a code-to-code verification with TURF.

The rest of the tutorial is broken up into the following sections. The [Description of the Example Problem](#) details the fundamental equations that govern the 1D normal shock problem. Next, [Setting up the DSMC Example](#) provides an explanation of the `world.list` file as well as a summary of downstream flow parameters for each Mach case, and finally an outline of the Ops used in each stage. The next section, [Operations](#), contains a series of subsections that breaks down the Ops used in simulation by its stage. The final section, [Comparison of Shock Profiles with Bird's DSMC Code](#), compares the results from TURF and DS1V.

1.7.3 Description of the Example Problem

In a fluid, disturbance information is communicated within a medium at the speed of sound, allowing the upstream flow field to adjust accordingly. However, when the flow velocity is greater than the speed of sound, the disturbance information cannot be communicated fast enough, resulting in a formation of a shock. The shock creates a “discontinuity” or a sudden change in flow properties such as velocity, pressure, and temperature. Across a shock, the pressure and temperature always increase while the velocity always decreases from upstream to downstream. The example to simulate with the DSMC module of TURF is the 1D normal shock problem, in which the shock forms in a plane perpendicular to the flow direction. In this problem, the flow properties at upstream and downstream regions with respect to the shock location are related through the following equations:

$$\begin{aligned}\rho_1 u_1 &= \rho_2 u_2 \\ p_1 + \rho_1 u_1^2 &= p_2 + \rho_2 u_2^2 \\ h_1 + \frac{1}{2} u_1^2 &= h_2 + \frac{1}{2} u_2^2\end{aligned}$$

where ρ is the density, u is the velocity, p is the pressure, h is the enthalpy, and subscripts 1 and 2 denote upstream and downstream, respectively. [Conservation Equations](#) are obtained by integrating the Euler equations, a set of conservation equations for mass, momentum, and energy that are applicable for such flows [1]. In a perfect gas, the speed of sound, a , can be determined using the isentropic relation,

$$a^2 = \left(\frac{\partial p}{\partial \rho} \right)_s = \frac{\gamma p}{\rho} = \gamma R_s T$$

where γ is the heat capacity ratio defined as $\gamma = 1 + 2/f$, f is the degree of freedom (i.e. 3 for a monatomic gas and 5 for a diatomic gas), R_s is the specific gas constant (i.e. 208.13 J/kg · K for argon), and T is the temperature. Using the [Conservation Equations](#) and the perfect gas assumption, the downstream flow properties can be determined if the upstream flow properties are known [25]

$$\begin{aligned}M_2^2 &= \frac{1 + \frac{\gamma-1}{2} M_1^2}{\gamma M_1^2 - \frac{\gamma-1}{2}} \\ \frac{n_2}{n_1} &= \frac{\rho_2}{\rho_1} = \frac{u_1}{u_2} = \frac{(\gamma+1)M_1^2}{(\gamma-1)M_1^2 + 2} \\ \frac{T_2}{T_1} &= 1 + \frac{2(\gamma-1)}{(\gamma+1)^2} \frac{\gamma M_1^2 + 1}{M_1^2} (M_1^2 - 1)\end{aligned}$$

where M is the Mach number defined as $M = u/a$ and n is the number density. In setting up the 1D normal shock problem, the downstream flow properties need to be evaluated and input in `operations.Mxx.list` prior to running TURF.

1.7.4 Setting up the DSMC Example

One way to set up the 1D normal shock problem is to introduce uniformly distributed gases upstream and downstream of the shock location. Given the upstream flow properties, appropriate downstream flow properties are determined by the [One Dimensional Shock Equations](#). Table 1.14 provides the downstream flow properties for argon gases with the following upstream conditions

- $T_1 = 293$,
- $n_1 = 1 \times 10^{22} \text{ m}^{-3}$, and
- $a_1 = 318.8 \text{ m/s}$

at Mach numbers, M_1 of 1.2, 1.4, 2.0, and 8.0.

The upstream flow velocities corresponding to the Mach number of 1.2, 1.4, 2.0, and 8.0 are 382.4, 446.2, 637.4, and 2549.6 m/s, respectively. In order to maintain the gas density and the shock location, the gas must also be flowing into the domain from the upstream boundary according to the upstream flow conditions. At the interface between the two gases at different flow properties, the properties are initially discontinuous, and they will develop smooth profiles as time evolves. These profiles can be compared with the profiles obtained by other DSMC models or fluid models. Examples of shock profiles are also provided in Bird Chapter 12, [11].

Table 1.14 Downstream flow properties for upstream Mach number of 1.2, 1.4, 2.0, and 8.0. The values are for argon gas.

Downstream Flow Property	Symbol	Unit	Upstream Mach Number, M1			
			1.2	1.4	2.0	8.0
Velocity	u_2	m/s	294.9	282.3	278.9	667.3
Speed of Sound	a_2	m/s	348.4	376.0	459.4	1456
Mach Number	M_2	-	0.85	0.75	0.61	0.46
Number Density	n_2	$1/\text{m}^3$	1.30×10^{22}	1.58×10^{22}	2.29×10^{22}	3.82×10^{22}
Temperature	T_2	K	350.1	407.8	608.9	6116

The script file `world.list` (shown below) includes global parameters that define the problem such as computational grid size, time-step, species, etc. In TURF, the number of grid cells is determined by finding how many `DELTA`s fit in the user-defined domain bounds (`BOUND_HI-BOUND_LO`), and the domain bounds are redefined such that the coordinates of the lowest corner of the domain box are placed at `ORIGIN`. Therefore, the domain box may not necessarily be bounded by the user-defined `BOUND_LO` and `BOUND_HI`. In this example, the grid contains 1000 cells in x-direction and a single cell for both the y- and z-directions. Here, `BOUND_HI` is set to slightly larger numbers than intended; this ensures that the domain size is not reduced by one cell due to the rounding down of the number of cells. Only argon gas (`Ar@g`) is used in this simulation, and the field parameters to be computed are `NC`, `NP`, `N`, `VX`, and `T` as defined in the 1.15. This example uses three stages (`INITIALIZE`, `MOVE`, and `PLOT`), and the script file `operations.Mxx.list` contains all the operations within each of the three stages, as listed in the 1.16. It should be noted that the field and stage names are not case sensitive.

```

DEFINE WORLD
NAME = DSMC_example
OP_FILE = operations.Mxx.list
COORDINATES = cartesian
ORIGIN = (0.0,0.0,0.0)
DELTA = (2.0e-5,2.0e-3,2.0e-3)
END_TIME = 8.00001e-4
START_DT = 1.0e-8

```

```

FIELDS = [NC NP N VX T]
MATERIALS = [Ar@g]
STAGES = [INITIALIZE, MOVE, PLOT]
PRINT_PROFILE_INFO = true
SAVE_PROFILE_INFO = true
END WORLD
#####
## Domain Geometry
#####
DEFINE DOMAIN DOM000
    BOUND_LO = (0.0,0.0,0.0)
    BOUND_HI = (2.00001e-2,2.00001e-3,2.00001e-3)
END DOMAIN

```

Table 1.15 Volume mesh fields^a computed in this DSMC example.

Field Name	Description
NC	Number of simulation particles.
NP	Number of physical particles.
N	Density.
V	Velocity. VX, VY, VZ are the x-, y-, and z-components.
T	Temperature. TX, TY, TZ are the x-, y-, and z-components.

^a For multi-species simulations, field data for each species can be obtained by adding species name followed by a underscore. For example, if there are two species A@g and B@g, partial densities for species A@g and B@g and total density are named as N_A@g, N_B@g, and N, respectively.

Table 1.16 List of operations in operations.Mxx.list.

Stage	Operation	Description
INITIALIZE	MSPDistInitOp	Create MSPDist object that contains particle distribution
	MSPDistBoxICOp	Fill particle inside a box
MOVEOP	MSPDistNormalMaxwellianOp	Injection of particles at Maxwellian distribution
	MSPDistMoveOp	Advancement of particles
	MSPDistBCSpecOp	Specular boundary condition
	MSPDistCellIDOp	Find cell ID associated with particle location
	MSPDistSortOp	Sort particles according to cell ID and remove particles outside the simulation domain
	MSPDistDSMCOp	DSMC collision calculation
	MSPDistSampleOp	Sample particles and compute volume mesh field data
POSTOP	LogicalFieldWriteVTKOp	Output to a vtk file for a 3D plot
	LogicalFieldWrite1DOp	Output to a csv file for a line plot
	MSPDistParticleCountOp	Output particle count

1.7.5 Operations

Stage: Initialize

In this stage, MSPDistInitOp creates a MSPDist object containing particle information, and MSPDistBoxICOp uniformly distributes particles within a box placed inside the simulation. The buffer size of the MSPDist object is specified by MAX_NP in MSPDistInitOp. Users must be careful when choosing a value for MAX_NP; if the particle count exceeds the buffer size TURF will crash with an error. In addition, SPECIES_NAMES must be the ones already defined

for MATERIAL in world.list. In this example, two MSPDist particle distributions named as P-DST and P-GST are created. P-GST is a temporary MSPDist object that is used when deleting particles in a later stage.

```

DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-DST
  MAX_NP = 1280000
  SPECIES_NAMES = Ar@g
END OPERATION

```

The operation, MSPDistBoxICOp, then adds particles into P-DST. An example of inputs for MSPDistBoxICOp is shown below. The box to be filled with particles is bounded by BOUND_LO and BOUND_HI, in which the lower and higher bounds in Cartesian coordinate are specified, respectively. In this operation, only one gas species can be added at a time, and for this particular case, the gas species to be added is argon. The real to computational particle weight ratio is defined by REAL_TO_COMPUTATIONAL. Furthermore, TEMPERATURE, NUMBER_DENSITY, and VEL are the temperature of the gas species, density, and the streaming velocity of the particles at Maxwellian distribution, respectively. In order to set up the 1D shock problem properly, the upstream and downstream regions inside the computational domain are filled with particles according to flow properties in regions 1 and 2. The initial density distribution obtained by the DSMC example is shown in the [Initial density figure](#). This example corresponds to the case with $M_1 = 1.2$, where the downstream flow properties are given in the [1.14](#).

```

DEFINE OPERATION
  TYPE = MSPDistBoxICOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = Ar@g
  BOUND_LO = (0.0,0.0,0.0)
  BOUND_HI = (1.0e-2,2.0e-3,2.0e-3)
  TEMPERATURE = 293.0
  NUMBER_DENSITY = 1.0e22
  REAL_TO_COMPUTATIONAL = 9.1892e8
  VEL = (382.447,0.0,0.0)
  INIT_ONLY = TRUE
END OPERATION

```

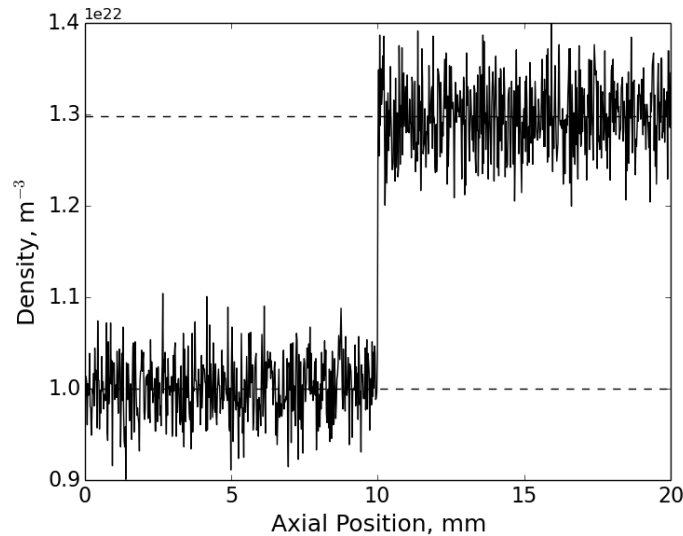


Figure 1.13 'Initial density distribution.'

Stage: Move

The MOVE stage includes several operations including MSPDistNormalMaxwellianOp, MSPDistMoveOp, MSPDistBCSpecOp, MSPDistCellIDOp, MSPDistSortOp, MSPDistDSMCOp, and MSPDistSampleOp.

MSPDistNormalMaxwellianOp injects particles at Maxwellian distribution from a virtual surface. If the surface mesh name (SURFACE_NAME) and component name (SURFACE_GROUP_NAME) are specified in this operation, the particles are injected from the surface mesh elements. Otherwise, the virtual surface is defined by its center (SOURCE_SURFACE_CENTER), normal vector (SOURCE_NORMAL_VECTOR), tangent vector (SOURCE_TANGENT_VECTOR), length (SOURCE_SURFACE_LENGTH), and type (SOURCE_SURFACE_TYPE). For the surface type, only "SQUARE" and "CIRCLE" surfaces are implemented in the current version. Other input parameters are similar to the ones in MSPDistBoxICOp.

```

DEFINE OPERATION
  TYPE = MSPDistNormalMaxwellianOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = Ar@g # Can only inject one species
  TEMPERATURE = 293.0
  NORMAL_VELOCITY = 382.447
  TANGENTIAL_VELOCITY = 0.0
  NUMBER_DENSITY = 1.0E22
  REAL_TO_COMPUTATIONAL = 9.1892e8
  SOURCE_SURFACE_CENTER = ( 0.0, 1.0e-3, 1.0e-3 )
  SOURCE_NORMAL_VECTOR = ( 1.0, 0.0, 0.0 )
  SOURCE_TANGENT_VECTOR = ( 0.0, 1.0, 0.0 )
  SOURCE_SURFACE_LENGTH = 2.0e-3
  SOURCE_SURFACE_TYPE = SQUARE
  VERBOSE = FALSE
END OPERATION

```

MSPDistMoveOp simply advances particles by one time-step. In this operations, only the species defined by SPECIES_NAMES in P_DST is advanced.

```

DEFINE OPERATION
  TYPE = MSPDistMoveOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_NAMES = Ar@g
END OPERATION

```

MSPDistBCSpecOp applies specular boundary conditions to the particles. This operation uses a box defined by BOUND_LO and BOUND_HI to specify the region to apply the boundary condition. The specular reflection is applied to particles that lie within the box, and particle positions are altered based on the box surface corresponding to the direction specified in DIRECTION. The directions that can be specified are xm, xp, ym, yp, zm, and zp, and the m and p in these directions refer to minus (from negative to positive direction) and "plus" (vice versa), respectively. Since the operator is only applied to the particles within the box, it is important to use a box size large enough to capture the fastest particles.

```

DEFINE OPERATION
  TYPE = MSPDistBCSpecOp
  MSPDIST_DATA_NAME = P-DST
  DIRECTION = ym
  BOUND_LO = (-0.1e-2, -1.0e-3, -1.0e-3)
  BOUND_HI = ( 2.1e-2, 0.0e-3, 3.0e-3)
END OPERATION

```

The particles leaving the simulation domain are removed by the combination of two operations, MSPDistCellIDOp and MSPDistSortOp. MSPDistCellIDOp determines the volume mesh cell ID based on where a particle resides. Then, MSPDistSortOp sorts the particle distribution in the order of cell ID. For the particles outside the simulation domain, particle cell ID is set to the maximum value, and these particles are copied to P-GST in MSPDistSortOp and thrown away later. Although removing the particle outside the simulation domain can be done by MSPDistRemovePartOp, MSPDistSortOp is used instead since the additional operation of sorting the particle distribution makes the DSMC algorithm more efficient.

```

DEFINE OPERATION
  TYPE = MSPDistCellIDOp
  MSPDIST_DATA_NAME = P-DST
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistSortOp

```

```

NAME = Sort_P-DST
MSPDIST_SRC_NAME = P-DST
MSPDIST_DST_NAME = P-GST
END OPERATION

```

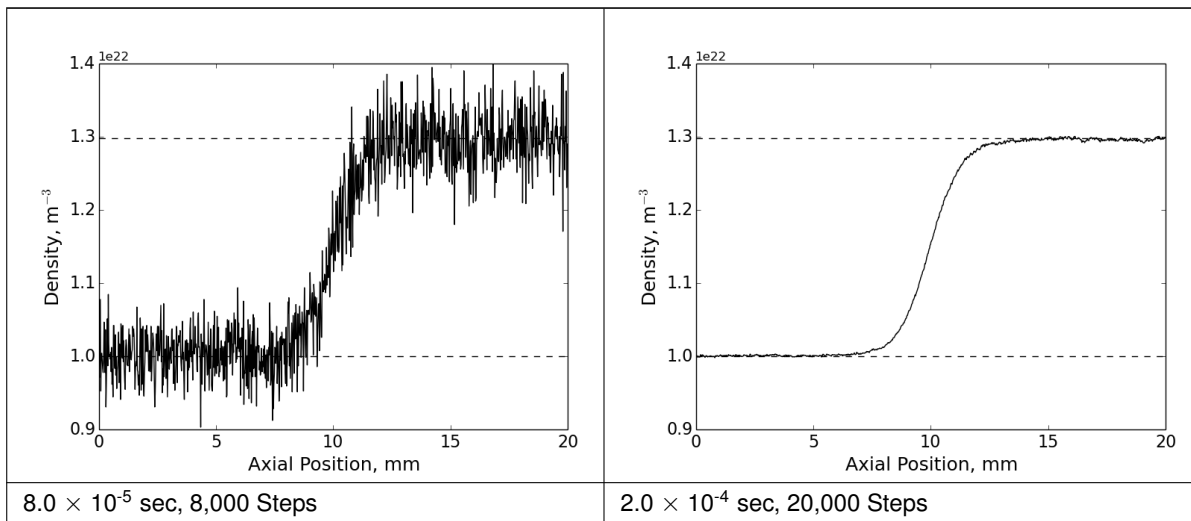
MSPDistDSMCOp finds the number of collisions to perform within all the grid cells and applies the DSMC method. Compared to the Monte Carlo Collision (MCC) method, the DSMC method performs a detailed balance collision in a way that the overall momentum is conserved within the whole system. The variable hard sphere (VHS) molecular model is used to determine the deflection of particles, and the VHS parameters are defined in `materials.list`. The VHS parameters include the reference temperature (T_{ref}), an empirical factor for the variable cross-section (α), and the reference diameter (d), and these parameters can be found in Ref.[11].

```

DEFINE OPERATION
TYPE = MSPDistDSMCOp
MSPDIST_DATA_NAME = P-DST
SPECIES_NAMES = Ar@g
FNUM = 9.1892e8
INIT_TEMPERATURE = 350.074
FREQUENCY_TO_RESAMPLE_MFS = 2000
SORT_OP_NAME = Sort_P-DST
END OPERATION

```

Table 1.17 Density distribution at different time-steps. Mixing is started at 8,000th time-step.



MSPDistSampleOp samples mass, momentum, and kinetic energy from a particle distribution and computes volume mesh field data from these sampled quantities. The quantity to be sampled depends on `FIELD_NAMES` defined in the input. For example, if the temperature field `T` does not need to be determined, the kinetic energy is not sampled from the particle distribution. In this way, the time to perform the sampling calculation can be reduced when less field data is needed. The frequencies of sampling and field calculation are defined by `STEPS_PER_SAMPLING` and `STEPS_PER_FIELD_CALCULATION`. `STEPS_PER_SAMPLING` is typically set to a value greater than one to remove the correlation of the flow field caused by numerics (e.g. particle injection). `STEPS_PER_FIELD_CALCULATION` is typically set to be the same as the output frequency defined by `SKIP` in operations such as `LogicalFieldWriteVTKOp` and `LogicalFieldWrite1DOp`. After the time defined by `MIX_START_TIME`, sampled quantities are mixed between iterations in order to improve the statistics. Therefore, this operation allows a smooth field distribution at the end of simulation without using a very large number of simulation particles. Here, `MIX_START_TIME` should be set to the time when the simulation becomes steady-state. The field distributions at 8,000 and 20,000 time-steps are shown in the [Distribution figure](#). In this example, mixing has been performed after 8,000 time-steps; the distribution is smoothed out significantly after mixing the field parameters for the last 12,000 iterations.

```

DEFINE OPERATION
  TYPE = MSPDistSampleOp
  FIELD_DATA_NAME = FieldData
  MSPDIST_DATA_NAME = P-DST
  FIELD_NAMES = NC NP N VX T
  MIX_START_TIME = 8.0e-5
  STEPS_PER_SAMPLING = 5
  STEPS_PER_FIELD_CALCULATION = 100
END OPERATION

```

Stage: Plot

In this tutorial, two different operations to output field data are used. `LogicalFieldWriteVTKOp` writes field data in vtk format such that the data can be loaded in visualization software like Paraview and VisIt. `LogicalFieldWrite1DOp` writes 1D field data along x-axis in a csv file to simplify the data extraction for post-processing. For both of these operators, the plotting frequency can be defined by `SKIP`. `LogicalFieldWriteVTKOp` has an additional input parameter, `PLOT_GHOST`, providing users with options to plot the field data within ghost layers.

```

DEFINE OPERATION
  TYPE = LogicalFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = Plot3D_M12/plt_
  FIELD_NAMES = NC NP N VX T
  PLOT_GHOST = FALSE
  SKIP = 100
END OPERATION
DEFINE OPERATION
  TYPE = LogicalFieldWrite1DOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = Plot1D_M12/plt_
  FIELD_NAMES = NC NP N VX T
  SKIP = 100
END OPERATION

```

The last operator is unnecessary to run this simulation but is very useful in setting up the simulation. `MSPDistParticleCountOp` may print the number of particles per species and domain to the terminal (`VERBOSE=TRUE`) and/or csv file (`LOG_FILE`). If the particle count reaches nearly constant between iterations, then the simulation has reached steady-state. Otherwise, necessary operations may be missing, or there may be a bug in an operation. If `GLOBAL` is set to true, `TURF` prints out the total number of particles across all the sub-domains for a multi-domain simulation.

```

DEFINE OPERATION
  TYPE = MSPDistParticleCountOp
  MSPDIST_DATA_NAME = P-DST
  QUANTITY_NAME = ParticleCount
  LOG_FILE = ParticleCount/M12.csv
  VERBOSE = FALSE
  GLOBAL = TRUE
END OPERATION

```

1.7.6 Comparison of Shock Profiles with Bird's DSMC Code

The 1D shock profiles from the DSMC module of `TURF` can be compared with other DSMC programs for code verification; in this tutorial, the results are compared with the 1D DSMC code, `DS1V`, developed by G. A. Bird (available at www.gab.com.au). The `DS1V` source code along with input files (`ds1vd.dat`) for the cases with $M_1 = 1.2, 1.4, 2.0$, and 8.0 are also provided in this tutorial. In order to obtain a smooth distribution at the end of simulation, `DS1V` is run twice.

- First, the simulation is started using the “new run” (#3) option in the terminal, the simulation is then stopped at a time greater than 2×10^{-5} sec, and

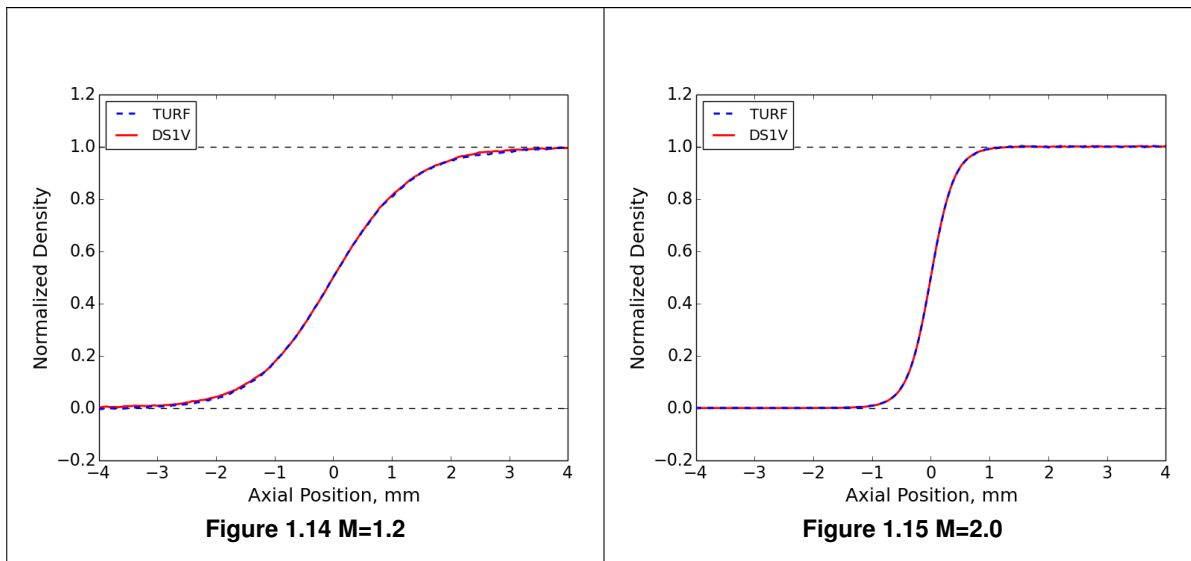
- Second the simulation is restarted using “new sample” (#2) followed by the “adapt the cells” (#1) option. The resulting profile of density as a function of position can be extracted from the output file, “PROFILE.DAT.”

This figure compares the shock profiles computed with TURF and DS1V for the upstream Mach number of 1.2 and 2.0 at 8.0×10^{-4} sec (80,000 time-steps in TURF). The values are normalized according to,

$$\tilde{n} = \frac{n - n_1}{n_2 - n_1}$$

It is readily seen from the [Shock figure](#) that the agreement between the two programs is quite satisfactory.

Table 1.18 Normalized density computed by TURF and DS1V.



1. A number corresponding to a different inflow Mach number is used in place of “xx”.

1.8 Sphere Immersed in a Plasma: Charging with Hybrid PIC/Boltzmann Fluid Model

Author

Samuel J. Araki

1.8.1 Introduction

This tutorial demonstrates a surface charging model coupled with a hybrid Particle-in-Cell (PIC)/fluid model. In the directory `src-TURF/tutorials/SphereInPlasma/HybridPIC`, there are seven files, and these files are briefly described in the following [table](#). More details of these files are given later in this tutorial.

Table 1.19 Description of files in this charging example

File	Description
<code>world.list</code>	TURF script file containing global parameters defining the entire simulation.
<code>operations.cond.list</code>	TURF script file containing individual operations and inputs.
<code>sphere_1.inp</code>	Surface mesh file.
<code>component_TURF.txt</code>	Parameters associated with surface mesh components.
<code>surf_int.txt</code>	Parameters for particle-surface interaction calculations.
<code>materials.list</code>	Parameters associated with material properties.
<code>PlotResults.py</code>	Paraview python script for post-processing the output data.

In this problem, a sphere is immersed in a plasma, and the floating potential on the sphere is to be determined. [Figure](#) below shows the simulation domain and the surface used in this example. Xenon ions are placed uniformly in the domain with temperature of 0.1 eV, and specular and diffuse reflections are applied at the domain boundary and the sphere, respectively. Electron temperature is set to 1 eV. As ions are moved within the simulation domain, ion particles impact the sphere, contributing to the positive current to the sphere. In contrast, negative current contributed by electrons is approximated by integrating the Maxwellian distribution for the appropriate energy range. Integrating the currents from ions and electrons, surface potential of the sphere can be determined. As the simulation progresses, the magnitude of ion and electron currents approach the same value, and the floating potential is obtained when the total current reaches

zero. Details of the surface charging model and more extensive analysis on this example is provided in Refs.[3] [2] .

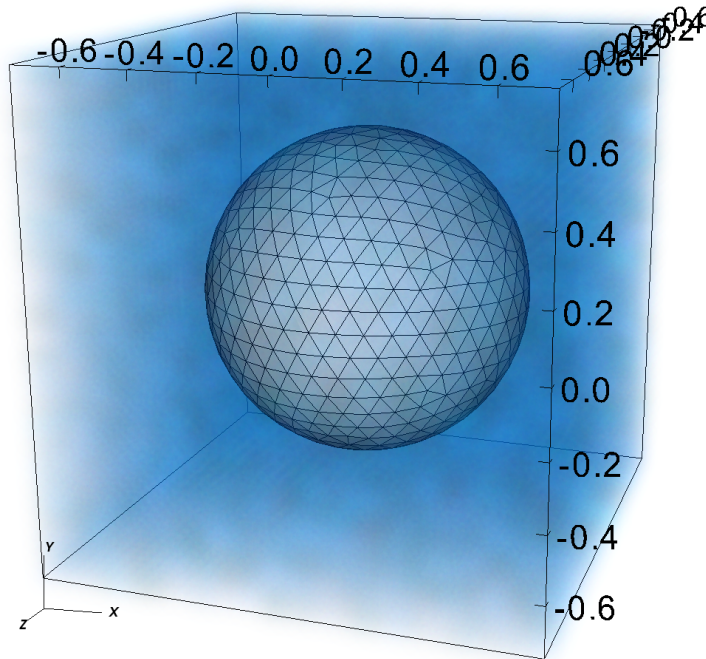


Figure 1.16 Simulation domain and surface mesh.

1.8.2 Defining the World

Parameters used in `world.list` file are defined in [the section followed by this link](#). The simulation domain is a cube with a side length of 1.5 meters, and the domain center is located at (0,0,0). One grid cell is 0.05 meters in each direction. The global simulation time-step used to advance particles is set to 5×10^{-8} s, and 5,000 time-steps are taken before the end of simulation. Six volume grid fields are defined in this simulation, and the description of the fields are given in the [table](#) below. Three materials (Xe+@g, e-, and Test_Solid) are listed in `world.list`, and these materials are defined in the material file, `materials.list`. This example is broken into four stages: INITIALIZE, FIELD, MOVE, and POSTMOVE.

```

DEFINE WORLD
  NAME = sphere_charging_example
  OP_FILE = operations.cond.list
  COORDINATES = cartesian
  ORIGIN = ( 0.0, 0.0, 0.0 )
  DELTA = ( 0.05, 0.05, 0.05 )
  END_TIME = 25.00001e-5
  START_DT = 5.0e-8
  FIELDS = [ VOL, n, n_Xe+, n_Fe, phi, TeeV, rho, rho_avg ]
  MATERIALS = [Xe+@g e- Test_Solid]
  MATERIAL_LIST = materials.list
  STAGES = [INITIALIZE, FIELD, MOVE, POSTMOVE]
  PRINT_PROFILE_INFO = TRUE
  SAVE_PROFILE_INFO = TRUE
END WORLD
#####
## Domain Geometry
#####
DEFINE DOMAIN DOM
  BOUND_LO = ( -0.75, -0.75, -0.75)
  BOUND_HI = ( 0.75, 0.75, 0.75)
  SUB_DOMAINS = (1,1,1)

```

END DOMAIN

Table 1.20 Field Variables: Volume mesh fields^{a,b} computed in this surface charging example

Field Name	Description
VOL	Volume of cells external to a geometry.
n_Xe+	Xenon ion density.
phi	Electric potential.
rho	Instantaneous charge density.
rho_avg	Time-averaged charge density.
TeeV	Temperature in electron volt.
^a FIELD_NAMES are not case sensitive.	
^b All fields are cell-centered otherwise noted.	

1.8.3 Additional Files

Parameters associated with the surface mesh components are defined in the component file (`component_TURF.txt`). In the surface mesh file (`sphere_1.inp`), names of components are defined, and these names are used in the component file to set parameters. In this example, only one component (EB1) is defined. The component EB1 is made out of `Test_Solid` whose material properties are defined in `materials.list`. The material properties defined herein are not really used in this example since the conductive model is used for surface charging, which is defined in the component file; however, `conductivity` and `rel_permittivity` are required when the charge propagation model is used. The component file also defines the geometry's temperature, potential, and thickness. Temperature is required when performing the surface interaction calculation. The surface interactions are defined in `surf_int.txt`, and the interaction is set to diffuse reflection for this example. The potential defined in `component_TURF.txt` is only the initial condition, and the surface potential is to be evolved as the simulation progresses. All of these parameters are input as JSON-like format, and TURF's operators can pick up the required information by identifying the keys such as `component`, `surface_impact`, and `material`. Therefore, the simulation will run even if the three files are combined into one file.

`component_TURF.txt`

```
component{name:EB1,material:Test_Solid,temperature:1160.8,potential:0.0,chargemodel:CONDUCTIVE,
thickness:0.01}
```

`surf_int.txt`

```
surface_impact{source:"Xe+@g",target:"Test_Solid",product:"Xe+@g",spwt_ratio:1.0,c_stick:0.0,c_rest:1.0,
c_accom:1.0,c_diff:1.0}
```

`materials.list`

```
material{name:Null,composition:*1.0@g}
material{name:Electron,composition:e-}
material{name:Test_Solid,composition:*63.546@s,atomic_number:29,conductivity:1.0E-1,rel_permittivity:1.0}
material{name:Xenon_Ion,composition:Xe+@g,dof:3}
```

1.8.4 TURF Operations

Stage: INITIALIZE

The list of operations in the INITIALIZE stage is provided in the [table](#) below.

Table 1.21 Summary of operations in stage INITIALIZE.

Operation	Description
UMeshImporterOp	Loads unstructured surface mesh (UMesh)
SurfaceComponentSetOp	Creates a database for geometry components
LogicalMeshSurfaceSugarcubeOp	Performs sugarcubing to determine the relationship between UMesh and a structured volume mesh (SMesh)
UFieldMSInitOp	Creates a UMesh field data object
UFieldICByComponentOp	Set volume mesh field based on surface field
MSPDistInitOp	Creates a MSPDist data object
MSPDistBoxICOp	Uniformly distribute particles in a box
NextStageOp	Goes to next stage

A sphere is first loaded into TURF by UMeshImporterOp. In the surface mesh file (Sphere_1.inp), a sphere of one meter in radius containing 1,204 triangular elements is defined. This sphere is shrunk by half the radius (set by SCALE=0.5) when loaded such that the sphere's radius is now 0.5 meters. The component parameters are loaded by SurfaceComponentSetOp. The operator LogicalMeshSurfaceSugarcubeOp finds the relationship between the surface and volume meshes. In particular, it figures out which volume cell is inside or outside of the sphere, which aids in efficiently determining whether particles impact surface elements or not. The operator UFieldMSInitOp allocates a container for surface fields. Fields defined by FIELD_NAMES are created for each species and total. On the other hand, fields defined by FIELD_NO_DOF_NAMES are not necessarily associated with species. Therefore, #FIELD_NAMES*#DOF1_NAMES+#FIELD_NO_DOF_NAMES number of fields are created here. The definition of the surface fields are provided in UField Table.

```

DEFINE STAGE INITIALIZE
#####
## Init Spacecraft Geometry ##
#####
DEFINE OPERATION
  TYPE = UMeshImporterOp
  UMESH_FILES = sphere_1.inp
  SCALE = 0.5
  TRANSFORM_CENTER = 0.0, 0.0, 0.0
  TRANSLATE = 0.0, 0.0, 0.0
  ROTATION_AXIS = 1.0, 0.0, 0.0
  ROTATION_DEGREES = 180.0
  UMESH_TYPE = Surface
  UMESH_FILE_TYPE = Abaqus
  UMESH_NAME = UMesh
  WRITE_VTK_MESH = FALSE
END OPERATION
DEFINE OPERATION
  TYPE = SurfaceComponentSetOp
  SURFACE_NAME = UMesh
  COMPONENT_MATERIAL_FILE = component_TURF.txt
END OPERATION
DEFINE OPERATION
  TYPE = LogicalMeshSurfaceSugarcubeOp
  SURFACE_NAME = UMesh
  SUGARCUBE_NAME = SC_UMesh
  SMESH_NAME = SMesh
  WRITE_VTK_MESH = TRUE
END OPERATION
#####
## Initialize UMesh Field ##
#####
DEFINE OPERATION
  TYPE = UFieldMSInitOp
  UFIELD_DATA_NAME = SurfaceField
  UMESH_NAME = UMesh
  FIELD_NAMES = NC NCIN NCOU NPFLUX NPFLUX-IN NPFLUX-OUT
  DOF1_NAMES = Xe+@g SUM
  DOF_TYPES = SPECIES
  FIELD_NO_DOF_NAMES = J_Ion J_IonAvg J_e- J_Total potIn Q
  VERBOSE = TRUE

```

END OPERATION

Table 1.22 Surface fields.

Variable Name	Description
NC	Number of simulation particle impacting a surface element, particle.
NPFLUX	Particle flux to a surface (particle/s · m ²)
J_Ion	Ion current density (A/m ²)
J_IonAvg	Time-averaged ion current density (A/m ²)
J_e-	Electron current density (A/m ²)
J_Total	Total current (A)
potIn	Surface potential (V)
Q	Surface charge (C)

Once the geometry is loaded, the next step is to create particle distributions. Three different containers for particles are created in this simulation. P-DST is the main particle distribution, and P-GST is the ghost particle distribution in which particles impacting surfaces are temporarily stored. P-DEL contains the particles to be deleted. After the particle distributions are created, MSPDistBoxICOp creates Maxwellian particles distributed uniformly within the box defined as an input. This operator is only applied once before the main iteration as INIT_ONLY is set to TRUE. Finally, NextStageOp tells TURF to advance to the next stage.

```
#####
## Particle Distributions and Ghost/Exchange Distributions ##
#####
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-DST
  MAX_NP = 1024000
  SPECIES_NAMES = Xe+@g
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-GST
  MAX_NP = 1024000
  SPECIES_NAMES = Xe+@g
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-DEL
  MAX_NP = 1024000
  SPECIES_NAMES = Xe+@g
END OPERATION
```

```
#####
## Make Particles ##
#####
DEFINE OPERATION
  TYPE = MSPDistBoxICOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = Xe+@g
  BOUND_LO = ( -0.75, -0.75, -0.75)
  BOUND_HI = ( 0.75, 0.75, 0.75)
  TEMPERATURE = 1163.0
  NUMBER_DENSITY = 1.0e16
  REAL_TO_COMPUTATIONAL = 5.0e11
  VEL = (0.0,0.0,0.0)
  INIT_ONLY = TRUE
  VERBOSE = TRUE
END OPERATION
```

```
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE INITIALIZE
```

Stage: FIELD

Table 1.23 Summary of operations in stage FIELD.

Operation	Description
MSPDistCellIDOp	Determines SMesh cell IDs based on particle positions
MSPDistMarkDelGeometryOp	Deletes particles inside the geometry
MSPDistRemovePartOp	Removes particles that stuck to a surface or reached the final time within the stage
FieldSetOp	Initialize charge density to zero
MSPDistChargeDepositionOp	Deposits particle charges to a grid
FieldArithmeticOp	Divides by cell volumes to get charge densities
FieldBlendTimeOp	Time-averages charge density
MSPDistCopyOp	Copies data in a MSPDist to other MSPDist
NextStageOp	Goes to next stage

In the stage FIELD, the first set of operations remove particles from the particle distribution. The operator `MSPDistCellIDOp` first figures out cells that particles reside in. This information is used by the next two operators to remove particles. The operator `MSPDistMarkDelGeometryOp` marks particles that are inside the sphere. `MSPDistBoxICOp` can only distribute particles in a box, so it creates a number of particles inside the sphere, which is taken care of by `MSPDistMarkDelGeometryOp`. The actual deletion of particles is performed by `MSPDistRemovePartOp`. This operator also deletes particles that escape from the simulation domain, which may occur if the global simulation time-step is such that particles move several cells in one time step.

```
DEFINE STAGE FIELD
#####
## Delete Particles ##
#####
# Update CellID
DEFINE OPERATION
  TYPE = MSPDistCellIDOp
  MSPDIST_DATA_NAME = P-DST
END OPERATION
# Mark particles inside geometry
DEFINE OPERATION
  TYPE = MSPDistMarkDelGeometryOp
  MSPDIST_DATA_NAME = P-DST
  SURFACE_NAME = UMesh
  SUGARCUBE_NAME = SC_UMesh
  FIRST_APPLY_ONLY = TRUE
  INIT_ONLY = FALSE
END OPERATION
# Remove particles with CellID>max
DEFINE OPERATION
  TYPE = MSPDistRemovePartOp
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-GST
  REMOVE_ZERO_WEIGHT = FALSE
  REMOVE_CELLID_GT_MAX = TRUE
  DISCARD_REMOVED_PARTICLE = TRUE
  VERBOSE = FALSE
END OPERATION
```

The next set of operators deal with the volume grid fields. The operator `FieldSetOp` first initializes `rho` (charge density) to zero. Then, the ion particles are deposited onto the cell-centers of the volume grid to determine charge, and finally the ion charge is converted to charge density by dividing the charge by the grid volume in `FieldArithmeticOp`. Since ions are solved by the particle method, the charge density distribution contains a large statistical noise. The noise is reduced by time-averaging the charge density using `FieldBlendTimeOp`, and the time-averaged charge density is stored in `rho_avg`. The other volume grid fields such as `TeeV` and `phi` are set to constant values in this example by `FieldSetOp`.

```
#####
## Set Volume Grid Fields ##
#####
# Initialize charge density
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  VALUE = 0.0
  OPERATION = SET
END OPERATION
# Deposit charge
DEFINE OPERATION
  TYPE = MSPDistChargeDepositionOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  MSPDIST_DATA_NAME = P-DST
  ORDER = 1
END OPERATION
# Divide by cell volume to get charge density
DEFINE OPERATION
  TYPE = FieldArithmeticOp
  FIELD_DATA_NAME = FieldData
  FIELD_SRC_NAME = rho
  FIELD_DST_NAME = rho
  OPERATION = DIV
  USE_VOLUME = TRUE
END OPERATION
# Time average rho after TIME_START_BLEND
DEFINE OPERATION
  TYPE = FieldBlendTimeOp
  FIELD_DATA_NAME = FieldData
  FIELD_SRC_NAME = rho
  FIELD_DST_NAME = rho_avg
  OPERATION = AVERAGE
  TIME_START_BLEND = 0.0
END OPERATION
# Set electron temperature
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = TeeV
  VALUE = 1.0
  OPERATION = SET
END OPERATION
# Set plasma potential
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = phi
  VALUE = 0.0
  OPERATION = SET
END OPERATION
```

The last operator in stage `FIELD`, `MSPDistCopyOp`, moves particle data from `P-DST` to `P-GST`, which is necessary in properly performing the surface interaction calculation.

```
#####
## Prep for Advance Particle Positions ##
#####
# Copy the distribution from P-DST to P-GST (P-DST is initialized)
DEFINE OPERATION
  TYPE = MSPDistCopyOp
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-GST
END OPERATION
```

```
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE FIELD
```

Stage: MOVE

Table 1.24 Summary of operations in stage MOVE.

Operation	Description
MSPDistMoveOp	Advances particles
MSPDistSugarcubeSurfIntersectionOp	Determines if particles intersect with surface mesh during a time-step
MSPDistThinSheathOp	Applies thin sheath condition to particles impacting the geometry
MSPDistSurfaceInteractionOp	Performs surface reflections
MSPDistRemovePartOp	Removes particles that stuck to a surface or reached the final time within the stage
MSPDistParticleCountOp	Particle count
CriteriaStageOp	Repeats this stage until particle count becomes zero

In this example, plasma potential is set to 0 V throughout the simulation. Therefore, there is no electric field that affects ion particle motion. For this reason, the simplest particle pusher, MSPDistMoveOp, is used to advance ion particles. For each time-step, particles close to the sphere geometry are tested by an efficient line-triangle intersection algorithm in MSPDistSugarcubeSurfIntersectionOp. Ion particles intersecting the surface elements are marked, and trajectories of these particles are altered by the next two operators, MSPDistThinSheathOp and MSPDistSurfaceInteractionOp. The operator MSPDistThinSheathOp reflects ion particles if ions do not have enough energy to overcome the potential barrier through a sheath. Otherwise, normal velocities of ions are changed based on the potential gain/drop through the sheath. Then, MSPDistSurfaceInteractionOp collects ions impacting the sphere and diffusely reflects them. In reality, ions recombine with electrons at the wall so the number of ions decreases. However, for this simulation, it is necessary to maintain the number of ions within the simulation domain, and this operation effectively reinjects ions from the sphere. The same result is expected if ions are killed at the sphere, and the same number of ions are reinjected from the domain boundary.

```
DEFINE STAGE MOVE
#####
## Keep Advancing Particles Until Reaching the Final Time Per Step ##
#####
# Advance P-GST for a full time-step
DEFINE OPERATION
  TYPE = MSPDistMoveOp
  MSPDIST_DATA_NAME = P-GST
  SPECIES_NAMES = Xe+@g
  USE_CFL_TIME_STEP = TRUE
  CFL_NUMBER = 0.8
END OPERATION
# Find intersection of particle trajectories with UMesh
# *** Starting from P-GST, particles are copied to P-DST once reaching the final time
# *** Then, P-GST contains particles bouncing at surface during the time-step
# *** And, P-DST contains particles making it to the final time per iteration
DEFINE OPERATION
  TYPE = MSPDistSugarcubeSurfIntersectionOp
  NAME = Intersection_P-GST
  MSPDIST_SRC_NAME = P-GST
  MSPDIST_DST_NAME = P-DST
  MSPDIST_ERROR_NAME = P-EXC
  SUGARCUBE_NAME = SC_UMesh
```

```

    SURFACE_NAME = UMesh
#    SURFACE_TEST = TRUE
    VERBOSE = FALSE
END OPERATION
# Reflection by thin sheath
DEFINE OPERATION
    TYPE = MSPDistThinSheathOp
    MSPDIST_DATA_NAME = P-GST
    FIELD_DATA_NAME = FieldData
    FIELD_POTENTIAL_NAME = phi
    UFIELD_DATA_NAME = SurfaceField
    UFIELD_POTENTIAL_NAME = potIn
    SURFACE_NAME = UMesh
    SURF_INTERSECTION_OP_NAME = Intersection_P-GST
    VERBOSE = FALSE
END OPERATION
# Surface interaction - Reflection, sticking, and sputtering
DEFINE OPERATION
    TYPE = MSPDistSurfaceInteractionOp
    NAME = Interaction_P-GST
    SURFACE_NAME = UMesh
    MSPDIST_DATA_NAME = P-GST
    UFIELD_DATA_NAME = SurfaceField
    SURF_INTERSECTION_OP_NAME = Intersection_P-GST
    FILE_NAME = surf_int.txt
    STEPS_PER_SAMPLING = 1
END OPERATION

```

In this stage, the particle distribution P-GST holds ions that have not reached the final time within this iteration due to surface reflections. Ion particles not intersecting any surface element are copied to P-DST and marked for deletion in P-GST by setting the particle weight to be zero. The operator MSPDistRemovePartOp removes these particles in P-GST. This stage is repeated until particles in P-GST are all copied to P-DST. The number of particles is obtained by MSPDistParticleCountOp, and whether to advance to the next stage or repeat the same stage is determined by CriteriaStageOp based on the number of particles remaining in P-GST.

```

# Remove particles with zero weight
DEFINE OPERATION
    TYPE = MSPDistRemovePartOp
    MSPDIST_SRC_NAME = P-GST
    MSPDIST_DST_NAME = P-DEL
    REMOVE_ZERO_WEIGHT = TRUE
    REMOVE_CELLID_GT_MAX = FALSE
    DISCARD_REMOVED_PARTICLE = TRUE
END OPERATION
#####
## Quantity Needs To Be Smaller Than CRITERIA_LONG ##
#####
# Obtain particle count in P-GST
DEFINE OPERATION
    TYPE = MSPDistParticleCountOp
    MSPDIST_DATA_NAME = P-GST
    QUANTITY_NAME = MSPDistCount
    VERBOSE = FALSE
    GLOBAL = FALSE
END OPERATION
#####
## Proceed to the Next Stage If Criterion is Satisfied ##
#####
# Go to next stage if P-GST->Nparts is smaller than 1
DEFINE OPERATION
    TYPE = CriteriaStageOp
    QUANTITY_NAME = MSPDistCount
    CRITERIA_LONG = 1 #Go to next stage if P-GST->Nparts is smaller than 1
    MAX_ITERATION = 100 #Max number of subcycles to prevent stuck particles
END OPERATION
END STAGE MOVE

```

Stage: POSTMOVE

Table 1.25 Summary of operations in stage POSTMOVE.

Operation	Description
MSPDistBCSpecularOp	Applies specular reflection at domain boundary
MSPDistSampleOp	Samples ion density
UFieldMSComputeOp	Samples surface fields
FieldArithmeticConstOp	Multiplies ion flux by elementary charge
FieldBlendTimeOp	Time-averages ion current density
UFieldSurfaceMaxwellianCurrentOp	Calculates electron current density
FieldArithmeticOp	Sum two current densities to get total. Also multiply by the area to get total current
ComponentChargeImplicitOp	Surface charging model
LogicalFieldWriteVTKOp	Writes 3D data in VTK format
UFieldWriteVTKOp	Writes 2D surface data in VTK format
NextStageOp	Goes to next stage

After boundary operations on particles at the sphere geometry are performed, particles are checked against the domain boundary, and the ones outside the domain are specularly reflected to ensure the particle count within the simulation domain is maintained. This is done by `MSPDistBCSpecularOp`, in which particles outside the box defined by `BOUND_LO` and `BOUND_HI` are specularly moved into the simulation domain. The box defined here has to be slightly smaller than the simulation domain size defined in `world.list` file to account for the rare case where particles land exactly on the edge of the simulation domain.

```

DEFINE STAGE POSTMOVE
#####
## Perform Particle Patch ##
#####
DEFINE OPERATION
  TYPE = MSPDistBCSpecularOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_NAMES = Xe+@g
  BOUND_LO = ( -0.749, -0.749, -0.749)
  BOUND_HI = ( 0.749, 0.749, 0.749)
END OPERATION

```

The next step involves calculating the volume and surface fields by `MSPDistSampleOp` and `UFieldMSComputeOp`, respectively. The sampling of volume fields is unnecessary but is performed for visualization purposes. The operator `UFieldMSComputeOp` calculates the surface fields defined by `FIELD_NAMES` in `UFieldMSInitOp`, which is sampled earlier by `MSPDistSurfaceInteractionOp`.

```

#####
## Sum to Fields for Output ##
#####
# Volume grid field
DEFINE OPERATION
  TYPE = MSPDistSampleOp
  FIELD_DATA_NAME = FieldData
  MSPDIST_DATA_NAME = P-DST
  FIELD_NAMES = VOL_n_Xe+
  SUGARCUBE_NAME = SC_UMesh
  MIX_START_TIME = 0.0
  STEPS_PER_SAMPLING = 1
  STEPS_PER_FIELD_CALCULATION = 100
END OPERATION
# Compute UField quantities
# *** This needs to be after surface interaction
DEFINE OPERATION
  TYPE = UFieldMSComputeOp
  UFIELD_DATA_NAME = SurfaceField
  SURF_INTERACTION_OP_NAME = Interaction_P-GST
  MIX_START_TIME = 4.0e100
  STEPS_PER_FIELD_CALCULATION = 1

```

```
END OPERATION
```

Once the ion flux is determined by `UFieldMSComputeOp`, the ion current density onto the surface elements are computed by simply multiplying the flux by the elementary charge using the operator `FieldArithmeticConstOp`. Then, the ion current density is time-averaged by `FieldBlendTimeOp` in order to reduce statistical noise.

```
#####
## Surface Charge ##
#####
# Convert flux to current density
DEFINE OPERATION
  TYPE = FieldArithmeticConstOp
  FIELD_DATA_NAME = SurfaceField
  FIELD_SRC_NAME = NPFLUX-IN_Xe+@g
  FIELD_DST_NAME = J_Ion
  CONSTANT_NAME = MKS_e
  OPERATION = MUL
END OPERATION
# Time-average ion current density
DEFINE OPERATION
  TYPE = FieldBlendTimeOp
  FIELD_DATA_NAME = SurfaceField
  FIELD_SRC_NAME = J_Ion
  FIELD_DST_NAME = J_IonAvg
  OPERATION = AVERAGE
  TIME_START_BLEND = 0.0
#   DELAY_RESET_BY_ONE_ITERATION = TRUE
END OPERATION
```

Electron current densities onto surface elements are determined by `UFieldSurfaceMaxwellianCurrentOp`, using the surface potential, plasma potential, plasma density, and electron temperature.

```
# Compute electron current density
DEFINE OPERATION
  TYPE = UFieldSurfaceMaxwellianCurrentOp
  SURFACE_NAME = UMesh
  SUGARCUBE_NAME = SC_UMesh
  UFIELD_DATA_NAME = SurfaceField
  UFIELD_CURRENT_DENSITY_NAME = J_e-
  UFIELD_POTENTIAL_NAME = potIn
  FIELD_DATA_NAME = FieldData
  FIELD_ELECTRON_TEMPERATURE_EV_NAME = TeeV
  FIELD_CHARGE_DENSITY_NAME = rho_avg
  FIELD_POTENTIAL_NAME = phi
END OPERATION
```

Once the ion and electron current densities onto each surface element are determined, the total current density is obtained by `FieldArithmeticOp`, and the current density is converted to current by multiplying the current density value by the surface element area using `FieldArithmeticOp`.

```
# Sum to get total current density
DEFINE OPERATION
  TYPE = FieldArithmeticOp
  FIELD_DATA_NAME = SurfaceField
  FIELD_SRCB_NAME = J_IonAvg
  FIELD_SRCC_NAME = J_e-
  FIELD_DST_NAME = J_Total
  OPERATION = ADD
  USE_VOLUME = FALSE
END OPERATION
# Convert current density to current
DEFINE OPERATION
  TYPE = FieldArithmeticOp
  FIELD_DATA_NAME = SurfaceField
  FIELD_SRC_NAME = J_Total
  FIELD_DST_NAME = J_Total
  OPERATION = MUL
  USE_VOLUME = TRUE
END OPERATION
```

Using the total current onto the surface geometry, the surface potential can be updated by `ComponentChargeImplicitOp`. It should be noted that the time-step size defined by `TOTAL_TIME_PER_APPLY` for surface charging has to be comparable to the electron time-scale or more specifically the plasma oscillation time-scale. In many simulations, the global time-step is set based on the ion time-scale, which is too large to be used in the surface charging model as surface potential overshoots due to the instantaneous response of electrons to the surface potential. `STEPS_PER_ITERATION` is the number of subcycling steps for the charge propagation model, which is not relevant for this example where the conductive model is used. The frequency and start time of the surface charging calculation can be set by `TIME_BETWEEN_APPLY` and `START_TIME`, respectively.

```
# Compute surface potential
DEFINE OPERATION
  TYPE = ComponentChargeImplicitOp
  UFIELD_DATA_NAME = SurfaceField
  UFIELD_POTENTIAL_NAME = potIn
  UFIELD_CURRENT_NAME = J_Total
  UFIELD_CHARGE_NAME = Q
  BUS_CAPACITANCE = 0.0
  TOTAL_TIME_PER_APPLY = 1.0e-11
  STEPS_PER_ITERATION = 1
#   TIME_BETWEEN_APPLY = 500.0e-6
#   START_TIME = 1000.0e-8
  VERBOSE = TRUE
END OPERATION
```

The last operators are used to output vtk format files for the volume and surface mesh fields.

```
#####
## Write VTK Output - Moved Pre-Push for Consistent Patched Particles ##
#####
# 3D data
DEFINE OPERATION
  TYPE = LogicalFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = plot3D/plt_
  FIELD_NAMES = VOL rho_avg phi n_Xe+ TeeV
  SKIP = 100
  PLOT_GHOST = FALSE
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
# Surface data
DEFINE OPERATION
  TYPE = UFieldWriteVTKOp
  UFIELD_DATA_NAME = SurfaceField
  FILE_HEAD = plotSurf/pltSurf_
  FIELD_NAMES = potIn J_Total J_Ion J_IonAvg J_e- Q
  SKIP = 5
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE POSTMOVE
```

1.8.5 Plotting Floating Potential Result

After the TURF simulation is run, the floating potential and total current on the sphere can be plotted with the python script included in `src-TURF/tutorials/SphereInPlasma/HybridPIC`. The script requires `ParaView` python included when installing `ParaView`, and the command to run the script is

```
pypython PlotResults.py -d directory_containing_output_files
```

The output from the script is shown in the figure below. The red line corresponds to the analytical solution obtained by the equation provided below. It can be seen that as simulation progresses, the computed floating potential solution asymptotes to the analytical value and the total current from plasma approaches zero.

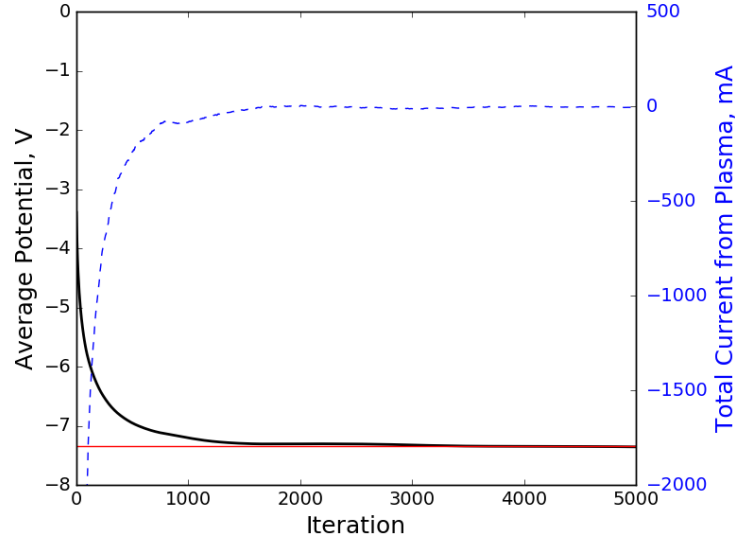


Figure 1.17 Convergence of floating potential as simulation progresses.

1.8.6 Analytical Solution

If the plasma potential is set to 0 V and the ion and electron temperatures are known, the floating potential can be determined analytically by equating the ion and electron current to the sphere. If the surface potential is lower than the plasma potential, only the electrons with sufficient energy can reach the surface, and the rest of electrons are repelled from the surface. The electron current to the surface can be determined by integrating the Maxwellian distribution for the appropriate energy range.

$$J_e = -\frac{en_e v_{e,th} A}{4} \exp\left(-\frac{e(\phi - \phi_w)}{k_B T_e}\right) \quad (1.1)$$

where e is the elementary charge, n_e is the electron density, $v_{e,th}$ is the mean speed of thermal electrons given by $v_{th} = \sqrt{8k_B T_e / \pi m_e}$, k_B is the Boltzmann constant, T_e is the electron temperature, m_e is the electron mass, A is the surface area, ϕ_w is the wall potential, and ϕ is the plasma potential. Electron density is determined by assuming quasi-neutrality such that $n_e = n_i$ where n_i is the ion density obtained directly from ion macro-particles. On the other hand, ions are attracted to the surface, so the ion current is space-charge-limited and simply equals the saturation current [29].

$$J_i = \frac{en_i v_{i,th} A}{4} \quad (1.2)$$

where $v_{i,th}$ is the mean speed of thermal ions given by $v_{i,th} = \sqrt{8k_B T_i / \pi m_i}$, T_i is the ion temperature, and m_i is the ion mass. Given the ion and electron temperatures and setting the plasma potential to be 0 V, the floating potential can be determined analytically by equating the ion and electron current to the sphere.

$$\phi_s = -\frac{K_B T_e}{e} \ln\left(\sqrt{\frac{T_i m_e}{T_e m_i}}\right) \quad (1.3)$$

Equation (3) is only valid when ϕ_s is lower than the plasma potential and the plasma density is sufficiently high such that the thin sheath approximation is appropriate. For this simulation condition, the floating potential is -7.344 V. It should be noted that Eq. (3) is not dependent on a plasma density.

1.9 Adaptive Mesh Refinement

Author

Samuel J. Araki

1.9.1 Introduction

This tutorial demonstrates the Adaptive Mesh Refinement (AMR) capability in TURF, providing rules and tips for defining the mesh refinement criteria, different input options available, and difference in the mesh refinement as a result of the corresponding input option. There are three AMR examples in this tutorial, which are listed in the [table](#) below along with the TURF input files.

Table 1.26 AMR Tutorial Examples

Example	Description	Files
1	Using coordinates and field data	world.demo.list, operations.demo.list
2	Extension to multi-domain simulation	world.demo_patch.list, operations.demo_↵ patch.list
3	Using sugarcube data	world.sugarcube.list, operations.↵ sugarcube.list, sphere_1.inp

1.9.2 TURF AMR Basics

This section covers the operators required to perform mesh refinement, details on the mesh refinement process, and the list of math operations, functions, and constants available for the refinement criteria.

Operator

In order to enable AMR in TURF, the coordinate system has to be set to AMR in the input file defining the world (world.demo.list, world.demo_patch.list, and world.sugarcube.list in the examples herein). Then, the operator `SMeshCreateOp` defined in the operations file will replace the default Cartesian mesh with a new Cartesian mesh compatible with AMR. Note that mesh information is stored using i, j, k indices in the default Cartesian mesh, while the adaptive Cartesian mesh stores data in a one dimensional array. The part of operations.list file defining `SMeshCreateOp` is shown below, and the description of the input keys is provided in the [table](#) below.

```

DEFINE STAGE MESH
#####
## Adaptive Cartesian Mesh ##
#####
DEFINE OPERATION
  TYPE = SMeshCreateOp
  SMESH_NAME = SMesh
  SUGARCUBE_NAME = SC_NAME
  GHOST_CELL = 3
  REFINEMENT_LEVEL = 1
  CRITERION = 1>0
  FILL_RATIO = 1.0
  SKIP = 1
  TIME_START = 0.0
  TIME_STOP = 1.0e100
  VERBOSE = TRUE
END OPERATION
### Default Criteria to Proceed to the Next Stage
DEFINE OPERATION
  TYPE = NextStageOp

```

```
END OPERATION
END STAGE MESH
```

Table 1.27 Input keys for SMeshCreateOp

Input Key	Description	Default
SMESH_NAME	Name of the mesh. ^a	SMesh
SUGARCUBE_NAME ^b	Name of sugarcube object to be updated every time the mesh refinement is performed	None
GHOST_CELL	Number of ghost cell layers	As defined in world.list, otherwise defaulted to 3
REFINEMENT_LEVEL	Finest refinement level	0
CRITERION	Refinement criteria	0>1, which leads to false, thus no refinement is performed
FILL_RATIO	Sets the size of one-dimensional arrays containing the mesh data. ^c	1.0
SKIP	Frequency of mesh refinement	1
TIME_START	Time to start mesh refinement	0.0
TIME_STOP	Time to end mesh refinement	MAX_REAL
VERBOSE	Flag to output timing data	FALSE
BOUND_LO ^d	Lower bound of mesh region	As defined in world.list
BOUND_HI ^d	Upper bound of mesh region	As defined in world.list
DELTA ^d	Cell size of newly added mesh	As defined in world.list
^a Using default name will replace the original Cartesian mesh, otherwise a new mesh will be added (Never been tested with multiple Cartesian meshes).		
^b Only necessary if 'sugarcubing' is to be performed between the adaptive Cartesian mesh and surface mesh.		
^c Size of array is FILL_RATIO*(# of base level cells)*(8 ^{REFINEMENT_LEVEL}). IMPORTANT: Too large or small value for FILL_RATIO leads to unexpected failure of simulation during initialization of operators.		
^d These inputs should not be defined unless multiple Cartesian meshes are to be used, but this capability has never been tested.		

If the simulation involves multiple sub-domains and requires 'sugarcubing' to be performed, SMeshCreateOp has to be placed in a stage before the stage containing LogicalASMeshSurfaceSugarcubeOp. This is because, in TURF, communication between subdomains is only performed between stages, which need to happen after the mesh refinement but before the sugarcubing.

After a new adaptive Cartesian mesh is created by calling SMeshCreateOp, the new field object compatible with AMR has to be created, which is done by LogicalFieldInitOp.

```
#####
## Create field data ##
#####
DEFINE OPERATION
  TYPE = LogicalFieldInitOp
  FIELD_DATA_NAME = FieldData
  SMESH_NAME = SMesh
  FIELD_NO_DOF_NAMES = phi
  VERBOSE = FALSE
END OPERATION
```

How the Mesh Refinement Is Performed in TURF

Mesh refinement in SMeshCreateOp is performed when the remainder after dividing the iteration count by SKIP is zero ($it \% \text{SKIP} == 0$ where it is the iteration count). In TURF, the iteration number is counted from zero, so the mesh refinement is always performed during the zeroth iteration. In addition, the mesh refinement at the first iteration is enforced, as it requires additional iterations for the mesh to be settled even when the field data remains the same. Furthermore, in the TURF AMR routine, the mesh cells can only be refined by one level per iteration except during the zeroth iteration where the mesh can be refined all the way to the finest level.

In the mesh refinement routine, all the cells meeting the refinement criteria are split first. The same field value as the parent is assigned to all the children cells. Then, the combine step checks against the same refinement criteria and the cells are combined when the eight cells with the same parent cell DO NOT satisfy the refinement criteria. Therefore, care must be taken when using the refinement level 'lv' in the refinement criteria. For example, when the refinement criteria is set to be `REFINEMET_CRITERIA=(lv==0)`, all the mesh cells are refined during the split step. However, since the children cells of (lv==1) do not satisfy the refinement criteria, they are all combined immediately. As a result, the mesh remains as if nothing happened. For this reason, additional flags are available for the use in the refinement criteria: 'refine' sets its value to 1 and 0 during split and combine steps, respectively, and 'combine' sets its value in the opposite manner. This allows for the specification of the finest refinement level for each criteria, which will be shown in the next section as well as the AMR examples below.

Mesh Refinement Criteria

All the math operations, functions, and constants that can be used in the mesh refinement criteria are defined in `Math←ExpressionParser.h` and `constant.h` in `src-TURF/src/Math` directory and are also listed in the 3.10. In addition, cell-center coordinates (x, y, z, -x, -y, -z), simulation time (t, -t), time-step (dt, -dt), cell refinement level (lv, -lv), and field data name can be used in the refinement criteria. The cell level can be used together with 'refine' and 'combine' flags to specify the finest refinement level for each criterion. For example, defining the refinement criteria in SMeshCreateOp as

```
CRITERION = (phi>0 && refine*lv<1)
```

will only allow the cells to be refined up to one level finer than the base cell size if the field data 'phi' is greater than zero. It is also important to note that minus sign can be used, but there is no guarantee if the equation is read correctly unless the expression with minus sign is wrapped around (any type of) parentheses.

```
CRITERION = phi>=(-1.0)
```

Finally, the field name containing math operations will not be read correctly unless the name is wrapped around curly parentheses.

```
CRITERION = {n_Xe+@g}>=1.0e15
```

1.9.3 Example 1: Using Coordinates and Field Data in Mesh Refinement Criteria

In this example, two separate criteria are specified: one is based on the field data 'phi' (ϕ) and another one is based on the cell-center coordinates.

$$\phi > 0 \quad (1.4)$$

$$\sqrt{(x+0.8)^2 + y^2 + z^2} < 1.2 \quad (1.5)$$

The finest refinement level for Eq. (2) is set to 1, which is accomplished by adding `refine * lv == 0` along with Eq. (2). The field ϕ is set to 1 by `FieldSetOp` in the box region specified by `BOUND_LO` and `Bound_HI`.

```

DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = phi
  VALUE = 1.0
  OPERATION = SET
  BOUND_LO = (-0.4,-0.4,-1.0)
  BOUND_HI = ( 1.6, 1.6, 1.0)
  VERBOSE = FALSE
END OPERATION

```

Three different cases can be set up by slightly modifying the inputs. Case (1) is run without any modification to `operations.demo.list`, which uses the following mesh refinement criteria.

```

CRITERION = (phi>0.0) || ( sqrt((x+0.8)*(x+0.8)+y*y+z*z)<1.2 && refine*lv==0 )

```

Since the finest refinement level (REFINEMENT_LEVEL) is set to 3 in this example, the mesh cells are refined all the way to the finest if the field data ϕ is greater than 0. Case (2) keeps the same finest refinement level but only allows up to the refinement level of 2 for Eq. (1).

```

CRITERION = (phi>0.0 && refine*lv<2) || ( sqrt((x+0.8)*(x+0.8)+y*y+z*z)<1.2 && refine*lv==0 )

```

Finally, Case (3) uses the same criteria as Case (1) but sets the `RUN_AT_INIT` option to true in `FieldSetOp`.

```

RUN_AT_INIT = TRUE

```

This will set the field data within the specified box to 1 during the operator initialization before the mesh refinement is performed for the first time, so the mesh refinement is performed all the way to the finest level. Then, cells that do not satisfy both Eqs. (1) and (2) are combined during the next iteration. It is interesting to note that the resulting mesh is different from the mesh in Case (1). The equation `refine*lv=0` is always false during the `combine` step, so the cells satisfying Eq. (2) never get combined.

Figures below show xy-planes of the meshes from the three cases at different iterations. At the mesh cells colored in blue, the values of the field ϕ are set to 1 by `FieldSetOp`.

Table 1.28 Case (1)

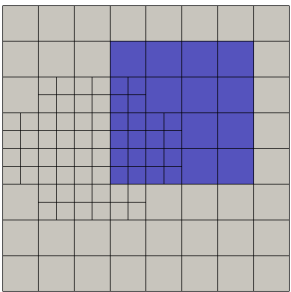
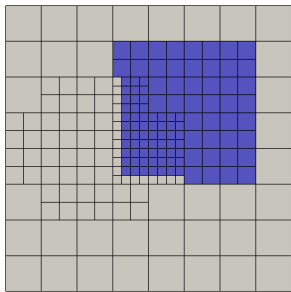
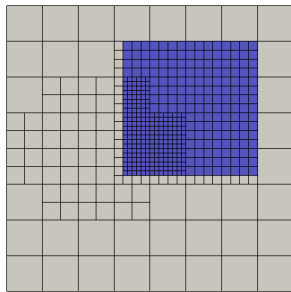
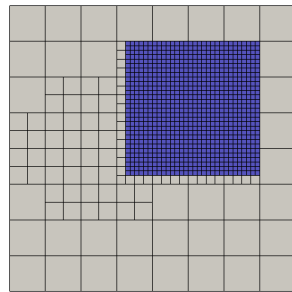
			
Iteration 0	Iteration 1	Iteration 2	Iteration 3

Table 1.29 Case (2)

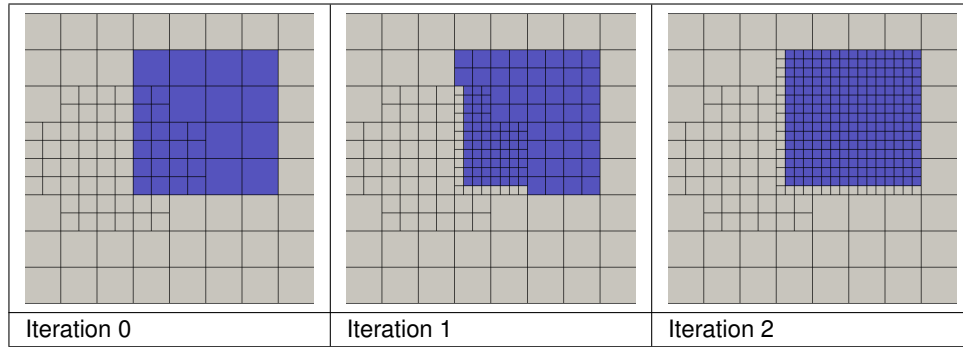
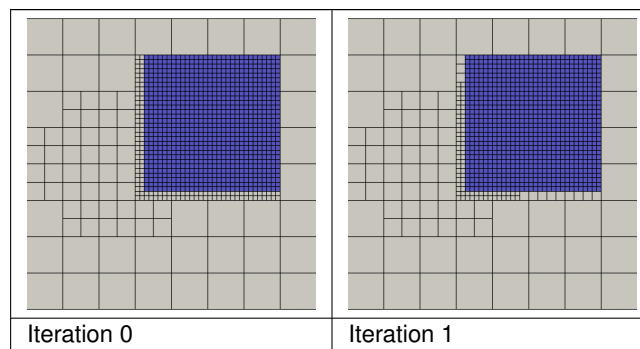


Table 1.30 Case (3)



These figures are generated from the vtk output data written by LogicalAMRFieldWriteVTKOp.

```
#####
## VTK Output For Visualization ##
#####
DEFINE OPERATION
  TYPE = LogicalAMRFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAMES = phi
  FILE_HEAD = Plot1/Field_
  PLOT_GHOST = FALSE
  LEAF_ONLY = TRUE
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
```

Here, setting PLOT_GHOST to TRUE will output the ghost cell data in addition to the interior cell data. Also, setting LEAF_ONLY to TRUE will only output the finest level cells, while setting it to FALSE will output all the parent cells as well, which can be useful when debugging.

1.9.4 Example 2: Extending Example 1 to Multi-Domain

In Example 2, the domain in Case (2) of Example 1 is split up into four subdomains as defined in `world.demo_` ← `patch.list`.

```
#####
## Domain Geometry
#####
DEFINE DOMAIN DOM
  BOUND_LO = (-2.0,-2.0,-2.0)
  BOUND_HI = ( 2.0, 2.0, 2.0)
  SUB_DOMAINS = (2,2,1)
END DOMAIN
```

In TURF, ghost cells are used to exchange data between subdomains. A ghost cell can either overlap with one or more subdomains' ghost or interior regions or be completely outside of the simulation region. Although not all the simulations

require the exchange of field data between subdomains, there are some occasions that this becomes necessary. For example, when solving the Poisson's equation, the field data in the ghost region can be updated to correspond with the values in the interior region, which imposes the Dirichlet boundary condition at the domain interface. Another example would be, when using the first order weighting in a particle simulation, some particles are assigned to the ghost cells, which needs to be added back into the interior cells of different sub-domain.

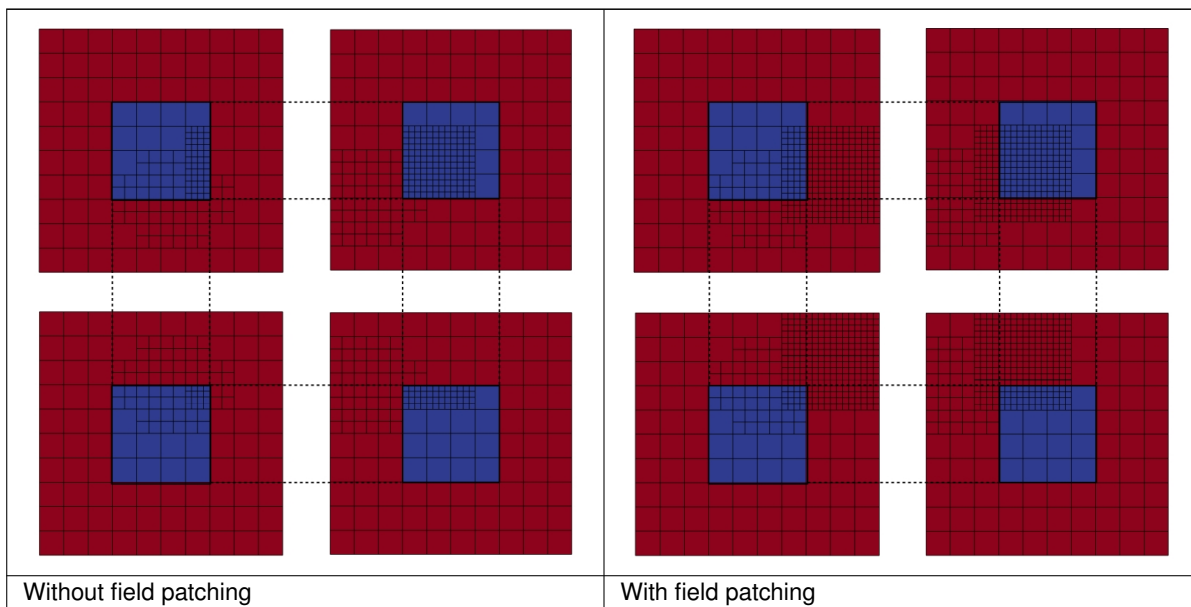
When the field data need to be exchanged between subdomains, the mesh cells across the overlapping regions have to be consistent across all the subdomains, otherwise the communication will not be performed correctly, as there would be a mismatch of cells. In order to prevent this situation, the field data used in the mesh refinement criteria has to be patched across the subdomains such that the field data in the ghost region are consistent with the values set in the interior region. The exchange of field data is performed if the patch flag is set accordingly in `LogicalFieldPatchOp`.

```
#####
## Patch rho across domains ##
#####
DEFINE OPERATION
  TYPE = LogicalFieldPatchOp
  FIELD_NAMES = phi
  PATCH_OPS = PATCH_SET
END OPERATION
```

For the AMR meshes to be consistent across ghost and interior regions of different subdomains, SET option is used, which copies the field value in the interior cells to the corresponding ghost cells. There are four additional options for field patching: MAX, MIN, SUM, and PROD.

In order to illustrate the difference in the mesh results, Example 2 is run with two cases where `LogicalFieldPatchOp` is turned off and on. For the purpose of this demonstration, the field values are only set for the interior cells by setting `INTERIOR_CELLS_ONLY` to be true in `FieldSetOp`. The figures below show the difference in the meshes between the two cases. The interior region is in blue, and the ghost region is in red. The mesh in the interior region is identical between the two cases, while the difference in the two meshes is clearly seen in the ghost region.

Table 1.31 AMR in multi-domain simulation



1.9.5 Example 3: Using Sugarcube Data in Mesh Refinement Criteria

This example uses the sugarcube data to refine the mesh on the cells that touch the surface mesh. Refining the mesh near the geometry is one of the common ways to reduce the error associated with misalignment of geometry with the Cartesian mesh in Computational Fluid Dynamics. The mesh cells are split based on the field data `io`, which contains

positive integers corresponding to the surface number, negative integers corresponding to the interior region, and zero if the cell is in a free-space region.

```
#####
## Adaptive Cartesian Mesh ##
#####
DEFINE OPERATION
  TYPE = SMeshCreateOp
  SMESH_NAME = SMesh
  SUGARCUBE_NAME = SC_surface
  GHOST_CELL = 3
  CRITERION = io>0.0
  REFINEMENT_LEVEL = 2
  FILL_RATIO = 1.0
  SKIP = 1
  VERBOSE = TRUE
END OPERATION
```

The regions of free-space, boundary, and inside a geometry are marked by the sugarcubing routine `LogicalASMeshSurfaceSugarcubeOp` after loading the geometry by `UMeshImporterOp`.

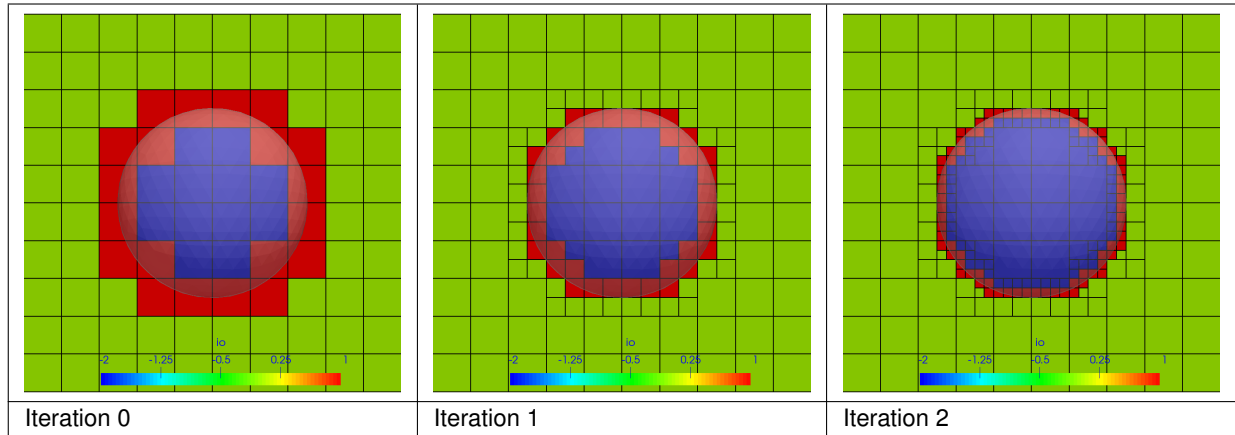
```
#####
## Initialize Geometry ##
#####
DEFINE OPERATION
  TYPE = UMeshImporterOp
  UMESH_FILES = sphere_1.inp
  UMESH_TYPE = Surface
  UMESH_FILE_TYPE = Abaqus
  UMESH_NAME = surface
  WRITE_VTK_MESH = FALSE
END OPERATION
DEFINE OPERATION
  TYPE = LogicalASMeshSurfaceSugarcubeOp
  SURFACE_FILE_TYPE = Abaqus
  SURFACE_NAME = surface
  SMESH_NAME = SMesh
  SUGARCUBE_NAME = SC_surface
  DOMID_FIELD_NAME = domID
  WRITE_VTK_MESH = TRUE
  VERBOSE = TRUE
END OPERATION
```

After the cells are marked with respect to the geometry, the data are extracted to the field data `io` by `FieldCopyKMatrixOp`.

```
#####
## Copy Array Data To Field ##
#####
DEFINE OPERATION
  TYPE = FieldCopyKMatrixOp
  FIELD_DATA_NAME = FieldData
  FIELD_DST_NAME = io
  GSOBJECT_NAME = SC_surface
  KMATRIX_NAME = in_or_out
  INTERIOR_CELLS_ONLY = FALSE
END OPERATION
```

The figures below show the evolution of mesh and the field data `io` as time progresses. The sphere geometry is also shown in the figures. The mesh refinement is an iterative process in that the refinement and sugarcubing routines have to be repeated until arriving at the final mesh.

Table 1.32 Mesh refinement based on sugarcubing data



1.10 EP Plume Simulation 1: Baseline

Author

Samuel J. Araki

1.10.1 Introduction

A series of electric propulsion (EP) plume simulations is provided in the TURF tutorial. Starting from the baseline plume simulation (this page), advanced models are incrementally added in the order of [the DSMC collision method](#), [spacecraft charging](#), and [adaptive mesh refinement and dynamic load balancing](#). There are also tutorials on how to create a geometry file using [Cubit](#) and how to visualize field data and particles using [Paraview](#). In order to help users to adapt these tutorials for their needs, "#<---" is added to the end of lines in the input files throughout this EP plume tutorial, indicating the ones to be changed. This tutorial covers the baseline simulation, which is slightly improved upon the version published in 2017.

The directory `src-TURF/tutorials/Plume/Plume_1.Base` contains 10 files to run the baseline simulation as well as supplemental simulations demonstrating the restart capability and how to extend it to multiple sub-domains and processes. For each simulation, there is a pair of `world.*.list` and `operations.*.list` files as listed in the [table](#) below. In addition, there are four additional input files listed in the [table](#) below, which are used among all the simulations in this directory.

Table 1.33 Baseline EP plume examples

Example	Description	Files
1	Baseline	<code>world.base.list</code> , <code>operations.base.list</code>
2	Restarting the simulation	<code>world.restart.list</code> , <code>operations.restart.list</code>
3	Extension to multi-domain simulation	<code>world.basex4.list</code> , <code>operations.basex4.list</code>

Table 1.34 Additional files to run the plume simulation

File	Description
<code>jaysat2_rotate.inp</code>	Geometry file
<code>RPA_HCT.csv</code>	RPA data for particle source
<code>component_TURF.txt</code>	Inputs associated with geometry components
<code>surf_int.txt</code>	Surface interactions between gas and solid materials

1.10.2 Defining the World

In order to establish the starting parameters and conditions of the simulation, the TURF calls the world file (`world.<list>`). This file contains the name of the simulation, points to an input file that contains the operators, the global parameters of the simulation, etc., and these are assigned to the World object in TURF. Keys used in `world.list` file are defined in [the section followed by this link](#). Note that the inputs can either be lower or upper case.

```

DEFINE WORLD
  NAME = plume_example
  OP_FILE = operations.base.list #<---
  COORDINATES = cartesian
  ORIGIN = ( 0.0, 0.0, 0.0 ) #<---
  DELTA = ( 0.25, 0.25, 0.25 ) #<---
  END_TIME = 4.00001e-3 #<---
  START_DT = 2.0e-6 #<---
  FIELDS = [ pflag, rho, rho_avg, phi, Enx, Eny, Enz ] #<---
  FIELDS = [ VOL, n_Xe@g, n_Xe+@g, n_Al@g ] #<---
  MATERIALS = [ Xe@g Xe+@g Al@g Al@s Cu@s ] #<---
  STAGES = [ INITIALIZE, MOVE, POSTMOVE, RHO, FIELD ] #<---
  PRINT_STATS_SKIP = 1
  PRINT_PROFILE_INFO = TRUE

```

```

    SAVE_PROFILE_INFO = TRUE
    PROFILE_FILE = profile1A.txt #<---
END WORLD
#####
## Domain Geometry
#####
DEFINE DOMAIN DOM
    BOUND_LO = (-3.5, -2.5, -2.5) #<---
    BOUND_HI = ( 5.0,  2.5,  7.5) #<---
    SUB_DOMAINS = (1,1,1) #<---
END DOMAIN

```

Table 1.35 Field Variables: Volume mesh fields^{a,b} computed in this plume example

Field Name	Description
pFlag	Flag to indicate which potential solver to use
rho	Charge density (C/m ³)
rho_avg	Time-averaged charge density (C/m ³)
phi	Electric potential (V)
Enx, Eny, Enz	Components of node-centered electric field in x-, y-, and z-directions (V/m)
VOL	Cell volume (m ³)
n_Xe@g, n_Xe+@g, n_Al@g	Species density (particle/m ³)
^a FIELDS are not case sensitive.	
^b All fields are cell-centered otherwise noted.	

1.10.3 Geometry Components

The parameters associated with each component of the spacecraft geometry can be specified in a component file (named as `component_TURF.txt` in this example). The file contains a list of components with associated parameters in a JavaScript Object Notation (JSON) like format.

```
component{name:SC_BODY, material:Fe@s, potential:0.0, temperature:300.0}
```

The JSON format is chosen to retain the flexibility as TURF capability is expanded. In this format, there are multiple pairs of keys and values separated by colon associated with a component. The keys are used within TURF operations to access the data when needed, and there is no distinction between upper or lower case for the keys. On the other hand, the values of different data types are stored as a TURF string object, and they are converted to integer, float, double, or string data type when they are accessed within TURF. The values as string format have to match exactly when referring to them in other parts of inputs; it distinguishes lower and upper case letters.

In the component file, the current version takes the name of the component (`name`) and associated `material`, `potential`, `temperature`, `chargemodel`, and `thickness`. The `material` and `temperature` are used when performing the surface interaction calculations and `potential` is used as Dirichlet boundary condition for the potential calculations within free-space. The `chargemodel` and `thickness` are used for the spacecraft charging calculations. Options for `chargemodel` are provided in [EP Plume Simulation 3: Spacecraft Charging](#). The file must use the names given in the surface mesh file; the names also correspond to the “blocks” defined in Cubit when generating the surface mesh file. If the names do not match exactly, the component specific inputs will not be read, and the default values established in the Initialization stage will be assigned. Similarly, the specified material must be one listed in the `materials` field in the World file.

1.10.4 Particle-Surface Interaction

The surface interaction file, `surf_int.txt`, informs the simulation what to do when gas particles come into contact with the surface of the spacecraft. Both the surface reflection and sputtering are integrated in the current version of TURF. The incident particles can stick to the surface or reflect off the surface specularly or diffusely. These particles can also cause sputtering of surface material. The surface interactions are defined in `surf_int.txt` in the JSON like format also used for `component_TURF.txt`.

```
surface_impact{source:Xe@g, target:Al@s, product:Xe@g, spwt_ratio:1.0, c_stick:1.0, c_rest:1.0, c_accom:0.9,
c_diff:1.0}
```

```
sputtering{model:yama, source:Xe@g, target:Al@s, product:Al@g, spwt_ratio:0.005, sput_c0:3.39, sput_c1:1.0,
sput_c2:2.17, sput_c3:2.5, sput_c4:1.8, sput_c5:2.56, emission:zhang, emit_c0:72.25, emit_v:5000}
```

The theory for Particle-Surface interactions can be found [Surface](#).

1.10.5 TURF Operations

In this baseline plume simulation, five different stages are used as defined in the [World file](#) shown above: INITIALIZE, MOVE, POSTMOVE, RHO, and FIELD. MPI communication is performed between stages. Therefore, using fewer stages results in less time spent on MPI communication, which leads to shorter run times.

For this reason, the name of the stage does not necessarily represent all the operations defined in the given stage. The operators provided in each stage are briefly explained in this section.

Stage: INITIALIZE

The INITIALIZE stage consists of operations that import geometries and create object data necessary for the simulation. The stage also injects particles and prepares to the stage MOVE by copying particles to different distributions. The operations are listed and briefly described in the [table](#) below.

Table 1.36 Summary of operations in stage INITIALIZE.

Operation	Description
UMeshImporterOp	Loads unstructured surface mesh (UMesh)
SurfaceComponentSetOp	Creates a database for geometry components
LogicalMeshSurfaceSugarcubeOp	Performs sugarcubing to determine the relationship between UMesh and a structured volume mesh (SMesh)
UFieldMSInitOp	Creates a UMesh field object class
UFieldICByComponentOp	Sets the surface field according to the component value
MSPDistInitOp	Creates a MSPDist object class
MSPDistNormalMaxwellianOp	Injection of particles at a Maxwellian distribution
MSPDistSourceRPAOp	Injection of particles according to RPA data
MSPDistCopyOp	Copies data in a MSPDist to other MSPDist
NextStageOp	Goes to next stage

The operation UMeshImporterOp imports the mesh file containing the spacecraft geometry. The spacecraft components can be made up of different materials and have parameters specific to those components as specified in an input file. The operation SurfaceComponentSetOp imports user-input parameters and values and creates a database. Then, the relationship between the surface mesh and the volume mesh is determined by LogicalMeshSurface↔SugarcubeOp.

```
DEFINE STAGE INITIALIZE
#####
## Load and Initialize Spacecraft Geometry ##
#####
DEFINE OPERATION
  TYPE = UMeshImporterOp
  UMESH_FILES = jaysat2_rotate.inp #<---
  SCALE = 0.012 #<---
  TRANSFORM_CENTER = 0.0, 0.0, 0.0 #<---
  TRANSLATE = -0.15, 0.0, 0.0 #<---
  ROTATION_AXIS = 0.0, 0.0, 1.0 #<---
  ROTATION_DEGREES = 0.0 #<---
  UMESH_TYPE = Surface
```

```

    UMESH_FILE_TYPE = Abaqus
    UMESH_NAME = UMesh
    WRITE_VTK_MESH = FALSE
END OPERATION
DEFINE OPERATION
    TYPE = SurfaceComponentSetOp
    SURFACE_NAME = UMesh
    #   DEFAULT_COMPONENT_MATERIAL_NAME = Al@s
    #   DEFAULT_SET_KEYS = potential, temperature
    #   DEFAULT_VALUES = 0.0, 500.0
    COMPONENT_MATERIAL_FILE = component_TURF.txt #<---
    VERBOSE = TRUE
END OPERATION
DEFINE OPERATION
    TYPE = LogicalMeshSurfaceSugarcubeOp
    SURFACE_NAME = UMesh
    SUGARCUBE_NAME = SC_UMesh
    SMESH_NAME = SMesh
    EXTERIOR_PTS = 0.0, 0.0, 4.0 #<---
    WRITE_VTK_MESH = TRUE
END OPERATION

```

- **SURFACE_FILE** points to the file of the desired satellite geometry.
- **SCALE** allows the user to account for unit conversions if the geometry was not built in meters.
- The **Transform**, **Translate**, and **Rotation** commands determine the position and orientation of geometry.
- **DEFAULT_COMPONENT_MATERIAL_NAME** sets the default material for all parts of the geometry. The input should be one of the materials named in the world file.
- **DEFAULT_SET_KEYS** specifies the list of keys to set defaults.
- **DEFAULT_VALUES** are the list of default values corresponding to the keys given for **DEFAULT_SET_KEYS**.
- **COMPONENT_MATERIAL_FILE** calls on the file containing information about the geometry components. If this command is used in addition to the default component initializations, values for components named in the material file have their default values overwritten.
- **EXTERIOR_PTS** are the x, y, and z coordinates of points exterior to the geometry. Multiple points can be defined as **EXTERIOR_PTS = x1, y1, z1, x2, y2, z2, ...**

A new unstructured mesh data field is created with **UFieldMSInitOp**. The initial surface field values are set in **UFieldICByComponentOp** according to the component data.

```

#####
## Initialize UMesh Field ##
#####
DEFINE OPERATION
    TYPE = UFieldMSInitOp
    UFIELD_DATA_NAME = SurfaceField
    UMESH_NAME = UMesh
    FIELD_NAMES = NC NCIN NCOU NPFLUX NPFLUX-IN NPFLUX-OUT PRES #<---
    DOF1_NAMES = Xe@g Xe+@g Al@g SUM #<---
    DOF_TYPES = SPECIES #<---
    FIELD_NO_DOF_NAMES = potIn #<---
    VERBOSE = TRUE
END OPERATION
#####
## Set Surface Potential from Inputs ##
#####
DEFINE OPERATION
    TYPE = UFieldICByComponentOp
    UFIELD_DATA_NAME = SurfaceField
    FIELD_NAMES = potIn #<---
    KEYS = potential #<---
    INIT_ONLY = TRUE
END OPERATION

```

The operator **UFieldMSInitOp** allocates unstructured field data according to the number of surface elements and number of surface fields. In this tutorial example, the name of the field data is determined by the inputs to **FIELD_**
_NAMES, **DOF1_NAMES**, and **FIELD_NO_DOF_NAMES**. By convention, the field base name is combined with the

degree of freedom through an underscore (e.g. NC_Xe@g where Xe@g is the degree of freedom of type SPECIES). Only three DOF_TYPES are available in this version: SPECIES, DERIVATIVE, and COORDINATE. To define field names, there are several other input keys not used in this example, such as FIELD_DOF1_NAMES, FIELD_DOF2←_NAMES, FIELD_DOF3_NAMES, DOF2_NAMES, DOF3_NAMES, and SPECIES_NAMES. FIELD_DOF#_NAMES only uses the degree of freedom defined by DOF#_NAMES, so the resulting field names only contain the base name and one degree of freedom. On the other hand, field names with all possible permutations of degree of freedom are created based on the inputs to FIELD_NAMES and FIELD_DOF#_NAMES. SPECIES_NAMES is deprecated but still works for backward compatibility, serving in the same way as FIELD_DOF1_NAMES when only one degree of freedom is defined. It should be noted that setting VERBOSE to true will output the list of field names, which is helpful when the simulation crashes with an error message that the field data name does not exist.

Some surface field data can be sampled from particles, and these names have to contain the names given in the [table](#) below. IN and OUT can be appended to names to specify the flux into or out from the surface. Without IN or OUT, the field data will contain the sum of the two (Note that the outflow field has the opposite sign as the inflow field).

Table 1.37 Surface fields that can be sampled from particles.

Variable Name	Description
NC	Number of simulation particle impacting a surface element
NP	Number of physical particle impacting a surface element
NPFLUX	Particle flux to a surface (particle/s · m ²)
MFLUX	Mass flux (kg/s · m ²)
PRES	Pressure (N/m ²)
TAU	Shear stress (N/m ²)
QFLUX	Heat flux (J/s · m ²)

In UFieldICByComponentOp, FIELD_NAMES should match the field names created by UFieldInitOp, whereas the inputs to KEYS must match the key defined in the component file, component_TURF.txt.

The MSPDist class objects that store particle information are created by MSPDistInitOp. P-DST is the primary particle distribution, P-GST and P-DEL are the temporary particle distributions, and P-EXC is the exchange particle distribution only used for particle patching between sub-domains. P-EXC is defined to be a global distribution, so only one buffer per MPI process is created and is shared among all the sub-domains.

```
#####
## Particle Distributions and Ghost/Exchange Distributions ##
#####
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-DST
  MAX_NP = 1024000 #<---
  SPECIES_NAMES = Xe@g Xe+@g Al@g #<---
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-GST
  MAX_NP = 1024000 #<---
  SPECIES_NAMES = Xe@g Xe+@g Al@g #<---
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-DEL
  MAX_NP = 1024000 #<---
  SPECIES_NAMES = Xe@g Xe+@g Al@g #<---
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistInitOp
  MSPDIST_DATA_NAME = P-EXC
  MAX_NP = 50000 #<---
  GLOBAL_DIST = TRUE
  SPECIES_NAMES = Xe@g Xe+@g Al@g #<---
END OPERATION
```

- MAX_NP sets the maximum number particles to be used in the simulation. It is important to ensure not to exceed

memory availability. If the particle count ever exceeds these values, TURF will crash with an error message.

- SPECIES_NAMES loads gas species that will be available in the particle distribution.

In this tutorial, neutral atoms are injected from one-sided Maxwellian distribution, while ions are injected according to the current density distribution measured by the Retarded Potential Analyzer (RPA).

```
#####
## Injection of Particles ##
#####
# Inject neutral atoms from surface
DEFINE OPERATION
  TYPE = MSPDistNormalMaxwellianOp
  MSPDIST_DATA_NAME = P-DST
  TEMPERATURE = 24920.0 #<---
  NORMAL_VELOCITY = 5000.0 #<---
  MASS_FLOW_RATE = 1.0e-4 #<---
  REAL_TO_COMPUTATIONAL = 1.0e13 #<---
  SPECIES = Xe@g # Can only inject one species #<---
  SURFACE_NAME = UMesh
  SURFACE_GROUP_NAME = HCT1_INJ #<---
  VERBOSE = FALSE
END OPERATION
# Inject ions from surface
DEFINE OPERATION
  TYPE = MSPDistSourceRPAOp
  SURFACE_NAME = UMesh #<---
  COMPONENT_AT_SPHERE_CENTER = HCT1_INJ #<---
  MSPDIST_DATA_NAME = P-DST
  SPECIES = Xe+@g #<---
  INPUT_CSV_FILE_NAME = RPA_HCT.csv #<---
  INJECTION_RADIUS = 0.3 #<---
  MASS_FLOW_RATE = 1.0e-4 #<---
  REAL_TO_COMPUTATIONAL = 1.0e13 #<---
  VERBOSE = FALSE
END OPERATION
```

- TEMPERATURE refers to the inflow temperature of the injected neutral atoms. Measured in Kelvin.
- NORMAL_VELOCITY is the streaming velocity with respect to the injection surface in m/s. Neutral atoms should nearly be thermal and are not likely to have a driven normal velocity, but the streaming velocity is set to a large value to accelerate the simulation in this example.
- MASS_FLOW_RATE dictates how many particles of each species are injected. Measured in kg/s.
- REAL_TO_COMPUTATIONAL represents the number of physical particles contained in a single computational particle. This ratio allows the simulation to scale the number of computational particles in the simulation to a manageable quantity. For most simulations tens of thousands to low millions of computational particles per MPI process leads to resonable simulation times. This value will also impact the accuracy of the simulation.
- SPECIES determines what kinds of particles are injected by each operation. Only one species can be injected for each block of operation.
- SURFACE_GROUP_NAME establishes where the particles are injected. It should be the name given to the thruster face's block in the geometry file.
- COMPONENT_AT_SPHERE_CENTER places the RPA source at the center of the given component. It should be the name given to the thruster face's block in the geometry file.
- INPUT_CSV_FILE_NAME calls on the csv file containing the RPA data and imports the information to provide an injection source.
- INJECTION_RADIUS sets the size of the RPA injection sphere.

Before proceeding with the simulation and beginning the MOVE stage, MSPDistCopyOp copies particles stored to P-DST to P-GST and is required before performing the particle-surface interaction calculation in TURF. The last operation in this stage, NextStageOp directs TURF to proceed to the next stage.

```
#####
## Prep for Advance Particle Positions ##
#####
# Copy the distribution from P-DST to P-GST (P-DST is initialized)
DEFINE OPERATION
  TYPE = MSPDistCopyOp
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-GST
END OPERATION
```

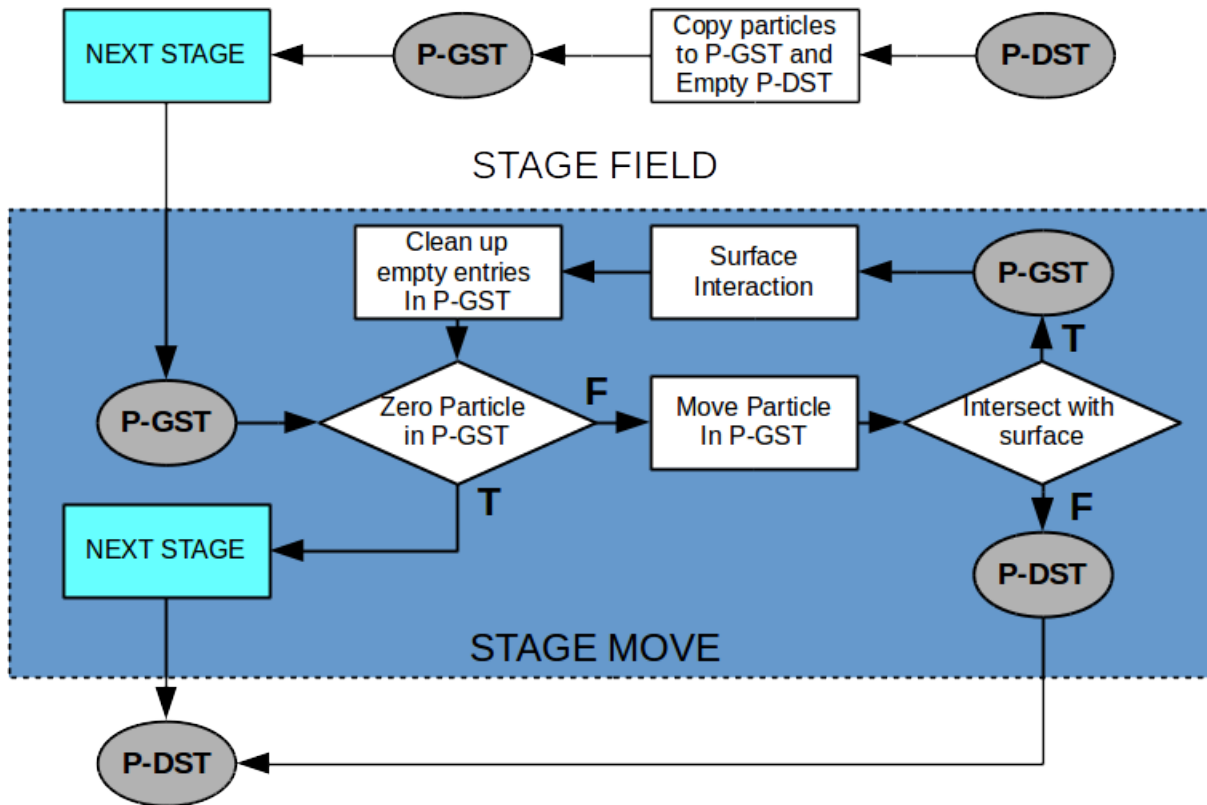
```
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE INITIALIZE
```

Stage: MOVE

The MOVE stage has operations that advance particles in an electric field and perform surface interaction calculations. This stage also performs calculations for numerical probes in order to compare the result with experimental data. The list of operations in the MOVE stage is given in the [table](#) below. Furthermore, the general flow of the surface interaction calculation in TURF is illustrated in the [figure](#) below.

Table 1.38 Summary of operations in stage MOVE

Operation	Description
MSPDistESPushOp	Advances particles using node-centered electric fields
MSPDistProbeStageSphericalOp	Spherical stage probe
MSPDistSugarcubeSurfIntersectionOp	Determines if particles intersect with surface mesh during a time-step
MSPDistProbeFixedOp	Probe fixed to a surface
MSPDistThinSheathOp	Applies thin sheath boundary condition to particles
MSPDistSurfaceInteractionOp	Performs sputtering and surface reflections
MSPDistRemovePartOp	Removes particles that stuck to a surface or reached the final time within the stage
MSPDistEmptyOp	Sets particle count in the distribution to zero
MSPDistCellIDOp	Determines cell ID that particles reside
MSPDistCombineOp	Combines two particle distributions
MSPDistParticleCountOp	Counts particles
CriteriaStageOp	Repeats this stage until particle count becomes zero



The first operation of stage MOVE pushes particles using the electric field stored at the nodes.

```

DEFINE STAGE MOVE
#####
## Keep Advancing Particles Until Reaching the Final Time Per Step ##
#####
# Advance P-GST for a full time-step
DEFINE OPERATION
    TYPE = MSPDistESPushOp
    FIELD_DATA_NAME = FieldData
    FIELD_EN_PREFIX = En
    FIELD_EN_DIRECTIONS = [x, y, z]
    MSPDIST_DATA_NAME = P-GST
    SPECIES_NAMES = Xe@g Xe+@g Al@g #<---
    USE_CFL_TIME_STEP = FALSE
    CFL_NUMBER = 0.5
END OPERATION

```

- SPECIES_NAMES specifies which particles are to be moved with the particle distribution during each time step.

TURF is able to mimic Faraday and RPA probes to measure current densities and energy distributions. These pseudo probes can be placed at any component of the spacecraft or along a virtual spherical surface and told to begin collecting data at any point in time in the simulation. Probes using MSPDistProbeStageSphericalOp operations take measurements across a virtual sphere, and both the Faraday and RPA type probes can be used.

```

# Probe sweeping
# *** This needs to be done before MSPDistSugarcubeSurfIntersectionOp
# *** since particles in P-GST are marked for deletion by setting the
# *** weight to zero in that routine, but the weight is needed for
# *** probe measurement
DEFINE OPERATION
    TYPE = MSPDistProbeStageSphericalOp
    NAME = FARADAY_Axi
    MSPDIST_DATA_NAME = P-GST
    PROBE_TYPE = FARADAY # FARADAY or RPA
    SURFACE_NAME = UMesh
    COMPONENT_AT_SPHERE_CENTER = HCT1_INJ #<---

```

```

    SWEEP_RADIUS = 0.1 #<---
    PROBE_POLAR_ANGLE = 1.0 #<---
    SWEEP_POLAR_ANGLE_START = 0.0 #<---
    SWEEP_POLAR_ANGLE_END = 120.0 #<---
    NUMBER_OF_POLAR_ANGLE_BINS = 120 #<---
    VIEW_ANGLE = 30.0 # in degrees #<---
    START_TIME = 2.0e-3 #<---
    STEPS_PER_SAMPLING = 1 # Samples every (SAMPLING) iteration #<---
    VERBOSE = FALSE
END OPERATION
DEFINE OPERATION
    TYPE = MSPDistProbeStageSphericalOp
    NAME = RPA_Axi
    MSPDIST_DATA_NAME = P-GST
    PROBE_TYPE = RPA # FARADAY or RPA
    SURFACE_NAME = UMesh
    COMPONENT_AT_SPHERE_CENTER = HCT1_INJ #<---
    SWEEP_RADIUS = 0.1 #<---
    PROBE_POLAR_ANGLE = 10.0 #<---
    SWEEP_POLAR_ANGLE_START = 0.0 #<---
    SWEEP_POLAR_ANGLE_END = 120.0 #<---
    NUMBER_OF_POLAR_ANGLE_BINS = 12 #<---
    MINIMUM_ENERGY = -0.5 # in eV - only for RPA #<---
    MAXIMUM_ENERGY = 500.5 # in eV - only for RPA #<---
    NUMBER_OF_ENERGY_BINS = 501 # - only for RPA #<---
    VIEW_ANGLE = 15.0 # in degrees #<---
    START_TIME = 2.0e-3 #<---
    STEPS_PER_SAMPLING = 1 # Samples every (SAMPLING) iteration #<---
    VERBOSE = FALSE
END OPERATION

```

- **COMPONENT_AT_SPHERE_CENTER** specifies the geometry component that the measurement sphere is referenced to. The center of the spherical surface collecting currents is placed at the center of the component specified here.
- **SWEEP_RADIUS** specifies the radius of the measurement sphere.
- **SWEEP_PROBE _ANGLE_START** and **SWEEP_PROBE _ANGLE_END** are the first and last polar angles of the measurement sphere to sweep the probe.
- **NUMBER_OF_BINS** divides the range of angles into the desired number of equally-sized sections. Measurements are taken at the center of each bin, and the thickness of each bin is determined by **PROBE_POLAR_↵ANGLE**.
- Current is only collected if the incident angle is smaller than the **VIEW_ANGLE**.
- **START_TIME** specifies the time to start the probe measurement.
- Measurements of current are only collected at multiples of the **STEPS_PER_SAMPLING** value.
- **MAXIMUM_ENERGY** and **MINIMUM_ENERGY** set the energy detection limits (in eV) for the RPA probe.
- **NUMBER_OF_ENERGY_BINS** divides the range of energies into the desired number of equally-sized sections.

The operation **MSPDistSugarcubeSurfIntersectionOp** determines whether particles impact surface mesh elements and moves these particle data to P-DST. The list of particle impacting elements is stored here and used when performing the surface reflection and/or sputtering calculations.

```

# Find intersection of particle trajectories with UMesh
# *** Starting from P-GST, particles are copied to P-DST once reaching the final time
# *** Then, P-GST contains particles bouncing at surface during the time-step
# *** And, P-DST contains particles making it to the final time per iteration
DEFINE OPERATION
    TYPE = MSPDistSugarcubeSurfIntersectionOp
    NAME = Intersection_P-GST
    MSPDIST_SRC_NAME = P-GST
    MSPDIST_DST_NAME = P-DST
    MSPDist_ERROR_NAME = P-EXC
    SUGARCUBE_NAME = SC_UMesh
    SURFACE_NAME = UMesh
    SURFACE_TEST = FALSE
    VERBOSE = FALSE

```

```
END OPERATION
```

The other probe operation is `MSPDistProbeFixedOp`, which attach an unmoving probe to a component of the spacecraft surface mesh. The surface component can be specified as a virtual surface by specifying the material as "virtual" (material:virtual) in `component_TURF.txt`, and using this surface for this operation. In this way, particles can fly through this surface while probe measurement is taken.

```
# Fixed probe
DEFINE OPERATION
  TYPE = MSPDistProbeFixedOp
  NAME = RPA_Fixed
  SURFACE_NAME = UMesh
  MSPDIST_DATA_NAME = P-GST
  SURF_INTERSECTION_OP_NAME = Intersection_P-GST
  COMPONENT_NAMES = SP_E1 #<---
  PROBE_TYPE = RPA # FARADAY or RPA
  VIEW_ANGLE = 90.0 # in degrees #<---
  MINIMUM_ENERGY = -0.5 # in eV - only for RPA #<---
  MAXIMUM_ENERGY = 500.5 # in eV - only for RPA #<---
  NUMBER_OF_ENERGY_BINS = 501 # - only for RPA #<---
  START_TIME = 2.0e-3 #<---
  STEPS_PER_SAMPLING = 1 # Samples every (SAMPLING) iteration #<---
  VERBOSE = FALSE
END OPERATION
```

- `COMPONENT_NAMES` dictates which spacecraft component the probe is attached to.
- Current is only collected if the incident angle is smaller than the `VIEW_ANGLE`.
- `MAXIMUM_ENERGY` and `MINIMUM_ENERGY` set the energy detection limits (in eV) for the RPA probe.

One important upgrade from the 2017 version is the application of thin sheath approximation to charged particles. In the past, electric potential on a cell intersecting a geometry was set to a fixed potential specified in `component_TURF.txt`. This introduced a significant potential drop from the free space cell to the boundary cell, resulting in a numerical sheath with the depth of the cell size and bending the slow ion trajectories toward the walls. Instead of setting the boundary cell potential to coincide with the surface potential, `LogicalFieldGeometrySetOp` defined later tries to minimize the artificial potential drop by extrapolating the potential from the neighboring free-space cell potential, and `MSPDistThinSheathOp` alters the charged particle speed and direction only at the time of surface impact.

```
# Reflection by thin sheath
DEFINE OPERATION
  TYPE = MSPDistThinSheathOp
  MSPDIST_DATA_NAME = P-GST
  FIELD_DATA_NAME = FieldData
  FIELD_POTENTIAL_NAME = phi
  UFIELD_DATA_NAME = SurfaceField
  UFIELD_POTENTIAL_NAME = potIn
  SURFACE_NAME = UMesh
  SURF_INTERSECTION_OP_NAME = Intersection_P-GST
  VERBOSE = FALSE
END OPERATION
```

The operation `MSPDistSurfaceInteractionOp` performs surface reflection and sputtering for different combinations of source and target species defined in `surf_int.txt`.

```
# Surface interaction - Reflection, sticking, and sputtering
DEFINE OPERATION
  TYPE = MSPDistSurfaceInteractionOp
  NAME = Interaction_P-GST
  SURFACE_NAME = UMesh
  MSPDIST_DATA_NAME = P-GST
  UFIELD_DATA_NAME = SurfaceField
  SURF_INTERSECTION_OP_NAME = Intersection_P-GST
  FILE_NAME = surf_int.txt
  STEPS_PER_SAMPLING = 1
  VERBOSE = FALSE
END OPERATION
```

While MSPDistSugarcubeSurfIntersectionOp moves particles reaching the end time within each time-step to P-GST, these particles still need to be deleted from P-DST. This is done by marking them in P-DST by setting their weights to be zero and actually removing them from P-DST by calling MSPDistRemovePartOp.

```
# Remove particles with zero weight
DEFINE OPERATION
  TYPE = MSPDistRemovePartOp
  MSPDIST_SRC_NAME = P-GST
  MSPDIST_DST_NAME = P-DEL
  REMOVE_ZERO_WEIGHT = TRUE
  REMOVE_CELLID_GT_MAX = FALSE
  DISCARD_REMOVED_PARTICLE = TRUE
END OPERATION
```

The following two operations monitor the particle count in P-GST; the simulation will continue the current stage until the particle count becomes zero.

```
#####
## Quantity Needs To Be Smaller Than CRITERIA_LONG ##
#####
# Obtain particle count in P-GST
DEFINE OPERATION
  TYPE = MSPDistParticleCountOp
  MSPDIST_DATA_NAME = P-GST
  QUANTITY_NAME = MSPDistCount
  VERBOSE = FALSE
END OPERATION
#####
## Proceed to the Next Stage If Criterion is Satisfied ##
#####
# Go to next stage if P-GST->Nparts is smaller than 1
DEFINE OPERATION
  TYPE = CriteriaStageOp
  QUANTITY_NAME = MSPDistCount
  CRITERIA_LONG = 1 #Go to next stage if P-GST->Nparts is smaller than 1
  MAX_ITERATION = 100 #Max number of subcycles to prevent stuck particles
END OPERATION
END STAGE MOVE
#####
```

Stage: POSTMOVE

The stage POSTMOVE starts the particle patching, which is completed at the first operator in the next stage.

Table 1.39 Summary of operations in stage POSTMOVE

Operation	Description
MSPDistPatchOp	Patches particle data across sub-domains
NextStageOp	Goes to next stage

After particles are advanced by one time-step, they can be outside of the sub-domain that they resided in the previous time-step. For a multi-domain simulation, these particles have either left the simulation domain or landed to another sub-domain. Therefore, before deleting particles that have left the simulation domain, the particle data need to be patched across the sub-domains. MSPDistPatchOp copies all of these particles to a global particle distribution, P-EXC.

```
#####DEFINE STAGE POSTMOVE
#####
## Perform Particle Patch ##
#####
# Particle CellID greater than maximum is copied to P-EXC (CellID set in MSPDistESPushOp)
DEFINE OPERATION
  TYPE = MSPDistPatchOp
  MSPDIST_SRC_NAME = P-DST
  GLOBAL_SRC_DIST = FALSE
  GLOBAL_MSPDIST_NAME = P-EXC
  VERBOSE = FALSE
END OPERATION
#####
```

```
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
    TYPE = NextStageOp
END OPERATION
END STAGE POSTMOVE
```

Stage: RHO

The RHO stage has operations which complete the particle patching, sample particles to compute the field data, output restart files, compute space charge, and set flags to perform the field patching. The list of operations in RHO stage is given in the [table](#) below.

Table 1.40 Summary of operations in stage RHO.

Operation	Description
MSPDistCopyOp	Copies particles to complete particle patching
MSPDistCellIDOp	Determines SMesh cell IDs based on particle positions
MSPDistSampleOp	Samples MSPDist data to compute SMesh fields
SampleFieldWriteVTKOp	Writes sampling field data in VTK format (for restart)
MSPDistWriteVTKOp	Writes MSPDist data in VTK format (for restart)
FieldSetOp	Set a SMesh field to a value
MSPDistChargeDepositionOp	Deposits particle charges to a grid
FieldArithmeticOp	Divides by cell volumes to get charge densities
LogicalFieldPatchOp	Patches field data across sub-domains
NextStageOp	Goes to next stage

The operation MSPDistCopyOp distributes particles in the global distribution to the local particle distribution owned by the appropriate sub-domain. Then, the cell ID is updated by MSPDistCellIDOp.

```
#####
## Combine Particles in Ghost Cells from Prior Step ##
#####
DEFINE OPERATION
    TYPE = MSPDistCopyOp
    MSPDIST_SRC_NAME = P-EXC
    GLOBAL_SRC_DIST = TRUE
    MSPDIST_DST_NAME = P-DST
    GLOBAL_DST_DIST = FALSE
    ONLY_COPY = TRUE
END OPERATION
# Update CellID
DEFINE OPERATION
    TYPE = MSPDistCellIDOp
    MSPDIST_DATA_NAME = P-DST
END OPERATION
```

The operation MSPDistSampleOp samples particle count, velocity, and kinetic energy and converts them to the various field data defined in the world file.

```
#####
## Sum to Fields for Output ##
#####
# Sampling of neutral atom is done more frequently for elastic collision calculation
DEFINE OPERATION
    TYPE = MSPDistSampleOp
    FIELD_DATA_NAME = FieldData
    MSPDIST_DATA_NAME = P-DST
    FIELD_NAMES = n_Xe@g #<---
    MIX_START_TIME = 2.0e-3 #<---
    STEPS_PER_SAMPLING = 1 #<---
    STEPS_PER_FIELD_CALCULATION = 1 #<---
END OPERATION
# Other fields only need to be computed when output
DEFINE OPERATION
```

```

TYPE = MSPDistSampleOp
NAME = Sample_P-DST
FIELD_DATA_NAME = FieldData
MSPDIST_DATA_NAME = P-DST
FIELD_NAMES = VOL n_Xe+@g n_Al@g #<---
MIX_START_TIME = 2.0e-3 #<---
STEPS_PER_SAMPLING = 1 #<---
STEPS_PER_FIELD_CALCULATION = 100 #<---
END OPERATION

```

- MIX_START_TIME should be set to approximately the time when particle steady-state is reached. This is to improve the statistics by taking the time-average through multiple time-steps.
- STEPS_PER_SAMPLING is the number of iterations conducted before sample data are collected.
- STEPS_PER_FIELD_CALCULATION is the number of iterations conducted before sample data are used for field calculations. Value should be equal to or a multiple of the operation's STEPS_PER_SAMPLING value.

SampleFieldWriteVTKOp writes the restart file for the volume mesh sampling field, and MSPDistWriteVTKOp outputs particle data that can be used as a restart file.

```

#####
## Output for Restart ##
#####
DEFINE OPERATION
  TYPE = SampleFieldWriteVTKOp
  SAMPLE_OP_NAME = Sample_P-DST
  MSPDIST_DATA_NAME = P-DST
  FILE_HEAD = Restart/sampleA
  SKIP = 500 #<---
  FORMAT = Binary
  COMPRESS_DATA = TRUE
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistWriteVTKOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = ALL
  FILE_HEAD = Restart/particleA
  SKIP = 500 #<---
  PARTICLE_SKIP = 1.0
  FORMAT = Binary
  COMPRESS_DATA = TRUE
  OVERWRITE = TRUE
END OPERATION

```

- SKIP tells how frequently the restart file should be written.
- FORMAT is the format of written data. Both ASCII and Binary are supported.
- COMPRESS_DATA is set TRUE to compress the binary data with zlib routine.
- OVERWRITE is set TRUE if only one restart file is kept.

The operations shown below compute the charge density. When the first order weighting scheme is used for charge deposition, the charge density field needs to be patched across the sub-domains using LogicalFieldPatchOp.

```

#####
## Deposit Charge for Boltzmann ##
#####
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  VALUE = 0.0
  OPERATION = SET
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistChargeDepositionOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  MSPDIST_DATA_NAME = P-DST
  ORDER = 1

```

```

END OPERATION
# Divide by cell volume to get charge density
DEFINE OPERATION
  TYPE = FieldArithmeticOp
  FIELD_DATA_NAME = FieldData
  FIELD_SRCB_NAME = rho
  FIELD_SRCC_NAME = VOL
  FIELD_DST_NAME = rho
  OPERATION = DIV
END OPERATION
#####
### Patch Rho before Boltzmann Solve ###
#####
DEFINE OPERATION
  TYPE = LogicalFieldPatchOp
  FIELD_NAME = rho
  PATCH_OP = PATCH_SUM
END OPERATION
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE RHO

```

Stage: FIELD

The FIELD stage contains many operations not limited to the ones involving field data. These operations unify the contiguous region across sub-domains, solve for electric potential via Boltzmann inversion, compute electric fields, apply collisions, and output data. The list of operations used in this stage is shown in the [table](#) below.

Table 1.41 Summary of operations in stage FIELD

Operation	Description
LogicalFieldPatchOp	Stops patching of field data
LogicalBCMirrorMoveInOp	Reassigns rho distributed to ghost cells to interior cells
FieldSetOp	Sets the minimum allowable density value
FieldBlendTimeOp	Time-averages charge density
LogicalPotentialSetInsideGeometryOp	Sets potential within a geometry
LogicalPotentialBoltzmannOp	Potential solve by a Boltzmann inversion
LogicalFieldGeometrySetOp	Extrapolates field values at geometry boundaries from free space to minimize numerical sheath
LogicalBCOp	Extrapolates field values from interior cells to ghost region
LogicalNodeGradientOp	Computes electric fields at nodes
MSPDistSortOp	Sort particles and remove particles outside the simulation domain
MSPDistMCCElasticFitOp	Performs MCC elastic collisions using a curve-fit to differential cross-section
LogicalFieldWriteVTKOp	Writes 3D data in VTK format
LogicalFieldWriteVTK2DOp	Writes 2D data in VTK format
MSPDistWriteVTKOp	Writes MSPDist data in VTK format
UFieldMSComputeOp	Computes surface mesh field
UFieldWriteVTKOp	Writes surface mesh field in VTK format
ProbeStageWriteOp	Writes spherical stage probe data to a csv file
ProbeFixedWriteOp	Writes fixed probe data to a csv file
MSPDistParticleCountOp	Counts particles in all the sub-domains
NextStageOp	Goes to next stage

The first operation in the FIELD stage is LogicalFieldPatchOp, which is called again to terminate patching after the RHO stage by setting PATCH_OPS to PATCH_NULL. Then the next three operations complete the calculation for the charge density for the given TURF time-step. The operation LogicalBCMirrorMoveInOp moves the charge density assigned to the cells outside of the simulation domain to the outermost cells inside the domain. This operation is only required if the order of particle weighting is greater than zero. Then, the operation FieldSetOp sets the floor to the charge density ρ_0 , which is effectively the background density of the plasma. This prevents unrealistic potential values predicted by the Boltzmann inversion model, since the model predicts negative infinity if ρ_0 is zero. Finally, FieldBlendTimeOp time-averages the charge density to obtain a smoother field data. Either the instantaneous or the time-averaged charge density (ρ_0 and ρ_{0_avg} , respectively) can be used later in the potential calculation.

```
#####
## Add Charge Distributed at Ghost to Interior ##
#####
DEFINE OPERATION
  TYPE = LogicalBCMirrorMoveInOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
END OPERATION
#####
## Potential Solve via Boltzmann Inversion ##
#####
# Adding background density for Boltzmann solve
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  VALUE = 1.602e-7 #<---
  OPERATION = FLOOR
END OPERATION
# Time average rho after TIME_START_AVERAGING
# *** rho_avg can be used for later calculations or just for plotting
DEFINE OPERATION
  TYPE = FieldBlendTimeOp
  FIELD_DATA_NAME = FieldData
  FIELD_SRC_NAME = rho
  FIELD_DST_NAME = rho_avg
  OPERATION = AVERAGE
  TIME_START_BLEND = 2.0e-3 #<---
END OPERATION
```

- TIME_START_BLEND is essentially the same as the MIX_START_TIME defined in MSPDistSampleOp, which should be the approximate beginning of particle steady state.

The next four operations calculate and set the electric potential value. The operation LogicalPotentialSetInsideGeometryOp uses the sugarcubing data to identify which volume cell to calculate the electric potential. This operator is also used to determine the validity of Boltzmann electron assumption and flags the invalid region where the Poisson equation solver should be used in place of the Boltzmann inversion model. The so-called "Quasi-Neutral" switch is likely needed for Geosynchronous Earth Orbit (GEO) but is not used in this tutorial example. The upgrade from the TURF 2017 release version includes LogicalFieldGeometrySetOp as mentioned above along with MSPDistThinSheathOp in the MOVE stage. This operator extrapolates the potential value from the free-space to minimize the numerical sheath near the geometry intersecting cells. Finally, LogicalBCOp extrapolates the potential values from outermost domain cells to the ghost region, which minimizes the electric field near the domain edge. Without this operation, the steep drop in the potential accelerates charged particles and the effect cascades across multiple cell layers from the edge of the simulation domain.

```
# Set potential inside geometries and flag poisson region
DEFINE OPERATION
  TYPE = LogicalPotentialSetInsideGeometryOp
  SUGARCUBE_NAME = SC_Umesh
  FIELD_DATA_NAME = FieldData
  FIELD_CHARGE_DENSITY_NAME = rho_avg #rho
  FIELD_POTENTIAL_NAME = phi
  POTENTIAL_MODEL = BOLTZMANN
  TEMPERATURE_MODEL = POLY # CONST #<---
  REFERENCE_GAMMA = 1.3 #<---
  REFERENCE_ELECTRON_DENSITY = 3.11135E17 #<---
  REFERENCE_ELECTRON_TEMPERATURE_EV = 8.01663 #<---
  REFERENCE_POTENTIAL = 35.7182 #<---
```

```

        FIELD_POTENTIAL_FLAG_NAME = pflag
        CALCULATE_BOUNDARY_FIELD = FALSE
    END OPERATION
    # Compute potential using Boltzmann relations
    DEFINE OPERATION
        TYPE = LogicalPotentialBoltzmannOp
        FIELD_DATA_NAME = FieldData
        FIELD_CHARGE_DENSITY_NAME = rho
        FIELD_POTENTIAL_NAME = phi
    END OPERATION
    # Reset geometry
    DEFINE OPERATION
        TYPE = LogicalFieldGeometrySetOp
        FIELD_DATA_NAME = FieldData
        FIELD_NAMES = phi
        SURFACE_NAME = UMesh
        SUGARCUBE_NAME = SC_UMesh
        SET_OPTION = FREE
        SKIP_BOUNDARY = FALSE
    END OPERATION
    # Copy Potential to Ghost Cells
    DEFINE OPERATION
        TYPE = LogicalBCOp
        FIELD_NAMES = phi
        BOUNDARY_TYPE = FLAT
    END OPERATION

```

The operation `LogicalNodeGradientOp` computes the electric field on the mesh nodes (also called vertices) from the eight adjacent cells.

```

#####
## Compute Electric Field ##
#####
    DEFINE OPERATION
        TYPE = LogicalNodeGradientOp
        FIELD_DATA_NAME = FieldData
        FIELD_POTENTIAL_NAME = phi
        FIELD_GRADIENT_PREFIX = En
        FIELD_MULTIPLY_CONSTANT = -1.0
        FIELD_GRADIENT_DIRECTIONS = [x, y, z]
        SUGARCUBE_NAME = SC_UMesh
    END OPERATION

```

The operation `MSPDistMCCElasticFitOp` performs the elastic collision calculations between high energy ions and the background neutral atoms via Monte Carlo Collision model. Here, the elastic collision calculations are split into two mechanisms, i.e. momentum exchange (MEX) and charge exchange (CEX) interactions, and each interaction is approximated independently by `MSPDistMCCElasticFitOp`.

```

#####
## Sort and remove particles outside of domain ##
#####
    DEFINE OPERATION
        TYPE = MSPDistSortOp
        MSPDIST_SRC_NAME = P-DST
        MSPDIST_DST_NAME = P-GST
    END OPERATION
    #####
    ## Collision Calculations ##
    #####
    # MEX Collision
    DEFINE OPERATION
        TYPE = MSPDistMCCElasticFitOp
        MSPDIST_DATA_NAME = P-DST
        SPECIES = Xe+@g
        FIELD_DATA_NAME = FieldData
        FIELD_NAME = n_Xe@g
        TEMPERATURE_OF_TARGET_SPECIES = 0.0 #<---
        MEX_OR_CEX = MEX
        SIGMA_COEFFS = -27.2 171.13 1.0e-20 #<---
        FIRST_FIT_EXPONENT_DIFF_SIGMA = -2.02 #<---
        SECOND_FIT_EXPONENT_DIFF_SIGMA = 3.24 #<---
        FIT_ANGLE_DIFF_SIGMA = 3.526E-5
    END OPERATION
    # CEX Collision
    DEFINE OPERATION

```

```

TYPE = MSPDistMCElasticFitOp
MSPDIST_DATA_NAME = P-DST
SPECIES = Xe+@g
FIELD_DATA_NAME = FieldData
FIELD_NAME = n_Xe@g
TEMPERATURE_OF_TARGET_SPECIES = 300.0 #<---
MEX_OR_CEX = CEX
SIGMA_COEFFS = -27.2 171.13 1.0e-20 #<---
FIRST_FIT_EXPONENT_DIFF_SIGMA = -1.098 #<---
SECOND_FIT_EXPONENT_DIFF_SIGMA = 1.53 #<---
FIT_ANGLE_DIFF_SIGMA = 1.375E-3
END OPERATION

```

- TEMPERATURE_OF_TARGET_SPECIES is the thermal velocity of the neutrals
- SIGMA_COEFFS is the coefficients to compute the total CEX cross-section given as $\sigma = (c_0 \log 10(v_{rel}) + c_1) * c_2$
- FIRST_FIT_EXPONENT_DIFF_SIGMA, SECOND_FIT_EXPONENT_DIFF_SIGMA, and FIT_ANGLE<_DIFF_SIGMA are the coefficients used in the curve-fit equation for the differential cross-section. (See Ref.[33] for details)

The operations LogicalFieldWriteVTKOp and LogicalFieldWriteVTK2DOp write field data to VTK format files. Data for the fields specified in FIELD_NAMES will be written files with the file head specified by FILE_HEAD.

```

#####
## Write VTK Output ##
#####
# 3D data
DEFINE OPERATION
  TYPE = LogicalFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = Plot3D1A/plt_ #<---
  FIELD_NAMES = VOL pflag rho rho_avg phi Enx Eny Enz n_Xe@g n_Xe+@g n_Al@g #<---
  SKIP = 100 #<---
  PLOT_GHOST = FALSE
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
# 2D data, snaps to a plane that goes through cell-centers
DEFINE OPERATION
  TYPE = LogicalFieldWriteVTK2DOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = PlotXZ1A/plt_ #<---
  FIELD_NAMES = VOL pflag rho rho_avg phi Enx Eny Enz n_Xe@g n_Xe+@g n_Al@g #<---
  PLANE_AXES = XZ #<---
  SKIP = 100 #<---
  PLOT_GHOST = FALSE
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION

```

The operation MSPDistWriteVTKOp writes particle data to a VTK format file. Here, the operation is commented out since this is very slow and should only be used for debugging purposes. After the volume mesh field data are computed, the surface data sampled in MSPDistSurfaceInteractionOp are used to compute the surface field data UFieldMSComputeOp and output by UFieldWriteVTKOp.

```

#####
## Sum to Fields for Output ##
#####
# Compute UField quantities
# *** This needs to be after surface interaction
DEFINE OPERATION
  TYPE = UFieldMSComputeOp
  UFIELD_DATA_NAME = SurfaceField
  SURF_INTERACTION_OP_NAME = Interaction_P-GST
  MIX_START_TIME = 2.0e-3 #<---
  STEPS_PER_FIELD_CALCULATION = 100 #<---
END OPERATION
#####
## Output Surface Field ##
#####
DEFINE OPERATION
  TYPE = UFieldWriteVTKOp
  UFIELD_DATA_NAME = SurfaceField

```

```

FILE_HEAD = PlotSurf1A/pltSurf_ #<---
FIELD_NAMES = potIn NPFLUX-IN_Xe@g NPFLUX-IN_Xe+@g NPFLUX-OUT_Al@g PRES_SUM #<---
SKIP = 100 #<---
FIRST_DOMAIN_ONLY = TRUE
REMOVE_DIR_AT_INIT = TRUE
END OPERATION

```

- MIX_START_TIME is the time to start averaging between iterations for the surface fields in order to improve statistics.
- FIELD_NAME specifies which fields are exported into the FILE_HEAD folder.
- SKIP tells TURF how many iterations to wait/skip before the field information is exported. This value should be equal to (or a multiple of) the STEPS_PER_CALCULATION value in the UFieldMSSComputeOp operation preceding it.

The operations ProbeStageWriteOp and ProbeFixedWriteOp export the data collected by the pseudo probes from the MOVE stage to csv files named in FILE_HEAD.

```

#####
## Output probe data ##
#####
DEFINE OPERATION
  TYPE = ProbeStageWriteOp
  PROBE_OP_NAME = FARADAY_Axi
  STEPS_PER_OUTPUT = 100 # Averages out (OUTPUT/SAMPLING) data #<---
  TIME_DEPENDENT_DATA = FALSE # Overwrites the same file if false
  RESET_SAMPLING = FALSE
  FILE_HEAD = Probela/FARADAY_Axi #<---
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
DEFINE OPERATION
  TYPE = ProbeStageWriteOp
  PROBE_OP_NAME = RPA_Axi
  STEPS_PER_OUTPUT = 100 # Averages out (OUTPUT/SAMPLING) data #<---
  TIME_DEPENDENT_DATA = FALSE # Overwrites the same file if false
  RESET_SAMPLING = FALSE
  FILE_HEAD = Probela/RPA_Axi #<---
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
DEFINE OPERATION
  TYPE = ProbeFixedWriteOp
  PROBE_OP_NAME = RPA_Fixed
  STEPS_PER_OUTPUT = 100 # Averages out (OUTPUT/SAMPLING) data #<---
  TIME_DEPENDENT_DATA = FALSE # Overwrites the same file if false
  RESET_SAMPLING = FALSE
  FILE_HEAD = Probela/RPA_Fixed #<---
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION

```

The last operation MSPDistParticleCountOp determines the total number of particles across all the sub-domains when GLOBAL is set to true.

```

#####
## Count Particles at the End of Iteration ##
#####
DEFINE OPERATION
  TYPE = MSPDistParticleCountOp
  MSPDIST_DATA_NAME = P-DST
  LOG_FILE = ParticleCount1A.csv #<---
  VERBOSE = TRUE
  GLOBAL = TRUE
END OPERATION

```

1.10.6 Restarting the Simulation

The current version of TURF can only perform the restart of the particle distribution and the volume mesh field data. The restart of the surface field data is not implemented but will potentially be included in the future version. The particle

data written to the VTK format file by MSPDistWriteVTKOp can be read by MSPDistReadVTKOp. The operation reads the particle data and assigns particles to the correct sub-domain. On the other hand, SampleFieldReadVTKOp reads the data file written by SampleFieldWriteVTKOp containing the accumulated field data sampled from particles. While the particle restart is not restricted to the same domain partitioning before and after the restart, the sampled field restart can only be performed with exactly the same domain partitioning and mesh.

```
#####
## Extract Particle Data from Restart File ##
#####
DEFINE OPERATION
  TYPE = MSPDistReadVTKOp
  MSPDIST_DATA_NAME = P-DST
  FILE_HEAD_PVTP = Restart/particleA
  UPDATE_WORLD_TIME = TRUE
  ADVANCE_TIME_BY_DT = TRUE
  REINDEX = FALSE
END OPERATION
```

```
#####
## Restart Sampled Field Data ##
#####
DEFINE OPERATION
  TYPE = SampleFieldReadVTKOp
  SAMPLE_OP_NAME = Sample_P-DST
  MSPDIST_DATA_NAME = P-DST
  FILE_HEAD = Restart/sampleA
  UPDATE_WORLD_TIME = FALSE #Taken care of by MSPDistReadVTKOp
  ADVANCE_TIME_BY_DT = FALSE
  RESET_SAMPLING = FALSE
END OPERATION
```

1.10.7 Extension to Multi-Domain Simulation

The baseline plume simulation is already set up to run with multiple MPI processes and sub-domains. The only modification is the domain definition in the world file. In this example, the simulation domain is partitioned into four sub-domains.

```
#####
## Domain Geometry
#####
DEFINE DOMAIN DOM
  BOUND_LO = (-3.5, -2.5, -2.5)
  BOUND_HI = ( 5.0,  2.5,  7.5)
  SUB_DOMAINS = (1,2,2)
END DOMAIN
```

1.11 EP Plume Simulation 2: Fractional DSMC Collision

Author

Samuel J. Araki

1.11.1 Introduction

A series of electric propulsion (EP) plume simulations is provided in the TURF tutorial. Starting from the [baseline plume simulation](#), advanced models are incrementally added in the order of the DSMC collision method (this page), [spacecraft charging](#), and [adaptive mesh refinement and dynamic load balancing](#). There are also tutorials on how to create geometry file using [Cubit](#) and how to visualize field data and particles using [Paraview](#). This tutorial covers how to convert the Monte Carlo Collision (MCC) model to the Direct Simulation Monte Carlo (DSMC) collision model. This tutorial also includes an extra example showing how to split primary ions and charge-exchange (CEX) ions.

The MCC model samples a target particle from a fluid, while the DSMC collision samples the target particle directly from the particle distribution. In the baseline plume simulation, the neutral density is computed by the neutral particles, and the collision is applied against the fluid neutral via the MCC model (i.e. the collision rate is computed from the fluid, and the post-collision velocity is determined by colliding against a randomly sampled neutral from the fluid). Between the two-step process, the neutral particle velocity information is lost, and sampling the neutral from the fluid does not perfectly recover the correct neutral velocity distribution. While this does not have much impact to the MEX collision (due to the resultant charged particle moving away quickly), it has been shown that it has a significant effect to the CEX collision.[4] The initial CEX ion velocity affects its trajectory significantly, even though the final speed is mostly attributed to the strong electric field at the plume edge, so incorrect velocity of target neutrals results in different flux distribution at the geometry walls. Another limitation of the MCC model is that it is only valid under an assumption that the background neutral fluid is at much higher density compared to the ion density, even though there is only two orders of magnitude difference in densities in the EP plume. This limitation can be resolved by reducing neutral particle's computational weight accordingly, but at this point, there is no advantage over the DSMC method with regards to numerical efficiency. In comparison, the DSMC method uses the correct target neutral velocity distribution and reduces the neutral density accordingly but is typically numerically more expensive as it requires sorting of particles.

In addition to the correct target neutral velocity and depletion of neutral density, recent advancement in the DSMC method between particles of different species and weights has led to an order of magnitude reduction in computational run-time while keeping the same level of statistical variance for the CEX ions.[5] The traditional DSMC method for particles with different weights alters the post-collision velocity according to the probably determined by the weight ratio. In contrast, the new interspecies fractional collision method splits the particles if there is any weight difference between the collision pair, resulting in four particles per collision. The post-collision particle count can be controlled by using a "collision multiplier", which effectively splits particles before the collision is applied and results in a much larger number of particles after the collision. This idea is adapted to the plume simulation by taking the hybrid of traditional and new methods; the number of CEX ions is bumped up by splitting, while the numbers of primary ions and neutrals are kept relatively small by using the traditional method with the weight ratio probability for the post-collision velocity. Although splitting of particles leads to an unsustainable number of particles if the simulation is closed, this problem does not apply to the plume simulation with an open boundary condition for particles.

The directory `src-TURF/tutorials/Plume/Plume_2.DSMC` contains twelve files to run (1) the EP plume simulation with the DSMC CEX collision method and two additional examples demonstrating (2) how to use different collision multipliers for fast and slow ions and (3) how to split the primary and CEX ions. For each simulation, there is a pair of `world*.list` and `operations*.list` files as listed in the [table](#) below.

Table 1.42 EP plume with Fraction DSMC Collision Method

Description	Files
Interspecies Fractional DSMC	<code>world.dsmcA.list</code> , <code>operations.dsmcA.list</code>
Using different collision multipliers for fast and slow ions	<code>world.dsmcB.list</code> , <code>operations.dsmcB.list</code>
Separating primary and CEX ions	<code>world.dsmcC.list</code> , <code>operations.dsmcC.list</code>

The four other input files remain the same as in the [baseline plume simulation](#) (see the [table](#) in EP Plume Simulation 1: Baseline), while there is an additional file, `materials.list`, containing the list of available materials. The user-

defined material database file can be read by defining as follows.

```
MATERIAL_LIST = materials.list
```

If the input to MATERIAL_LIST is not specified, TURF reads the default material database file located in bin/database/materials.list.

Comparing the operations.*.list files will be helpful in seeing the modification from the baseline plume simulation to extend it with the DSMC method for the CEX collision. If Meld is set up in the system, one can use the following command in the terminal.

```
tutorial/Plume/Plume_2.DSMC> meld operations.DSMC.list ../Plume_1.Base/operations.base.list
```

1.11.2 Interspecies Fractional DSMC

The wing of the plume angular distribution is susceptible to the collision model since the initial velocity of the CEX ions has a significant effect on their trajectories. On the other hand, the collision model does not have a significant effect on the profile in the core of the plume, as long as the accurate scattering model is used (e.g. the classical scattering model with the ab initio interatomic potential). Note that the phenomenological model such as the Variable Hard Sphere (VHS) model predicts a larger population with moderately large angles in the plume profile. For this reason, only the model for the CEX collision is changed from the MCC to the DSMC method. Switching the collision model requires defining MSPDistSortTwoOp and MSPDistFDSMCCEXOp in place of MSPDistSortOp and MSPDistMCCCElasticFitOp, respectively. The operation MSPDistSortTwoOp sorts particles by the cell ID and then by the species index, while the operation MSPDistSortOp does not use the species index to sort particles. NAME of the sort operator has to be defined, so the objects in MSPDistSortTwoOp can be accessed later by MSPDistFDSMCCEXOp.

```
#####
## Sort and remove particles outside of domain ##
#####
DEFINE OPERATION
  TYPE = MSPDistSortTwoOp
  NAME = Sort2_P-DST
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-GST
  VERBOSE = FALSE
END OPERATION
```

```
# CEX Collision
DEFINE OPERATION
  TYPE = MSPDistFDSMCCEXOp
  NAME = CEXCollisionOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_ION = Xe+@g
  SPECIES_NEUTRAL = Xe@g
  SPECIES_CEX = Xe+@g
  SORT_OP_NAME = Sort2_P-DST
  SUGARCUBE_NAME = SC_UMesh
  SIGMA_COEFFS = -27.2 171.13 1.0e-20
  INIT_RELATIVE_VELOCITY = 20000.0
  COLLISION_MULTIPLIER = 8.0
  SPLIT_NEUTRAL = FALSE
  VERBOSE = FALSE
END OPERATION
```

- SIGMA_COEFFS is the coefficients to compute the total CEX cross-section given as $\sigma = (c_0 \log 10(v_{rel}) + c_1) * c_2$
- INIT_RELATIVE_VELOCITY is used to set the upper bound of $\sigma \cdot v_{rel}$ where σ is the collision cross-section and v_{rel} is the relative speed.
- COLLISION_MULTIPLIER controls the post-collision particle count by effectively splitting the particles before a collisional event.
- SPLIT_NEUTRAL sets a flag to split neutrals after collisions or simply use the probability from the computational weight ratio to decide whether to alter the post collision neutral velocity. The former conserves the moments exactly but results in the rapid growth of neutral particle count.

1.11.3 Using Different Collision Multipliers for Fast and Slow Ions

The simulation described in the [section](#) above uses the same collision multiplier values for all the CEX collisions between ions and neutrals. The CEX ions are more susceptible to collisions because of their slow speed and the larger collision cross-section, which can potentially lead to unexpected growth of the particle count. Although the above example does not lead to such a behavior, this section demonstrates a way to mitigate the effect by splitting fast and slow ions right before the application of collisions.

In order to split fast and slow ions before collisions, an additional species tag needs to be available for the particle distribution. Here, Xe+Prim is defined as an additional species for MATERIALS. The default material database file could not be used in this example, because Xe+Prim is not defined in the file. For this reason, a new database file is created to include the species Xe+Prim with the corresponding material data, and the material database file is specified by MATERIAL_LIST=materials.list. The new species also needs to be available to the particle distribution, so the species name is also included in SPECIES_NAMES of MSPDistInitOp.

The operation MSPDistSplitSpeciesOp changes the particle species index when the user-defined condition is met. In this example, the species index is changed from Xe+@g to Xe+Prim when ion speed is greater than 5,000 m/s, where the speed is set by the input to SPEED_TO_SPLIT. MSPDistSplitSpeciesOp can also take a user-defined equation with a key, CRITERION (All the functions and constants in the [3.10](#) are available). After the species tag is changed, the particle distribution needs to be sorted according to the cell ID and species index using MSPDistSortTwoOp. After the CEX collisions are performed, the fast particles need to be combined into the original particle distribution, which is again executed by MSPDistSplitSpeciesOp (SPEED_TO_SPLIT=-1 will convert all the species of Xe+Prim to Xe+@g).

```
# CEX Collision
### Split fast ions
DEFINE OPERATION
  TYPE = MSPDistSplitSpeciesOp
  MSPDIST_DATA_NAME = P-DST
  SRC_SPECIES = Xe+@g
  DST_SPECIES = Xe+Prim
  SPEED_TO_SPLIT = 5000.0
  SET_ZERO_WEIGHT = FALSE
END OPERATION
### Sort particles to prepare for DSMC collision
DEFINE OPERATION
  TYPE = MSPDistSortTwoOp
  NAME = Sort2_P-DST
  MSPDIST_SRC_NAME = P-DST
  MSPDIST_DST_NAME = P-GST
  VERBOSE = FALSE
END OPERATION
### Larger collision multiplier for fast ions
DEFINE OPERATION
  TYPE = MSPDistFDSMCCEXOp
  NAME = CEXCollisionOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_ION = Xe+Prim
  SPECIES_NEUTRAL = Xe+@g
  SPECIES_CEX = Xe+Prim
  SORT_OP_NAME = Sort2_P-DST
  SUGARCUBE_NAME = SC_UMesh
  SIGMA_COEFFS = -27.2 171.13 1.0e-20
  INIT_RELATIVE_VELOCITY = 20000.0
  COLLISION_MULTIPLIER = 8.0
  SPLIT_NEUTRAL = FALSE
  VERBOSE = FALSE
END OPERATION
### Collision multiplier of 1 for slow ions
DEFINE OPERATION
  TYPE = MSPDistFDSMCCEXOp
  NAME = CEXCollisionOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_ION = Xe+@g
  SPECIES_NEUTRAL = Xe+@g
  SPECIES_CEX = Xe+@g
  SORT_OP_NAME = Sort2_P-DST
  SUGARCUBE_NAME = SC_UMesh
  SIGMA_COEFFS = -27.2 171.13 1.0e-20
  INIT_RELATIVE_VELOCITY = 20000.0
  COLLISION_MULTIPLIER = 1.0
```

```

        SPLIT_NEUTRAL = FALSE
        VERBOSE = FALSE
    END OPERATION
    ### Change the species back to original
    DEFINE OPERATION
        TYPE = MSPDistSplitSpeciesOp
        MSPDIST_DATA_NAME = P-DST
        SRC_SPECIES = Xe+Prim
        DST_SPECIES = Xe+@g
        SPEED_TO_SPLIT = -1.0
        SET_ZERO_WEIGHT = FALSE
    END OPERATION

```

1.11.4 Separating Primary and CEX Ions

The example here also applies two different collision multipliers to ions. However, they are applied to slightly different populations, namely the primary and CEX ions instead of the fast and slow ions. Unlike the previous example, the primary and CEX ion species are completely separated throughout the simulation, enabling the computation of the volume and surface mesh fields separated by the two species. Similarly to the previous example, an additional species Xe+CEX is defined in the world file as well as the material database file. Also, the species is made available to the particle distributions by adding the species to SPECIES_NAMES of MSPDistInitOp.

After the ions are injected by MSPDistSourceRPAOp, the species tag needs to be changed from Xe+@g to Xe+←Prim using MSPDistSplitSpeciesOp in the INITIALIZE stage.

```

# Inject ions from surface
### The RPA input file uses Xe+@ as the ion name
DEFINE OPERATION
    TYPE = MSPDistSourceRPAOp
    SURFACE_NAME = UMesh
    COMPONENT_AT_SPHERE_CENTER = HCT1_INJ
    MSPDIST_DATA_NAME = P-DST
    SPECIES = Xe+@g
    INPUT_CSV_FILE_NAME = RPA_HCT.csv
    INJECTION_RADIUS = 0.3
    MASS_FLOW_RATE = 1.0e-4
    REAL_TO_COMPUTATIONAL = 1.0e13
    VERBOSE = FALSE
END OPERATION
### We want these ions to be tagged as Xe+Prim
DEFINE OPERATION
    TYPE = MSPDistSplitSpeciesOp
    MSPDIST_DATA_NAME = P-DST
    SRC_SPECIES = Xe+@g
    DST_SPECIES = Xe+Prim
    SPEED_TO_SPLIT = -1.0
    SET_ZERO_WEIGHT = FALSE
END OPERATION

```

If the total ion field data are needed, FieldArithmeticOp can be used to add two fields for the primary and CEX ions. In this example, the operation is executed in the RHO and FIELD stages for the volume and surface field data, respectively.

```

#####
## Compute Total Ion Density ##
#####
DEFINE OPERATION
    TYPE = FieldArithmeticOp
    FIELD_DATA_NAME = FieldData
    FIELD_DST_NAME = n_Xe+@g
    FIELD_SRCB_NAME = n_Xe+Prim
    FIELD_SRCC_NAME = n_Xe+CEX
    OPERATION = ADD
END OPERATION

```

```

#####
## Sum Ion Flux ##
#####
DEFINE OPERATION
    TYPE = FieldArithmeticOp

```

```

FIELD_DATA_NAME = SurfaceField
FIELD_DST_NAME = NPFLUX-IN_Xe@g
FIELD_SRCB_NAME = NPFLUX-IN_Xe+Prim
FIELD_SRCC_NAME = NPFLUX-IN_Xe+CEX
OPERATION = ADD
END OPERATION

```

Since the primary and CEX ions are treated separately, the MEX and CEX collisions are applied to the two species separately.

```

#####
## Collision Calculations ##
#####
# MEX Collision
DEFINE OPERATION
  TYPE = MSPDistMCElasticFitOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = Xe+Prim
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = n_Xe@g
  TEMPERATURE_OF_TARGET_SPECIES = 0.0
  MEX_OR_CEX = MEX
  SIGMA_COEFFS = -27.2 171.13 1.0e-20
  FIRST_FIT_EXPONENT_DIFF_SIGMA = -2.02
  SECOND_FIT_EXPONENT_DIFF_SIGMA = 3.24
  FIT_ANGLE_DIFF_SIGMA = 3.526E-5
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistMCElasticFitOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES = Xe+CEX
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = n_Xe@g
  TEMPERATURE_OF_TARGET_SPECIES = 0.0
  MEX_OR_CEX = MEX
  SIGMA_COEFFS = -27.2 171.13 1.0e-20
  FIRST_FIT_EXPONENT_DIFF_SIGMA = -2.02
  SECOND_FIT_EXPONENT_DIFF_SIGMA = 3.24
  FIT_ANGLE_DIFF_SIGMA = 3.526E-5
END OPERATION
# CEX Collision
### Larger collision multiplier for fast ions
DEFINE OPERATION
  TYPE = MSPDistFDSMCCEXOp
  NAME = CEXCollisionOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_ION = Xe+Prim
  SPECIES_NEUTRAL = Xe@g
  SPECIES_CEX = Xe+CEX
  SORT_OP_NAME = Sort2_P-DST
  SUGARCUBE_NAME = SC_UMesh
  SIGMA_COEFFS = -27.2 171.13 1.0e-20
  INIT_RELATIVE_VELOCITY = 20000.0
  COLLISION_MULTIPLIER = 8.0
  SPLIT_NEUTRAL = FALSE
  VERBOSE = FALSE
END OPERATION
### Collision multiplier of 1 for slow ions
DEFINE OPERATION
  TYPE = MSPDistFDSMCCEXOp
  NAME = CEXCollisionOp
  MSPDIST_DATA_NAME = P-DST
  SPECIES_ION = Xe+CEX
  SPECIES_NEUTRAL = Xe@g
  SPECIES_CEX = Xe+CEX
  SORT_OP_NAME = Sort2_P-DST
  SUGARCUBE_NAME = SC_UMesh
  SIGMA_COEFFS = -27.2 171.13 1.0e-20
  INIT_RELATIVE_VELOCITY = 20000.0
  COLLISION_MULTIPLIER = 1.0
  SPLIT_NEUTRAL = FALSE
  VERBOSE = FALSE
END OPERATION

```

1.12 EP Plume Simulation 3: Spacecraft Charging

Author

Samuel J. Araki

1.12.1 Introduction

A series of electric propulsion (EP) plume simulations is provided in the TURF tutorial. Starting from the [baseline plume simulation](#), advanced models are incrementally added in the order of [the DSMC collision method](#), spacecraft charging (this page), and [adaptive mesh refinement and dynamic load balancing](#). There are also tutorials on how to create a geometry file using [Cubit](#) and how to visualize field data and particles using [Paraview](#). This tutorial covers how to couple the spacecraft charging model to [the EP plume simulation with the fractional CEX DSMC method](#), which involves five steps of adding (1) new materials, (2) surface component properties, (3) more SMesh and UMesh fields, (4) current calculation, and (5) surface charging model. The description of the TURF spacecraft charging model can be found in [Modules/Charging](#) and is also published in Refs. [3] [2]

The directory `src-TURF/tutorials/Plume/Plume_3.Charging` contains seven files to run the EP plume simulation with the spacecraft charging and the fractional CEX DSMC collision. The input files for material database, geometry components, and surface interactions are modified from [the last tutorial example](#). The main two modifications in these files are related to the newly defined material `Test_Solid` and the model and parameter used specifically for the charging calculation. The changes can be seen when using text file comparison programs such as Meld. In the charging model, three models are available: (1) DIELECTRIC, (2) CONDUCTIVE, and (3) PROPAGATE. One of these three models can be defined for each geometry component by specifying in the `component_TURF.txt` with the key, `chargemodel`. While all the charging calculation can be handled by the charge propagation model (PROPAGATE), dielectric and conductive models are available to improve the computational efficiency in the limits of zero and infinite electrical conductivity. The electrical conductivity (`conductivity`) and the relative permittivity (`rel_permittivity`) are the material properties defined in the material database file, `materials.list`.

1.12.2 Changes in the World File

Comparing the world files in this directory (`world.charging.list`) and [the last tutorial](#) (`world.dsmcA.list`), the CHARGING stage is added to the list of stages and the field `T_e-` is added to `FIELDS`. Furthermore, the new material `Test_Solid` is also defined in the list of materials used in this simulation.

```

DEFINE WORLD
  NAME = plume_example
  OP_FILE = operations.charging.list
  COORDINATES = cartesian
  ORIGIN = ( 0.0, 0.0, 0.0 )
  DELTA = ( 0.25, 0.25, 0.25 )
  END_TIME = 8.00001e-3
  START_DT = 2.0e-6
  FIELDS = [ pflag, rho, rho_avg, phi, Enx, Eny, Enz ]
  FIELDS = [ VOL, n_Xe@g, n_Xe+@g, n_Al@g, T_e- ]
  MATERIALS = [ Xe@g Xe+@g Al@g Al@s Cu@s Test_Solid ]
  STAGES = [ INITIALIZE, MOVE, POSTMOVE, RHO, FIELD, CHARGING ]
  MATERIAL_LIST = materials.list
  PRINT_STATS_SKIP = 1
  PRINT_PROFILE_INFO = TRUE
  SAVE_PROFILE_INFO = TRUE
  PROFILE_FILE = profile3.txt
END WORLD

```

1.12.3 Operations for Spacecraft Charging

Comparing the operations files in this directory (`operations.charging.list`) and [the last tutorial](#) (`operations.dsmcA.list`), there are several operations that are added for the spacecraft charging calculation. The operations that are newly defined in this tutorial include `UFieldSurfaceMaxwellianCurrentOp`, `ComponentChargeImplicitOp`, `GlobalFieldPatchFillOp`, and `GlobalFieldPatchEmptyOp`. In addition, several field operations are added, which include `FieldArithmeticConstOp`, `FieldSetOp`, and `FieldArithmeticOp` to apply basic math operations to field data. Besides these operations, listed below are the

new surface field data defined in UFieldMSInitOp, in addition to the surface field variables defined in the previous table.

Table 1.43 Surface Field Variables

Field Name	Description
J_ion	Ion current density (A/m ²)
J_e-	Electron current density (A/m ²)
J_e-Old	Electron current density from the previous iteration (A/m ²).
J_Total	Total current density (A/m ²)
I_Total	Total current (A)
Q	Surface charge (C)

The electric potential is computed by inverting the Boltzmann relation, and the electron temperature is required in this model. Two temperature models can be assumed for this potential model, where the electron temperature is assumed either to be constant or to follow the polytropic relationship. The electron temperature is also used when computing the electron flux to walls, so the information needs to be stored in the container for the field data. This is done by specifying FIELD_TEMPERATURE_NAME in LogicalPotentialBoltzmannOp in the FIELD stage.

```
# Set potential inside geometries and flag Poisson region
DEFINE OPERATION
  TYPE = LogicalPotentialSetInsideGeometryOp
  SUGARCUBE_NAME = SC_UMesh
  FIELD_DATA_NAME = FieldData
  FIELD_CHARGE_DENSITY_NAME = rho_avg #rho
  FIELD_POTENTIAL_NAME = phi
  POTENTIAL_MODEL = BOLTZMANN
  TEMPERATURE_MODEL = POLY # CONST
  REFERENCE_GAMMA = 1.3
  REFERENCE_ELECTRON_DENSITY = 3.11135E17
  REFERENCE_ELECTRON_TEMPERATURE_EV = 8.01663
  REFERENCE_POTENTIAL = 35.7182
  ADD_FLAG_TO_FIELD = TRUE
  FIELD_POTENTIAL_FLAG_NAME = pflag
  CALCULATE_BOUNDARY_FIELD = TRUE
END OPERATION

# Compute potential using Boltzmann relations
DEFINE OPERATION
  TYPE = LogicalPotentialBoltzmannOp
  FIELD_DATA_NAME = FieldData
  FIELD_CHARGE_DENSITY_NAME = rho
  FIELD_POTENTIAL_NAME = phi
  FIELD_TEMPERATURE_NAME = T_e-
END OPERATION
```

Towards the end of the FIELD stage, ion flux to walls computed by UFieldMSComputeOp is converted to the current density value by FieldArithmeticConstOp. Also, the electron current density is calculated assuming the sheath is thin and electrons are Maxwellian using UFieldSurfaceMaxwellianCurrentOp.

```
#####
## Sum to Fields for Output ##
#####
# Compute UField quantities
# *** This needs to be after surface interaction
DEFINE OPERATION
  TYPE = UFieldMSComputeOp
  UFIELD_DATA_NAME = SurfaceField
  SURF_INTERACTION_OP_NAME = Interaction_P-GST
  MIX_START_TIME = 2.0e-3
  STEPS_PER_FIELD_CALCULATION = 100
END OPERATION

#####
## Ion Current Density ##
#####
# Convert to ion current
DEFINE OPERATION
```

```

        TYPE = FieldArithmeticConstOp
        FIELD_DATA_NAME = SurfaceField
        FIELD_SRC_NAME = NPFLUX-IN_Xe+@g
        FIELD_DST_NAME = J_Ion
        CONSTANT_NAME = MKS_e
        OPERATION = MUL
    END OPERATION
#####
## Electron Current Density ##
#####
# Store J_e- from last iteration
DEFINE OPERATION
    TYPE = FieldSetOp
    FIELD_DATA_NAME = SurfaceField
    FIELD_NAME = J_e-Old
    FIELD_SRC_NAME = J_e-
    OPERATION = SET
END OPERATION
# Calculate J_e- assuming Maxwellian
DEFINE OPERATION
    TYPE = UFieldSurfaceMaxwellianCurrentOp
    SURFACE_NAME = UMesh
    SUGARCUBE_NAME = SC_UMesh
    UFIELD_DATA_NAME = SurfaceField
    UFIELD_CURRENT_DENSITY_NAME = J_e-
    UFIELD_POTENTIAL_NAME = potIn
    FIELD_DATA_NAME = FieldData
    FIELD_ELECTRON_TEMPERATURE_EV_NAME = T_e-
    FIELD_CHARGE_DENSITY_NAME = rho_avg
    FIELD_POTENTIAL_NAME = phi
    VERBOSE = TRUE
END OPERATION

```

After the current densities of the two species are computed, they are combined and converted to total current by FieldArithmeticOp.

```

#####
## Total Current ##
#####
# Sum currents
DEFINE OPERATION
    TYPE = FieldArithmeticOp
    FIELD_DATA_NAME = SurfaceField
    FIELD_SRCB_NAME = J_Ion
    FIELD_SRCC_NAME = J_e-
    FIELD_DST_NAME = J_Total
    OPERATION = ADD
    USE_VOLUME = FALSE
END OPERATION
# Multiply current density by element area to get current
DEFINE OPERATION
    TYPE = FieldArithmeticOp
    FIELD_DATA_NAME = SurfaceField
    FIELD_SRC_NAME = J_Total
    FIELD_DST_NAME = I_Total
    OPERATION = MUL
    USE_VOLUME = TRUE
END OPERATION

```

In the current version of TURF, copies of the entire surface field data are owned by all the active domains, instead of partitioning sections of the data. In other words, the surface field calculation cannot be parallelized using the domain decomposition strategy (this is not planned in the future versions unless there is a strong need). Nevertheless, the simulation domain is partitioned, so the current densities are only computed for the section of the surface mesh immersed in the sub-domain. Therefore, the surface field data needs to be unified across all the sub-domains before performing the spacecraft charging calculation, which is done by two operations, GlobalFieldPatchFillOp and GlobalFieldPatchEmptyOp. The operation GlobalFieldPatchFillOp copies the field data into a buffer and then the data in the buffer for each sub-domain are unified between the two stages, FIELD and CHARGING. Once the communication is completed, the operation GlobalFieldPatchEmptyOp copies the data back into the field data.

```

#####
## Patch I_total ##
#####

```

```

DEFINE OPERATION
  TYPE = GlobalFieldPatchFillOp
  NAME = Patch_Current
  FIELD_DATA_NAME = SurfaceField
  FIELD_NAMES = I_Total
  PATCH_OP = SUM
END OPERATION
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE FIELD
#####
DEFINE STAGE CHARGING
#####
## Extract Patched I_total Data ##
#####
DEFINE OPERATION
  TYPE = GlobalFieldPatchEmptyOp
  FIELD_DATA_NAME = SurfaceField
  PATCH_FILL_OP_NAME = Patch_Current
END OPERATION

```

The charging calculation is performed by only the first active sub-domain. The details of the charging model are provided in [Modules/Charging](#) and are also published in Refs.[3] [2]. It is important to note that the time-step in the charging calculation (TOTAL_TIME_PER_APPLY) can be chosen to be different from the global time-step defined in the world file ('START_DT'). This is because the time-scale for the charging can be wildly different from that for the plume simulation, leading to an unstable simulation behavior. Although the time-accurate solution cannot be obtained with different time-step sizes, it has been shown that the correct steady-state solution can still be obtained (Even with the same time-step sizes, the solution is unlikely time-accurate because of assumptions typically used for capacitance values).

```

#####
## Surface Charging ##
#####
# Only performed by the first active sub-domain
DEFINE OPERATION
  TYPE = ComponentChargeImplicitOp
  UFIELD_DATA_NAME = SurfaceField
  UFIELD_POTENTIAL_NAME = potIn
  UFIELD_CURRENT_NAME = I_Total
  UFIELD_CHARGE_NAME = Q
  BUS_CAPACITANCE = 1.0e-5
  TOTAL_TIME_PER_APPLY = 1.0e-8
  STEPS_PER_ITERATION = 1
#   TIME_BETWEEN_APPLY = 500.0e-6
  START_TIME = 2.0e-3
  VERBOSE = TRUE
END OPERATION

```

- TOTAL_TIME_PER_APPLY is the total time that the charges are advanced per iteration.
- STEPS_PER_ITERATION is the number of sub-steps in each iteration for the charge propagation model.
- TIME_BETWEEN_APPLY is the global time between the charging calculations. This allows for the plume solution to reach steady-state every time the surface potential is updated in this operation.
- START_TIME sets the time to start the charging calculation.

Finally, going back to the earlier stages, the potential solution computed by the charging model needs to be copied across all the active sub-domains.

```

#####
## Patch Surface Potential ##
#####
DEFINE OPERATION
  TYPE = GlobalFieldPatchFillOp
  NAME = Patch_Potential
  FIELD_DATA_NAME = SurfaceField
  FIELD_NAMES = potIn
  PATCH_OP = SET
END OPERATION

```

```
#####  
## Proceed to the Next Stage ##  
#####  
DEFINE OPERATION  
    TYPE = NextStageOp  
END OPERATION  
END STAGE INITIALIZE  
#####  
DEFINE STAGE MOVE  
#####  
## Extract Patched Surface Potential ##  
#####  
DEFINE OPERATION  
    TYPE = GlobalFieldPatchEmptyOp  
    FIELD_DATA_NAME = SurfaceField  
    PATCH_FILL_OP_NAME = Patch_Potential  
END OPERATION
```

1.13 EP Plume Simulation 4: AMR and Dynamic Load Balancing

Author

Samuel J. Araki

1.13.1 Introduction

A series of electric propulsion (EP) plume simulations is provided in the TURF tutorial. Starting from the [baseline plume simulation](#), advanced models are incrementally added in the order of [the DSMC collision method](#), [spacecraft charging](#), and adaptive mesh refinement and dynamic load balancing (this page). There are also tutorials on how to create geometry file using [Cubit](#) and how to visualize field data and particles using [Paraview](#). This tutorial covers how to convert the uniform mesh to the Cartesian mesh with adaptive mesh refinement (AMR) for the EP plume simulation, basing off of [the EP plume simulation with the spacecraft charging](#). There is a separate [tutorial on the AMR mesh](#), which provides additional information not covered in this tutorial.

The directory `src-TURF/tutorials/Plume/Plume_4.AMR` contains eleven files to run three examples as listed in the [table](#) below.

Table 1.44 EP plume with AMR and dynamic load balancing

Description	Files
Using AMR in EP plume simulation with spacecraft charging	<code>world.AMR.list</code> , <code>operations.AMR.list</code>
Reordering operations to update fields after mesh refinement	<code>world.AMR2.list</code> , <code>operations.AMR2.list</code>
Dynamic load balancing	<code>world.LB.list</code> , <code>operations.LB.list</code>

In order to use the AMR capability in the EP plume simulation with spacecraft charging, two lines need to be modified in the world list. First, the coordinate system is changed from Cartesian to AMR. Then, another stage `MESH` is added in the beginning of the stage list, which is exclusively for the mesh refinement operation.

```

DEFINE WORLD
  NAME = plume_example
  OP_FILE = operations.AMR.list
  COORDINATES = AMR
  ORIGIN = ( 0.0, 0.0, 0.0 )
  DELTA = ( 0.25, 0.25, 0.25 )
  END_TIME = 8.00001e-3
  START_DT = 2.0e-6
  FIELDS = [ pflag, rho, rho_avg, phi, Enx, Eny, Enz ]
  FIELDS = [ VOL, n_Xe@g, n_Xe+@g, n_Al@g, T_e- ]
  MATERIALS = [ Xe@g Xe+@g Al@g Al@s Cu@s Test_Solid ]
  STAGES = [ MESH, INITIALIZE, MOVE, POSTMOVE, RHO, FIELD, CHARGING ]
  MATERIAL_LIST = materials.list
  PRINT_STATS_SKIP = 1
  PRINT_PROFILE_INFO = TRUE
  SAVE_PROFILE_INFO = TRUE
  PROFILE_FILE = profile4A.txt
END WORLD

```

Many of the operations are upgraded to work with the uniform mesh and the AMR mesh, while some of the operations can only work with either type of mesh. These specific operations are listed below. If the uniform mesh is defined in the world but the AMR specific operations are called, the operators will call the appropriate operations that work for the uniform mesh. This means that, once the simulation with the AMR mesh is set up, the simulation can be switched to use the uniform mesh by only modifying the coordinate system in the world file. Note that there is no AMR version of the 2D plotter (`LogicalFieldWriteVTK2DOp`).

Table 1.45 Mesh specific operations

Uniform	AMR
<code>LogicalMeshSurfaceSugarcubeOp</code>	<code>LogicalASMeshSurfaceSugarcubeOp</code>
<code>MSPDistSampleOp</code>	<code>MSPDistASampleOp</code>
<code>SampleFieldWriteVTKOp</code>	<code>SampleAMRFieldWriteVTKOp</code>

SampleFieldReadVTKOp	SampleAMRFieldReadVTKOp
LogicalFieldWriteVTKOp	LogicalAMRFieldWriteVTKOp

1.13.2 Converting the Uniform Cartesian Mesh to the Cartesian Mesh with AMR

TURF creates a uniform mesh before initializing each operator, and this default mesh needs to be replaced with the new AMR mesh object (called `ASMesh`, i.e. adaptive structured mesh). The operation `SMeshCreateOp` creates another uniform mesh but with the `ASMesh` object during the operator initialization. Then, the mesh refinement is performed in the beginning of specified iterations. The input keys are explained in the [table](#) in the [tutorial on the AMR mesh](#).

```

DEFINE STAGE MESH
#####
## Adaptive Cartesian Mesh ##
#####
### Create Mesh
DEFINE OPERATION
  TYPE = SMeshCreateOp
  SMESH_NAME = SMesh
  SUGARCUBE_NAME = SC_UMesh
  CRITERION = (x>=(-0.25) && x<=1.25 && y>=(-0.5) && y<=1.0 && z>=1.0 && z<2.5) || (io>=1 && io<=11 &&
    refine*lv==0) || ( {n_Al@g}>1.0e12 && x<=(-0.5) && y>=(-0.75) && y<=0.75 && z>1.25 && t>=2e-3 &&
    refine*lv==0)
  REFINEMENT_LEVEL = 2
  FILL_RATIO = 0.05
  SKIP = 200
  TIME_STOP = 3.0e-3
  VERBOSE = TRUE
END OPERATION
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE MESH

```

Right after the MESH stage, the field data object compatible with the `ASMesh` is created by `LogicalFieldInitOp`, and the default field object is replaced with this object. A new field `io` is defined in this simulation that contains the sugarcube data used in the mesh refinement criteria.

```

#####
## Replace the Default SMesh Field with AMR Field Data ##
#####
DEFINE OPERATION
  TYPE = LogicalFieldInitOp
  FIELD_DATA_NAME = FieldData
  SMESH_NAME = SMesh
  FIELD_DOF1_NAMES = n
  DOF1_NAMES = Xe@g Xe+@g Al@g
  DOF_TYPES = SPECIES
  FIELD_NO_DOF_NAMES = pflag, rho, rho_avg, phi, Enx, Eny, Enz, VOL, T_e-, io
  VERBOSE = TRUE
END OPERATION

```

The operation `FieldCopyKMatrixOp` copies the data stored in the array `in_or_out` in the sugarcube object `SC_UMesh`. The array `in_or_out` contains integers that are zero if the cells are in free-space, positive if the cells intersect the surface mesh, and negative if the cells are inside the geometry.

```

#####
## Copy Array Data To Field ##
#####
DEFINE OPERATION
  TYPE = FieldCopyKMatrixOp
  FIELD_DATA_NAME = FieldData
  FIELD_DST_NAME = io
  GSOBJECT_NAME = SC_UMesh
  KMATRIX_NAME = in_or_out
  INTERIOR_CELLS_ONLY = FALSE

```

```
END OPERATION
```

Unlike [the previous tutorial](#), the operation `MSPDistCellIDOp` is added in the `INITIALIZE` stage. When the mesh refinement is performed, the cell ID assigned to each particle needs to be updated. Otherwise, the mismatch of particle cell ID with the field data leads to incorrect electric field in `MSPDistESPushOp`.

```
#####
## Update CellID in Case Mesh Has Changed ##
#####
DEFINE OPERATION
  TYPE = MSPDistCellIDOp
  MSPDIST_DATA_NAME = P-DST
END OPERATION
```

For the charge deposition, the same operation `MSPDistChargeDepositionOp` as the uniform mesh case is used. However, the current version with `ASMesh` only supports the zeroth order weighting. The first order weighting at the mesh interface with different mesh resolution is subject to future research.

```
DEFINE OPERATION
  TYPE = MSPDistChargeDepositionOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  MSPDIST_DATA_NAME = P-DST
  ORDER = 0 # AMR currently only supports zeroth order weighting
END OPERATION
```

The input keys for the `ASMesh VTK writer LogicalAMRFieldWriteVTKOp` are similar to the one for the uniform mesh `LogicalFieldWriteVTKOp`. The only new input key is `LEAF_ONLY`, which flags whether to output the parent cell data. Outputting the data can be useful in debugging, which can be enhanced by using the "Threshold" operation in Paraview with a specified refinement level.

```
#####
## Write VTK Output ##
#####
# 3D data
DEFINE OPERATION
  TYPE = LogicalAMRFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = Plot3D4A/plt_
  FIELD_NAMES = VOL pflag rho rho_avg phi n_Xe@g n_Xe+@g n_Al@g T_e-
  FIELD_NODE_NAMES = Enx Eny Enz
  SUGARCUBE_NAME = SC_UMesh
  SKIP = 100
  PLOT_GHOST = FALSE
  LEAF_ONLY = TRUE
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
```

The restart of a simulation is also possible using `ASMesh` just like [the restart for the uniform mesh](#). In addition to the particle data and the volume mesh field data, the mesh needs to be recovered from the simulation before the restart, which is performed by the mesh data written by `LogicalAMRFieldWriteVTKOp`. All the cell data need to be written out, which include the ghost and parent cells, enabled by setting `PLOT_GHOST` and `LEAF_ONLY` as `True` and `False`, respectively. If the value for `SKIP` is set to a smaller number, the output will be more frequent but will slow down the simulation.

```
# Mesh restart file
DEFINE OPERATION
  TYPE = LogicalAMRFieldWriteVTKOp
  FIELD_DATA_NAME = FieldData
  FILE_HEAD = Restart/plt_
  SKIP = 100000 # The file will be written at the end of simulation
  PLOT_GHOST = TRUE # Need to include ghost for restart
  LEAF_ONLY = FALSE # Need to include non-leaf cells for restart
  REMOVE_DIR_AT_INIT = TRUE
END OPERATION
```

In order to read the mesh data, the operation `ASMeshReadVTKOp` is required, which can be placed after the operation `LogicalFieldInitOp`.

```
#####
## Read Mesh and Update ##
#####
DEFINE OPERATION
  TYPE = ASMeshReadVTKOp
  SMESH_NAME = SMesh
  FILE_HEAD_PVTP = Restart/plt_04001
  MESH_ARRAY_LENGTH_RATIO = 1.0
  UPDATE_WORLD_TIME = TRUE
  ADVANCE_TIME_BY_DT = FALSE
END OPERATION
```

- MESH_ARRAY_LENGTH_RATIO sets the size of arrays containing the mesh data. The value of 1.0 will allocate arrays with the length exactly the same as in the restart file.
- UPDATE_WORLD_TIME updates the simulation time to the value in the restart file if this is set to True. ADVANCE_TIME_BY_DT increases the world time by one time-step if this is set to True.

1.13.3 Reordering Operations to Update Field Data after Mesh Refinement

In this example, the calculations of field data such as the charge density, the electric potential, and the electric field are moved towards the beginning of the operations file. This will ensure that the correct field data are used after the mesh refinement is performed, otherwise the field data will be misaligned with the cell data until they are recalculated. Since the mesh refinement is not performed frequently, reordering the operations might not have a significant impact to the results. The block of operations shown below is moved right after MSPDistInitOp. Also, the stage names and order are changed in this case. The text file comparison program will help visualize the modification from [the simulation above](#).

```
tutorial/Plume/Plume_4.AMR> meld operations.AMR.list operations.AMR2.list
```

```
#####
## Deposit Charge for Potential Calculation ##
#####
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  VALUE = 0.0
  OPERATION = SET
END OPERATION
DEFINE OPERATION
  TYPE = MSPDistChargeDepositionOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  MSPDIST_DATA_NAME = P-DST
  ORDER = 0 # AMR currently only supports zeroth order weighting
END OPERATION
# Divide by cell volume to get charge density
DEFINE OPERATION
  TYPE = FieldArithmeticOp
  FIELD_DATA_NAME = FieldData
  FIELD_SRCB_NAME = rho
  FIELD_SRCC_NAME = VOL
  FIELD_DST_NAME = rho
  OPERATION = DIV
END OPERATION
#####
## Patch Rho before Boltzmann Solve ##
#####
DEFINE OPERATION
  TYPE = LogicalFieldPatchOp
  FIELD_NAMES = rho
  PATCH_OPS = PATCH_SUM
END OPERATION
#####
## Proceed to the Next Stage ##
#####
DEFINE OPERATION
  TYPE = NextStageOp
END OPERATION
END STAGE RHO
#####
DEFINE STAGE FIELD
```

```

#####
## Stop Patching ##
#####
DEFINE OPERATION
  TYPE = LogicalFieldPatchOp
  FIELD_NAMES = rho
  PATCH_OPS = PATCH_NULL
END OPERATION
#####
## Add Charge Distributed at Ghost to Interior ##
#####
### Not needed because of zeroth order weighting
#DEFINE OPERATION
#  TYPE = LogicalBCMirrorMoveInOp
#  FIELD_DATA_NAME = FieldData
#  FIELD_NAME = rho
#END OPERATION
#####
## Potential Solve via Boltzmann Inversion ##
#####
# Adding background density for Boltzmann solve
DEFINE OPERATION
  TYPE = FieldSetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = rho
  VALUE = 1.602e-7 # Minimum density
  OPERATION = FLOOR
END OPERATION
# Time average rho after TIME_START_AVERAGING
# *** rho_avg can be used for later calculations or just for plotting
DEFINE OPERATION
  TYPE = FieldBlendTimeOp
  FIELD_DATA_NAME = FieldData
  FIELD_SRC_NAME = rho
  FIELD_DST_NAME = rho_avg
  OPERATION = AVERAGE
  TIME_START_BLEND = 2.0e-3
END OPERATION
# Set potential inside geometries and flag poisson region
DEFINE OPERATION
  TYPE = LogicalPotentialSetInsideGeometryOp
  SUGARCUBE_NAME = SC_UMesh
  FIELD_DATA_NAME = FieldData
  FIELD_CHARGE_DENSITY_NAME = rho_avg #rho
  FIELD_POTENTIAL_NAME = phi
  POTENTIAL_MODEL = BOLTZMANN
  TEMPERATURE_MODEL = POLY # CONST
  REFERENCE_GAMMA = 1.3
  REFERENCE_ELECTRON_DENSITY = 3.11135E17
  REFERENCE_ELECTRON_TEMPERATURE_EV = 8.01663
  REFERENCE_POTENTIAL = 35.7182
  ADD_FLAG_TO_FIELD = TRUE
  FIELD_POTENTIAL_FLAG_NAME = pflag
  CALCULATE_BOUNDARY_FIELD = TRUE
END OPERATION
# Compute potential using Boltzmann relations
DEFINE OPERATION
  TYPE = LogicalPotentialBoltzmannOp
  FIELD_DATA_NAME = FieldData
  FIELD_CHARGE_DENSITY_NAME = rho
  FIELD_POTENTIAL_NAME = phi
  FIELD_TEMPERATURE_NAME = T_e-
END OPERATION
# Reset geometry
DEFINE OPERATION
  TYPE = LogicalFieldGeometrySetOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAMES = phi
  SURFACE_NAME = UMesh
  SUGARCUBE_NAME = SC_UMesh
  SET_OPTION = FREE
  SKIP_BOUNDARY = FALSE
END OPERATION
# Copy Potential to Ghost Cells
DEFINE OPERATION
  TYPE = LogicalBCOp
  FIELD_NAMES = phi
  BOUNDARY_TYPE = FLAT
END OPERATION
#####

```

```

## Compute Electric Field ##
#####
DEFINE OPERATION
  TYPE = LogicalNodeGradientOp
  FIELD_DATA_NAME = FieldData
  FIELD_POTENTIAL_NAME = phi
  FIELD_GRADIENT_PREFIX = En
  FIELD_MULTIPLY_CONSTANT = -1.0
  FIELD_GRADIENT_DIRECTIONS = [x, y, z]
  SUGARCUBE_NAME = SC_UMesh
END OPERATION

```

1.13.4 Dynamic Load Balancing

Enabling the dynamic load balancing requires an addition of one operator, `LoadBalanceDomainOp`. The load balancing can also be performed using `LoadBalanceDomainZoltanOp` together with the Trilinos Zoltan library (Trilinos needs to be enabled during `cmake`). The current version only uses the number of particles in each sub-domain as the weight to determine the load.

```

#####
## Load Balance ##
#####
DEFINE OPERATION
  TYPE = LoadBalanceDomainOp
#   TYPE = LoadBalanceDomainZoltanOp
  MSPDIST_DATA_NAME = P-DST
  WEIGHT_EMPTY_DOMAIN = 10.0
  SKIP = 100
  SHIFT = 1
  THRESHOLD = 0.05
  VERBOSE = TRUE
END OPERATION

```

- `WEIGHT_EMPTY_DOMAIN` is the minimum weight per sub-domain used for load balancing.
- `SKIP` sets the frequency of load balancing.
- `SHIFT` shifts when the load balancing is performed, which is executed when $(\text{ITERATION} + \text{SHIFT}) \text{SKIP} == 0$. This allows for offset of iterations when mesh refinement and dynamic load balancing are performed.
- `THRESHOLD` is the threshold when the load balancing is performed. More specifically, it is executed if the equation below is satisfied.

$$\frac{\max(N_r) - \min(N_r)}{\max(N_d) - \min(N_d)} \geq \text{THRESHOLD} \quad (1.6)$$

where N_r and N_d are the number of particles per rank and sub-domain, respectively.

In addition to the `LoadBalanceDomainOp`, the operation `FieldSetDomainOp` can copy the domain ID, MPI rank, and total number of particles in the sub-domains to the field data, which is only used for visualization.

```

#####
## Store DomID, Rank, and Particle Count for Visualization ##
#####
DEFINE OPERATION
  TYPE = FieldSetDomainOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = domID
  PARAMETER = DOMID
END OPERATION
DEFINE OPERATION
  TYPE = FieldSetDomainOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = Rank
  PARAMETER = Rank
END OPERATION
DEFINE OPERATION
  TYPE = FieldSetDomainOp
  FIELD_DATA_NAME = FieldData
  FIELD_NAME = NPart
  PARAMETER = Particle

```

```
MSPDIST_DATA_NAME = P-DST  
END OPERATION
```

1.14 EP Plume Simulation: Cubit

Author

Kari A. Kawashima

1.14.1 Introduction

A series of electric propulsion (EP) plume simulations is provided in the TURF tutorial. Starting from the [baseline plume simulation](#), advanced models are incrementally added in the order of [the DSMC collision method](#), [spacecraft charging](#), and [adaptive mesh refinement and dynamic load balancing](#). There are also tutorials on how to create geometry file using Cubit (this page) and how to visualize field data and particles using [Paraview](#).

This tutorial serves as a guide to creating and exporting a mesh geometry of a spacecraft to be used in a simulation. TURF requires a triangulated surface mesh in order to handle spacecraft geometries; while TURF can handle several different formats including abaqus, exodus, gambit, and off, only the abaqus format (.inp filetype) is fully supported in the current version. The abaqus format file can be exported from a CAD meshing program such as Cubit [12], which is used for the purposes of this example.

First, a general overview of Cubit's functions and capabilities will be explained in the context of creating the initial CAD model. Depending on the kind of thruster and injection source, additional steps may be necessary to create an accurate geometry. Three generic spacecraft CAD files are provided in `tutorial/Plume/CubitExample/` to give the user optional geometries to start with. Then, the surface mesh will be created, grouped, and named appropriately, and the file will be ready to be exported for use in a simulation.

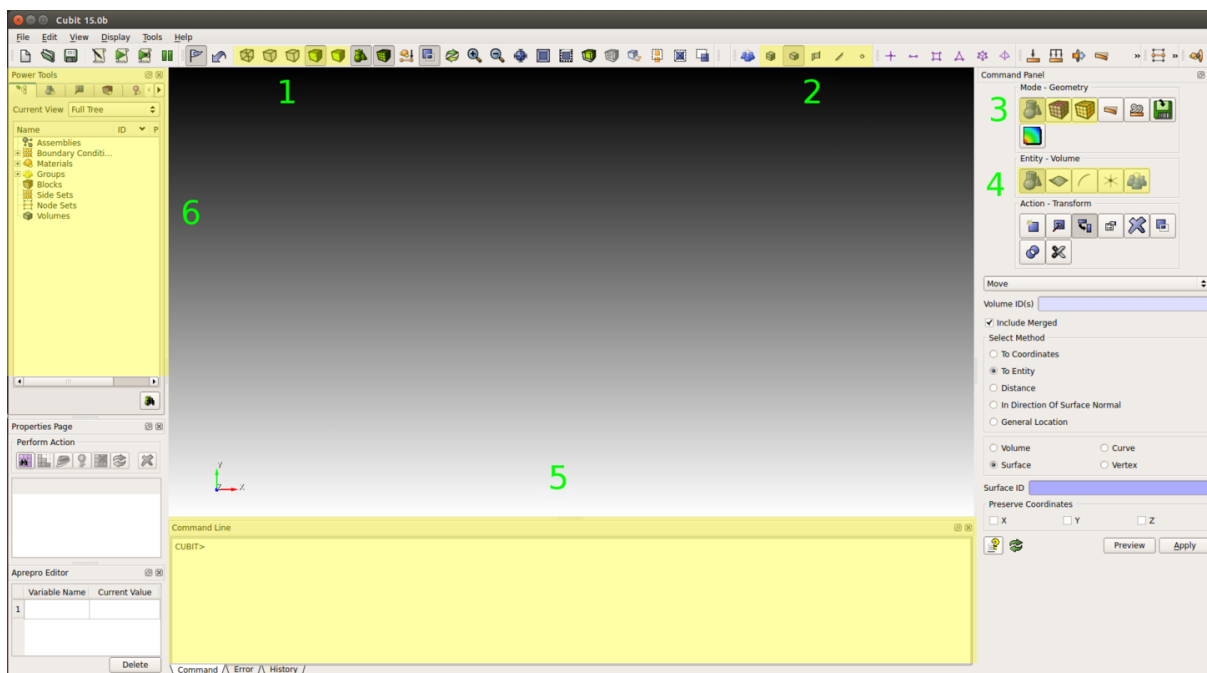


Figure 1.18 The main Cubit interface

1.14.2 General Geometry

Cubit's interface layout is pictured in the [Cubit interface](#):

- Area 1 is a set of visual settings buttons that change the transparency and presentation of volumes.
- Area 2 determines which kinds of entities can be selected when clicking on the geometry: volumes, surfaces, curves, or vertices.

- Area 3 highlights the three management modes used for this tutorial: geometry, mesh, and blocks.
- Area 4 is similar in function to those in Area 2, but they are specific to the creation and manipulation of specific entities.
- Area 5 contains the command line, error, and command history and is located bottom of Cubit's interface.
- Area 6 is the tree summarizing the entities in the Power Tools toolbar on the left side of the main window.

To make a new volume, select the Volume management buttons in Areas 3 and 4, and then click the first option "Create volumes." Options for the kind of volume to be created appear in a dropdown menu.

Manipulating an entity uses the Transform button under the Action toolbar. There are several particularly useful commands that will help make the desired geometry more quickly and accurately. One command is "copy," found as the last option under "create volumes" list, which allows the user to make a duplicate of selected volumes and move, rotate, etc., the copy without displacing the original volume. Another command is "reflect," found in the "transform" list, which moves an entity to its mirror position across a plane.

Also useful are the functions under the "webcutting operations" tab, which allow you to use entities to cut a volume into pieces. For example, the [Chop Figure](#), depicts one face of a cube "chopping" a hollow sphere into two half-dome volumes shown in the [Dome figure](#). This is done by selecting the sphere as the "Webcut Target" and the cube as the "Cutting Tool," leaving separate halves of the sphere.

Note as well that commands can be entered in the command line directly. For the sake of accuracy, commands such as "Measure between surface 3 5" will respond with the measured distance between surface 3 and surface 5, and the graphics display will show a red 3D arrow to indicate which measurement was taken.

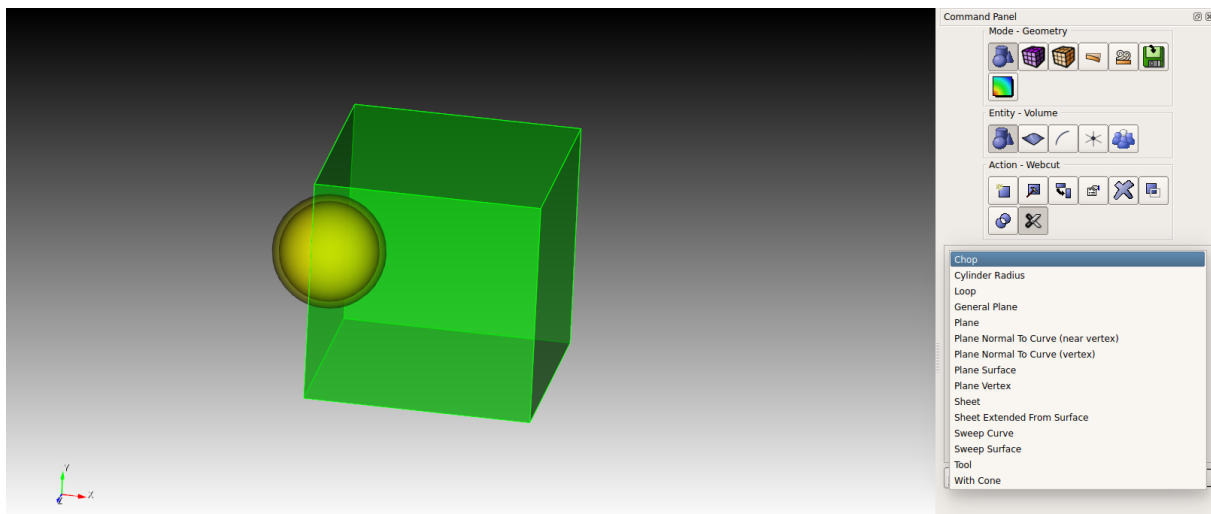


Figure 1.19 Sphere and cube to be used in the "chop" function

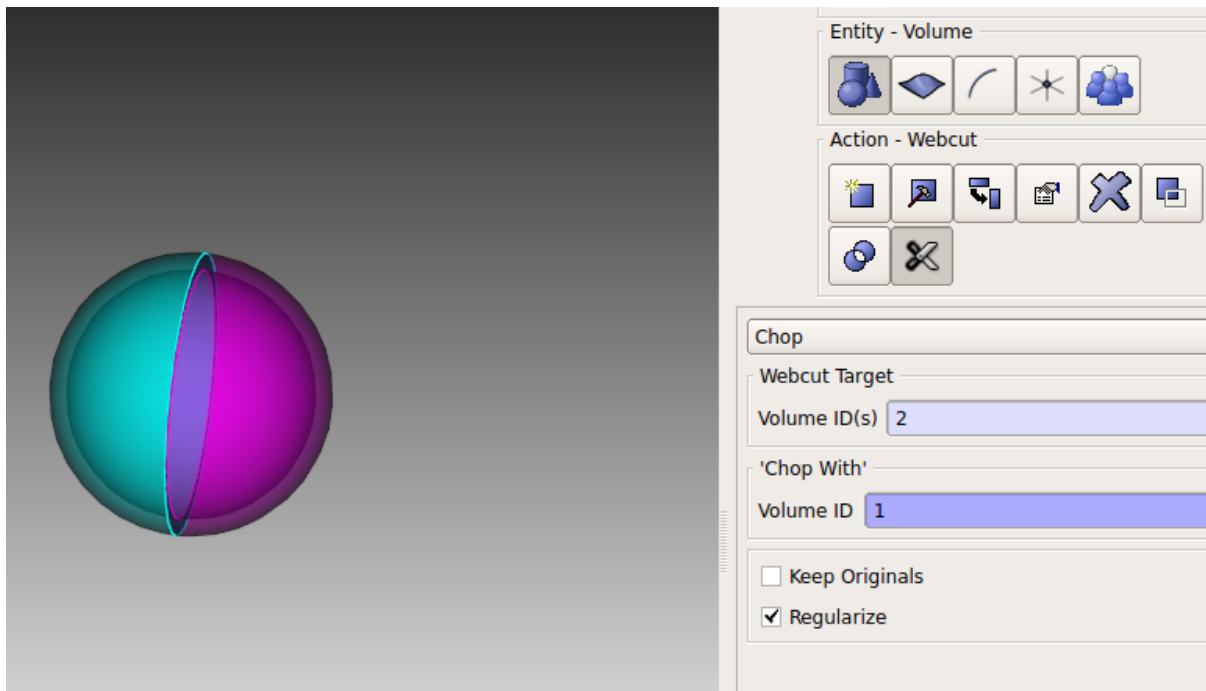


Figure 1.20 The halves resulting from the chop operation shown in the above figure

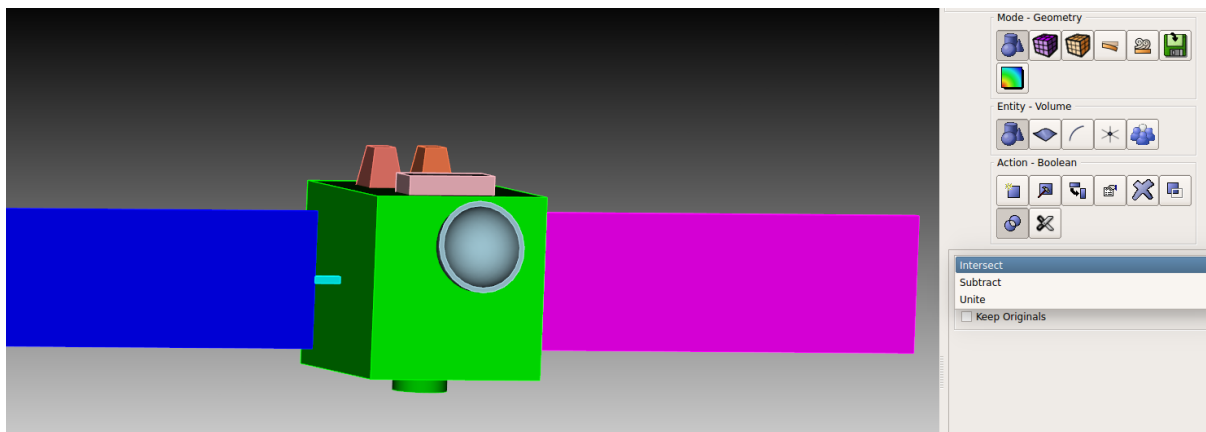


Figure 1.21 Satellite geometry before volumes are united

Once the desired satellite geometry has been created just as the one shown in the [Unite figure](#), all the shapes must be combined to make a single volume entity and eliminate overlapping parts. The “Unite” function is under the “boolean operations for volumes” tab, as shown in the [Unite figure](#). Type “all” into the Volume ID box. The result is a single volume, indicated by a change of the entire geometry to be the same color (unless there are unattached volumes floating in space). The united spacecraft geometry shown in the [United satellite geometry figure](#) has been saved as `example1.cub` for the user’s convenience, and can be found in `tutorial/Cubit/CubitExample/`.

If the geometry must be altered after the volumes have been united, surfaces must be separated from the rest of the geometry so the changes can be made. “Separate” is found under the Modify toolbar for Surface selection. The chosen

surfaces will form a separate “Sheet Body” volume that can be manipulated in the same manner as a normal volume.

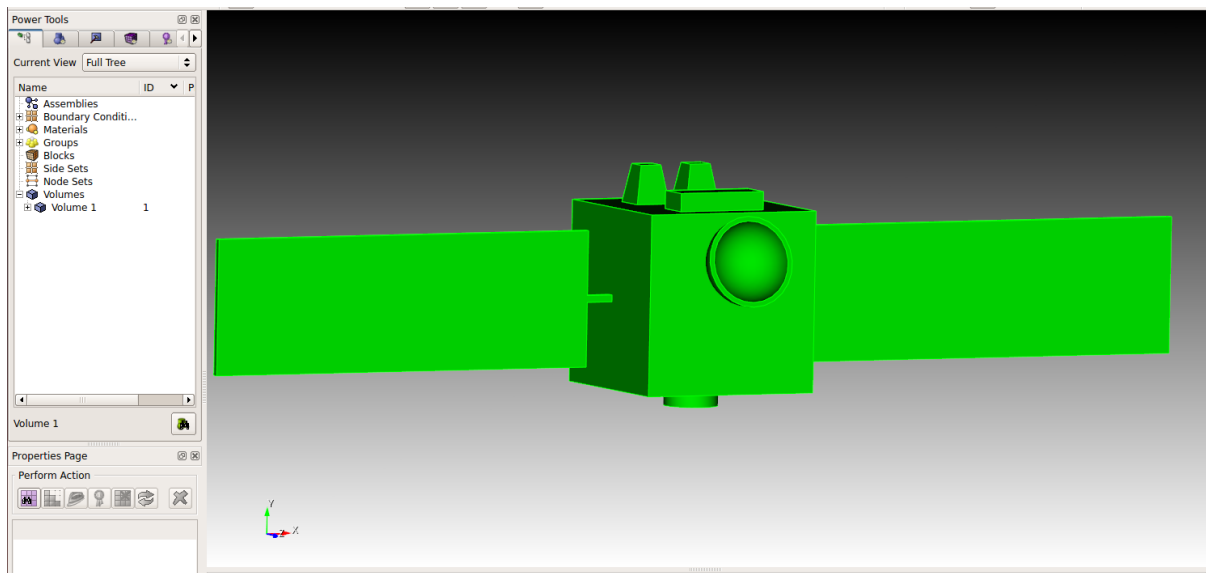


Figure 1.22 United satellite geometry. Note the tree in the sidebar lists only one volume.

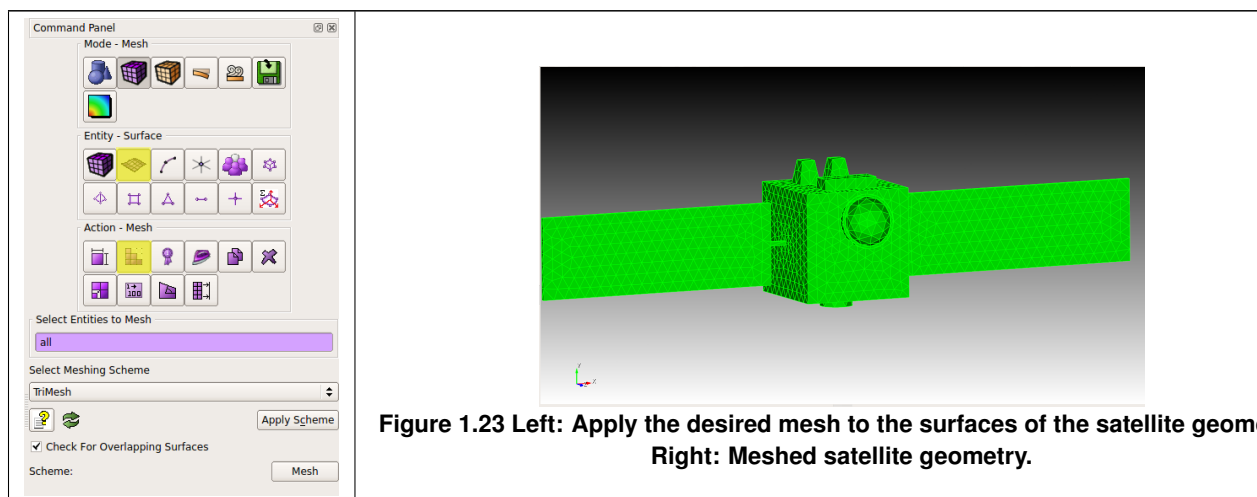


Figure 1.23 Left: Apply the desired mesh to the surfaces of the satellite geometry. Right: Meshed satellite geometry.

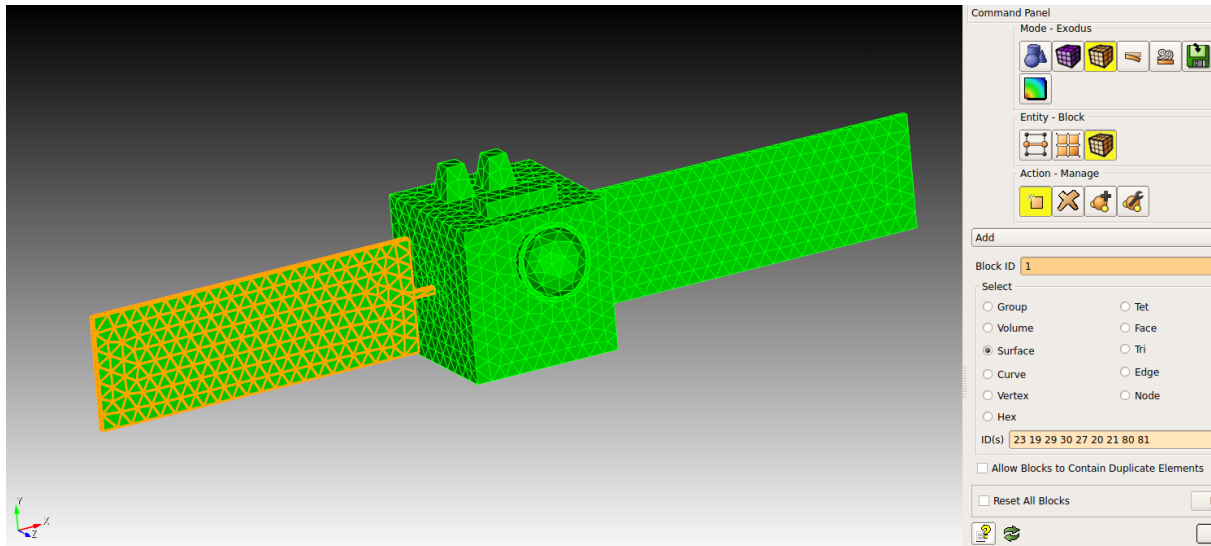


Figure 1.24 The surfaces of one solar array are grouped into a block.

1.14.3 Mesh

After the satellite geometry is properly united, its surfaces can be meshed. Select the “Specify meshing schemes and attributes” function under Mesh Mode, as shown in the [meshed geometry figure](#) (left) and change the meshing scheme from “Automatic” to “Trimesh” for best results. Tell Cubit to mesh all surfaces, click “Apply Scheme”, then hit “Mesh.” The results should resemble the geometry shown in the [Mesh figure](#) (right).

It is important to adjust the mesh resolution in order to maximize the TURF simulation efficiency. Too coarse surface mesh does not represent the curved surfaces accurately, while increasing the resolution will require more triangles to scan through in determining particle-surface intersections in TURF. The mesh resolution can be adjusted in Action “Define intervals and sizes.”

1.14.4 Blocks

When all surfaces of the spacecraft geometry have been meshed, they must be organized into groups and named so they can be specified in TURF’s input files. Cubit refers to such groups as Blocks. To create blocks, select the Exodus Mode in the command panel, select Block, and choose the Add function under the Manage operation. Give the new block an ID number not already in use and select the surfaces desired for the block. Group surfaces into blocks in a logical way (e.g. the surfaces of a solar array, the main body of the spacecraft, etc.), as shown in the [block figure](#). When the block is created, change the name of the block in the component tree (Area in [Cubit interface](#)) to describe the part (e.g. *SatelliteDish*). TURF’s input files refer to these block names, so make them distinct and straightforward.

When creating blocks for the spacecraft, make separate blocks for the thruster’s face and body. Doing so allows them to be assigned separate material properties for a simulation in the input files. It also allows injection sources and probes to be placed at specific locations, which is explained in greater detail in Part One of this tutorial.

The other groups can be as general or as specific as desired. It is beneficial to make separate blocks for satellite components that will either have distinct electric potentials for the simulation, or have a high level of concern with respect to damage due to ion sputtering and flyback.

1.14.5 Export

After the geometry is meshed, blocked, and named as necessary, the file is ready to be exported in an abaqus file format (.inp) compatible with TURF. A dialogue box with export options will pop up when the user hits save. Uncheck the option to export the file using Cubit IDs; the option will refer to the blocks by the ID numbers used in Cubit rather than the names the user has assigned, which is undesirable.

Make sure *all* surfaces of the geometry are in a block before the geometry is exported. If a surface is forgotten, there will be a hole in the exported mesh geometry that will cause the simulation to crash.

1.14.6 Advanced Examples

Ion Thruster Grid

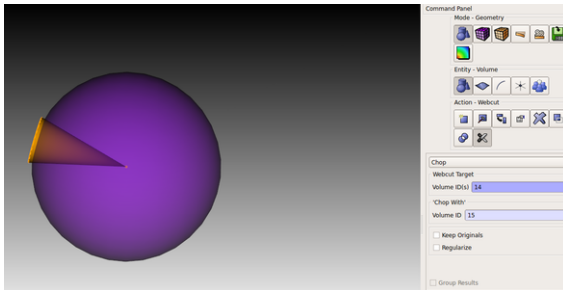


Figure 1.25 (a) Chop a slightly-oversized cone.

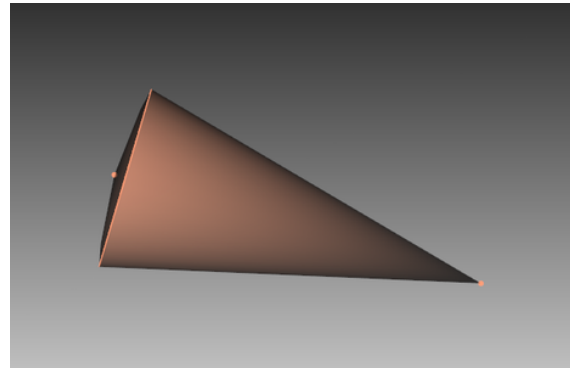


Figure 1.26 (b) The resulting volume. Note the slight curve on the cone's bottom face.

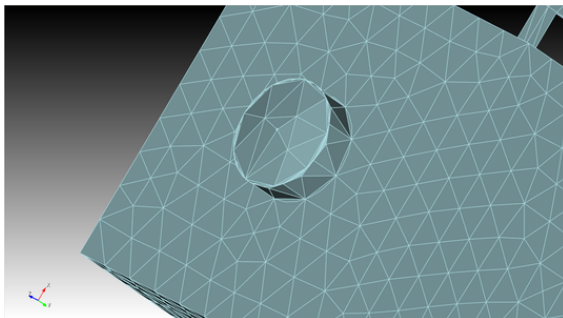


Figure 1.27 (c) The curved thruster face as part of the satellite's surface mesh.

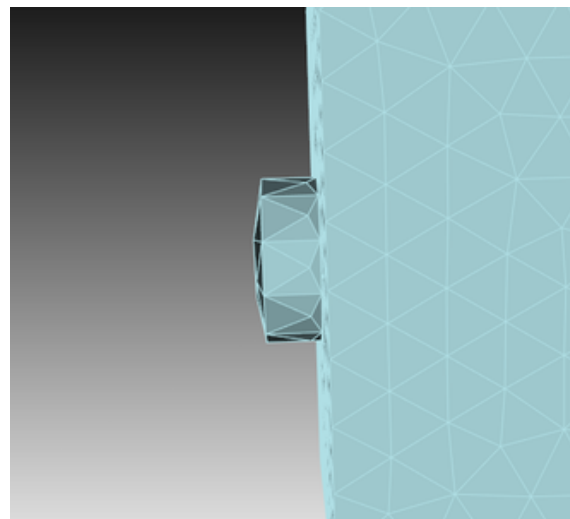


Figure 1.28 (d) Another view of the thruster face to better display the slight curve of the surface mesh.

For a spacecraft using an ion thruster, a special geometry consideration may need to be taken into account. Conventional ion thrusters of the size greater than (10) cm in diameter use curved grids in order to increase the structural integrity. Although the beam divergence is typically much less compared to Hall Thrusters, the ion beam still expands due to the potential field within the grids. On top of this, the curvature of the grids contributes significantly to the beam divergence.

A surface with the appropriate curvature can be created by using the chop function with a sphere and a cone. When creating a cone, its bottom face needs to be extended past the sphere such that the cone volume can be chopped using the sphere volume. The process is displayed in the [cone figure](#).

Rescale, move, separate, and unite volumes and surfaces as necessary until the curve is properly assimilated into the satellite geometry. Then apply the surface mesh. An unmeshed geometry file with this thruster face has been saved as `example2.cub` for the user's convenience.

RPA Source

Particle injection using a Retarding Potential Analyzer (RPA) source occurs at a certain distance from the thruster face. Essentially, ions are injected from a sphere above the thruster face rather than the thruster face itself. Because of this injection sphere, there is a volume of empty space underneath it. The size of the injection sphere is determined by the size of the thruster and the maximum angle to which the RPA data is provided: its radius must be large enough to ensure particles are not injected within the thruster's geometry, but small enough so the injection sphere does not interfere with other components nearby.

Due to the empty space underneath the spherical injection surface, the electric field establishes at the edge, possibly pulling slow ions toward the thruster face. In order to prevent this, a physical representation of the vacuum space under the injection sphere is useful for simulation purposes. This geometry is used solely in setting the potential within such that electric field does not establish within the region. Once an appropriate size for the injection sphere is determined, add a *slightly smaller* sphere centered at the thruster face to the geometry. However, the mesh of the sphere should be kept independent of the satellite's geometry. In other words, do not unite the spacecraft volume with the vacuum sphere volume. The correct setup is saved as `example3.cub`, and also shown in the [RPA figure](#).

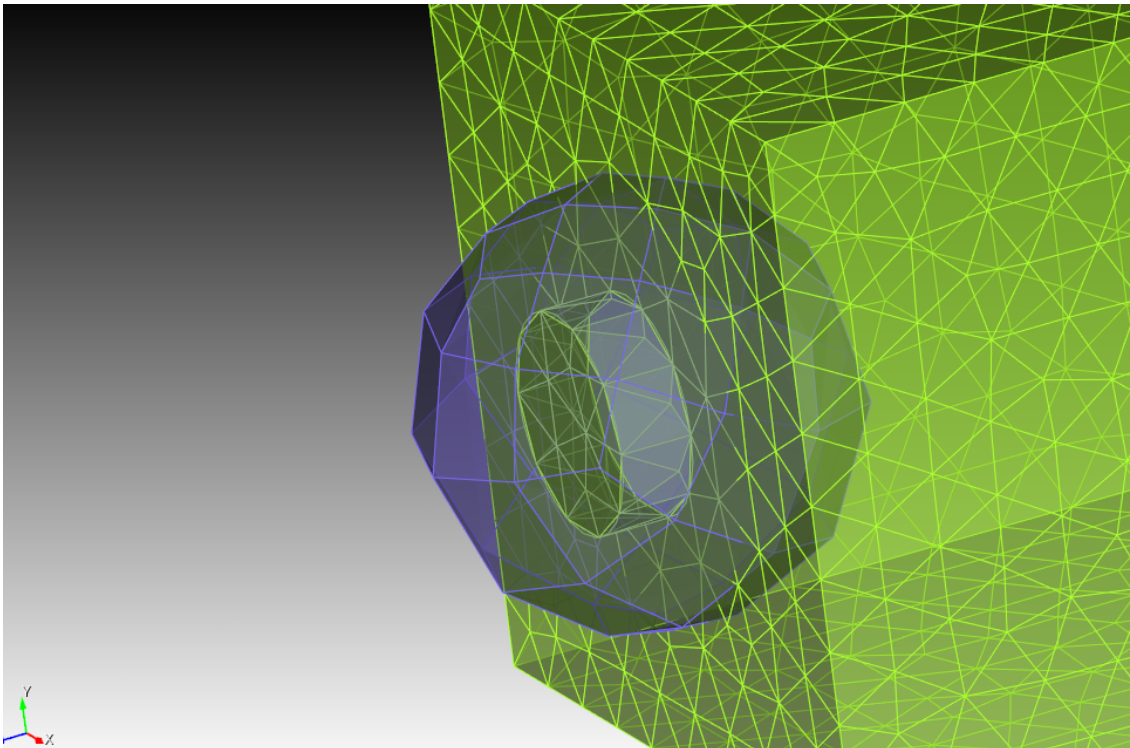


Figure 1.29 A vacuum space sphere placed over a thruster face using an RPA source. Note that the sphere mesh overlaps the satellite mesh and is not a part of the entity.

1.14.7 References

1. TIP: Instead of typing out all the surface IDs, *all* can be used in "Select Entities to Mesh" to point to all the surfaces.

1.15 EP Plume Simulation: ParaView

Author

Kari A. Kawashima and Samuel J. Araki

1.15.1 Introduction

A series of electric propulsion (EP) plume simulations is provided in the TURF tutorial. Starting from the [baseline plume simulation](#), advanced models are incrementally added in the order of [the DSMC collision method](#), [spacecraft charging](#), and [adaptive mesh refinement and dynamic load balancing](#). There are also tutorials on how to create geometry file using [Cubit](#) and how to visualize field data and particles using Paraview (this page). This tutorial serves as a guide to displaying the simulation results in ParaView [8].

Table 1.49 Folders created by various TURF operations to output results.

File/Folder	TURF Operation	Type of Data
UMesh.vtu	LogicalMeshSurface↔ SugarcubeOp	Surface mesh loaded to TURF
UMesh_sugarcube.pvtr	LogicalMeshSurface↔ SugarcubeOp	Points to UMesh_sugarcube_↔ DOM.vtr files output by each sub-domain
UMesh_sugarcube_DOM.vtr	LogicalMeshSurface↔ SugarcubeOp	Sugarcube information
Plot3D	LogicalFieldWriteVTKOp	3D volume mesh field data
PlotSurf	UFieldWriteVTKOp	Surface mesh field data
PlotTraj	MSPDistWriteVTKOp	Particle data
PlotXZ	LogicalFieldWriteVTK2DOp	2D volume mesh field data
Probe	ProbeStageWriteOp/Probe↔ FixedWriteOp	Probe data

Once started the EP plume simulation from `tutorial-TURF/Plume/PlumeExample/`, various files and folders are created as listed in the [files table](#).

The newly created folders (except `Probe`) contain files with extensions of `.pvts`, `.pvtu`, `.pvtp`, `.vtm`, `.vts`, `.vtu`, and `.vtp`. For the case of structured grid, TURF outputs one `.pvts` file and multiple `.vts` files, which the number of `.vts` files correspond to the number of sub-domains at a given iteration. The `.vts` file contains the data that belongs to a single sub-domain, and the `.pvts` file points to multiple `.vts` files, representing the whole simulation domain. There is an additional file with `.vtm` extension, which can be read in place of `.pvts` file.⁴ The viewing of these files requires software compatible with the VTK format files; this tutorial uses ParaView as the viewing software, but VisIt [22], is a program with similar functionality.

1.15.2 ParaView

When TURF runs a simulation, it will indicate when the imported surface mesh has been successfully sugarcubed.⁵ This means that the geometry mesh has been fully imported and the output geometry file (`UMesh.vtu`) can be opened for viewing in ParaView, even before the time steps begin advancing. When opened and clicking “Apply” in the sidebar, the satellite geometry should appear on the interface as shown in the [Geometry figure](#). This figure highlights five areas in ParaView:

1. Area 1 allows the user to display a specific frame of data in the simulation,
2. Area 2 is the Active Variable Controls toolbar,

⁴Loading ‘.pvts’ file to the visualization software generally works, but some visualization software may work better with ‘.vtm’ file.

⁵Once the surface mesh is loaded, the relationship between this mesh and the volume mesh is determined. This includes the number of surface elements contained within each volume mesh cell, cells that are inside and outside of geometries, etc. We call this process, Sugarcubing in this document.

3. Area 3 is the Common toolbar,
4. Area 4 displays a tree listing all the existing entities, and
5. Area 5 is the Properties sidebar.

The bounds of the domain can also be made visible by importing the `UMesh_sugarcube.pvtr` file.

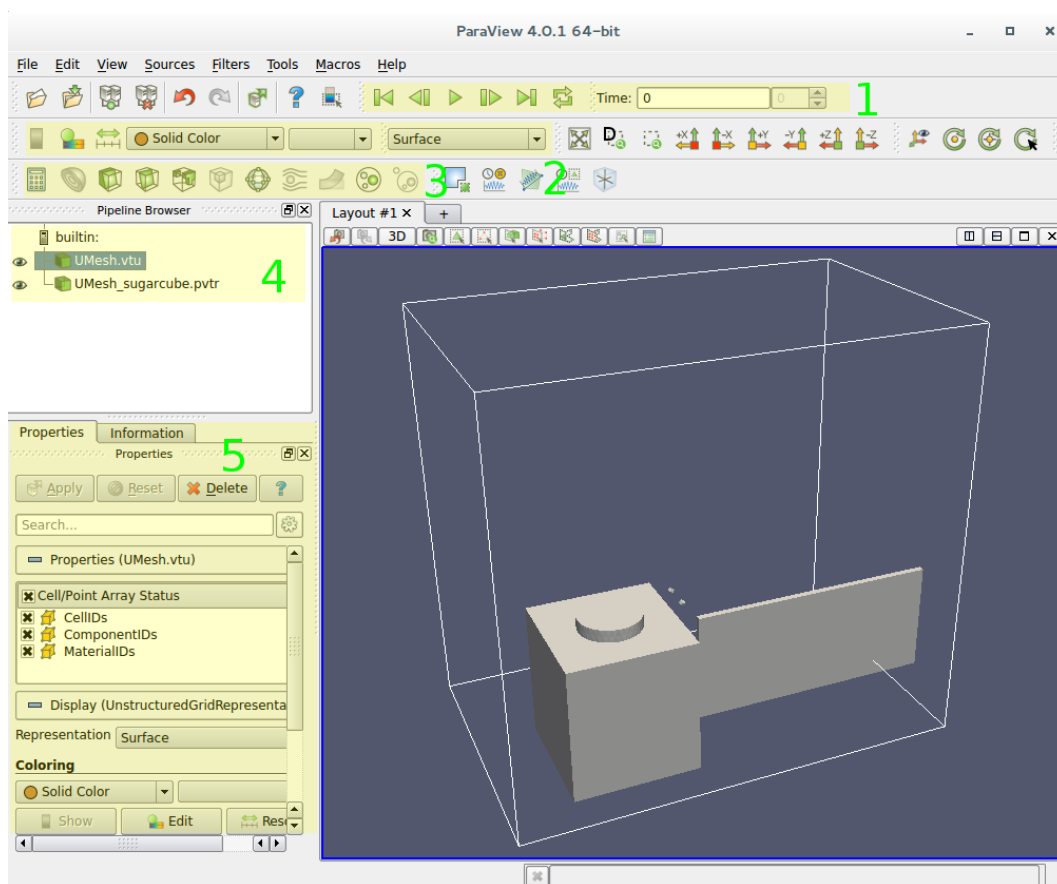


Figure 1.30 'Display of satellite geometry and a box representing the simulation domain. Some of ParaView's basic functions are highlighted in yellow.'

Particle Trajectory

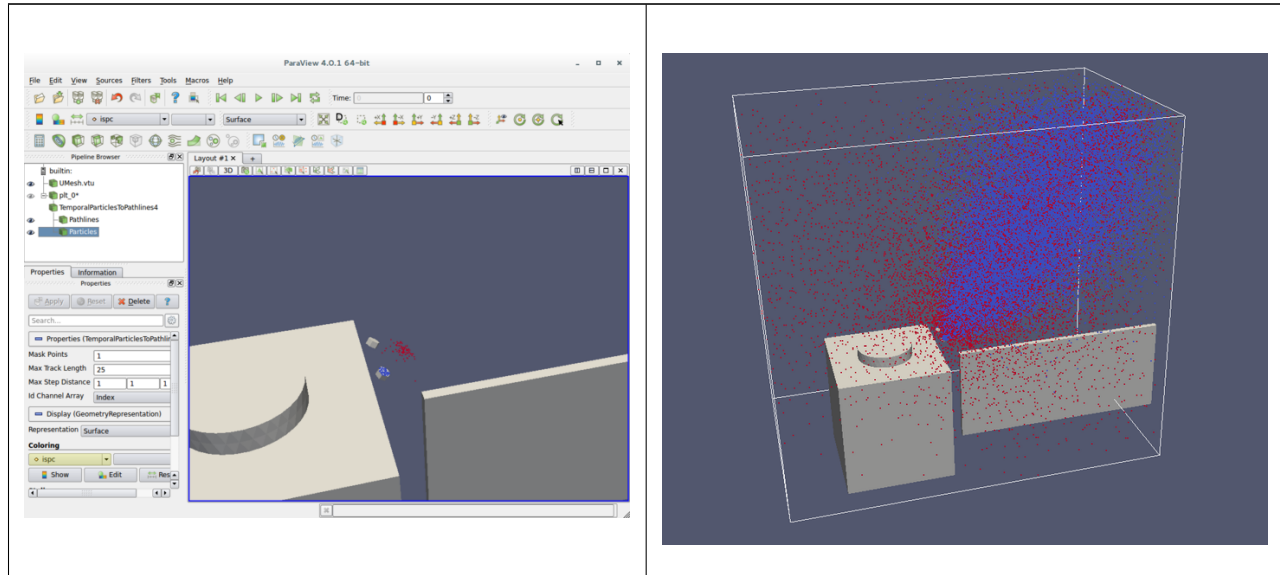
To see the progression of particles for the duration of the simulation, open the `plt_*.pvtp` collection of files inside the `PlotTraj` folder. With the `plt_0*` in Area selected, go to the Filters tab in the taskbar and apply the "Temporal Particles To Pathlines" filter to the data. This filter should be able to be found under "Alphabetical" category under the Filters tab or by using "Search..." tool.

In the Properties bar, change the "Mask Points" value to (1) so the simulation does not inflate the number of particles in the display.⁶ Also change "Id Channel Array" to "Index" to ensure the progression of particles is based on the correct position data. Small dots should appear on the face of the simulation's injection surface once the filter is applied (by clicking "Apply" button in Area 5). It is useful to change the colors of the particles depending on their gas species. Selecting Particles in the tree and changing the color display to "ispc" gives different colors to the neutral atoms and ions, as shown in the [Particle Figure](#).⁷

The buttons in Area can be used to display the simulation frame-by-frame, skip to the end or beginning, or simply play them chronologically as a short video. For a clearer display of the particles, the visibility of the pathways can be turned off by clicking the eye icon next to its name in the tree, and vice versa to displays the particles at the end of simulation.

⁶This value can be set to larger number if too many particles are displayed.

⁷The color display is under "coloring" in Area 5 which is by default chosen to be "Solid Color".



Surface Field Data

Data collected at the surface mesh, such as particle flux, can also be displayed. This can be particularly useful if there is concern that, for example, ion flyback could potentially damage mission-critical equipment mounted to the spacecraft. Import the `pltSurf_.pvtu` file collection from the `PlotSurf` folder and hit "Apply". For clearer viewing of the flux, turn off the visibility of data from other files. Then, change the display to show the flux of the desired particle in Area (e.g. `NPFLUX-IN_Xe+@g`). Go to the last time-step by clicking the forward button in Area 1. In case the color map does not cover the appropriate range for the flux data, click "Rescale" in Area 2. At this point, the surface flux of xenon ions should be displayed as shown in the \ref FIG_EPPlumeParaview_ParaView_Flux "Flux figure". Click "Show" in Area or Area to display the legend for the color gradient. "Edit" should also be used to manipulate the colormap as necessary

Volume Mesh Field Data

In order to visualize the field data,

1. Import the `plt_.vts` file collection from the `Plot3D` folder and hit "Apply".
2. Select the corresponding `plt_0*` that appears in the tree, and then click the Slice button. This command will create a plane within the domain at a specified location. For this geometry's orientation, a plane defined by the $\hat{Y}$ Normal shows the feature of plume.
3. Change the coloring from "Solid Color" to the field variable of choice. This shows the xenon ion density along the mid-plane of the simulation domain.

Alternatively, TURF can use `LogicalFieldWriteVTK2Dop` to save a 2D slice. These files are much smaller than

the 3D field data and can be saved much more often. Examples of this data are found in the `plotXZ` folder.

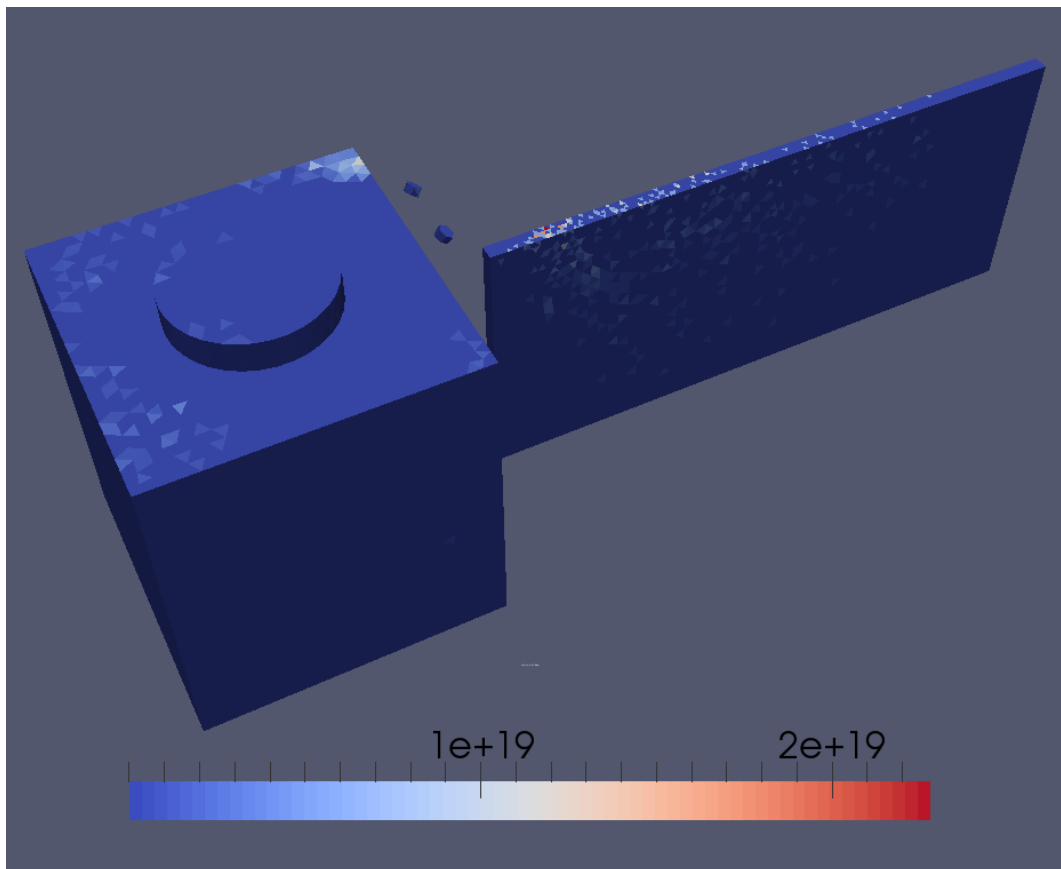


Figure 1.31 Xenon ion density along the mid-plane of the simulation domain.

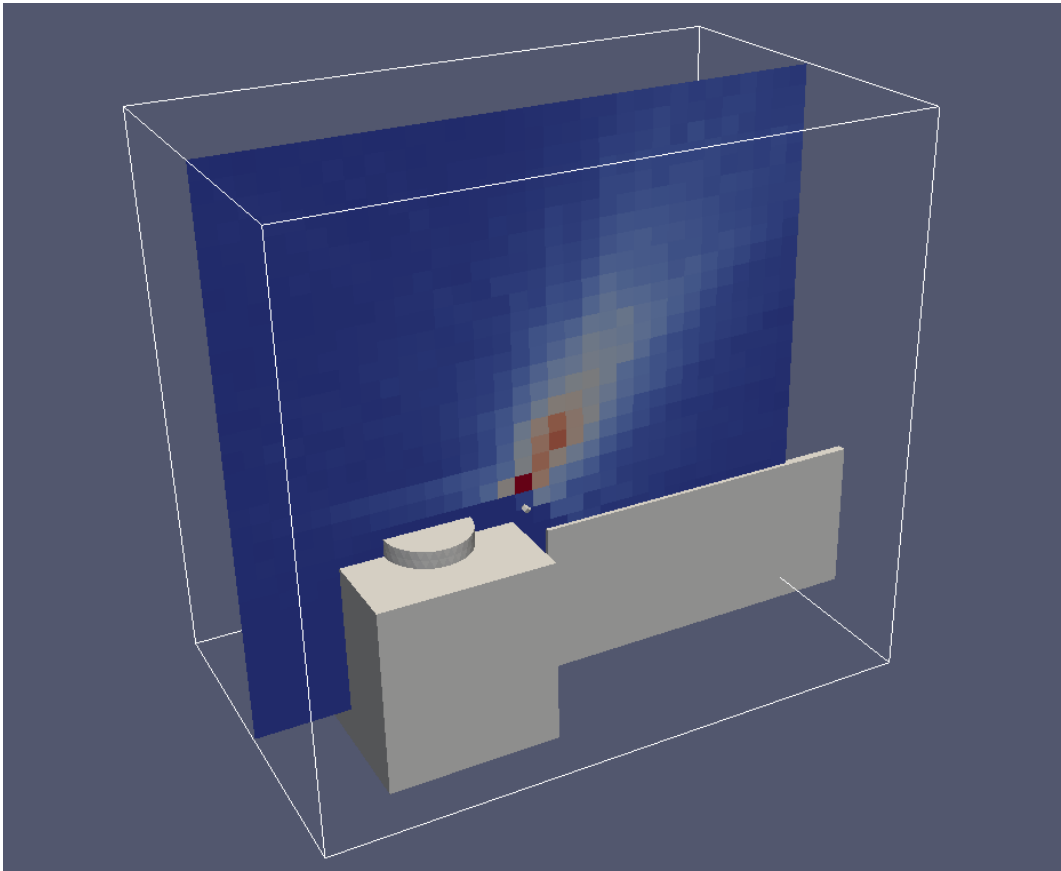


Figure 1.32 Xenon ion density along the mid-plane of the simulation domain.

Chapter 2

List of Operators in TURF

Module	Operation	Description
Charging	ComponentChargeImplicitOp	Perform charging calculations on surface mesh
Charging	ComponentChargeOp	Perform charging calculations on surface mesh
Charging	FieldSimpleAverageOp	An example Operator that calculates the average of the surrounding cells.
DSMC	MSPDistDSMCCEXOp	DSMC method for CEX collisions
DSMC	MSPDistDSMCCPUOp	DSMC collision calculation
DSMC	MSPDistDSMCClassicalOp	DSMC method using the classical scattering with the spin-orbit free potential
DSMC	MSPDistDSMCCoulombNanbuOp	Perform Nanbu method for Coulomb collision
DSMC	MSPDistDSMCCoulombSubOp	Perform Nanbu method for Coulomb collision in collision sub-cells
DSMC	MSPDistDSMCOp	DSMC collision calculation on GPU
DSMC	MSPDistFDSMCCEXOp	Fractional DSMC method for CEX collision
DSMC	MSPDistFDSMCCPUOp	Fractional DSMC collision calculation
Field	CalculateHallMHDOp	Calculate perpendicular and cross-flow components of mobility.
Field	CalculateHallMHD RHSOp	RHS for Ohm's law solve (IEPC-2015-314)
Field	CalculateIonizationRateOp	Calculates ionization rates from field data.
Field	CalculateMobilityComponentsOp	Evaluate mobility components (IEPC-2015-314)
Field	ConstantAnnulusPotentialOp	Apply annularly symmetric potential to field
Field	ConstantSphericalPotentialOp	Apply spherically symmetric potential to field
Field	ConstantWellPotentialOp	Apply constant parabolic well potential to field
Field	FieldArithmeticCoeffOp	Perform arithmetic operation to two fields with two coefficients
Field	FieldArithmeticConstOp	Perform arithmetic operation to one field and one scalar value
Field	FieldArithmeticOp	Perform arithmetic operation to two field data
Field	FieldBlendTimeOp	Blend two fields over time
Field	FieldBoundaryConditionsOp	This class handles simple boundary conditions such as, periodic, extrapolate, Neumann, Dirichlet, for both structured and unstructured meshes.
Field	FieldCopyKMatrixOp	Copy kMatrix to field data
Field	FieldEvaluateEqnOp	Evaluate a math equation and assign the result to field data

Module	Operation	Description
Field	FieldGradientOp	Compute gradient on field data
Field	FieldMathFunctionOp	Apply math function on one field data
Field	FieldMathFunctionThreeArgOp	Apply math function that requires three field data
Field	FieldMathFunctionTwoArgOp	Apply math function that requires two field data
Field	FieldSetDomainOp	Assign domain related parameters to field data
Field	FieldSetOp	Set unstructured or structured field data
Field	GlobalFieldPatchEmptyOp	Create a global field to sum
Field	GlobalFieldPatchFillOp	Patch Ufield
Field	LogicalBCMirrorMoveInOp	Move and add field ghost data over reflecting surface
Field	LogicalBCOp	Set a physical boundary to extrapolation
Field	LogicalDensityBoltzmannOp	Find potential from Boltzmann relations
Field	LogicalDensityNeuRayTraceOp	Compute steady-state neutral density field by view factor model
Field	LogicalDensityNeuRayTraceVDFOp	Compute steady-state neutral density field by view factor model with arbitrary VDF for source
Field	LogicalDensityNeuRayTraceVDFProbeOp	Compute steady-state neutral density field by view factor model with octree based particle VDF compression
Field	LogicalDirectPoissonEmptyOp	Read Solution of Sparse Direct Poisson into LogicalField
Field	LogicalDirectPoissonFillOp	Load RHS from LogicalField for Sparse Direct Poisson Solve
Field	LogicalFieldGeometrySetOp	Set field on geometry boundary to neighboring free-space value
Field	LogicalFieldInitOp	Create new Logical Field associated with a given SMesh
Field	LogicalFieldPatchOp	Write to output files for 3D plots in .vts format
Field	LogicalFieldStreamingDMDOp	None
Field	LogicalGlobalFieldPatchEmptyOp	Copy data from the global field
Field	LogicalGlobalFieldPatchFillOp	Create a global field to sum the data.
Field	LogicalGradientOp	Calculate cell centered gradient
Field	LogicalNodeGradientOp	Calculate node-centered gradient of cell center field
Field	LogicalNormOp	Calculate L^p -norm of field variable
Field	LogicalPoissonBoltzmannStrip1DOp	Solve for the electric field assuming a Boltzmann electrons
Field	LogicalPoissonOp	Red/Black Gauss-Seidel Poisson Solve
Field	LogicalPoissonPeriodicStripOp	Solve Poisson equation with periodic boundary conditions, requires cuFFT.
Field	LogicalPoissonStrip1DOp	Red/Black 1D strip Poisson solve w/o transverse cells
Field	LogicalPoissonStripOp	Red/Black line relaxing Poisson solve
Field	LogicalPotentialBoltzmannOp	Find potential from Boltzmann relations
Field	LogicalPotentialSetInsideGeometryOp	Integer flag for regions in/out body to select potential solver
Field	LogicalPotentialSetRefOp	Update reference values for Boltzmann inversion
Field	LogicalResidualOp	Calculate residual of Poisson solve
Field	UFieldICByComponentOp	Set UField data by component
Field	UFieldMSInitOp	Create unstructured field data
Field	UFieldSumByComponentOp	Sum UField data by component

Module	Operation	Description
Fluid	CeseLinearAdvectionOp	None
Fluid	CeseOp	None
Fluid	FluidConstantICOp	Set field parameters based off fluid definition
Fluid	FluidESPushOp	Advance fluid state with electrostatic forcing
Fluid	FluidESwBPushOp	Advance fluid state with Lorentz forcing with external E & B fields
Fluid	FluidESwFixedBPushOp	Advance fluid state with Lorentz forcing with external E & B fields
Fluid	FluidRayleighTaylorICOp	Initialize fluid properties for the Rayleigh Taylor problem
Fluid	FluidSphereICOp	Set fluid parameters for cells within a sphere
Fluid	IntegrateFluxOp	Sum flux variables to advance fluid state
Fluid	LinearSemiLagrangianOp	Linear advection using semi-Lagrangian advance
Fluid	MP3FluxCalcOp	Fluid flux calculation using the MP3 scheme
Fluid	PlumeElectronEnergyUnmagNOXOp	Solve steady-state electron fluid energy equation for EP plume using Trilinos nonlinear solver
Fluid	PlumeElectronEnergyUnmagOp	Solve steady-state electron fluid energy equation for EP plume
Fluid	PlumeElectronMassUnmagOp	Solve steady-state electron fluid continuity equation for EP plume
Fluid	PlumeElectronMomentumUnmagOp	Solve steady-state electron fluid momentum equation for EP plume
Fluid	RoeFluxCalc2Op	Calculate 2nd order Roe flux
Fluid	RoeFluxCalcHLL2Op	Calculate 2nd order Roe flux with HLL2 limiter
Fluid	RoeFluxCalcHLL2EOP	Calculate 2nd order Roe flux with HLL2E limiter
Fluid	RoeFluxCalcOp	Calculate 1st order Roe flux
Fluid	TVDFluxCalcOp	Calculate total variation diminishing flux
Geometry	LogicalASMeshSurfaceSugarcubeOp	Create sugarcube surface mesh/AMR mesh intersection.
Geometry	LogicalMeshGlobalRecolorOp	Complete sugarcubing for multi-domain simulation
Geometry	LogicalMeshSurfaceBBoxSplotchOp	Mark surface triangle bounding box on cells
Geometry	LogicalMeshSurfaceSugarcubeOp	Create sugarcube surface mesh/structured mesh intersection
Geometry	MSPDistMarkDelGeometryOp	Set CellID==Max to mark deletion when particle is inside geometries
Geometry	MSPDistSugarcubeSurfBCMoveOp	Perform particle surface reflection
Geometry	MSPDistSugarcubeSurfIntersectionOp	Determine if particles intersect with surface elements
Geometry	SMeshCreateOp	Generate 1D/2D/3D SMesh, for now supports only Cartesian mesh but eventually plan adding support for loading nodes from a file
Geometry	SurfaceComponentSetOp	Add component list to surface mesh object
Geometry	SurfaceMeshMergerOp	Merge two or more surface meshes
Geometry	SurfaceNormalCheckOp	Check if surface normal is consistent in its direction for all UMesh elements
Geometry	UMeshImporterOp	Read unstructured volume or surface mesh file
Particle	FMSPDistInitOp	Create new filtered particle distribution of FMSPDist class
Particle	MSPDistASampleDensOp	Sample particles to obtain density data.

Module	Operation	Description
Particle	MSPDistASampleOp	Sample particles to obtain field data on AMR mesh.
Particle	MSPDistBCPeriodicOp	Periodic boundary condition for particles
Particle	MSPDistBCSpecOp	Specular boundary condition for particles
Particle	MSPDistBCSpecularOp	Specular boundary condition for particles
Particle	MSPDistBCStreamOp	Stream boundary condition for particles
Particle	MSPDistBoxICOp	Initialize particles uniformly in a physical box
Particle	MSPDistBoxICStreamOp	Initialize particles in a physical box with an imposed field in given direction
Particle	MSPDistCellIDOp	Find cell ID associated with particle location
Particle	MSPDistChargeDepositionFlagOp	Accumulates charge to cells for one dimensional problems
Particle	MSPDistChargeDepositionOp	Accumulate charge to cells
Particle	MSPDistCombineOp	Combine two particle distributions into one
Particle	MSPDistConstantICOp	Create uniform constant number of particles/cell within a box
Particle	MSPDistCopyOp	Copy srcdist to dstdist, and set srcdist particle count to zero
Particle	MSPDistDirectCellMergeOp	Default RMS x&v random sign merge
Particle	MSPDistDirectCellSplitOp	Default RMS x&v random sign split
Particle	MSPDistEMBorisOp	Electromagnetic particle push using Boris method.
Particle	MSPDistESPhiCNPuShOp	Crank Nicolson electrostatic particle push using cell center potential field
Particle	MSPDistESPushOp	Electrostatic particle push using node electric field
Particle	MSPDistEmptyOp	Set particle count to be zero
Particle	MSPDistGonoskovMergeOp	Default RMS x&v random sign merge
Particle	MSPDistInitOp	Create new particle distribution of MSPDist class
Particle	MSPDistMCCCEXOp	Apply MCC CEX collision by only exchanging velocities between the collision pair
Particle	MSPDistMCCElasticFitOp	Apply MCC elastic collision
Particle	MSPDistMCCElasticTableOp	Apply MCC elastic collision
Particle	MSPDistMCCElasticVHSOp	Apply MCC elastic collision
Particle	MSPDistMCCEleNeuOp	Electron-neutral collision with MCC/Null collision and Okhrimovskyy's scattering model
Particle	MSPDistMoveOp	Advance particles
Particle	MSPDistOrbiterICOp	Initialize particles in a physical box with synchronized Orbits
Particle	MSPDistParticleCountOp	Obtain global count of particles
Particle	MSPDistPatchOp	Inter-domain particle patch
Particle	MSPDistPerturbedCellIDOp	Find cell ID associated with particle location
Particle	MSPDistRemovePartOp	Remove particles in MSPDist with $w=0$ or $cellID > mesh \rightarrow N_{cells}$
Particle	MSPDistSampleOp	Sample particles to obtain field data.
Particle	MSPDistSortOp	Sort MSPDist particles for cellID
Particle	MSPDistSortTreeOp	Sort particles in an octree velocity mesh
Particle	MSPDistSortTwoOp	Sort MSPDist for cellID then speciesID
Particle	MSPDistSplitOp	Split particle distributions
Particle	MSPDistSplitSpeciesOp	Split species by user defined criterion

Module	Operation	Description
Particle	MSPDistSubCellIDOp	Find cell ID associated with particle location
Particle	MSPDistSubSortOp	Sort MSPDist particles for arbitrary cellID including subcells
Particle	MSPDistTotalMomentsOp	Compute total moments
Particle	MSPDistTreeChargeDepositionOp	Accumulate particle weights to Fluid Data
Particle	MSPDistVSortOp	Sort particles by cell and velocity octant
Plotting	ASMeshReadVTKOp	Read AMR mesh data from the VTK file
Plotting	LogicalAMRFieldWriteVTKOp	Write to output files for 3D plots in .vtu format
Plotting	LogicalFieldCatalystOp	Kitware Catalyst plotting connection
Plotting	LogicalFieldReadVTKOp	Read LogicalField data from VTK file created by LogicalFieldWriteVTKOp.
Plotting	LogicalFieldWrite1DOp	Write to output files for line plots
Plotting	LogicalFieldWriteTecplotOp	Write logical field data in Tecplot format
Plotting	LogicalFieldWriteVTK2DOp	Plot 2D slice of 3D data
Plotting	LogicalFieldWriteVTKOp	Write to output files for 3D plots in .vts format
Plotting	LogicalFieldWriteVTKROp	Export the field data in .vtr format
Plotting	LogicalVlasovFluidWrite1DProbeVDFOp	Write particle VDF from particles in probe region
Plotting	MSPDistReadVTKOp	Read MSPDist from VTK file created by MSPDist↔WriteVTKOp.
Plotting	MSPDistReadVTKSplitOp	Read MSPDist from VTK file created by MSPDist↔WriteVTKOp.
Plotting	MSPDistWriteVTKOp	Output MSPDist information such that it can be read for restart
Plotting	ProbePointOp	Output field data time series from a given point
Plotting	SampleAMRFieldReadVTKOp	Read sample field data from VTK file
Plotting	SampleAMRFieldWriteVTKOp	Write sample field data to a file in VTK file format
Plotting	SampleFieldReadVTKOp	Read sample field data from VTK file
Plotting	SampleFieldWriteVTKOp	Write sample field data to a file in VTK file format
Plotting	UFieldReadVTKOp	Read UField data from VTK file created by UField↔WriteVTKOp.
Plotting	UFieldWriteTecplotOp	General unstructured field writer
Plotting	UFieldWriteVTKOp	General unstructured field writer
Plotting	UPCWriteVTKOp	Output UPC PIC data to vtk file.
Plotting	VolumeRenderOp	Single cubic domain realtime volume rendering
Probe	MSPDistProbeFixedOp	Faraday/RPA numerical probes attached to a surface
Probe	MSPDistProbeStageSphericalOp	Faraday/RPA numerical probes on a virtual spherical stage
Probe	ProbeFixedWriteOp	Write fixed probe data
Probe	ProbeStageWriteOp	Write spherical stage probe data
SourceModel	MSPDistBeamletImportOp	None
SourceModel	MSPDistFieldSourceOp	Samples a particle field in terms of 'z r vz vr vt'
SourceModel	MSPDistIonThrusterOp	Ion thruster source
SourceModel	MSPDistNormalMaxwellianOp	Maxwellian stream source
SourceModel	MSPDistNormalMaxwellianStreamOp	Maxwellian stream source from triangulated surfaces.
SourceModel	MSPDistSourceCosineOp	Cosine distribution particle source for rarefied flows
SourceModel	MSPDistSourceRPAOp	Inject particle according to the current density profile

Module	Operation	Description
SourceModel	MSPDistSourceStageSphericalOp	Particle source on a virtual spherical stage
SourceModel	MSPDistSourceVDFOp	Inject particles based on input VDF
Surface	MSPDistOutgassingOp	Outgassing from surfaces
Surface	MSPDistSurfaceInteractionOp	Perform reflection and sputtering particle-surface interactions
Surface	MSPDistThinSheathOp	Apply particle boundary condition assuming thin sheath
Surface	UFieldMSComputeOp	Compute surface field
Surface	UFieldSurfaceMaxwellianCurrentOp	Compute current of Maxwellian species to surface elements
Surface	ViewFactorCosineOp	Compute steady-state neutral density field by view factor model
UPC	SPDistConstantVlasBCOp	Constant boundary conditions for VlasovPIC
UPC	SPDistConstantVlasICOp	Constant initial conditions for VlasovPIC
UPC	SPDistEvaluateUPCBCOp	Evaluated boundary conditions for UPC
UPC	SPDistEvaluateUPCICOp	Evaluated initial conditions for UPC
UPC	SPDistVlas2wOp	Density to weights for VlasovPIC
UPC	SPDistVlasDensityToFieldOp	Integrate density to field data for VlasovPIC
UPC	SPDistw2VlasOp	Weights to density for VlasovPIC
UPC	UPCMetricsOp	Export UPC metric data for mass and energy conservation
Universe	LoadBalanceDomainOp	Determine MPI rank to assign domains
Universe	LoadBalanceDomainZoltanOp	Determine MPI rank to assign domains using the Trilinos Zoltan library
Utility	ConfigurationTestOp	Determines accuracy of an operator or combination thereof and reports failure to class in world.
Utility	ConstantOperatorOp	Work to allow generic constant parsing
Utility	CriteriaStageOp	Continue to next stage if quantity below criteria
Utility	FromGPUOp	Recursive GSObject transfer from GPU
Utility	GSOPatchOp	Inter-domain particle patch
Utility	GSPrimeEvaluateEqnOp	Evaluate Math equation that is stored in GSPrime objects
Utility	HelloMPIWorldOp	Hello world to test MPI
Utility	NextStageOp	Continue to next stage
Utility	ToGPUOp	Recursive GSObject transfer to GPU
Vlasov	CreateVlasovVariableOp	Create new Vlasov fluid variable
Vlasov	FokkerPlanckExplicit3rdOp	3rd order Rosenbluth-Fokker-Planck model for Vlasov solver
Vlasov	FokkerPlanckExplicit4thOp	4th order Rosenbluth-Fokker-Planck model for Vlasov solver
Vlasov	FokkerPlanckExplicitNewOp	2nd order explicit Rosenbluth-Fokker-Planck model for Vlasov solver upgraded to use the trick for positivity preserving
Vlasov	FokkerPlanckExplicitNormOp	2nd order explicit Rosenbluth-Fokker-Planck model for Vlasov solver using normalized quantities
Vlasov	FokkerPlanckExplicitOp	2nd order explicit Rosenbluth-Fokker-Planck model for Vlasov solver

Module	Operation	Description
Vlasov	FokkerPlanckImplicitOp	2nd order implicit Rosenbluth-Fokker-Planck model for Vlasov solver Based on Tiatano et al., JCP 297 (2015) 357-380
Vlasov	LogicalBCPeriodicOneDomVlasovOp	Periodic BC for Vlasov
Vlasov	LogicalBCVlasovExtrapolateOp	Set a velocity boundary conditions to extrapolation
Vlasov	LogicalFluidToLogicalVlasovFluidOp	Create Vlasov VDF from Fluid Variables
Vlasov	LogicalVlasov2DWriterOp	Exports a 2D phase-space plot
Vlasov	LogicalVlasov3DWriterOp	Export a 3D phase-space plot
Vlasov	LogicalVlasovBGKOp	Applies the BGK collision operator to a Vlasov variable
Vlasov	LogicalVlasovCalcFluidVariablesOp	Calculates field variables given a velocity distribution
Vlasov	LogicalVlasovFieldVolumetricMulOp	Multiply or divide Vlasov data by cell volumes
Vlasov	LogicalVlasovFluidBoltzmannSetOp	Set initial conditions of a Vlasov field using a Boltzmann distribution
Vlasov	LogicalVlasovFluidConstantICOp	Create or initialize a new Vlasov fluid
Vlasov	LogicalVlasovFluidFunctionICOp	Initialize Vlasov data using external function
Vlasov	LogicalVlasovFluidSetOp	Set Vlasov field data to constant
Vlasov	LogicalVlasovMaxwellianStreamOp	Maxwellian stream inflow condition for Vlasov data
Vlasov	LogicalVlasovReadVTKOp	Read LogicalVlasovFluid data from VTK file created by LogicalVlasov3DWriterOp.
Vlasov	MSPDistToLogicalVlasovFluidOp	Accumulate particle weights to Vlasov VDF
Vlasov	MSPDistTreeToLogicalVlasovFluidOp	Accumulate particle weights to Vlasov VDF
Vlasov	SPDistToLogicalVlasovFluidOp	Accumulate particle weights to Vlasov VDF
Vlasov	Vlasov1D1VSLop	Advect a Vlasov variable using the Semi-Lagrangian method
Vlasov	VlasovMetricsOp	Export Vlasov metric data for mass and energy conservation

Chapter 3

Module Documentation

3.1 Charging

The Charging module is responsible for calculating the migration of electrical charge through conductive and dielectric materials.

3.1.1 Surface Charging

The Surface Charging operators model the migration of charge along the surface of a material. To provide greater flexibility the surface is triangulated and the charge migrates between adjoining triangles. The surface mesh must be imported using the `UMeshImporterOp` operator. Details of the charging model are provided in Ref. [2]

Intended Use

The original use of this module is to predict the surface charging on a satellite at LEO or GEO while an electric propulsion device is firing. As such, the intended source of the charging is electrons and ions emitted from the thruster, which then impacts the surface on the spacecraft.

Limitations and Assumption

- In a dielectric, charge moves between adjoining triangles and to a common ground.
- In a conducting material, the charge instantaneously equilibrates.
- Charges can migrate between different components. However, it is assumed that the components are connected perfectly, and the average values are used in the calculations (i.e. thickness, conductivity, relative permittivity, etc).

Governing Physics

There are three different surface charging models in TURF:

- **Dielectric**, this model accepts charge from ion and electron currents but does not conduct charge to neighboring triangles.
- **Conductive**, this model accepts charge from ion and electron currents and instantaneously distributes the charge over the surface.
- **Propagate**, this model accepts charge from ion and electron currents and conducts charge over a timescale that is significant in comparison to the simulation times.

Dielectric On dielectric surfaces, charge is deposited on a surface element but does not migrate to the neighboring surface elements during simulation. The surface potential is determined by the charge remaining from the last time-step and the current deposited onto the surface element from the plasma.

$$q_i^{n+1} = \phi_i^n C_i + J_i^n \Delta t_{\text{ch}}, \quad \phi_i^{n+1} = q_i^{n+1} / C_i \quad (3.1)$$

where q is the surface charge, ϕ is the surface potential, J is the current to the surface from a plasma, Δt_{ch} is the charging simulation time-step, i is the index of a surface element, n is the index for the surface charging iteration. Here, the capacitance of the surface element is obtained by approximating the element as a plate capacitor, and the capacitance is given by $C_i = \epsilon_m A_i / \tau$ where ϵ_m is the permittivity of the surface material, A is the area of the surface element, and τ is the thickness of the surface element specified in the input.

Conductive Unlike the dielectric model, the conductive model uniformly distributes charges over all of the conducting surfaces. First, total charge from all the conductive surface elements are obtained and transferred to the spacecraft bus.

$$q_{\text{bus}}^{k=0} = q_{\text{bus}}^n + \sum_{j=1}^{N_c} q_j^n = q_{\text{bus}}^n + \sum_{j=1}^{N_c} \phi_j^n C_j \quad (3.2)$$

where k is the index for charging sub-step and N_c is the total number of conducting surface elements. In the conductive model, the capacitance of the surface element is obtained by approximating the element as an isolated sphere capacitor, and the capacitance is given by $C_i = 4\pi\epsilon_0 r_i$ where ϵ_0 is the vacuum permittivity and $r_i \approx (3A_i\tau/4\pi)^{1/3}$. The current model assumes that the conductive surfaces are always connected to the spacecraft bus. For the case where the same potential is not shared between the conducting objects, one needs to implement an appropriate circuit model (e.g. see Ref.[14]), which has not been implemented.

Second, the charge on the bus is increased or decreased based on the current from or to the semi-conductive surface approximated by the charge propagation model. For every iteration of the surface charge calculation as $n \rightarrow n+1$, the charge propagation calculation is performed for $k_{\max} + 1$ sub-steps, and the bus potential ϕ_{bus}^k is updated accordingly. For each sub-step k , the bus charge is updated based on sum of the currents to the bus and conducting surfaces.

$$\phi_{\text{bus}}^k = q_{\text{bus}}^k / C_{\text{total}} \quad (3.3)$$

$$q_{\text{bus}}^{k+1} = q_{\text{bus}}^k + \sum_{j=1}^{N_c} J_j^n \delta t - \sum_{p=1}^{N_p} \sigma_p \frac{\phi_{\text{bus}}^k - \phi_p^{k+1}}{\tau} A_p \delta t \quad (3.4)$$

where δt is the time-step size for each sub-step k of the charging time-step Δt_{ch} , N_p is the total number of surface elements modeled by the charge propagation model, σ_p is the electrical conductivity of the element, and C_{total} is the total capacitance and is the sum of bus capacitance, C_{bus} , and capacitances of conducting surfaces,

$$C_{\text{total}} = C_{\text{bus}} + \sum_j^{N_c} C_j \quad (3.5)$$

The bus capacitance C_{bus} is specified in the input. Note that the potential in the charge propagation model is solved implicitly so that the currents from the semi-conductive surfaces are calculated based on the potential at the $(k+1)$ th sub-step. For this reason, the charge propagation calculation is performed between the two equations (ϕ_{bus} and q_{bus}).

After the charge propagation calculation is completed for $k_{\max} + 1$ sub-steps (i.e. $\Delta t_{\text{ch}} = (k_{\max} + 1)\delta t$), the final step is to distribute the remaining charge in the bus to all the conducting surface elements according to their capacitance.

$$q_i^{n+1} = q_{\text{bus}}^{n+1} C_i / C_{\text{total}} \quad (3.6)$$

where $q_{\text{bus}}^{n+1} = q_{\text{bus}}^{k=k_{\max}}$.

Propagate Due to the wide range of electrical conductivity that materials can have, the charge propagation model is solved implicitly. In this way, the charge propagation model can handle dielectric, conductive, and semi-conductive surfaces without adversely impacting numerical stability. However, the implicit scheme requires solving a linear system of equations at every sub-step k , thus using the conductive and dielectric models helps in reducing the system size and increases computational efficiency. It should be noted that overall surface charging calculation is only semi-implicit since currents from a plasma is determined explicitly.

The rate of change in surface charge on surface element i is equal to currents from a plasma (J_i), neighboring elements (I_{ji}), and spacecraft bus ($I_{\text{bus},i}$).

$$\frac{dq_i}{dt} = J_i + \sum_{j=1}^{N_n} I_{ji} + I_{\text{bus},i} \quad (3.7)$$

where N_n is the number of elements neighboring element i . In order for this model to be stable for a wide range of electrical conductivity, currents from the neighboring elements and spacecraft bus are expressed with surface potentials at $(k+1)$ th sub-step.

$$\begin{aligned} \frac{dq_i}{dt} &\approx \frac{q_i^{k+1} - q_i^k}{\delta t} \\ &= J_i^n + \sum_{j=1}^{N_n} \sigma_{ji} \frac{\phi_j^{k+1} - \phi_i^{k+1}}{L_{ji}} d_{ji} \tau + \sigma_i \frac{\phi_{\text{bus}}^k - \phi_i^{k+1}}{\tau} A_i \end{aligned} \quad (3.8)$$

where σ_{ji} is approximated as the average conductivity between elements j and i , L_{ji} is length between the element centers, and d_{ji} is the edge length shared by the two elements. By expressing the surface potential in terms of surface charge and capacitance with elements approximated as plate capacitors, a linear system of equations ($Aq = b$) is formed. The matrix components and right-hand side element are given as

$$A_{ii} = 1 + \left(\sum_{j=1}^{N_n} \sigma_{ji} \frac{d_{ji}\tau}{L_{ji}C_i} + \sigma_i \frac{A_i}{\tau C_i} \right) \delta t \quad (3.9)$$

$$A_{ij} = -\sigma_{ji} \frac{d_{ji}\tau}{L_{ji}C_j} \delta t \quad \text{for } i \neq j \quad (3.10)$$

$$b_i = J_i^n \delta t + q_i^k + \sigma_i \frac{\phi_{bus}^k A_i}{\tau} \quad (3.11)$$

If a fixed time-step is used such that δt is constant, the matrix components can be precomputed and stored. Once surface charges at $(k+1)$ th sub-step are calculated by solving the linear system of equations, the total current to the bus in the equation can be determined. It should be noted that this model handles charge migration between neighboring components with different thickness and/or semi-conductive materials by assuming that the components are perfectly in contact and using the average material properties at the interface.

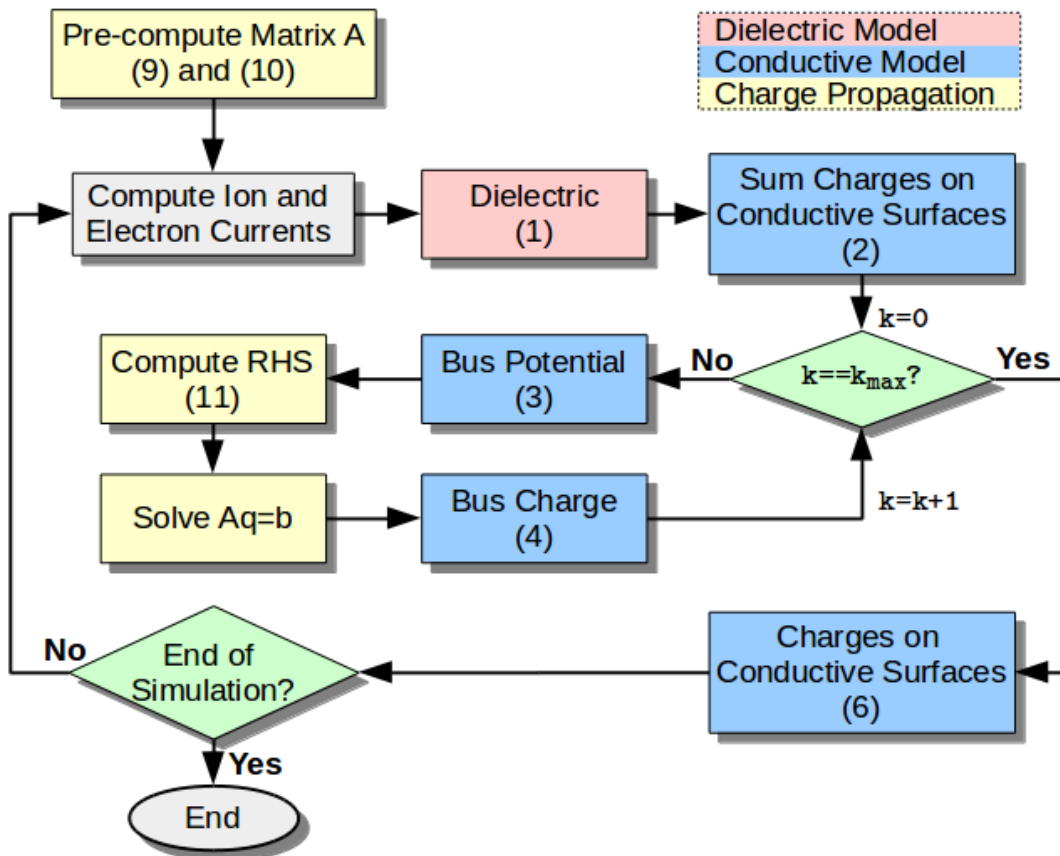


Figure 3.1 Flowchart of the spacecraft charging calculation with dielectric, conductive, and charge propagation models. Equation numbers are also given in the flowchart.

Implementation Since spacecraft components can be made out of different materials with different electrical conductivity values, a combination of three charging models may be required. The figure above shows the flowchart of the spacecraft charging model. The EP plume simulation is performed to determine ion and electron currents for every

iteration of the charging calculation. For the case of dielectric model, the surface charge on an element is independent of the charge of the bus or surrounding surfaces so the calculation can be decoupled from the other models. On the other hand, the conductive and charge propagation models are coupled through spacecraft bus potential. Therefore, these calculations have to be performed together.

3.2 DSMC

The DSMC module contains models to compute different types of collisions using the DSMC algorithm.

3.2.1 The DSMC Method

Direct Simulation Monte Carlo (DSMC) is a collision algorithm used by particle methods. It is a statistical approach to collisions that approximates the solution of the Boltzmann collision integral. The algorithm randomly samples possible collisions between macro-particles in a computational cell. This is in contrast to the Monte Carlo Collision (MCC) collision algorithm that collides each particle against a background gas. The DSMC method is considered to be more accurate than the MCC method but is also more computationally expensive. This module contains several models including the momentum and charge-exchange collisions for ions and neutrals, Coulomb collisions for charged species, elastic collisions for neutrals with the variable hard-sphere (VHS) collision method, and (interspecies and intraspecies) fractional collision method. Most of the DSMC model in this module uses the no-time counter (NTC) algorithm, which first determines the maximum number of collisions N_{coll} and then uses the probability function P to sort out the fictitious collisions. Assuming that all the particles have the same specific weight (the ratio between the number of physical to the number of computational particles), the maximum number of collisions is given by the following equation.

$$N_{coll} = N_{pair} w \langle \sigma v_r \rangle_{max} dt / \Delta V \quad (3.12)$$

where w is the specific weight, σ is the collision cross-section, v_r is the relative velocity, dt is the time-step, and ΔV is the cell volume. Also, N_{pair} is the total number of possible pairs in a given cell, which is expressed as $N_{pair} = N(N-1)/2$ or $N_{pair} = N_\alpha N_\beta$ when pairs are chosen from the same or different population, respectively. Here N is the number of particles in a cell, and N_α and N_β are the number of particles of species α and β , respectively.

The next step involves picking N_{coll} collision pairs and altering the velocities if the collision is to take place. The collision calculation between particles i and j is executed if the probability function P is greater than a random number U between 0 and 1.

$$P_{ij} = \frac{\langle \sigma v_r \rangle_{ij}}{\langle \sigma v_r \rangle_{max}} > U \quad (3.13)$$

During this calculation, $\langle \sigma v_r \rangle_{max}$ is updated for subsequent time steps. The post-collision velocity depends on the scattering model. The most commonly used one is the variable hard-sphere model, utilizing the isotropic scattering just as the hard-sphere model but adjusting the cross-section to match the viscosity data. Therefore, this model yields a computationally efficient method and is considered as a phenomenological model. The VHS model is implemented in `MSPDistDSMCOp` (`MSPDistDSMCCPUOp`). There are slightly more sophisticated models including variable soft sphere (VSS), variable sphere (VS)[27], and generalized soft-sphere (GSS)[21] to capture the diffusion and viscosity cross-sections. As the gas becomes less dense, these phenomenological models can no longer produce accurate results, as the data to fit the coefficients are rarely available if any and not enough collisions occur in this regime. `MSPDistDSMCClassicalOp` uses a collision model based on the classical scattering theory with ab initio interatomic potential [6] [7], which provide an accurate scattering model in very rarefied simulations such as a plume. `MSPDistDSMCClassicalOp` also includes the charge-exchange (CEX) reactions. `MSPDistDSMCCEXOp` can be considered as a simplified version of `MSPDistDSMCClassicalOp`, in which only the charge (and no momentum) is exchanged between the collision pair to approximate the CEX reaction.

More advanced models that take variable weight particles are also included in this module (`MSPDistFDSMCCEXOp` and `MSPDistFDSMCCPUOp`). In these models, N_{coll} as well as P_{ij} are modified and can be found in [26] and [5]. Finally, `MSPDistDSMCCoulombNanbuOp` implements the Coulomb collisions between charged particles based on [30]. This model is different from the others in that all the charged particles are collided.

3.3 Field

The Field module is responsible for calculating generic field operations, e.g. gradients, arithmetic operations between two field variables, array creation, and generic initialization.

The field module contains a collection of support algorithms that act directly on field variables. Operations such as Poisson solvers, gradient calculators, boundary conditions, and field initialization ops are stored in this module.

- Operators that start with "Logical" only work on structured meshes.
- Operators that start with "UField" only work on unstructured meshes.
- Operators that start with "Field" will work with both structured and unstructured meshes.

Note that surface meshes are considered to be unstructured meshes.

3.3.1 Arithmetic Operations

The list of arithmetic operators and available operations are shown in the tables below.

Table 3.1 List of arithmetic operators

Operator	Description
FieldSetOp	3.2
FieldArithmeticOp	3.3
FieldArithmeticConstOp	3.4
FieldArithmeticCoeffOp	3.5
FieldMathFunctionOp	3.6
FieldMathFunctionTwoArgOp	3.7
FieldMathFunctionThreeArgOp	3.8
FieldBlendTimeOp	3.9
FieldEvaluateEqnOp	3.10 {

Table 3.2 List of Operations Available in FieldSetOp

Operation	Math equation
SET	$f_1 = C \cdot f_2 \quad (f_1 = C)$
CEIL	$f_1 = C$ if $f_2 > C$, otherwise $f_1 = f_2$
FLOOR	$f_1 = C$ if $f_2 < C$, otherwise $f_1 = f_2$
ABSCEIL	$f_1 = \text{sign}(f_2) \cdot C$ if $ f_2 > C$
ABSFLOOR	$f_1 = \text{sign}(f_2) \cdot C$ if $ f_2 < C$
CEILF	$f_1 = C \cdot f_2$ if $f_1 > C \cdot f_2$
FLOORF	$f_1 = C \cdot f_2$ if $f_1 < C \cdot f_2$
FIELD_NAME= f_1 , FIELD_SRC_NAME= f_2 , VALUE=C	
If FIELD_SRC_NAME is undefined, f_1 is used in place of f_2 (or the equation in parentheses is evaluated for the SET operation).	

Table 3.3 List of Operations Available in FieldArithmeticOp

Operation	Math equation
ADD	$f_1 = f_2 + f_3$ ($f_1 = f_2 + f_1$, $f_1 = f_1 + f_3$)
SUB	$f_1 = f_2 - f_3$ ($f_1 = f_2 - f_1$, $f_1 = f_1 - f_3$)
MUL	$f_1 = f_2 \cdot f_3$ ($f_1 = f_2 \cdot f_1$, $f_1 = f_1 \cdot f_3$)
DIV	$f_1 = f_2 / f_3$ ($f_1 = f_2 / f_1$, $f_1 = f_1 / f_3$)
DIV2	$f_1 = f_3 / f_2$ ($f_1 = f_1 / f_2$, $f_1 = f_3 / f_1$)
FIELD_DST_NAME= f_1 , FIELD_SRC_NAME=FIELD_SRCB_NAME= f_2 , FIELD_SRCC_NAME= f_3	
If FIELD_SRCB_NAME or FIELD_SRCC_NAME is undefined, equations in parentheses are evaluated.	

Table 3.4 List of Operations Available in FieldArithmeticConstOp

Operation	Math equation
ADD	$f_1 = f_2 + C$
SUB	$f_1 = f_2 - C$
SUB2	$f_1 = C - f_2$
MUL	$f_1 = f_2 \cdot C$
DIV	$f_1 = f_2 / C$
DIV2	$f_1 = C / f_2$
FIELD_DST_NAME= f_1 , FIELD_SRC_NAME= f_2	
If FIELD_SRC_NAME is undefined, f_1 is used in place of f_2 .	
CONSTANT_NAME can take constants shown in the 3.10 below. It also supports "time", "dt" and "n_domains", which are the simulation time, time-step, and the number domains, respectively.	

Table 3.5 List of Operations Available in FieldArithmeticCoeffOp

Operation	Math equation
ADD	$f_1 = C_2 \cdot f_2 + C_3 \cdot f_3$
SUB	$f_1 = C_2 \cdot f_2 - C_3 \cdot f_3$
MUL	$f_1 = C_2 \cdot f_2 \cdot f_3$
DIV	$f_1 = C_2 \cdot f_2 / f_3$
DIV2	$f_1 = C_2 \cdot f_3 / f_2$
FIELD_DST_NAME= f_1 , FIELD_SRC_NAME=FIELD_SRCB_NAME= f_2 , FIELD_SRCC_NAME= f_3	
If FIELD_SRCB_NAME or FIELD_SRCC_NAME is undefined, f_1 is used in place of f_2 or f_3 .	

Table 3.6 List of Operations Available in FieldMathFunctionOp

Operation	Math equation
-----------	---------------

SQRT	$f_1 = \sqrt{f_2}$
EXP	$f_1 = \exp(f_2)$
POW	$f_1 = f_2^C$
LN	$f_1 = \ln(f_2)$
LOG10	$f_1 = \log 10(f_2)$
LOG	$f_1 = \log(f_2, C)$
SIN	$f_1 = \sin(f_2)$
COS	$f_1 = \cos(f_2)$
TAN	$f_1 = \tan(f_2)$
ABS	$f_1 = \text{abs}(f_2)$
ERF	$f_1 = \text{erf}(f_2)$
FIELD_DST_NAME= f_1 , FIELD_SRC_NAME= f_2 , NUMBER= C	
If FIELD_SRC_NAME is undefined, f_1 is used in place of f_2 .	

Table 3.7 List of Operations Available in FieldMathFunctionTwoArgOp

Operation	Math equation
MAX	$f_1 = \max(f_2, f_3)$
MIN	$f_1 = \min(f_2, f_3)$
MAG	$f_1 = \sqrt{f_2^2 + f_3^2}$
FIELD_DST_NAME= f_1 , FIELD_SRC_NAME=FIELD_SRCB_NAME= f_2 , FIELD_SRCC_NAME= f_3	
If FIELD_SRCB_NAME or FIELD_SRCC_NAME is undefined, f_1 is used in place of f_2 or f_3 .	

Table 3.8 List of Operations Available in FieldMathFunctionThreeArgOp

Operation	Math equation
MAX	$f_1 = \max(f_2, f_3, f_4)$
MIN	$f_1 = \min(f_2, f_3, f_4)$
MAG	$f_1 = \sqrt{f_2^2 + f_3^2 + f_4^2}$
FIELD_DST_NAME= f_1 , FIELD_SRCB_NAME= f_2 , FIELD_SRCC_NAME= f_3 , FIELD_SRCD_NAME= f_4	
If FIELD_SRCB_NAME, FIELD_SRCC_NAME, or FIELD_SRCD_NAME is undefined, f_1 is used in place of f_2 , f_3 , or f_4 , respectively.	

Table 3.9 List of Operations Available in FieldBlendTimeOp

Operation	Math equation
AVERAGE	$f_1 = \frac{\Delta t f_2 + (t - \Delta t) f_3}{t}$

EXP	$f_1 = f_2(1 - \exp(-dt/hl) + f_3 \exp(-dt/hl)$
FIELD_DST_NAME= f_1 , FIELD_SRC_NAME=FIELD_SRCB_NAME= f_2 , FIELD_SRCC_NAME= f_3 , HALF_LIFE_ $\leftrightarrow$ SRCB_COEFF=hl	
If FIELD_SRCC_NAME is undefined, f_1 is used in place of f_3 .	

Table 3.10 Available math operations, functions, and constants^a

Operations	arithmetic: + (addition), - (subtraction), * (multiplication), / (division)
	% (modulus)
	^ (exponent)
	comparison: == (equal to), != (not equal to), < (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to)
	&& (and)
	(or)
Functions	powers: sqrt (square-root), exp (exponential)
	logarithm: ln (natural logarithm), log (logarithm of base 10)
	abs (absolute value)
	trigonometric: sin (sine), cos (cosine), tan (tangent)
	rounding: ceil (round up to the nearest integer), floor (round down to the nearest integer), round (round to the nearest integer)
	max (maximum of two arguments)
	min (minimum of two arguments)
Constants ^b	MKS_c (speed of light)
	MKS_eps0 (vacuum permittivity)
	MKS_mu0 (vacuum permeability)
	MKS_e (elementary charge)
	MKS_me (electron mass)
	MKS_mp (proton mass)
	MKS_amu (atomic unit)
	MKS_mec (relativistic momentum scale)
	MKS_mec2 (relativistic energy scale)
	MKS_Rb (Boltzmann constant)
	MKS_Avo (Avogadro number)
	MKS_kb (Boltzmann constant)
	MKS_hPlanck (Planck constant)
	MKS_hbar (Planck constant/(2*pi))
	MKS_mole (mole)
	MKS_atm (1 atm in Pascal)
	MKS_eV_K (1 eV temperature in Kelvin)
	MKS_Rcm (1/cm to Joule)
	MKS_KcalJ (Kcal to Joules)
	MKS_cm, MKS_cm2, MKS_cm3, MKS_us
	MKS_TeraHz
	PI_d (Archimedes constant)

^a Field data names in curly parentheses, coordinates (x,y,z), velocity (vx,vy,vz), time-step (dt), and time (t) can also be used in FieldEvaluateEqnOp.

^b See constant.h for values.

3.4 Geometry

The Geometry module handles the import of meshes and their properties.

3.4.1 Introduction

The Geometry module provides support for coordinate systems, structured meshes, unstructured volume meshes, unstructured surface meshes, and structured-surface mesh intersection. These classes are some of the oldest in TURF and were created before the syntax style was formalized.

3.4.2 Structured Mesh

Structured mesh support is provided by the gSMesh class. This class has the capability to support both Cartesian and curvilinear meshes but TURF does not currently take advantage of the curvilinear capabilities.

Of note is that a two-dimensional mesh is nothing more than a 3D mesh with a distance of one in the z direction. Similarly, a 1D mesh is a 3D mesh with a distance of one in the y and z directions. This is done to make the particle push consistent across one-, two- and three-dimensions.

3.4.3 Unstructured Mesh

Unstructured mesh support, both volumetric and surface, is provided by the gUMesh class.

Volumetric Meshes

Volumetric meshes can either be two- or three-dimensional and supports mix element meshes, e.g. a mesh that contains both triangles and squares in two-dimensions or tetrahedral and prisms in three-dimensions.

The nomenclature used to define an unstructured mesh is given from the standpoint of a CFD code. So a cell is the geometric shape used for conservation, a face is a hyperplane that enforces flux, and nodes define discrete points in space that makes up a face. In a 3D mesh the cell could be a tetrahedral, box, pyramid, or prism; a face could be a square or a triangle, and nodes are the vertices of a face. In a 2D mesh cells consists of squares or triangles, faces are lines, and nodes are each end of a face.

This nomenclature was chosen so that a consistent naming procedure can be used by both 2D and 3D CFD codes. As a side note, this makes a lot of sense from a CFD perspective.

Surface Mesh

A surface mesh is essentially the skin of a three-dimensional volume mesh and only support triangular elements, square elements are not permitted. Square elements are problematic as they over define a plane.

Unstructured Mesh Import

- Abaqus - 3D surface meshes.
- Cart3D - 3D surface meshes.
- Exodus - 2D/3D volume meshes.
- OFF - 3D surface mesh, no connectivity information.
- SMesh - Reads in a structured smesh and converts to an unstructured mesh.

Abaqus format

The Abaqus format is the primary format used to import 3D surface meshes into TURF, as such, TURF is designed around the capabilities that this format provides. Please refer to the tutorials on how to generate meshes that TURF can import.

Cart3D format

The Cart3D mesh format is defined [here](#). The wetted surface format should be used because the connectivity between cells is guaranteed.

TURF keeps track of boundaries and surface/cell groups through its name, not the number that is was added. The Cart3D format does not natively allow for named groups (components in their file format) to accommodate this, TURF will set each surface's name to the number listed in the file (e.g. "1", "2", ...) for the configuration, wetted, and annotated format and will assign all cells to the group "1" in the case of a single component. An alternative is to include a file "<meshname.i.tri>.component" with the following format,

Group# Name

e.g.

1 SolarPannel

2 SCBuss

3 Dish_main

Exodus format

The Exodus file format is used to read in 2D and 3D volume meshes and requires linking with the external NETCDF library when compiling TURF.

OFF format

The OFF file format can be used to import 3D surface meshes. This format is popular in 3D graphics programs but does not provide any connectivity information or cell groups so it can only be used to bounce particles off of. This format should be used as a last resort when testing out new meshes.

SMesh format

This will convert an SMesh to the UMesh file format.

3.5 Particle

The Particle module is responsible for handling the creation, advection, sorting, and deletion of particles. Support for the single species particle distribution has been superseded by the multi-species particle distribution. Since TURF is a multi-purpose framework, background tasks like particle deletion must be manually enabled.

3.5.1 Introduction

The particle module contains operators that model non-Maxwellian flow using the Particle-In-Cell (PIC) approach. For a complete listing of available operators, please see [Particle](#).

3.5.2 Theory

The governing equation used for non-equilibrium flow in the presence of electric and magnetic fields is:

$$\frac{\partial \mathbf{f}}{\partial t} + \mathbf{v} \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{x}} + \frac{q}{m} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{v}} = C(\mathbf{f})$$

where $\mathbf{f}$ is the distribution function, $\mathbf{v}$ is the velocity, q is the charge, m is the mass, $\mathbf{E}$ is the electric field, $\mathbf{B}$ is the magnetic field, and C is the collision function.

The PIC method treats the distribution function of as a series of discrete macro particles, i , whose position and velocity are defined by,

$$\begin{aligned} m_i \frac{d\mathbf{v}_i}{dt} &= q_i (\mathbf{E} + \mathbf{v}_i \times \mathbf{B}) \\ \frac{d\mathbf{x}_i}{dt} &= \mathbf{v}_i \end{aligned}$$

3.5.3 Current Capabilities

Particle Creation

Particle creation is handled by the MSPDistInitOp operator. This operator allows the user to specify the maximum number of particles in the distribution per domain (MAX_NP) and the species present (SPECIES_NAMES) in the distribution. **Note** that the maximum number of particles present is the total number of particles, not the number of particles per species. Alternatively, you can use the keyword GLOBAL_MAX_NP to specify the maximum number of particles for all computational domains. The GLOBAL_MAX_NP keyword is useful when using strong scaling.

Particle Generation

Particles can be generated in a number of ways. The first is through the MSPDistBoxICOp operator that allows macro particles to be generated in a particular location, (x, y, z), at a particular temperature. The particles velocity will be sampled from a Maxwellian distribution. This operator is typically used as an initial condition. A similar option is the MSPDistBoxICStreamOp operator that allows for the particles to be generated with an applied electric field. This allows the particles to move as soon as they are created which is useful when the particles are generated during the simulation.

Another option for particle generation is to create them at a boundary. There are several operators that allow this including, MSPDistNormalMaxwellianOp, MSPDistNormalMaxwellianStreamOp, MSPDistSourceCosineOp, MSPDistSourceRPAOp, and MSPDistSourceVDFOp. These operators create particles at a surface, either virtual or real, by sampling from a VDF. A virtual surface is a disk defined by a center, radius, and normal vector; a real surface will generate particles from each triangle on that surface.

One of the most important parameters is the REAL_TO_COMPUTATIONAL keyword. This variable controls how many physical particles are in a computational macro-particle. Smaller values will include fewer physical particles per macro-particle, which decrease the numerical noise and increase the computational cost. There are no set rules to determine the optimum value for this parameter but here are some guidelines.

- For steady state simulations using one of the box IC operators, each cell should have a minimum of 5 macro-particles per cell.
- For steady state simulations, the surface generating operators should inject a minimum of 3-5 particles per time step.
- If running a multi-species simulation with collisions each species should inject about the same number of macro-particles per time step.

Time accurate simulations require a significant increase in the number of macro-particle per cell, typically 1-2 orders of magnitude higher than a steady state simulation. For example, the collisionless shock case contains more than a thousand particles per cell; other cases tend to have about a hundred macro-particles per cell.

Particle Push

TURF currently has both electro-static and magneto-static particle pushing algorithms. The electro-static push is callable through the MSPDistESPushOp operator while the magneto-static push is callable through the MSPDistEMBorisOp operator.

3.6 Plotting

The Plotting module is responsible for exporting field data to files.

The Plotting module contains operators that saves and reads structured fields, unstructured fields, and particles to the disk. The preferred output formats are the various VTK xml file formats but there is limited support for Tecplot. The VTK files can be opened with Paraview and Visit while the Tecplot files can only be opened by Tecplot. The VTK file format is preferred for two reasons, first the file format is open and well defined, and second there is better support for distributed output files allowing each computational domain to write its own output file. Neither the VTK or Tecplot output routines requires linking with external libraries but the VTK file format can link with the zlib library to compress the output, saving disk space.

There are nine main VTK writers:

- Logical "Structured" Field writers
 - **LogicalAMRFieldWriteVTKOp** writes 3D field data for adaptive mesh refinement meshes.
 - **LogicalFieldWriteVTKOp** is the main 3D field data writer for structured meshes.
 - **LogicalFieldWriteVTK2DOp** is similar to **LogicalFieldWriteVTKOp** but saves a 2D slice of a 3D field.
 - **LogicalFieldWriteVTKROp** is similar to **LogicalFieldWriteVTKOp** but saves in the rectilinear format. This was written to support the Vlasov module and shouldn't be used for saving field data.
- Particle Writer
 - **MSPDistWriteVTKOp** writes particle data to a file.
- Sample Writer
 - **SampleAMRFieldWriteVTKOp** is similar to **LogicalAMRFieldWriteVTKOp** but is intended to save all sampling data associated with a PIC Particle. Useful for saving the exact state of a system for future import by **SampleAMRFieldReadVTKOp**
 - **SampleFieldWriteVTKOp** is the same as **SampleAMRFieldWriteVTKOp** but for use by non-AMR fields.
- Unstructured Field Writer
 - **UFieldWriteVTKOp** saves field data associated with an unstructured field.
- UPC Particle Writer
 - **UPCWriteVTKOp** saves UPIC particle data, can only be used with a UPC particle.

There are seven VTK readers associated with the above writers, their purpose is to read in the files saved by their equivalent Write Ops and transfer that data to their associated object (i.e. **MSPDist**, **LogicalField**, **UField**,...). The associated readers are: **ASMeshReadVTKOp** **LogicalFieldReadVTKOp** **MSPDistReadVTKOp** **MSPDistReadVTKSplitOp** **SampleAMRFieldReadVTKOp** **SampleFieldReadVTKOp** **UFieldReadVTKOp**. There are two Ops that can write Tecplot files, **LogicalFieldWriteTecplotOp** & **UFieldWriteTecplotOp**, which are the Tecplot equivalent to **LogicalFieldWriteVTKOp** and **UFieldWriteVTKOp**.

There are five additional plotting ops that haven't been covered yet.

- **LogicalFieldWrite1DOp** writes 1D data saved on a structured field.
- **LogicalFieldCatalystOp** and **VolumeRenderOp** are Ops that allow for real time visualization. They are not actively used and should be used with caution.
- **ProbePointOp** Saves the time history at a single point in a Logical field.
- **LogicalVlasovFluidWrite1DProbeVDFOp** similar to **ProbePointOp** but for use with a Vlasov object.

3.7 Probe

The Probe module allows the user to place a virtual probe in the simulation.

The Probe module provides routines that extract particle information from a specific region of space or surface. These probes are intended to mimic the behavior of probes used in plasma experiments. The Retarding Probe Analyzer (RPA) and Faraday probe have been implemented as well as a generic Velocity Distribution Function (VDF) probe.

There are two different Ops that collect probe information from a region, `MSPDistProbeStageSphericalOp` and `MSPDistProbeFixedOp`. Both of these Ops can mimic a Faraday and RPA probe while the `MSPDistProbeStageSphericalOp` can also collect particle VDF. The difference between the two is how they specify the region of collection. `MSPDistProbeStageSphericalOp` defines a hemispherical region and `MSPDistProbeFixedOp` uses surfaces groups to collect particle info. These Ops collect probe data but do not save the info, to write the probe data to a file a different Op is needed. `ProbeStageWriteOp` writes data collected by `MSPDistProbeStageSphericalOp`, and `ProbeFixedWriteOp` writes data collected by `MSPDistProbeFixedOp`.

3.8 Surface

The Surface module provides operators that model the interaction of particles with a surface, including outgassing.

3.8.1 Introduction

The surface interactions module handles the collision between a particle and a surface. TURF currently handles four different possibilities:

- Reflection, with or without species changing, e.g., Xenon Ion -> Xenon neutral.
- Sticking, when a particle sticks to a surface.
- Virtual, particle moves through the surface but can collect information as it passes through. This is useful when modeling probes.
- Sputtering, e.g Xenon Ion + Beam dump (Carbon) -> Carbon (gas) + Xenon.

3.8.2 Particle-Surface Interaction

For both the surface reflection and sputtering, source, target, and product materials are specified. The specific weight ratio (`spwt_ratio`) is the ratio of product to source particle weight, where the particle weight is the ratio of physical to computational particles. A sputter weight ratio less than one creates more computational particles. For surface impacts, a sputter weight ratio of 1 is normal but for sputtering a value much less than 1 may be required. For sputtering values as small as 10^{-6} are not uncommon but it is recommended to start with values closer to unity, e.g. 0.1.

Reflection

For surface reflection, four coefficients are defined; C_{stick} , C_{rest} , C_{accom} , and C_{diff} are coefficients of sticking, restitution, thermal accommodation, and diffuse reflection, respectively.

If $U < C_{\text{stick}}$ where U is a random number between 0 and 1, then the particle is killed and is no longer tracked. A C_{stick} of 0 will reflect all particles and a value of 1 will absorb all particles.

Otherwise, the particle is reflected off the surface. The rest of coefficients are used to smoothly mix the velocity direction ($\hat{v}$) and magnitude (v) between specular and diffuse reflections.

$$\begin{aligned}\hat{v} &= \hat{v}_{\text{spec}} + C_{\text{diff}}(\hat{v}_{\text{diff}} - \hat{v}_{\text{spec}}) \\ v &= C_{\text{rest}}v_{\text{spec}} + C_{\text{accom}}(v_{\text{diff}} - C_{\text{rest}}v_{\text{spec}})\end{aligned}$$

Here, the velocity due to the diffuse reflection, $\hat{v}_{\text{diff}}$, is computed based on the wall temperature and by the Box-Muller algorithm.

Angular Distribution

TURF supports the cosine and Zhang angular distribution models for the sputtered particles.

Model	Key	Required Coefficients	Reference
Cosine	COS	V	
Zhang	ZHANG	V and C_0	[37]

Cosine The `Cosine` angular distribution results in peak atom fluxes along the surface normal of the target. In the `Cosine` model, angles are randomly sampled using the equation [17]

$$\begin{aligned}\sin \phi &= \sqrt{U_1} \\ \cos \phi &= \sqrt{1 - \sin^2 \phi} \\ \theta &= 2\pi U_2\end{aligned}$$

Where U_1 and U_2 are random numbers. This results in the polar angular probability distribution function,

$$g(\phi) = \sin 2\phi$$

This model requires the sputtered particle speed V as an input.

Zhang This model uses a modification to the redeposition algorithm developed by Zhang [37]. The particular feature of the modified Zhang model is that it assumes a data-driven fit for the E^* coefficient rather than using a physically motivated sputtering threshold energy. This is a relatively minor change so it is possible to use the modified Zhang sampling model to replicate the results of classical Yamamura sputtering by simply using the physically motivated E^* coefficient instead.

$$\begin{aligned}f(E, \beta, \alpha, \phi) &= \frac{1}{1 - \cos(\beta)\sqrt{\frac{E^*}{E}}} \frac{\cos(\alpha)}{\pi} \left[1 - \frac{1}{4} \sqrt{\frac{E^*}{E}} \left(\cos(\beta)\gamma(\alpha) + \frac{3}{2} \pi \sin(\beta) \sin(\alpha) \cos(\phi) \right) \right] \\ \gamma(\alpha) &= \frac{3 \sin^2(\alpha) - 1}{\sin^2(\alpha)} + \frac{\cos^2(\alpha)(3 \sin^2(\alpha) + 1)}{2 \sin^3(\alpha)} \ln \left(\frac{1 + \sin(\alpha)}{1 - \sin(\alpha)} \right)\end{aligned}$$

where β is the polar angle of incident particle with respect to surface normal, α is the polar angle of ejected particle with respect to the surface normal, and ϕ is the azimuthal angle of the ejected particle.

3.8.3 Virtual

A `virtual` surface does not impede the motion of particles and is used

- to save the flux through a particular region or
- to remove part of a surface without remeshing.

3.8.4 Sputtering

Sputtering occurs when a particle hits a surface and causes part of that surface to break off. Ten sputtering models, listed below, have been implemented in TURF.

Model	Key	Required Coefficients	Nomenclature	Reference
Constant	CONST	C_0	SPUT_CO	
Linear	LINEAR	C_0	SPUT_CO SPUT_C1	
Matsunami	MATSU	C_0 and C_1	SPUT_CO SPUT_C1	[34], [28]
Yamamura	YAMA	$C_0, C_1, C_2, C_3, C_4,$ and C_5	SPUT_CO SPUT_C1 SPUT_C2 SPUT_C3 SPUT_C4 SPUT_C5	[35],[13]
Kannenbergl	KANNEN	$C_0, C_1, C_2, C_3,$ and C_4	SPUT_CO SPUT_C1 SPUT_C2 SPUT_C3 SPUT_C4	[18]
Roussel	ROUSSEL	C_0 and C_1	SPUT_CO SPUT_C1	[32]
Garnier	GARNIER	$C_0, C_1, C_2, C_3, C_4,$ and C_5	SPUT_C0 SPUT_C1 SPUT_C2 SPUT_C3 SPUT_C4 SPUT_C5	[16]
Pencil	PENCIL	$C_0, C_1, C_2, C_3,$ and C_4	SPUT_C0 SPUT_C1 SPUT_C2 SPUT_C3 SPUT_C4	[31]

Model	Key	Required Coefficients	Nomenclature	Reference
Eckstein	ECKSTEIN	Q, λ, μ, E_{th} , and ϵ	SPUT_Q SPUT_LAMBDA SPUT_ $\leftrightarrow$ MU SPUT_ETH	[36]
Hofer	HOFER	C_0, E_{th}	SPUT_C0 SPUT_ETH	

In addition to the ten normal-yield sputtering models, 5 angular dependencies have also been added. An angular dependency can be included in the yield by use of an underscore. For example, to invoke a normal Yamamura yield with a Cheng angular dependency, the key is defined as `model : YAMA_CHENG`.

Model	Key	Required Coefficients	Nomenclature	Reference
Poly4	POLY4	C_1, C_2 , & C_3	ANG_C1, ANG_C2, ANG_C3	
Bermudez	BERM_P4	C_1, C_2, C_4, C_5 , & C_6	ANG_C1, ANG_C2, ANG_C4, ANG_C5, ANG_C6	[9]
Cheng	CHENG	C_0, C_1, C_2, C_3, C_4 , & C_5	ANG_US ANG_FS ANG_R0	[13]
Eckstein	_ECKSTEIN	f, b , & c	ANG_F ANG_B ANG_C	[36]
Wei	WEI	β/α & a/α	ANG_BETA_ALPHA ANG_A_ALPHA	[36]

Some sputtering models have built in angular dependencies or skip the angular dependency. These can be used with the keys listed in the next table:

Model	Normal Yield	Angular Yield	Key
Yamamura-Cheng	YAMA	CHENG	YAMA_CHENG
Hofer-Bermudez	GARNIER	BERM_P4	GARNIER_BERM_P4
Yamamura- (normal only)	YAMA	(none)	YAMA_NOANGLE

Constant

This is the simplest among all the sputter models implemented in TURF. The total yield Y is simply a constant value specified in the input.

$$Y = C_0$$

Matsunami

Although we call this model "Matsunami", the original work was actually published by Yamamura, Matsunami, and Itoh [34]. Only two inputs (C_0 and C_1) are required for the model. When initializing TURF, four constants (A_0, A_1, A_2 , and A_3) are precomputed based off the inputs and material properties.

$$\begin{aligned}
 A_0 &= \frac{0.03255}{Z_1 Z_2 (Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_2}{M_1 + M_2} \\
 A_1 &= 0.079 \frac{(M_1 + M_2)^{2/3}}{M_1^{3/2} M_2^{1/2}} \frac{Z_1^{2/3} Z_2^{1/2}}{(Z_1^{2/3} + Z_2^{2/3})^{3/4}} \\
 A_2 &= U_s (1.9 + 3.8(M_2/M_1)^{-1} + 0.134(M_2/M_1)^{1.24}) \\
 A_3 &= \alpha^* Q K
 \end{aligned}$$

where

$$\begin{aligned}
 \alpha^* &= 0.08 + 0.164(M_2/M_1)^{0.4} + 0.0145(M_2/M_1)^{1.29} \\
 K &= 8.478 \frac{Z_1 Z_2}{(Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_1}{M_1 + M_2}
 \end{aligned}$$

Here U_s and Q are used in place of C_0 and C_1 , respectively, in order to keep the symbols used in the original reference. These parameters can be found in Ref. [34]. Z is the atomic number in amu and M is the atomic mass in amu. Subscripts 1 and 2 denote source and target particles. Then, the total yield is computed for each sputtering event by

$$Y = 0.42 \frac{A_3 s_n}{U_s (1 + 0.35 U_s s_e)} \left[1 - \sqrt{\frac{A_2}{E}} \right]^{2.8}$$

where s_n and s_e are

$$s_n = \frac{3.441 \sqrt{\epsilon} \ln(\epsilon + 2.718)}{1 + 6.355 \sqrt{\epsilon} + \epsilon(-1.708 + 6.882 \sqrt{\epsilon})}$$

$$s_e = A_1 \sqrt{\epsilon}$$

where $\epsilon = A_0 E$ and E is the energy of incident particle in electronvolt. Matsunami should not be used with molecules.

Yamamura

Yamamura and Tawara [35] upgraded the model originally developed by Yamamura, Matsunami, and Itoh (referred as Matsunami model herein). Six inputs (C_0 to C_5) are required for this model, and seven constants are stored during the initialization stage.¹

$$B_0 = 0.42 \alpha^* Q K / U_s$$

$$B_1 = \frac{W}{1 + (M_1/7)^3} 0.079 \frac{(M_1 + M_2)^{3/2}}{M_1^{3/2} M_2^{1/2}} \frac{Z_1^{2/3} Z_2^{1/2}}{(Z_1^{2/3} + Z_2^{2/3})^{3/4}}$$

$$B_2 = E_{th}$$

$$B_3 = \frac{0.03255}{Z_1 Z_2 (Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_2}{M_1 + M_2}$$

$$B_4 = s$$

$$B_5 = E_{th, \theta} = 1.5 \frac{U_s}{\gamma} \left[1 + 1.38 \left(\frac{M_1}{M_2} \right)^h \right]^2$$

$$B_6 = (a/R_0)^{1.5} \sqrt{Z_1 Z_2 / (Z_1^{2/3} + Z_2^{2/3})^{1/2}}$$

$$B_7 = f_s$$

where

$$\alpha^* = \begin{cases} 0.249(M_2/M_1)^{0.56} + 0.0035(M_2/M_1)^{1.5} & M_1 \leq M_2 \\ 0.0875(M_2/M_1)^{-0.15} + 0.165(M_2/M_1) & \text{Otherwise} \end{cases}$$

$$K = 8.478 \frac{Z_1 Z_2}{(Z_1^{2/3} + Z_2^{2/3})^{1/2}} \frac{M_1}{M_1 + M_2}$$

$$\gamma = \frac{4M_1 M_2}{(M_1 + M_2)^2}$$

$$E_{th} = \begin{cases} U_s \frac{1.0 + 5.7 M_1/M_2}{\gamma} & M_1 \leq M_2 \\ U_s \frac{6.7}{\gamma} & \text{Otherwise} \end{cases}$$

¹Yamamura's paper has typo in his equation where the coefficient should be 0.042 in the equation for Y(E)

$$h = \begin{cases} 0.834 & M_1 \leq M_2 \\ 0.18 & \text{Otherwise} \end{cases}$$

$$a = \begin{cases} \frac{0.4685}{(Z_1^{1/2} + Z_2^{1/2})^{2/3}} & Z_1 > 3 \\ \frac{0.4685}{(Z_1^{2/3} + Z_2^{2/3})^{1/2}} & \text{Otherwise} \end{cases}$$

Here U_s , Q , W , s , f_s , R_0 are used in place of C_0 , C_1 , C_2 , C_3 , C_4 , and C_5 , respectively. The input parameter for various combinations of source and target materials can be found in Table 4 and 17 of Ref. [34] and Table 1 of Ref. [35]. Then, the yield for every incident particle is computed by

$$Y(E) = \beta \frac{B_0 s_n}{1 + B_1 \epsilon^{0.3}} \left(1 - \sqrt{\frac{B_2}{E}} \right)^{B_4}$$

where

$$\epsilon = B_3 E, \quad s_n = \frac{3.441 \sqrt{\epsilon} \log(\epsilon + 2.718)}{1.0 + 6.355 \sqrt{\epsilon} + \epsilon(-1.708 + 6.882 \sqrt{\epsilon})}$$

$$\nu = 1 - \sqrt{B_5/E}, \quad f = B_7 \frac{1 + 2.5(1 - \nu)}{\nu}$$

$$\sigma = f \cos(\theta_{\text{opt}}), \quad \theta_{\text{opt}} = 0.5\pi - 4.9916(B_6/\sqrt{E})^{0.45}$$

$$\beta = \frac{\exp(-\sigma(1/\cos \theta_i - 1))}{\cos^f \theta_i}$$

Here, E is the incident particle energy in electronvolt. Yamamura should not be used with molecules.

Kannenberg

The Kannenberg sputter yield [18] is modeled as a function of energy, E , and incident angle, β . The total yield in atoms/ion is given as follows.

$$Y(E, \beta) = S(E)g(\beta)$$

$$S(E) = C_0 + C_1 E$$

$$g(\beta) = C_2 \cos \theta + C_3 \cos^2 \theta + C_4 \cos^3 \theta$$

If the energy threshold ($E_{th} = -a/b$) is negative, the fit is rejected on physical grounds and is instead constrained to be zero. The reference paper shows a lot of data for ITO but has no coefficients in it. Assume that all AFRL Kannenberg data fits are Lockheed proprietary.

Roussel

The sputtering yield of standard coverglass (cerium-doped borosilicate) as a function of incidence angle, β , was measured and fit to a polynomial (for energy 300 eV) [32]. This resulted in the following yield where β is in degrees and Y is in atoms/ion:

$$Y(E = 300 \text{ eV}, \beta) = 0.42 - 0.0053\beta + 0.0015\beta^2 - 7 \times 10^{-6}\beta^3 - 10^{-7}\beta^4$$

The coverglass had a normal yield of 0.42 for 300 eV impact ions and a linear dependence from a threshold energy of 50 eV. This allows for separation of the yield into

$$Y(E, \beta) = S(E)g(\beta)$$

where $S(E) = -0.084 + 0.0017E$. This form is generalized as follows.

$$\begin{aligned}
Y(E, \beta) &= S(E)g(\beta) \\
S(E) &= C_0 + C_1 E \\
g(\beta) &= 1 - 0.723\beta + 11.724\beta^2 - 3.13\beta^3 - 2.57\beta^4
\end{aligned}$$

where β is redefined to be in radians. Unlike the `Kannenberg` sputter model, $g(\beta)$ is a polynomial function in β instead of $\cos \beta$. Therefore, the set of coefficients may not yield a sputter yield of zero at $\beta = 90^\circ$.

Garnier

The `Garnier` sputtering algorithm performs yield calculations according to Garnier, et al for Boron Nitride [16]. `Garnier` yield fits are simplified equations from `Yamamura` which have a separate angle dependence and energy dependence, both of which are multiplied together to find the total yield. The angular yield has a polynomial dependence on incident angle making it easier to fit experimental data. The equations for the total yield, Y , in atoms/ion is given as

$$\begin{aligned}
Y(e, \beta) &= S(E)g(\beta)C_6 \\
S(E) &= \sqrt{E} \left(1 - \sqrt{\frac{E_{th}}{E}} \right)^{2.5} \\
g(\beta, E) &= C_1 + C_2\beta + C_3\beta^2 + C_4\beta^3
\end{aligned}$$

Where E_{th} is the threshold energy and is another input parameter defined by C_5 . This model also uses a polynomial function in β instead of $\cos \beta$ such that the set of coefficients may not yield a sputter yield of zero at $\beta = 90^\circ$.

Hofer-Bermudez

The `Hofer-Bermudez` model [9] is a variation of `Garnier` model with additional constraints and free parameters.

$$\begin{aligned}
Y(e, \beta) &= S(E)g(\beta) \\
S(E) &= C_0 \sqrt{E} \left(1 - \sqrt{\frac{E_{th}}{E}} \right)^{2.5} \\
g(\beta) &= 1 + C_1\beta^{C_4} + C_2\beta^{C_5} + C_3\beta^{C_6}
\end{aligned}$$

where E_{th} , C_0 , C_1 , C_2 , C_4 , C_5 , and C_6 are free parameters and C_3 is selected such that $g(\pi/2) = 0$,

$$C_3 = -C_1 \left(\frac{\pi}{2} \right)^{C_4 - C_6} - C_2 \left(\frac{\pi}{2} \right)^{C_5 - C_6} - \left(\frac{\pi}{2} \right)^{-C_6}.$$

Pencil

In the work by `Pencil` at NASA Glenn Research Center, an SPT-100 thruster was used as a contamination source [31]. There were a lot of data about coated CMX, a ceria-doped borosilicate that is used as a cover slide material for silicon solar cells (but no coefficients for CMX). The materials with coefficients from Ref. [31] are

Normal incidence: Ag, Fe, Si, Borosil, (Quartz coverglass is studied as Si_2). The normal yield for Si must be multiplied by 0.57 to recover the normal yield for SiO_2 .

Angular incidence: Ag, Fe, SiO_2 and Borosil. These are not implemented in the code.

The general form for the `Pencil` model is as follows.

$$\begin{aligned}
Y(E, \beta) &= S(E)g(\beta) \\
S(E) &= C_0 E^{0.25} \left(1 - \frac{C_1}{E} \right)^{3.5} \\
g(\beta) &= 1 + C_2(1 - \cos(C_3\beta))^{C_4}
\end{aligned}$$

Sputter yields below the threshold energy, C_1 , are zero.

Eckstein

Eckstein introduced an empirical based correction to account for low energy sputtering data. The formulation used was taken from Yim's 2017 IEPC paper [36]. One of the difference with this implementation is the use of a reduced stopping power based on the Krypton-Carbon (KC) potential that is more accurate low impact energies.

$$Y(E) = Q s_n \frac{\left(\frac{E}{E_{th}} - 1\right)^\mu}{\frac{\lambda}{w} + \left(\frac{E}{E_{th}} - 1\right)^\mu}$$

$$w = \epsilon + 0.1728\sqrt{\epsilon} + 0.008\epsilon^{0.1504}$$

with,

$$s_n = \frac{0.5 \ln(1 + 1.2288\epsilon)}{w}$$

$$\epsilon = \frac{a_L}{Z_1 Z_2} \frac{4\pi\epsilon_0}{e^2} \frac{M_2}{M_1 + M_2} E$$

$$a_L = \left(\frac{9\pi^2}{128}\right)^{1/3} a_0 \left(Z_1^{2/3} + Z_2^{2/3}\right)^{-1/2}$$

The free parameters are Q , λ , μ , and E_{th} . a_0 is the Bohr radius (0.529 Å), e is the elementary charge, ϵ_0 is the vacuum permittivity ($8.854 \times 10^{-12} \text{ F/m}$, $1.42 \times 10^{-40} \text{ C}^2/\text{eV}\cdot\text{Å}$)

Eckstein's angular dependence model is also included in TURF. The formulation used was taken from Yim's 2017 IEPC paper [36].

$$Y'(\theta) = (\cos \theta^c)^{-f} \exp[b(1 - \cos \theta^c)^{-1}]$$

The free parameters are f , b , and c .

Poly4

The Poly4 angular dependency model has the form

$$Y(e, \beta) = S(E)g(\beta)$$

$$g(\beta, E) = 1.0 + C_1\beta + C_2\beta^2 + C_3\beta^3 + C_4\beta^4$$

where the normal sputtering yield $S(E)$ needs to be specified separately. The coefficients C_1 , C_2 , and C_3 need to be provided by the user. The value of C_4 is used to enforce $g(\pi/2) = 0$ and calculated via:

$$C_4 = -(\pi/2)^{-4} - C_1(\pi/2)^{-3} - C_2(\pi/2)^{-2} - C_3(\pi/2)^{-1}$$

Wei

The manuscript of Yim's 2017 IEPC paper [36] includes a reformulation of an angular sputter model developed by Wei. Yim's formulation reduces Wei's model to 2 parameters and provides curve fit coefficients for common materials. This re-formulation is:

$$Y' = \frac{1}{\sqrt{1 + \frac{\beta}{\alpha} \tan^2 \theta}} \exp\left(\frac{1}{2} \left(\frac{\alpha^2}{\beta} \left[1 - \frac{1}{1 + \frac{\beta}{\alpha} \tan^2 \theta}\right]\right)\right)$$

The free parameters for TURF are $\frac{\beta}{\alpha}$, and $\frac{\alpha}{\beta}$.

3.8.5 Implementation

Particle surface interactions are defined by the MSPDistSurfaceInteractionOp operator.

```

DEFINE OPERATION
  TYPE = MSPDistSurfaceInteractionOp
  NAME = Interaction_P-GST
  SURFACE_NAME = UMesh
  MSPDIST_DATA_NAME = P-GST
  UFIELD_DATA_NAME = SurfaceField
  SURF_INTERSECTION_OP_NAME = Intersection_P-GST
  FILE_NAME = surf_int.txt
  STEPS_PER_SAMPLING = 1
  VERBOSE = FALSE
END OPERATION

```

Where,

Keyword	Description
SURFACE_NAME	The name of the surface defined in the UMeshImporterOp operator.
MSPDIST_DATA_NAME	The name of the particle distribution, defined in the MSPDistInitOp operator.
UFIELD_DATA_NAME	The name of the uField object that holds the unstructured data, defined in the UFieldMSInitOp operator.
SURF_INTERSECTION_OP_NAME	The operator name of the MSPDistSugarcubeSurfIntersectionOp used.
FILE_NAME	Defines the particle species-surface interaction type in the form shown below

The surface interaction file contains two keywords formatted in a json like style. The `surface_impact` keyword defines the reflection of a particle, while the `sputtering` keyword defines the sputtering of the surface material. An example is shown below.

```

surface_impact{source:"Xe@g", target:"Al@s",product:"Xe@g",spwt_ratio:1.0,
  c_stick:0.0, c_rest:1.0, c_accom:0.9, c_diff:1.0
}
sputtering{model:"yama", source:"Xe@g", target:"Fe@s", product:"Fe@g", spwt_ratio:0.000001,
  sput_c0:4.28, sput_c1:0.75, sput_c2:1.2, sput_c3:2.5, sput_c4:1.84, sput_c5:2.28,
  emission:"zhang", emit_c0:52.2, emit_v:5000
}

```

For the surface impact a xenon ion gas, Xe@g, hits solid aluminum, Al@s, and produces xenon gas at a sputter ratio of 1, a sticking coefficient of 0 (all particles reflect), a restitution coefficient of 1, an accommodation factor of 0.9, and a diffusive factor of 1.0. The sputtering entry defines the interaction of a xenon ion gas with an iron surface and produces iron gas at a ratio of 1 to 1 million using the Yamamura model and a zhang distribution with coefficients `emit_c0` and `emit_v`.

Bibliography

- [1] J. D. Jr. Anderson. *Modern Compressible Flow with Historical Perspective*. McGraw-Hill, New York, New York, 3rd edition, 2003. [35](#)
- [2] S. J. Araki. Multiscale Coupling of Spacecraft Charging Model with Electric Propulsion Plume Simulation. *IEEE Transactions on Plasma Science*, 47(11):4898–4908, 2019. [44](#), [89](#), [92](#), [122](#)
- [3] S. J. Araki and A. C. Barrie. Electric Propulsion Plume Simulation Coupled with Spacecraft Charging. In *54th AIAA/SAE/ASEE Joint Propulsion Conference*, Cincinnati, Ohio, July 2018. AIAA 2018-4906. [44](#), [89](#), [92](#)
- [4] S. J. Araki, D. L. Bilyeu, J. W. Koo, R. S. Martin, K. Kubota, K. Koga, S. Cho, K. Kinefuchi, S. Hatta, and H. O. Ueda. Convergence of Stochastic Models for Electric Propulsion Plume Simulation. In *36th International Electric Propulsion Conference*, University of Vienna, Austria, September 2019. IEPC 2019-156. [84](#)
- [5] S. J. Araki and R. S. Martin. Interspecies Fractional Collisions. *Physics of Plasmas*, 27(033504), 2020. [84](#), [126](#)
- [6] S. J. Araki and R. E. Wirz. Ion-Neutral Collision Modeling Using Classical Scattering With Spin-Orbit Free Interaction Potential. *IEEE Transactions on Plasma Science*, 41(3):470–480, 2013. [126](#)
- [7] S. J. Araki and R. E. Wirz. Modeling of Elastic Collisions between High Energy and Slow Neutral Atoms. In *3th International Electric Propulsion Conference*, pages 1–13, Kobe, Japan, July 2015. IEPC 2015–202. [126](#)
- [8] U. Ayachit. *The ParaView Guide*. Kitware Inc., 2015. Available at www.paraview.org. [108](#)
- [9] Luis Bermudez, Kevin J. Barnhart, Christine A. Agathon-Burton, Debasis Basak, and Michale J. Glogowski. A comprehensive numerical approach to the modeling and simulation of plume interaction effects on solar electric propulsion spacecraft. In *IEPC Conference Proceedings*, volume 238. IEPC, 2017. IEPC 2017-238. [140](#), [143](#)
- [10] M. Bettencourt and A. Greenwood. Performance improvements for efficient electromagnetic particle-in-cell computation on 1000s of CPUs. *IEEE Transactions on Antennas and Propagation*, 56(8):2178–2186, August 2008. [26](#)
- [11] G. A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Clarendon Press, Oxford, 1994. [34](#), [36](#), [40](#)
- [12] T. Blacker, S. J. Owen, M. L. Staten, R. W. Quadros, B. Hanks, B. Clark, R. J. Meyers, C. Ernst, K. Merkley, R. Morris, C. McBride, C. Stimpson, M. Plooster, and S. Showman. *CUBIT: Geometry and Mesh Generation Toolkit. 15.2 User Documentation*. Sandia National Laboratory, 2016. SAND2016-1649. [101](#)
- [13] S. Cheng. Computational modeling of a hall thruster plasma plume in a vacuum tank. Master's thesis, Massachusetts Institute of Technology, 2002. [139](#), [140](#)
- [14] F. Cichocki, A. Domínguez-Vázquez, M. Merino, and E. Ahedo. Hybrid 3D Model for the Interaction of Plasma Thruster Plumes with Nearby Objects. *Plasma Sources Science and Technology*, 26(125008):1–19, 2017. [123](#)
- [15] Lord Cole. *Combustion and Magnetohydrodynamic Processes in Advanced Pulse Detonation Rocket Engines*. PhD dissertation, University of California, Los Angeles, 2012. [3](#)

- [16] Y. Garnier, V. Viel, J.-F. Roussel, and J. Bernard. Low-energy xenon ion sputtering of ceramics investigated for stationary plasma thrusters. *Journal of Vacuum Science and Technology A*, 17(6):3246–3254, November 1990. 139, 143
- [17] John Greenwood. The Correct and Incorrect Generation of a Cosine Distribution of Scattered Particles for Monte-Carlo Modelling of Vacuum Systems. volume 67, pages 217–222, 2002. 138
- [18] K. Kannenberg, V. Khayms, B. Emgushov, L. Werthman, and J. Pollard. Validation of hall thruster plume sputter model. In *37th Joint Propulsion Conference*, Salt Lake City, Utah, 2001. AIAA 2001-3986. 139, 142
- [19] Michael G. Kapper. Development of advanced numerical algorithms for modeling complex flows. Master's thesis, San Jose State University, 2005. 3
- [20] Michael G. Kapper. *A high-order transport scheme for collisional radiative and nonequilibrium plasma*. PhD dissertation, The Ohio State University, 2009. 3
- [21] Jae Gang Kim, Oh Joon Kwon, and Chul Park. Modification and expansion of the generalized soft-sphere model to high temperature based on collision integrals. 20(1):017105, 2008. 126
- [22] Lawrence Livermore National Laboratory. *Visit User's Manual*, 2005. UCRL-SM-220449. Available at <https://visit.llnl.gov>. 108
- [23] Hai Le. Development of a chemically reacting flow solver on the graphic processing unit. Master's thesis, San Jose State University, 2011. 3
- [24] Hai Le. *Hydrodynamic models for multicomponent plasmas with collisional-radiative kinetics*. PhD dissertation, University of California, Los Angeles, 2011. 3
- [25] H. W. Liepmann and A. Roshko. *Elements of Gasdynamics*. Dover Publications, Mineola, New York, 1957. 35
- [26] Robert Martin. Conservative bin-to-bin fractional collisions. *AIP Conference Proceedings*, 1786(1):090003, 2016. 126
- [27] Hiroaki Matsumoto. Variable sphere molecular model in the monte carlo simulation of rarefied gas flow. *AIP Conference Proceedings*, 663(1):358–365, 2003. 126
- [28] N. Matsunami, Y. Yamamura, Y. Itikawa, N. Itoh, Y. Kazumata, S. Miyagawa, K. Morita, R. Shimizu, and H. Tawara. Energy dependence of the ion-induced sputtering yields of monatomic solids. *Nuclear Data Tables*, 31:1–80, 1984. 139
- [29] R. L. Merlino. Understanding Langmuir Probe Current-Voltage Characteristics. *American Journal of Physics*, 75(12):1078–1085, 2007. 55
- [30] K. Nanbu. Theory of cumulative small-angle collisions in plasmas. *Phys. Rev. E*, 55:4642–4652, Apr 1997. 126
- [31] E. J. Pencil, T. Randolph, and D. Manzella. End-of-life stationary plasma thruster far-field plume characterization. In *32nd Joint Propulsion Conference*, Lake Buena Vista, Florida, 1996. AIAA 1996-2709. 139, 143
- [32] J.-F. Roussel, J. Bernard, and Y. Garnier. Numerical simulation of induced environment, sputtering, and contamination of satellite due to electric propulsion. In *Proceedings Second European Spacecraft Propulsion Conference*, pages 517–522, August 1997. 139, 142
- [33] M. K. Scharfe, J. W. Koo, and G. Azarnia. DSMC implementation of experimentally-based $\text{Xe}^+ + \text{Xe}$ differential cross sections for electric propulsion modeling. In *AIP Conference Proceedings*, volume 1333, pages 1085–1090. AIP, 2011. 81
- [34] Y. Yamamura, N. Matsunami, and N. Itoh. Theoretical studies on an empirical formula for sputtering yield at normal incidence. *Radiation Effects*, 71, 1983. 139, 140, 141, 142
- [35] Y. Yamamura and H. Tawara. Energy dependence of the ion-induced sputtering yields of monatomic solids at normal incidence. *Nuclear Data Tables*, 62:149–253, 1996. 139, 141, 142

- [36] J. Yim. A survey of xenon ion sputter yield data and fits relevant to electric propulsion spacecraft integration. In *IEPC Conference Proceedings*, volume 060. IEPC, 2017. IEPC 2017-060. [140](#), [144](#)
- [37] Z. L. Zhang and L. Zhang. Anisotropic angular distributions of sputtered atoms. *Radiation Effects and Defects in Solids*, 159:301–307, 2004. [138](#), [139](#)