REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*

2. REPORT TYPE
Technical Paper

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE
Thermophysics Universal Research Framework (TURF) Tutorial Package

5a. CONTRACT NUMBER
In-House

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Robert Martin; Justin Koo

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
Q0A5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory (AFMC)
AFRL/RQRS
1 Ara Drive
Edwards AFB, CA 93524

8. PERFORMING ORGANIZATION REPORT NO.

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory (AFMC)
AFRL/RQR
5 Pollux Drive
Edwards AFB, CA 93524

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-RQ-ED-OT-2015-108

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
PA Case Number: 15200; Clearance Date: 15 Apr 2015.

14. ABSTRACT

TURF development has been ongoing at AFRL/RQRS for over two years and has reached a stage where most of the underlying code data-structures and interfaces are at a level of maturity at which it is possible for collaborators to begin writing external modules for the framework. To facilitate bringing these collaborators up to speed in writing modules for TURF, the TURF-IR has the capabilities in place to demonstrate bare-bones kinetic and fluid simulations, including a few 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 equivalent real-world capabilities to the Coliseum/HPHall suite. As such, this software by itself cannot be used to design or analyze real systems.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:

a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Justin Koo
Unclassified	**Unclassified**	**Unclassified**	SAR	58	**19b. TELEPHONE NO** *(include area code)*

THERMOPHYSICS UNIVERSAL RESEARCH FRAMEWORK – INFRASTRUCTURE RELEASE – (TURF-IR v1.0)

ROBERT MARTIN, ERC INC, AND JUSTIN KOO, AFRL/RQRS

Contents

1. Introduction 1
2. Core Structure 1
3. Key Data-structures 2
4. Basic Functionality 2
5. Capabilities 3
6. Extended Capabilities 3
7. Conclusion 3
References 7

1. Introduction

TURF development has been ongoing at AFRL/RQRS for over two years and has reached a stage where most of the underlying code data-structures and interfaces are at a level of maturity at which it is possible for collaborators to begin writing external modules for the framework. To facilitate bringing these collaborators up to speed in writing modules for TURF, the TURF-IR has the capabilities in place to demonstrate bare-bones kinetic and fluid simulations, including a few 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 equivalent real-world capabilities to the Coliseum/HPHall suite[6, 7, 4]. As such, this software by itself cannot be used to design or analyze real systems.

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 though 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

Date: 10 Nov 2014.

Distribution A: Approved for public release; unlimited distribution; PA (Public Affairs) Clearance Number 15200 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 in parallel on an accelerator such as the GPU.

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 are 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 C-R 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 the 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.

3. 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 largely tied to the sort of data being stored (particle/field data) and the sort of mesh on which it is being stored (mesh-free/structured mesh/unstructured mesh) but are all based on a custom multi-dimensional matrix object called gMatrix, a GSObject encapsulation and extension of the matrix objects used extensively in the AFRL/RQRS's prior research codes[9, 8, 10, 5, 11]. 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 Masters theses (refs?). In addition to mesh objects, the TURF-IR includes examples of basic particle and field data objects with similar header/gMatrix compound structure.

4. Basic Functionality

For this version of the TURF-IR, the computational domain is centered around 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, or 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.

5. Capabilities

The ultimate capability of TURF-IR is really dictated by how the parts are used together. While TURF-IR is not crippled and is intended to provide reasonable parallel scaling to look at fairly computationally challenging problems. This scaling will be a primary thrust for verification in future infrastructure releases. However, the TURF-IR simply lacks the physics modules to perform real-world simulations. In particular, it does not have any of the sputtering capability nor HPHall interfaces which make the Coliseum suite a tool for realistic spacecraft-thruster interaction modeling. A similar situation exists for the fluid and Vlasov capability in the TURF-IR. They are limited to a low-order and low-dimensionality, respectively, to offer only limited capability for real-world simulation. They do provide enough framework to allow our external collaborators to add different type of high-order solvers at our discretion and to run many fundamental test cases used in evaluating numerical methods.

The visualization capability provided by TURF-IR is the ability to write various VTK output formats for structured/unstructured grids and fields as well as some line plot data output. TURF also includes a compilation flag that allows the framework to be compiled with VisIt (DoE) libSim[15] libraries for near-realtime in-situ visualization. When included, all visualizeable data-structures such as particle distributions, meshes, and field data are accessible automatically if attached to the GSObject hierarchy from the World. Because this process is automatic, access includes temporary buffered versions of the objects such as ghost and exchange particle distributions which can aid in debugging. Access to data is currently only available between iterations through the entire operation stack, but future versions of TURF may be modified to allow visualization after each operation while in debug mode to facilitate visual debugging. The libSim functionality is still experimental and currently only operational on a single MPI process, though the interface was designed for full parallel scalability.

Table 1 below provides a list of Operations included in the TURF-IR with short descriptions which are used in the accompanying tutorials covering a particle heatbath[13, 14], direct simulation Monte Carlo 1D shock [1], a 1D1V collisionless electrostatic shock [3], and a 3D3V electrostatic collisionless particle in cell test case[12].

6. Extended Capabilities

Beyond the material covered in the tutorial, several additional tutorials are currently being developed. These include a PIC and Fluid versions of the collisionless shock tutorial using the same problem setup[2], ionizing breakdown using MCC collisions with particle merging and splitting, 1D Euler shock tube and 3D Euler shock-bubble fluid test cases, 3D DSMC bow-shock examples for both geometrically prescribed bodies as well as triangulated surface meshes, and a GPU accelerated version of the heatbath tutorial. Many of the Operations required for development of these tutorials are already complete but are not yet ready to be incorporated into the infrastructure release. The intent is to transition many of these Operations into the infrastructure in the next infrastructure release. These operations may nevertheless be available to select developers as part of the thermophysics universal research framework development (TURF-DEV) package on a case-by-case basis. Brief descriptions of the operations are provided in Table 2.

In addition to the TURF-DEV operations nearing incorporation in the infrastructure, another class of TURF-DEV operations are available that are still in experimental phases of development or are potentially too specific research problem dependent to warrant inclusion in the infrastructure. Brief descriptions of these operations are included in Table 3 as reference for some future areas of development.

7. Conclusion

The TURF-IR is an interim release intended to stimulate collaboration by demonstrating code data structures and interfaces in the context of very basic fluid and kinetic simulation capabilities; however, by itself, this software cannot be used to design or analyze real systems.

Table 1. Summary of operations included in `TURF-IR`.

Module	Operation	Description
DSMC	`SPDistDSMCConstantICOp`	Initial distribution of particles inside the domain
DSMC	`SPDistDSMCOp`	DSMC collision calculation
DSMC	`SPDistDSMCSampleOp`	Blend running average of field data
DSMC	`SPDistDSMCSampleOp`	Sample particles
Field	`LogicalBCConstantOp`	Set value of cell centers in box every iteration
Field	`LogicalBCXtrapOp`	Sets a physical boundary to extrapolation
Field	`LogicalFieldAddOp`	Adds one field variable to another
Field	`LogicalFieldScalarMulOp`	Multiplies field by scalar constant
Field	`LogicalFieldSetOp`	Set field values to constant
Field	`LogicalFieldSetOp`	Initialize the field parameters
Field	`LogicalFieldVolumetricMulOp`	Multiplies or divides field data by cell volumes
Field	`LogicalGradientCellCenterOp`	Calculates the gradient of a field vector
Field	`LogicalNodeGradientOp`	Calculate node-centered gradient of cell center field
Field	`LogicalNormOp`	Calculate L^p-norm of field variable
Field	`LogicalPoissonBoltzmannStrip1DOp`	Solves for the electric field assuming Boltzmann electrons
Field	`LogicalPoissonStripOp`	Red/Black line relaxing Poisson solve
Field	`LogicalResidualOp`	Calculate residual of Poisson solve
Particle	`SPDistBCSpecOp`	Specularly reflecting boundary condition
Particle	`SPDistCellIDOp`	Find cell ID associated with particle location
Particle	`SPDistCombineOp`	Unifies the particles from different distributions
Particle	`SPDistConstantBCOp`	Add Particles in box with uniform cell density via weights
Particle	`SPDistConstantICOp`	Initialize particle distribution & add SPDistConstantBCOp
Particle	`SPDistDensityToFieldOp`	Sum real and computational particles/cell to field
Particle	`SPDistDSMCConstantBCOp`	Injection of constant weight particles
Particle	`SPDistESPushOp`	Electrostatic particle push using node electric field
Particle	`SPDistMoveOp`	Linear particle push
Particle	`SPDistPatchOp`	Transfers particles between domains
Particle	`SPDistSortOp`	Sort particles according to cell ID
Particle	`SPdistSplitOp`	Split particle distribution into two by cell ID flag
Particle	`SPDistToFieldOp`	Sum particle charges to field entry
Plotting	`LogicalFieldWrite1DOp`	Write to output files for line plots
Plotting	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots in .vts format
Plotting	`LogicalFieldWriteVTKROp`	Exports the field data in .vtr format
Plotting	`LogicalVlasov2DWriterOp`	Exports a 2D phase-space plot
Plotting	`VlasovMetricsOp`	Exports Vlasov metrics for mass and energy conservation
Plotting	`VolumeRenderOp`	Single cubic domain realtime volume rendering
Utility	`CriteriaStageOp`	Continue to next stage if quantity below criteria
Utility	`NextStageOp`	Continue to next stage
Vlasov	`LogicalBCVlasovExtrapolateOp`	Sets a velocity boundary conditions to extrapolation
Vlasov	`LogicalVlasovCalcDensityOp`	Calculates the density given a velocity distribution
Vlasov	`LogicalVlasovCalcFluidVariablesOp`	Calculate field variables given a velocity distribution
Vlasov	`LogicalVlasovFluidBoltzmannSetOp`	Boltzmann distribution initial condition for Vlasov fluid
Vlasov	`LogicalVlasovFluidConstantICOp`	Creates or initializes a new Vlasov fluid
Vlasov	`Vlasov1D1VSLOp`	Semi-Lagrangian Vlasov advection

TABLE 2. Summary of operations included in `TURF-DEV`.

Module	Operation	Description
Field	`BCFaceXtrapOp`	Extrapolate field over face direction
Field	`LogicalBCMirrorMoveInOp`	Move and add field ghost data over reflecting surface
Field	`LogicalBCPeriodicOneDomOp`	Copy ghost data for single domain periodic BC
Field	`LogicalFieldAccumulateOp`	Add one field data to another field
Field	`LogicalGradientOp`	Calculate cell centered gradient
Field	`LogicalICConstantOp`	Sets field values in box to constant
Field	`LogicalPoissonOp`	Red/Black Gauss-Seidel Poisson Solve
Field	`LogicalPoissonStrip1DOp`	Red/Black 1D strip Poisson solve w/o transverse cells
Field	`UFieldConstantICOp`	Set unstructured field data to constant value
Fluid	`FluidConstantICOp`	Set field parameters based off fluid definition
Fluid	`FluidESPushOp`	Advance fluid state with electrostatic forcing
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	`RoeFluxCalc2Op`	Calculate 2nd order Roe flux
Fluid	`RoeFluxCalcHLLE2Op`	Calculate 2nd order Roe flux with HLLE2 limiter
Fluid	`RoeFluxCalcHLLEPOp`	Calculate 2nd order Roe flux with HLLEP limiter
Fluid	`RoeFluxCalcOp`	Calculate 1st order Roe flux
Fluid	`TVDFluxCalcOp`	Calculate total variation diminishing flux
Geometry	`LogicalMeshSurfaceSugarcubeOp`	Create sugarcube surface mesh/structured mesh intersection
Geometry	`SPDistBCSurfBoxICOp`	Add particles outside sugarcubed surface only
Geometry	`SPDistSugarcubeBCMoveOp`	Linear particle advance with specular reflection of marked cells
Geometry	`SPDistSugarcubeSurfBCMoveOp`	Linear particle advance with triangulated surface reflection
Particle	`SPDistBCBoxICOp`	Initialize particles uniformly in physical box except flagged cells
Particle	`SPDistBoxICOp`	Initialize particles uniformly in physical box
Particle	`SPDistDirectCellMergeOp`	Default RMS x & v random sign merge
Particle	`SPDistDirectCellSplitOp`	Default RMS x & v random sign split
Particle	`SPDistESPhiCNPushOp`	Crank Nicolson electrostatic particle potential push
Particle	`SPDistESPhiNCNPushOp`	Nonlinear Crank Nicolson electrostatic particle potential push
Particle	`SPDistESPhiNCNSpherePushOp`	Nonlinear Crank Nicolson in spherical ES-potential push
Particle	`SPDistESPhiPushOp`	Explicit electrostatic potential particle push
Particle	`SPDistMCCOp`	Monte Carlo Collision operator
Particle	`SPDistPatchICOp`	Inter-domain particle patch
Particle	`SPDistPerturbedCellIDOp`	Particle cell ID with added perturbation for smoothing
Particle	`SPDistSortedStatToFieldOp`	Accumulate cell velocity moments from sorted distribution
Particle	`SPDistTemperatureToFieldOp`	Calculate cell temperature from distribution
Particle	`SPDistToEMFieldOp`	Accumulate cell charge and current from distibution
Particle	`SPDistToFieldLinear1DOp`	Accumulate 1D linear weight charge to cells
Particle	`SPDistToFieldLinearOp`	Accumulate linearly weighted charge to cells
Particle	`SPDistVSortOp`	Sort particles by cell and velocity octant
Plotting	`LogicalFieldStatWriteVTKOp`	Write accumulated statistic data to VTK file
Plotting	`LogicalFieldWriteVTKUOp`	Write unstructured field data to .vtu file
Plotting	`LogicalVlasovFluidWrite1DProbeVDFOp`	Write particle VDF from particles in probe region
Plotting	`UFieldWriteVTKOp`	General unstructured field writer
SourceModel	`SPDistNormalMaxwellianStreamOp`	Maxwellian stream source from triangulated surfaces
Vlasov	`LogicalBCPeriodicOneDomVlasovOp`	Periodic BC for Vlasov
Vlasov	`LogicalFluidToLogicalVlasovFluidOp`	Create Vlasov VDF from Fluid Variables
Vlasov	`LogicalVlasovFieldVolumetricMulOp`	Multiply or divide Vlasov data by cell volumes
Vlasov	`LogicalVlasovFluidSetOp`	Set Vlasov field data to constant
Vlasov	`SPDistToLogicalVlasovFluidOp`	Accumulate particle weights to Vlasov VDF

Table 3. Summary of additional experimental (E) and research (R) operations included in TURF-DEV.

Module	Operation	Description
Field	ConstantAnnulusPotentialOp	(R) Apply annularly symmetric potential to field
Field	ConstantSphericalPotentialOp	(R) Apply spherically symmetric potential to field
Field	ConstantWellPotentialOp	(R) Apply constant parabolic well potential to field
Field	LogicalICFunctionOp	(E) Set field value using external function
Geometry	LogicalMeshSurfaceBBoxSplotchOp	(E) Mark surface triangle bounding box on cells
Particle	SPDistBCChildLangmuirOp	(E) Child Langmuir surface emission
Particle	SPDistBCSCLOp	(E) Space charge limited flux boundary
Particle	SPDistBCSecondaryOp	(E) Secondary emission boundary condition
Particle	SPDistBCTransOp	(E) Translate particles in box to new location
Particle	SPDistCFEBCOp	(E) Fowler-Nordheim cold field emission BC
Particle	SPDistCopyPosBoxICOp	(E) Initialize particles with VDF but positions from second dist
Particle	SPDistESPhiPushVerletHalfOp	(E) Verlet electrostatic potential particle push
Particle	SPDistDirectCellMergeMixOp	(R) Position sign from xv moment merge
Particle	SPDistDirectCellMergeMixXVOp	(R) Position from xv moment merge
Particle	SPDistDirectCellMergePCAOp	(R) Principal component analysis merge
Particle	SPDistDirectCellMergeXxVOp	(R) Angular momentum preserving merge
Particle	SPDistDirectCellSplitMixOp	(R) Position sign from xv moment split
Particle	SPDistDirectCellSplitMixXVOp	(R) Position from xv moment split
Particle	SPDistDirectCellSplitPCAOp	(R) Principal component analysis split
Particle	SPDistDirectCellSplitXxVOp	(R) Angular momentum preserving split
Particle	SPDistOrbiterICOp	(R) Period synchronized particle initial condition
Plotting	LogicalFieldCatalystOp	(E) Kitware Catalyst plotting connection
Utility	ConstantOperatorOp	(E) Work to allow generic constant parsing
Utility	FromGPUOp	(E) Recursive GSObject transfer from GPU
Utility	ToGPUOp	(E) Recursive GSObject transfer to GPU
Vlasov	CreateVlasovVariableOp	(E) Create new Vlasov fluid variable
Vlasov	LogicalVlasovFluidFunctionICOp	(E) Initialize Vlasov data using external function
VlasovPIC	SPDistConstantVlasBCOp	(R) Constant boundary conditions for VlasovPIC
VlasovPIC	SPDistConstantVlasICOp	(R) Constant initial conditions for VlasovPIC
VlasovPIC	SPDistVlas2wOp	(R) Density to weights for VlasovPIC
VlasovPIC	SPDistVlasDensityToFieldOp	(R) Integrate density to field data for VlasovPIC
VlasovPIC	SPDistw2VlasOp	(R) Weights to density for VlasovPIC

References

[1] Jun Araki. *TURF 1D SHOCK WAVE EXAMPLE - DSMC*. Air Force Research Laboratory (AFRL/RQRS), Version 1.0.0 edition, Dec 2014.

[2] David Bilyeu. *TURF COLLISIONLESS ELECTROSTATIC SHOCK - PART 1: EXPERIMENTAL SETUP*. Air Force Research Laboratory (AFRL/RQRS), Version 1.0.0 edition, Dec 2014.

[3] David Bilyeu. *TURF COLLISIONLESS ELECTROSTATIC SHOCK - PART 2: VLASOV*. Air Force Research Laboratory (AFRL/RQRS), Version 1.0.0 edition, Dec 2014.

[4] L. Brieda, R. Kafafy, J. Pierru, and J. Wang. Development of the draco code for modeling electric propulsion plume interactions. In *40th AIAA Joint Propulsion Conference*, volume AIAA 2004-3633, 2004.

[5] Lord Cole. *Combustion and Magnetohydrodynamic Processes in Advanced Pulse Detonation Rocket Engines*. PhD dissertation, University of California, Los Angeles, 2012.

[6] J. Fife, M. Gibbons, D. VanGilder, and D. Kirtley. The development of a flexible, usable plasma interaction modeling system. In *38th AIAA Joint Propulsion Conference*, volume AIAA 2002-4267, 2002.

[7] J.M. Fife. *Hybrid-PIC Modeling and Electrostatic Probe Survey of Hall Thrusters*. PhD dissertation, Massachusetts Institute of Technology, 1998.

[8] Michael Kapper. *A high-order transport scheme for collisional radiative and nonequilibrium plasma*. PhD dissertation, The Ohio State University, 2009.

[9] Michael G. Kapper. Development of advanced numerical algorithms for modeling complex flows. Master's thesis, San Jose State University, 2005.

[10] Hai Le. Development of a chemically reacting flow solver on the graphic processing unit. Master's thesis, San Jose State University, 2011.

[11] Hai Le. *Hydrodynamic models for multicomponent plasmas with collisional-radiative kinetics*. PhD dissertation, University of California, Los Angeles, 2011.

[12] Robert Martin. *TURF 3D-ESPIC EXAMPLE - PART 1: GROUNDED BOX*. Air Force Research Laboratory (AFRL/RQRS), Version 1.0.0 edition, Feb 2015.

[13] Jonathan Tran. *TURF HEATBATH EXAMPLE - PART 1: INITIALIZATION*. Air Force Research Laboratory (AFRL/RQRS), Version 1.0.0 edition, Nov 2014.

[14] Jonathan Tran. *TURF HEATBATH EXAMPLE - PART 2: EVOLUTION*. Air Force Research Laboratory (AFRL/RQRS), Version 1.0.0 edition, Nov 2014.

[15] Brad Whitlock. *Getting Data Into VisIt*. Lawrence Livermore National Laboratory, Version 2.0.0 edition, July 2010. LLNL-SM-446033.

TURF 1D SHOCK WAVE EXAMPLE - DSMC

SAMUEL J. ARAKI, ERC INC, AFRL/RQRS

Contents

1. Introduction 1
2. Description of the Example Problem 2
3. Setting up the DSMC Example 2
4. DSMC Operations 3
4.1. SPDistDSMCConstantICOp 3
4.2. SPDistDSMCConstantBCOp 5
4.3. SPDistDSMCOp 5
4.4. SPDistDSMCSampleOp 6
4.5. SPDistDSMCSample2Op 6
5. Comparison of Shock Profiles with Bird's DSMC Code 9
References 9

1. Introduction

This tutorial provides steps in setting up the DSMC example provided in the following directory: `tutorial-TURF/DSMC/DSMC_1DShock`. This folder contains two subfolders, `DS1V` and `TURF`. The `DS1V` contains reference cases for Argon shocks using the DS1V code described in Section 5 below. The `TURF` folder contains input files to run the same Argon shock cases within TURF. For example, the `TURF/M1.2` folder contains the input files for running the Mach 1.2 shock test case. Once in the specific case directory, `world.list` should be pointed to the file `world.dsmc1Dshock.list` by typing

```
    projects/dat-TURF/> ln -s world.dsmc1Dshock.list world.list
```

in the command line, as in other examples. The script file `world.dsmc1Dshock.list` will execute the DSMC operations listed in `operations.dsmc1Dshock.list` to simulate the 1D normal shock problem using the DSMC method. In order to run DSMC simulations properly, the value for `FNUM` must be set to the same value for all the DSMC operations, where `FNUM` is the number of physical particles represented by a single simulation particle. Therefore, when introducing particles into the domain, operations that require the user specified `FNUM` must be used; these operations are named as `SPDistDSMCConstantICOp` and `SPDistDSMCConstantBCOp`. The operation `SPDistDSMCConstantICOp` distributes particles uniformly inside the simulation domain. On the other hand, the operation `SPDistDSMCConstantBCOp` creates particles outside the domain, and a fraction of these particles flow into the domain. The DSMC collision calculation is done in `SPDistDSMCOp`, and the same `FNUM` used in the other operations must be used. Furthermore, `SPDistDSMCSampleOp` triggers the mixing of the field values between iterations, allowing a smoother distribution at the end of simulation. These DSMC operations are explained in this tutorial.

2. 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 then 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 part 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 [3].

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

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. Equation (1) is 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.

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

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 Eq. (1) and the perfect gas assumption, the downstream flow properties can be determined if the upstream flow properties are known [3].

$$M_2^2 = \frac{1 + \frac{\gamma-1}{2}M_1^2}{\gamma M_1^2 - \frac{\gamma-1}{2}} \tag{3}$$

$$\frac{n_2}{n_1} = \frac{\rho_2}{\rho_1} = \frac{u_1}{u_2} = \frac{(\gamma+1)M_1^2}{(\gamma-1)M_1^2 + 2} \tag{4}$$

$$\frac{T_2}{T_1} = 1 + \frac{2(\gamma-1)}{(\gamma+1)^2}\frac{\gamma M_1^2 + 1}{M_1^2}\left(M_1^2 - 1\right) \tag{5}$$

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.list` prior to running TURF.

3. 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 Eqs. (3) to (5). Table 1 provides the downstream flow properties for argon gases of $T_1 = 293$, $n_1 = 1 \times 10^{22}$ m^{-3}, and $a_1 = 318.8$ m/s at 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 should also be flowing into the domain from the upstream boundary according to the flow 1. At the interface between the two gases at different flow properties, the properties are initially discontinuous, while 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 Ref. [2].

The script file `world.dsmc1Dshock.list` includes important parameters that define the problem, including the information related to computational grid, time-step, species, etc, as shown below. Referring to `world.dsmc1Dshock.list`, the number of interior cells can be found by dividing (`bound_hi-bound_lo`) by `delta` for each direction. In this example, the grid contains 1000 cells in x-direction and a single cell for both the y- and z-directions. The gas

Table 1. Downstream flow properties for upstream Mach number of 1.2, 1.4, 2.0, and 8.0. The values are for argon gas.

Downstream			Upstream Mach Number, M_1			
Flow Property	Symbol	Unit	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/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

species is argon. Furthermore, the parameters to be output are the number of computational and physical particles which are specified as `NAr` and `CNAr`, respectively. This example uses three stages: INITIALIZE, MOVE, and POSTOP. The script file `operations.dsmc1Dshock.list` contains all the operations within each of the three stages, as listed in Table 2. This tutorial only covers the DSMC operations including `SPDistDSMCConstantICOp`, `SPDistDSMCConstantBCOp`, `SPDistDSMCOp`, and `SPDistDSMCSampleOp`. Descriptions of the other operations can be found in other tutorials.

```
DEFINE WORLD
      NAME = DSMC_example
      op_file = operations.dsmc1Dshock.list
      coordinates = cartesian
      origin = (0.0,0.0,0.0)
      delta = (2.0e-5,2.0e-3,2.0e-3)
      end_time = 1.0e-4
      start_dt = 1.0e-8
      fields = [NAr, CNAr]
      species = [Ar]
      stages = [INITIALIZE, MOVE, POSTOP]
      start_iteration = 0 # Number of Poisson Iteration Before Start
END WORLD

############################################################################
## Domain Geometry
############################################################################

DEFINE DOMAIN DOM000
      bound_lo = (0.0,0.0,0.0)
      bound_hi = (2.0e-2,2.0e-3,2.0e-3)
END DOMAIN
```

4. DSMC Operations

4.1. **SPDistDSMCConstantICOp.** This operation sets up uniformly distributed particles within a box placed inside the simulation domain. An example of inputs for `SPDistDSMCConstantICOp` are shown below. Unlike `SPDistConstantICOp`, the real to computational particle wieght ratio,`FNUM`, is specified as an input. 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. The gas species is argon, and the corresponding mass of each molecule is 39.659 times the proton mass, `Mp`. 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 1 and 2. The initial particle distribution obtained by the DSMC example is shown in Fig. 1. This example corresponds to the case with $M_1 = 1.2$, where the downstream flow properties are given in Table 1. Note that `NAr` is related to the number

Table 2. Summary of operations listed in `operations.dsmc1Dshock.list`.

Stage	Operation	Description
INITIALIZE	`SPDistDSMCConstantICOp`	Initial distribution of particles inside the domain
	`SPDistConstantICOp`	Null particle distribution in ghost cells
MOVEOP	`SPDistDSMCConstantBCOp`	Injection of particles
	`SPDistMoveOp`	Advancement of particles
	`SPDistBCSpecOp`	Specular boundary condition
	`SPDistCellIDOp`	Find cell ID associated with particle location
	`SPDistSortOp`	Sort particles according to cell ID
	`SPDistDSMCOp`	DSMC collision calculation
	`LogicalFieldSetOp`	Initialize the field parameters
	`SPDistDensityToFieldOp`	Sum real and computational particles/cell to field
	`LogicalFieldVolumetricMulOp`	Multiplies or divides field data by cell volumes
	`SPDistDSMCSample2Op`	Blend running average of field data
POSTOP	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots
	`LogicalFieldWrite1DOp`	Write to output files for line plots

density n such that $n = \mathtt{NAr}/\Delta V$ where ΔV is the size of a cell in m^{-3}. There is a statistical noise associated with the number of computational particles as the particle locations are determined using the random number generator; the noise can be reduced by increasing the number of simulation particles.

```
DEFINE OPERATION
      TYPE = SPDistDSMCConstantICOp
      DATA_NAME = Ar-DST
      MAX_NP = 12800000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (1.0e-2,2.0e-3,2.0e-3)
      TEMPERATURE = 293.0
      Z = 0
      MASS = 39.659 Mp
      NUMBER_DENSITY = 1.0e22
      FNUM = 9.1892e8
      VEL = (382.447,0.0,0.0)
END OPERATION
```

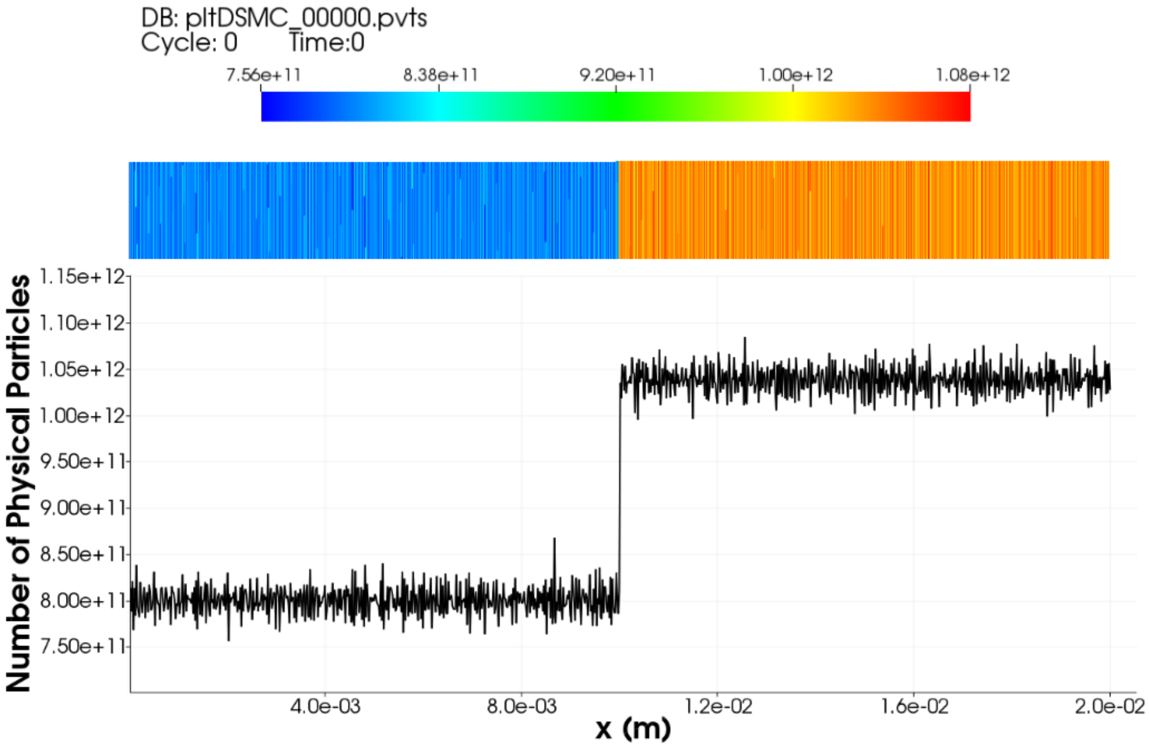


Fig. 1. Initial particle distribution.

4.2. **SPDistDSMCConstantBCOp.** This operation sets up uniformly distributed particles within a box placed outside the simulation domain. The size of box is modified such that it lies within the ghost cell region. An example of inputs for SPDistDSMCConstantBCOp are shown below. Similar to SPDistDSMCConstantICOp, the box to be filled with particles is bounded by BOUND_LO and BOUND_HI.

In the DSMC example, particles flowing out the simulation domain from ±x boundaries are simply discarded, while fraction of particles created in the box flows into the simulation domain. When the simulation is at steady-state, the particle counts flowing in and out the domain are maintained to be nearly equal.

```
DEFINE OPERATION
      TYPE = SPDistDSMCConstantBCOp
      DATA_NAME = Ar-DST
      TEMPERATURE = 293.0
      NUMBER_DENSITY = 1.0E22
      FNUM = 9.1892e8
      VEL = (382.447,0.0,0.0)
      BOUND_LO = (-0.01e-2, 0.0e-3, 0.0e-3)
      BOUND_HI = ( 0.00e-2, 2.0e-3, 2.0e-3)
END OPERATION
```

4.3. **SPDistDSMCOp.** This operation finds the number of collisions to perform within all the grid cells and apply collisions based on the DSMC method. The variable hard sphere (VHS) molecular model is used to determine the deflection of particles which requires inputs of ALPHA and DIAM where ALPHA and DIAM are the empirical factors that determine the diameter variation and reference diameter of the molecule, respectively. The DIAM is the reference molecular diameter at $273K$ such that a hard sphere of that diameter would have the correct fluid viscosity and

`ALPHA` is viscosity-temperature power law coefficient, ω, minus the hard sphere value of 0.5. It is important to note that `ALPHA` should not be confused with the variable soft sphere (VSS) model's α parameter. These parameters can be found in Ref. [2].

```
DEFINE OPERATION
      TYPE = SPDistDSMCOp
      SPDIST_DATA_NAME = Ar-DST
      MASS = 39.659 Mp
      ALPHA = 0.31
      FNUM = 9.1892e8
      KOVERM = 208.132
      DIAM = 4.17e-10
      FREQUENCY_TO_RESAMPLE_MFS = 2000
      SORT_OP_NAME = Sort_Ar-DST
END OPERATION
```

4.4. **SPDistDSMCSampleOp.** This operation is similar to `SPDistDensityToFieldOp` except that it starts to mix the fields after the time specified as an input. An example of inputs for `SPDistDSMCSampleOp` is shown below. In the DSMC example, the fields `NAr` and `CNAr` are computed from the particle distribution at the computational grid. After the time given to `MIX_START_TIME`, mixing between iterations is initiated. `MIX_START_TIME` should be set to the time when the simulation becomes steady-state. This operation allows a smooth field distribution at the end of simulation without using a very large number of simulation particles. The field distributions at 2,000 and 10,000 time-steps are shown in Figs. 2 and 3, respectively. 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 2,000 iterations.

```
DEFINE OPERATION
      TYPE = SPDistDSMCSampleOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = Ar-DST
      PSORT_NAME = Sort_Ar-DST
      FIELD_NAME = NAr CNAr
      MIX_START_TIME = 8.0e-5
END OPERATION
```

4.5. **SPDistDSMCSample2Op.** This operation is another version of `SPDistDSMCSampleOp` needed for long sampling times. Because TURF uses single precision floating point numbers for particles and fields for the sake of GPU performance, the direct DSMC sampling starts to loose accuracy for large numbers of samples. For the direct sample operation, the value of the sampled average field is first scaled by $(n_{sample} - 1)/n_{sample}$ and then particle weights are accumulated into the sample average scaled by w_p/n_{sample}. This means that in each cell, a number of order $(particles/cell) * n_{sample}$ smaller than the total is added for each particle during the accumulation phase. At approximately $O(1,000 - particles/cell)$ and $O(10,000)$ samples, the added pieces are $O(1e7)$ times smaller than the total resulting in lost single precision digits and bulk fluctuation of the sample values. Instead, the `SPDistDSMCSample2Op` uses two density buffers to help alleviate this issue, `NAr` and `NbarAr` (along with particle/cell counterparts). The instantaneous density is first accumulated in `NAr` using the standard `SPDistDensityToFieldOp` operation, and then the sample averaged value is updated via $\bar{N}_{Ar} = (N_{Ar} + (n_{sample} - 1)\bar{N}_{Ar})/n_{sample}$ for each cell. Furthermore, all calculations on the right hand side are performed in double precision prior to rounding to attempt to help retain as many digits of precision as possible during the calculation which is impossible with the sum updated in memory per particle as in the original version. The example shown also demostrates the use of the `LogicalFieldVolumetricMulOp` to divide the value of `NAr` by the cell volume to convert absolute number of real particles per cell into number density. These modifications are not necessary for the simple tutorial versions of the shock case because of the minimal sampling performed, but help in converging the results to the DS1V solution as part of the code verification process in the next section.

```
DEFINE OPERATION
      TYPE = SPDistDensityToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = Ar-DST
      FIELD_NAME = NAr CNAr # Computational and Physical Number per cell
END OPERATION

DEFINE OPERATION
      TYPE = LogicalFieldVolumetricMulOp
      DATA_NAME = FieldData
      FIELD_NAME = NAr
      OP_OPTION = DIV # Divide by Volume (Default is Multiply)
      RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
      TYPE = SPDistDSMCSample2Op
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = Ar-DST
      PSORT_NAME = Sort_Ar-DST
      SRC_FIELD_NAME = NAr CNAr # Instantaneous Computational and Physical Number/Cell
      DST_FIELD_NAME = NbarAr CNbarAr # Sampled Average Computational and Physical Number/Cell
      MIX_START_TIME = 8.0e-5
      SKIP = 5
END OPERATION
```

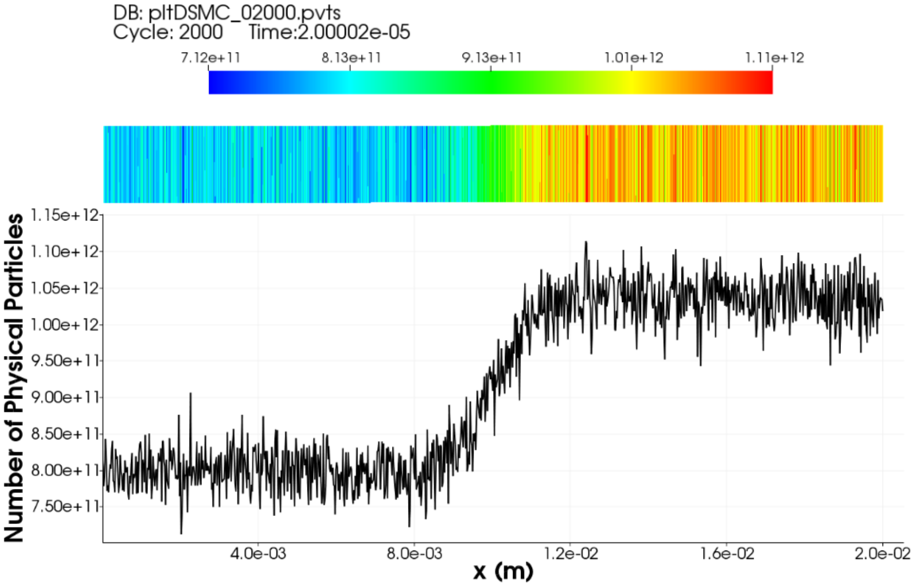


Fig. 2. Particle distribution after 2,000 steps.

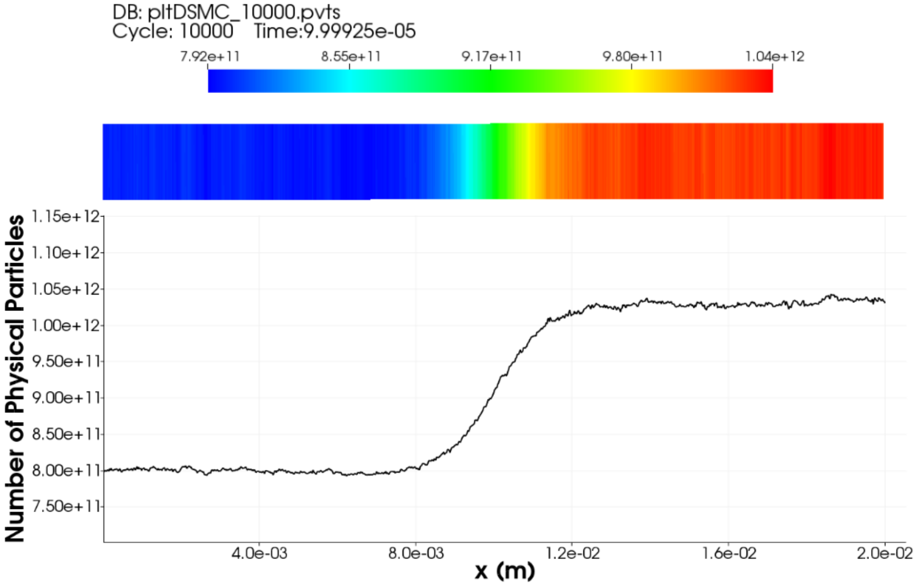


Fig. 3. Particle distribution after 10,000 steps.

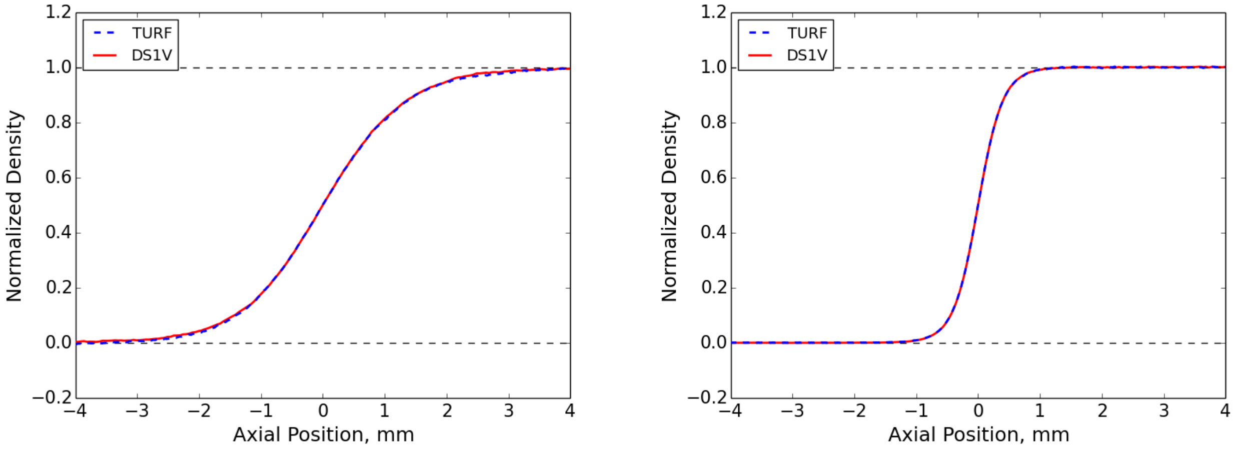
5. Comparison of Shock Profiles with Bird's DSMC Code

The 1D shock profiles from the DSMC part 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 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 simulaiton, 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 finally 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."

Figure 4 compares the shock profiles computed with TURF and DS1V for the upstream Mach number of 1.2 and 2.0. The values are normalized according to,

$$\tilde{\rho} = \frac{\rho - \rho_1}{\rho_2 - \rho_1} \tag{6}$$

The original profile obtained by TURF fluctuated considerably compared to the profile by DS1V when time averaged only over the last $20\mu s$ as shown in Figure 2. With the modification to the sampling procedure for $720\mu s$ of time averaging (similar to DS1V) as described in Section 4.5, Figure 4 show the agreement between the two programs is quite satisfactory.



(a) M=1.2 (b) M=2.0

Fig. 4. Normalized density computed by TURF and DS1V.

References

[1] J D Jr. Anderson. *Modern Compressible Flow with Historical Perspective.* McGraw-Hill, New York, New York, 3rd edition, 2003.

[2] G. A. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows.* Clarendon Press, Oxford, 1994.

[3] H W Liepmann and A Roshko. *Elements of Gasdynamics.* Dover Publications, Mineola, New York, 1957.

TURF COLLISIONLESS ELECTROSTATIC SHOCK - PART 1: EXPERIMENTAL SETUP

DAVID BILYEU, AFRL/RQRS

Contents

1. Introduction 1
2. The Collisionless Electrostatic Shock 1
References 2

1. Introduction

The purpose of this tutorial is to provide information on the setup of the collisionless electrostatic shock test case. By this point it is assumed that you already have an basic understanding of how the operators in TURF work and the purpose of both the `world.list` and the various `operations.*.list` files. For a review of the purpose of these files please refer to the heatbath example tutorials. Instead this tutorial will focus on aspects that are specific to this simulation and will only provide a cursory overview of basic topics. This tutorial is divided up into three main sections, Section 2 provides an overview of the collisionless electrostatic shock experiment.

2. The Collisionless Electrostatic Shock

The collisionless electrostatic shock simulation is based on the experiment by Taylor, et.al, [2]. Figure 1 shows the experimental setup as planned for upcoming validation experiments at AFRL/RQRS including a pulse shape needed to drive a soliton used a related problem also originally performed in the same experimental device[1]. In this experiment, Argon gas is fully ionized and separated into driver and target sections of a vacuum chamber. The separation is maintained by a negatively biased grid held at a fixed potential. The number density of the driver side is higher then the target but they share the same ion and electron temperature. At the start of the experiment a ramp potential is applied to the driver side and a shock moves into the target side. For all cases the ion temperature and driven number density was held at 0.2 eV and 10^9 cm^{-3} respectively. A sweep of parameter space was accomplished by varying the electron temperature and the density of the driver gas. The electron to ion temperature varies between 6 and 20 while the density varies from 1 to 20 percent. The initial setup that saw the most study was at a density jump of 25 percent and an electron temperature of 7.5 and 15 eV.

There are several favorable parameters unique to this experiment that makes it an ideal test-case for a Vlasov-Poisson simulation.

- Ion-Ion collisions can be neglected because the mean-free-path between collisions is on the order of 300 Debye lengths (λ_D) and the total domain of the chamber is about $1000\lambda_D$.
- There are no applied magnetic fields and induced currents can be assumed to be neglibible which obviate the need to solve the full Maxwell equations.
- The spatial symmetries of the experiment limit variations to be in only one direction so that a 1D1V Vlasov simulation of the flow is sufficient.

In addition to these parameters, several additional assumptions are made which will be tested throughout the upcoming TURF code validation and verification campaign.

- Modeling ion kinetic effects is important due to the collisionless nature of the plasma. This is to be validated through comparison of kinetic and fluid solutions.

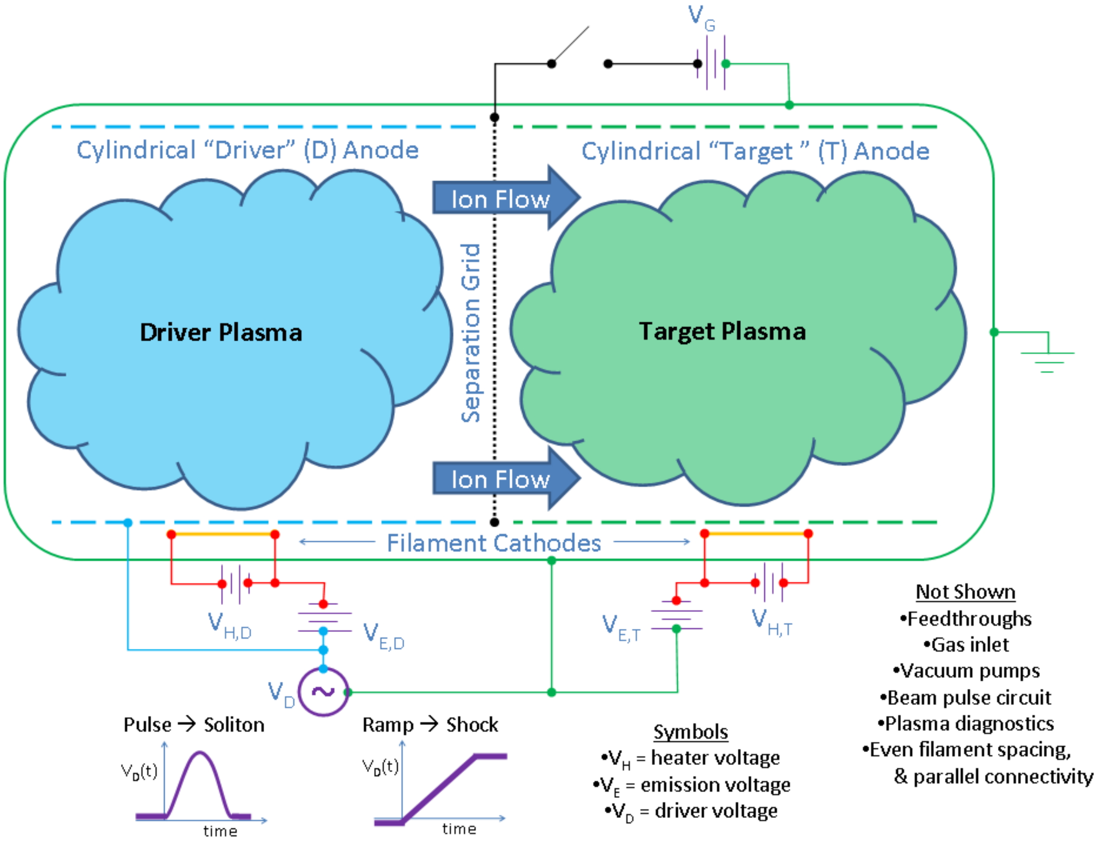


Fig. 1. Design of AFRL/RQRS double plasma device experiments.

- The electrons can be accurately modeled as a Boltzmann equilbirum fluid sothat the fastest time scale that needs to be resolved is the ion plasma frequency. This will be tested through comparison with experimental results as well as future fully kinetic simulations which include electron kinetic effects.
- The flow is robust against spontaneously generated transverse modes so that the one-dimensional character of the flow is preserved far enough from the boundaries.

References

[1] H. Ikezi, R. J. Taylor, and D. R. Baker. Formation and interaction of ion-acoustic solitons. *Physics Review Letters*, 25(1):11–14, July 1970.

[2] R. J. Taylor, D. R. Baker, and H. Ikezi. Observation of collisionless electrostatic shocks. *Physics Review Letters*, 24(5):206–208, February 1970.

TURF COLLISIONLESS ELECTROSTATIC SHOCK - PART 2: VLASOV

DAVID BILYEU, AFRL/RQRS

Contents

1. Introduction 1
2. World.list 1
3. operations.vlasov.list 2
3.1. operations.vlasov.list Initialization 2
3.2. operations.vlasov.list Move 3
3.3. operations.vlasov.list Postop 6
4. Appendix 8
References 8

1. Introduction

The purpose of this tutorial is to provide an overview of how to setup and run a Vlasov simulation in the Thermophysics Universal Research Framework (TURF). By this point it is assumed that you already have an basic understanding of how the operators in TURF work and the purpose of both the `world.list` and the various `operations.*.list` files. For a review of the purpose of these files please refer to the heatbath example tutorials. Instead this tutorial will focus on aspects that are specific to this simulation and will only provide a cursory overview of basic topics. This tutorial is divided up into three main sections: Section 1 of this tutorial provides an overview of the collisionless electrostatic shock experiment, Section 2 provides an explination of the `world.list` file, and finally Section 3 details the operators necessary to run the collisionless electrostatic shock simulation with the Vlasov solver within TURF.

2. World.list

The `world.list` file defines important parameters that are not unique to any one particular operator. This includes information about the mesh, the time step and total run time, as well as the global variables. The only new information included in the `world.list` that is unique to Vlasov solvers is the definition of the velocity space origin and mesh spacing. These variables are defined via:

```
    velocity_origin = (0.0,-0.5,-0.5)
    velocity_delta = (15.625,1.0,1.0)
```

At this time each species can have its own unique bounds in velocity space but they must share the same mesh spacing and origin.

This simulation uses seven different field variables including the electron and ion densities, the electric field vector, the electric potential, a density for the electric field solver, mean Velocity, and temperature and are defined:

```
fields = [rhoE, rhoI, Ex, Ey, Ez, phi, rho_source]
fields = [Vmeanx, Vmeany, Vmeanz, Temperature]
```

3. OPERATIONS.VLASOV.LIST

This section explains how to solve this problem using Vlasov methods within TURF. It is assumed that the reader has a basic understanding of how to setup a simulation in TURF and only the information relevant to Vlasov and this simulation in particular are detailed. The operations.vlasov.list file are broken up into three stages, INIT, MOVE, and POSTOP which are responsible for the initialization, solvers, and plotting respectively.

3.1. operations.vlasov.list Initialization.

In the initialize stage it is necessary to set the initial conditions which includes defining a new phase-space variable. The phase space variable, fAR+, is defined using:

```
DEFINE OPERATION
    TYPE = CreateVlasovVariableOp
    DATA_NAME = VlasovFluidData
    VBOX_LO = (-6000.0,-0.5,-0.5)
    VBOX_HI = ( 6000.0, 0.5, 0.5)
    VBOX_NGHOST = [3 0 0]
    SPECIES_NAMES = fAr+
    SPECIES_COMPOSITION = Ar
END OPERATION
```

The different fields are relatively self explanatory, but it should be noted that VBOX_LO/HI are in units of meters per second and the unused dimensions, V_y and V_z, need to have a length of one. Another important parameter is the VBOX_NGHOST which set the number of ghost cells in each velocity direction. If left unset, the default value of 3 ghost cells in the unused direction increases memory requirements by a factor of 49. The variable name is defined in SPECIES_NAMES, multiple species can be defined in this field. At a minimum the solver needs to know the mass and charge of each species. This is accomplished in one of two ways. The preferred method is to define the individual species that makes up each SPECIES_NAMES. This is defined in SPECIES_COMPOSITION. SPECIES_COMPOSITION will parse a chemical formula and calculate its mass and charge using an internal database of elements. Any values defined in M and Z will be ignored. The second method is to manually set the mass and charge via M in kg and Z respectively. For this method to work you must set the SPECIES_COMPOSITION to None. These two methods can be used together, but place holder will be needed in M and Z for the species defined via their SPECIES_COMPOSITION. For example the following code will use the internal database to calculate the mass and charge of fAr+ and fAr but will set FakeVar using the values in M and Z. Note that place holders in M and Z are required.

```
    SPECIES_NAMES = fAr+, FakeVar, fAr
    SPECIES_COMPOSITION = Ar, None, Ar
    M = -1.0, 1.0e-12, 23.3
    Z = 1, -2, 0
```

This operator will not set the initial value of fAR+ so we will need to call another operator to set the distribution. This is accomplished by:

```
DEFINE OPERATION
    TYPE = LogicalVlasovFluidBoltzmannSetOp
    PHASESPACE_TYPE = VlasovFluidData
    BOUND_LO = (-100.0e-3,-0.5,-0.5)
    BOUND_HI = ( 0.0e-3, 0.5, 0.5)
    PHASESPACE_NAME = fAr+
    TEMPERATURE_K = 2320.8 # 1.5ev
    NUMBER_DENSITY = 1.25e15
    INIT_ONLY = true
    NUMBER_OF_DIMENSIONS = 1
END OPERATION
```

This sets up the initial VDF with a Boltzmann distribution for the driver side. The target side is set using the same operator but with different BOUND_LO/HI values and density. The important parameters are TEMPERATURE_K and `NUMBER_DENSITY` which sets the temperature in Kelvin and the total number density per cell in m^{-3}. It should be noted that the molecular mass does not need to be defined because that information is stored within the variable `fAr+`. Another important parameter is `INIT_ONLY` which tells the operator that it should only run once at the beginning of the simulation. Otherwise the operation will overwrite the update with the initial conditions, though this could also be used as Dirichlet boundary condition at domain edges in other simulations.

3.2. **operations.vlasov.list Move.** The next stage to run is the `MOVE` stage. This stage contains the advection of the fluid in phase space as well as the electric field solver and is broken up into four main steps: (1) X-advection (half Δt), (2) Electric field solver, (3) V_x-advection (full Δt), (4) X-advection (half Δt). This dimensionally split procedure was originally developed for the Vlasov equation by Cheng and Knorr and provides a second order integration in time[1]. The advection in phase space uses a Semi-Lagrangian method with WENO style interpolation and was developed by Qiu and Christlieb[2]. This method was chosen because it was found to be an accurate and efficient solver. The X-advection consists of two different operations, the first sets the ghost cells while the second advects the fluid in the X direction.

```
DEFINE OPERATION
    TYPE = LogicalBCVlasovExtrapolateOp
    NAME = PeriodicBCX1
    DATA_NAME = VlasovFluidData
    FIELD_NAME = fAr+
    DIRECTION = X
END OPERATION

DEFINE OPERATION
    TYPE = Vlasov1D1VSLOp
    NAME = Vlasov1D1VSL_X1
    DIRECTION = X
    VARIABLE_NAME = fAr+
    TIME_SCALE = 0.5
    VARIABLE_TYPE = VlasovFluidData
END OPERATION
```

It should be noted that `TIME_SCALE` is set to 0.5 which indicates that only a half time step should be taken.

The next set of operations are used to calculate the electric field using Boltzmann equilibrium electrons. Many of the variables are self explanatory but one parameter, `RUN_AT_INIT`, needs further explanation. `RUN_AT_INIT` signifies that the `apply` function of the operator should also be run during the initialization stage. Typically, during the initialization stage the operator will only parse the input file, create the required memory, and if necessary set the initial conditions. In most cases this is enough, but some variables, such as the electric filed, its value is not explicitly known and a routine must be used to calculate it. Since the method used to calculate the initial electric field is the same used during the simulation it is more practical to define these operators once during the `MOVE` stage and set the `RUN_AT_INIT` to true.

```
DEFINE OPERATION
## Calculates several useful field variables from a velocity distribution
## including density, mean velocity and temperature. This routine may be used
## instead of LogicalVlasovCalcDensityOp
    TYPE = LogicalVlasovCalcFluidVariablesOp
    NAME = CalcFluidVariables
    PHASESPACE_TYPE = VlasovFluidData
    PHASESPACE_NAME = fAr+
    DENSITY_TYPE = FieldData
    DENSITY_NAME = rho_tmp
    MEAN_V_PREFIX = Vmean
    TEMPERATURE_NAME = Temperature
    MEAN_V_DIRECTIONS = [x, y, z]
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Calculates Density by integrating over velocity space
    TYPE = LogicalVlasovCalcDensityOp
    PHASESPACE_TYPE = VlasovFluidData
    PHASESPACE_NAME = fAr+
    DENSITY_TYPE = FieldData
    DENSITY_NAME = rhoI
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Sets electric potential (phi) to zero
    TYPE = LogicalFieldSetOp
    DATA_NAME = FieldData
    VALUE = 0.0
    FIELD_NAME = phi
    INIT_ONLY = false
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Add electron and ion density
   TYPE = LogicalFieldAddOp
   DATA_NAME = FieldData
   FIELD_SRCB_NAME = rhoE
   FIELD_SRCC_NAME = rhoI
   FIELD_DST_NAME = rhoE
   RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Multiples the ion density by the mass of an electron to convert
## from number density to mass density and set to rho_source
    TYPE = LogicalFieldScalarMulOp
    DATA_NAME = FieldData
    FIELD_SRC_NAME = rhoI
    FIELD_DST_NAME = rho_source
    SCALAR = 1.602189200e-19
    RUN_AT_INIT = true
END OPERATION

DEFINE OPERATION
## Finds mass of ions in each cell, i.e., no longer density
      TYPE = LogicalFieldVolumetricMulOp
      DATA_NAME = FieldData
      FIELD_NAME = rho_source
      RUN_AT_INIT = true
END OPERATION
```

```
DEFINE OPERATION
## Sets Electric field boundary conditions on the left hand side
      TYPE = LogicalBCXtrap
      NAME = Neumann-X-
      DATA_NAME = FieldData
      FIELD_NAME = phi
       BOUND_LO = (-120.0e-3,-1.0,-1.0)
       BOUND_HI = (-95.0e-3, 1.0, 1.0)
END OPERATION

DEFINE OPERATION
## Sets Electric field boundary conditions on the right hand side
      TYPE = LogicalBCConstantOp
      NAME = Electrode-X+
      value = 0.0 #
      DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = ( 95.0e-3,-1.0,-1.0)
      BOUND_HI = ( 120.0e-3,1.0, 1.0)
END OPERATION

DEFINE OPERATION
    TYPE = LogicalPoissonBoltzmannStrip1DOp
    FIELD_NAME = phi
    SOURCE_NAME = rho_source
    NUMBER_DENSITY_REF_CGS = 1.0e9
    ELECTRON_TEMPERATURE_CGS = 3.0
    ELECTRON_DENSITY_NAME = Ne-
    NEUMANN_LEFT = TRUE
       SUBCYCLE = 1
# INIT_ONLY = TRUE
END OPERATION

DEFINE OPERATION
## Finds the electric filed from the gradient of the electric
## potential and multiplies by a constant
    TYPE = LogicalGradientCellCenterOp
    FIELD_DATA_NAME = FieldData
    FIELD_POTENTIAL_NAME = phi
    FIELD_GRADIENT_PREFIX = E
    FIELD_MULTIPLY_CONSTANT = -2.415365e6 #ec/(MW*amu) -2.415365e6
    FIELD_GRADIENT_DIRECTIONS = [x, y, z] ## Ex,Ey,Ez
    RUN_AT_INIT = true
    BOUNDARY_TYPE = EXTRAPOLATE
END OPERATION
```

The V_x-advection consists of two different operations, the first sets the ghost cells while the second advects the fluid in the V_x direction.

```
DEFINE OPERATION
    TYPE = LogicalBCVlasovExtrapolateOp
    NAME = PeriodicBCY
    DATA_NAME = VlasovFluidData
    FIELD_NAME = fAr+
    DIRECTION = VX
END OPERATION

DEFINE OPERATION
    TYPE = Vlasov1D1VSLOp
    NAME = Vlasov1D1VSL_Vx
    DIRECTION = VX
    VARIABLE_NAME = fAr+
    WAVE_SPEED_NAME = Ex
    TIME_SCALE = 1.0
    VARIABLE_TYPE = VlasovFluidData
END OPERATION
```

The final step of the `MOVE` stage is the second advection in the X-direction, which uses the same two operations used before.

```
DEFINE OPERATION
    TYPE = LogicalBCVlasovExtrapolateOp
    NAME = PeriodicBCX2
    DATA_NAME = VlasovFluidData
    FIELD_NAME = fAr+
    DIRECTION = X
END OPERATION

DEFINE OPERATION
    TYPE = Vlasov1D1VSLOp
    NAME = Vlasov1D1VSL_X2
    DIRECTION = X
    VARIABLE_NAME = fAr+
    TIME_SCALE = 0.5
    VARIABLE_TYPE = VlasovFluidData
END OPERATION
```

3.3. **operations.vlasov.list Postop.** The third and final stage is the `POSTOP` stage. This stage is responsible for preparing and saving the results to various output files.

The first output file is a two-dimensional phase-space plot. The purpose of this operator is to take any two dimensions, $X, Y, Z, V_x, V_y,$ or V_z and a coordinate in phase space and slice along those planes. This is controlled by the following variables, `SPACE_CORD`, `VELOCITY_CORD`, `X_PLOT_DIRECTION`, and `Y_PLOT_DIRECTION` which sets the spatial coordinate, phase space coordinate the coordinate to plot along the "X" axis and the phase space coordinate to plot along the "Y" axis respectively. This operator looks like:

```
DEFINE OPERATION
    INCLUDE_GHOST = false
    TYPE = LogicalVlasov2DWriterOp
    DATA_NAME = VlasovFluidData
    FILE_HEAD = shockdata/phase_
    FIELD_NAME = fAr+
    SKIP = 5
    SPACE_CORD = (0.0, 0.0, 0.0)
    VELOCITY_CORD = (-5.0, 0.0, 0.0)
    X_PLOT_DIRECTION = X
    Y_PLOT_DIRECTION = VX
    BINARY = false
    RUN_AT_INIT = true
END OPERATION
```

The next plotting operator is a bit of a hack and could be changed in future releases. The operator is designed to save the spatial data, e.g., density, electric field, and it is desirable to use the same operator regardless of the number of spatial dimensions. Unfortunately the VTK file format does not have a convenient mechanism to save one-dimensional data. To get around this an additional parameter SAVE_AS_CSV was added that saves the data in csv file format rather than the standard VTR format. The operator is set via:

```
DEFINE OPERATION
    TYPE = LogicalFieldWriteVTKROp
    DATA_NAME = FieldData
    FILE_HEAD = shockdata/field_data
    FIELD_NAME = rhoI, phi, Ex, rho_source, Vmeanx, Vmeany, Vmeanz, Temperature, Ne-
    SKIP = 2
    DIMENSIONS = 2
    nFIELD_NAME = 4
    RUN_AT_INIT = true
    SAVE_AS_CSV = true
END OPERATION
```

The final operator in this stage is used to save various metrics including the density, energy, entropy and electric filed norms over time. The operator is set up via:

```
DEFINE OPERATION
    TYPE = VlasovMetricsOp
    PHASESPACE_TYPE = VlasovFluidData
    SPACE_TYPE = FieldData
    PHASESPACE_NAME = fAr+
    DENSITY_NAME = rhoI
    E_FIELD_PREFIX = E
    E_FIELD_DIRECTIONS = [x, y, z]
    FILE_NAME = shockdata/norms.csv
    SKIP = 1
    RUN_AT_INIT = true
END OPERATION
```

4. Appendix

Table 1. Summary of operations listed for the collisionless shock Vlasov example.

Stage	Operation	Description
INITIALIZE	`CreateVlasovVariableOp`	Create a Vlasov variable
	`LogicalVlasovFluidBoltzmannSetOp`	Sets initial conditions of a Vlasov variable using a Boltzmann distribution
MOVE	`LogicalVlasovCalcFluidVariablesOp`	Calculate field variables given a velocity distribution
	`LogicalBCVlasovExtrapolateOp`	Sets a velocity boundary conditions to extrapolation, i.e. simple non-reflecting
	`Vlasov1D1VSLOp`	Advects a Vlasov variable using the Semi-Lagrangian method
	`LogicalVlasovCalcDensityOp`	Calculates the density given a velocity distribution
	`LogicalFieldSetOp`	Set field values to constant
	`LogicalFieldAddOp`	Adds one field variable to another
	`LogicalFieldScalarMulOp`	Multiplies field by scalar constant
	`LogicalFieldVolumetricMulOp`	Multiplies or divides field data by cell volumes
	`LogicalBCXtrapOp`	Sets a physical boundary to extrapolation
	`LogicalBCConstantOp`	Sets a physical boundary to be a constant
	`LogicalPoissonBoltzmannStrip1DOp`	Solves for the electric filed assuming a Boltzmann electron
	`LogicalGradientCellCenterOp`	Calculates the gradient of a field vector
POSTOP	`LogicalVlasov2DWriterOp`	Exports a 2D phase-space plot
	`LogicalFieldWriteVTKROp`	Exports the filed data, e.g, density, velocity,...
	`VlasovMetricsOp`	Exports Vlasov metrics data e.g., mass and energy conservation

References

[1] C.Z Cheng and Georg Knorr. The integration of the vlasov equation in configuration space. *Journal of Computational Physics*, 22(3):330–351, November 1976.

[2] Jing-Mei Qiu and Andrew Christlieb. A conservative high order semi-lagrangian WENO method for the vlasov equation. *Journal of Computational Physics*, 229(4):1130–1149, February 2010.

TURF 3D-ESPIC EXAMPLE - PART 1: GROUNDED BOX

ROBERT MARTIN, ERC INC, AFRL/RQRS

Contents

1. Introduction 1
2. world.list 2
3. operations.list 3
3.1. INITIALIZE 3
3.2. SOLVE 6
3.3. MOVE 9
3.4. PLOT 10
4. results 12
5. Appendix 14
References 14

1. Introduction

This tutorial demonstrates running a simple 3D electrostatic particle in cell (PIC) case in the Thermophysics Universal Research Framework (TURF). This tutorial assumes familiarity with the simple heatbath tutorials. New users are referred to those tutorials for further explanation. The TURF input files can be located in `tutorial-TURF/GroundedBox/ES-PIC`. You should see several files with the `.list` extension, which act as the scripting files for TURF.

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 [1]. The initial conditions are a uniform unit meter cube of zero velocity protons at a density of $1e10/m^3$. In one octant of the cube, the proton charge is neutralized with $1e10/m^3$ electrons with neither velocity nor thermal velocity. 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.

Date: 1/31/2015.

2. world.list

Running the TURF executable in the working directory will have TURF search for the default script file, `world.list`, and parses it automatically. We can take a look at the script by opening the file. The first block that defines the `WORLD` is shown below.

```
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)
      end_time = 10.0e-6
      start_dt = 2.50e-9
# Names Should not be fully contained in an earlier name for Plotting
      fields = [rho, Enx, Eny, Enz, phi, residual_phi, Np+, Ne-, CNp+, CNe-]
      stages = [INITIALIZE, SOLVE, MOVE, PLOT]
END WORLD
```

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 sample uses the `operations.list` operations file to define the simulation algorithm which will be discussed below. The remainder of the world definition sets a global cartesian coordinate system with 2cm cells along with 2.5ns time steps up to a final simulation time of 10μs. The next line defines 10 field variables for 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+,Ne-,CNp+,CNe-`).

The example run is broken into 4 stages named `INITIALIZE`, `SOLVE`, `MOVE`, and `PLOT`. The two additional stages compared to the heatbath example are to accommodate an iterative electrostatic potential solve stage (`SOLVE`) and to ensure synchronization prior to the plotting operation stage (`PLOT`) though the later is not strictly necessary.

The last section of `world.list` defines the active domain for the simulation. In this example, it is simply a 1m unit cube starting from the coordinate origin. Using the global mesh spacing of 2cm from the `WORLD` definition results in a 50x50x50 active cell cube with the default 3 "ghost"-cells added to the high and low side in each direction for application of boundary conditions.

```
############################################################################
## Domain Geometry
############################################################################
DEFINE DOMAIN DOM000
      bound_lo = (0.0,0.0,0.0)
      bound_hi = (1.0,1.0,1.0)
END DOMAIN
```

3. OPERATIONS.LIST

In this tutorial, the operations file will be considered in stages.

3.1. INITIALIZE.

This stage creates new particle electron and proton particle distributions. The `SPDistConstantICOp` operation should be familiar from the heatbath example. Notable differences are that the charges, `Z`, are non-zero and the electron mass is defined in units of electron mass instead of proton mass.

```
DEFINE STAGE INITIALIZE

######################################################################
## Initial Particle Distributions and Ghost/Exchange Distributions ##
######################################################################

DEFINE OPERATION
      TYPE = SPDistConstantICOp
      DATA_NAME = e-DST
      MAX_NP = 2000000
      FILL_RATIO = 1.0 #4.0
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (0.5,0.5,0.5)
      temperature = 0.0 #11604.5059 # 1ev
      number_density = 1.e10
      Z = -1
      Mass = 1.0 Me
      vel = (0.,0.,0.)
END OPERATION

DEFINE OPERATION
      TYPE = SPDistConstantICOp
      DATA_NAME = p+DST
      FILL_RATIO = 0.0625
      MAX_NP = 2000000
      BOUND_LO = (0.0,0.0,0.0)
      BOUND_HI = (1.0,1.0,1.0)
      temperature = 0.0 #11604.5059 # 1ev
      Z = 1
      Mass = 1.0 Mp
      number_density = 1.e10
      vel = (0.,0.,0.)
END OPERATION
```

Next, empty “ghost” particle distributions are created again using `SPDistConstantICOp`. These are empty buffers where particles that have escaped the domain get copied later on in the sort.

```
DEFINE OPERATION
      TYPE = SPDistConstantICOp
      DATA_NAME = e-GST
      MAX_NP = 2000000
END OPERATION

DEFINE OPERATION
      TYPE = SPDistConstantICOp
      DATA_NAME = p+GST
      MAX_NP = 2000000
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 that stage. The `SPDistCellIDOp` operation identifies which cell the particle resides in and saves it to the "CellID" variable within the particle distribution. The `SPDistSortOp` operation sorts the particles by their "CellID" and any particle that has escaped the domain gets separated into the ghost distribution.

```
################################################
## Initial Sort Sets Cell Edges for Fast Sums ##
################################################

DEFINE OPERATION
      TYPE = SPDistCellIDOp
      DATA_NAME = e-DST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistCellIDOp
      DATA_NAME = p+DST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistSortOp
      NAME = Sort_e-DST
      SRC_NAME = e-DST
      DST_NAME = e-GST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistSortOp
      NAME = Sort_p+DST
      SRC_NAME = p+DST
      DST_NAME = p+GST
END OPERATION
```

The next sections accumulates particle quantities into the cell field variables. Before the data can be accumulated, the field variables must be cleared using the `LogicalFieldSetOp`. In future version of TURF, these operations may be optionally fused, but the separate combination is more general. The actual accumulation of field data is performed by the `SPDistDensityToFieldOp` and `SPDistToFieldOp`. The first is used to set diagnostic fields for number of real (`Nx`) and computational (`CNx`) particles per cell. It is worth noting that these numbers are both raw sums. To obtain the density, "n" from `N`, a `LogicalFieldVolumetricMulOp`. This usage can be seen in the Vlasov collisionless shock tutorial. The second operation multiples by particle charge while doing the accumulation such that the charge density is computed. More computationally efficient calculations can now be performed using field multiplication and summation operations, but the example in this tutorial was created using an early version of TURF that existed before those operations had been completed. This functionality can also be seen in the Vlasov collisionless shock tutorial.

```
###################
## Sum to Fields ##
###################

# Clear Variables First
```

```
DEFINE OPERATION
      TYPE = LogicalFieldSetOp
      DATA_NAME = FieldData
      FIELD_NAME = CNe-
      VALUE = 0.0
END OPERATION

DEFINE OPERATION
      TYPE = LogicalFieldSetOp
      DATA_NAME = FieldData
      FIELD_NAME = CNp+
      VALUE = 0.0
END OPERATION

DEFINE OPERATION
      TYPE = LogicalFieldSetOp
      DATA_NAME = FieldData
      FIELD_NAME = Ne-
      VALUE = 0.0
END OPERATION

DEFINE OPERATION
      TYPE = LogicalFieldSetOp
      DATA_NAME = FieldData
      FIELD_NAME = Np+
      VALUE = 0.0
END OPERATION

# Accumulate

DEFINE OPERATION
      TYPE = SPDistDensityToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = e-DST
      PSORT_NAME = Sort_e-DST
      FIELD_NAME = Ne- CNe- # Computational and Physical Number per Cell
END OPERATION

DEFINE OPERATION
      TYPE = SPDistDensityToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = p+DST
      PSORT_NAME = Sort_p+DST
      FIELD_NAME = Np+ CNp+ # Computational and Physical Number per Cell
END OPERATION


DEFINE OPERATION
      TYPE = LogicalFieldSetOp
      DATA_NAME = FieldData
      FIELD_NAME = rho
      VALUE = 0.0
END OPERATION

DEFINE OPERATION
      TYPE = SPDistToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = e-DST
      FIELD_NAME = rho
END OPERATION

DEFINE OPERATION
      TYPE = SPDistToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = p+DST
      FIELD_NAME = rho
END OPERATION
```

Finally, the “INITIALIZE” stage is completed with the NextStageOp operation to proceed to the “SOLVE” stage.

```

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

END STAGE INITIALIZE

############################################################################
```

3.2. **SOLVE.** 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 LogicalBCConstantOp operation. The operation is relatively straightforward. The phi variable of the default FieldData object is set to a potential of 0 Volts inside of the box defined by BOUND_LO and BOUND_HI. In this configuration of TURF, the potential is assumed to be cell centered. More specifically, the potential is set to 0 for every cell which has a cell center inside the physically defined box. 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. This same approach is used when creating the domains which snap to the nearest approximation of cells based on the physical constraints independent of the underlying mesh resolution. Once again, the NAME variable for the operation is simply a designator label for output readability and the value in the NAME is not evaluated by the code to influence application of the operation. 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 SOLVE

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-X-
      DATA_NAME = FieldData
      FIELD_NAME = phi
      VALUE = 0.0 #
      BOUND_LO = (-0.1,-0.1,-0.1)
      BOUND_HI = (0.0,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-X+
      VALUE = 0.0 #
      DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (1.0,-0.1,-0.1)
      BOUND_HI = (1.1,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Y+
      VALUE = 0.0 #
      DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (-0.1,1.0,-0.1)
      BOUND_HI = (1.1,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Y-
      VALUE = 0.0 #
      DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (-0.1,-0.1,-0.1)
      BOUND_HI = (1.1,0.0,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Z+
      VALUE = 0.0 #
      DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (-0.1,-0.1,1.0)
      BOUND_HI = (1.1,1.1,1.1)
END OPERATION

DEFINE OPERATION
      TYPE = LogicalBCConstantOp
      NAME = Electrode-Z-
      VALUE = 0.0 #
      DATA_NAME = FieldData
      FIELD_NAME = phi
      BOUND_LO = (-0.1,-0.1,-0.1)
      BOUND_HI = (1.1,1.1,0.0)
END OPERATION
```

After the boundary conditions have been set, the actual Poisson solve can be performed. Currently, the set of elliptic solvers in TURF is relatively minimal and includes only red-black Gauss-Seidel and tri-diagonal ADI-type solvers as indicated below by the LogicalPoissonStripOp operation. There is also 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. The only non-default options for the solver selected were to not cycle sweep directions and to sub-cycle the operation 3 times before continuing. The operation is also 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.

```


DEFINE OPERATION
      TYPE = LogicalPoissonStripOp
      DATA_NAME = FieldData #Default
      FIELD_NAME = phi  #Default
      SOURCE_NAME = rho #Default
      MESH_NAME = SMesh  #Default
      DIRECTION = 0 #Start with X-sweep
      SUBCYCLE = 3
      DIRCYCLE = FALSE # Default TRUE
END OPERATION
```

The last part of the SOLVE stage is defining the criteria to iterate the stage or continue to the next. First, the residual of phi is computed in every cell and stored in residual_phi using the LogicalResidualOp. The LogicalNormOp operation calculates the norm of the residual. The NORM parameter defines the power p for any L^p-norm. The operation creates a new scalar variable SUMresidual_phi_L2.00e+00 based off the name of the field in which the residual resides and the power of the norm to store the accumulated total residual. Finally, the CriteriaStageOp evaluates whether the summed residual is below the required threshold CRITERIA. Each domain applies this operation independently. At the end of each 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_NAME = phi #Default
      RESIDUAL_NAME = residual_phi #Default+(FIELD_NAME)
      SOURCE_NAME = rho #Default
END OPERATION

DEFINE OPERATION
      TYPE = LogicalNormOp
      FIELD_NAME = phi #Default
      RESIDUAL_NAME = residual_phi #Default+(FIELD_NAME)
      NORM = 2.0 #Default
END OPERATION

DEFINE OPERATION
# Threshold Criteria to Proceed to the Next Stage
      TYPE = CriteriaStageOp
      QUANTITY_NAME = SUMresidual_phi_L2.00e+00
      CRITERIA = 5.0e-4 # High Density!
```

```
END OPERATION

END STAGE SOLVE

############################################################################
```

3.3. **MOVE.** In the next section, the particle positions are updated using the field solved in the prior step. To do this, the node centered electric field, `En`, 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 `En` 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 two operations use the field to advance the electron and ion positions. The inputs are similar to the basic linear push described in the heatbath tutorials with extra field options so that the operator knows which field data to use for the acceleration.

```
DEFINE OPERATION
      TYPE = SPDistESPushOp
      FIELD_DATA_NAME = FieldData
      FIELD_EN_PREFIX = En
      FIELD_EN_DIRECTIONS = [x, y, z]
      SPDIST_DATA_NAME = e-DST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistESPushOp
      FIELD_DATA_NAME = FieldData
      FIELD_EN_PREFIX = En
      FIELD_EN_DIRECTIONS = [x, y, z]
      SPDIST_DATA_NAME = p+DST
END OPERATION
```

After the push, the particle distribution is split between particles that remain within the domain and particles that escaped into the grounded wall. Particle that escape are marked with a `CellID` flag set within the push. This push does not actually test boundary intersections during the push which is a fast method for simple boundary conditions. Triangulated boundary surface intersecting pushes with and without field are still in testing and will appear in future revisions of the TURF-IR. The last operation in the stage is another `NextStageOp`.

```
DEFINE OPERATION
      TYPE = SPDistSplitOp
      SRC_NAME = e-DST
      DST_NAME = e-GST
END OPERATION

DEFINE OPERATION
      TYPE = SPDistSplitOp
      SRC_NAME = p+DST
      DST_NAME = p+GST
END OPERATION

##############################################

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

END STAGE MOVE
############################################################################
```

3.4. **PLOT.** The last stage of the simulation is plotting. The first operation in the plotting section is the CUDA accelerated real-time volume ray tracing operation, `VolumeRenderOp`. The code is primarily a wrapped version of the NVIDIA CUDA SDK's VolumeRender example. The infrastructure launches that set of code in a separate window. When the operation is applied during the code's main thread loop, a second buffer is filled from the field variable specified by the `DATA_NAME` and `FIELD_NAME` parameters. It then signals the visualization thread to swap buffers. It is restricted to single cubic domains in this version of the infrastructure because it uses the rendering kernels from the example with few modifications to apply in other geometries. Most of the settings for producing the coloring and view were obtained by interacting with the visualization to determine a 'good' view. This mode of interaction is described below the file listing. Other options include the commented `FILE_HEAD` and `SAVE_IMG` options. If re-enabled, the operation outputs a '.ppm' image file for every iteration that is drawn. Iteration skipping can be adjusted by the `SKIP` parameter to reduce the number of files. The `VIEW_ORBIT` parameter tells the visualization to rotate by the indicated number of degrees once per iteration automatically in addition to the interactive rotations to help make the 3D nature of the volume rendering more intuitive.

```
############################################################################
DEFINE STAGE PLOT

DEFINE OPERATION
      TYPE = VolumeRenderOp
      DATA_NAME = FieldData
# FILE_HEAD = VolViz0T
# SAVE_IMG = TRUE
      FIELD_NAME = Ne-
      SKIP = 1
# INVERT = TRUE
      DENSITY = 0.04
      BRIGHTNESS = 1.7
      TRANSFERUPPERBOUND = 3.8e5
      TRANSFERLOWERBOUND = 2.5e3
      LOG_PLOT = FALSE
      INVERT = FALSE
      VIEW_TRANSLATION = (0.0,0.0,-3.6)
      VIEW_ROTATION = (0.4,51.6,0.0)
      VIEW_ORBIT = (0.0,-2.0,0.0)
      WINDOW_SIZE = (960,960)
END OPERATION
```

An example of the output displayed with the default settings by the realtime visualization can be seen in Figure 1. This shows the electron cloud density in the box after 828 timesteps using the default visualization parameters.

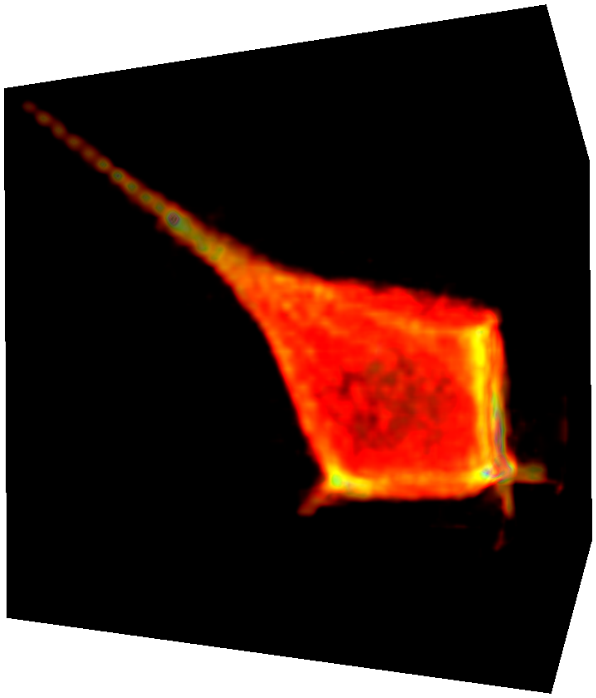


Fig. 1. Volume rendering example output of electron density in grounded box

Left-clicking and dragging the mouse rotates the visualization. Right-clicking and dragging the mouse scales the view. Center or simultaneous left and right clicking while dragging the mouse pans the viewport. The '-+' keys adjust the density for the ray tracing. Lower values make the electron cloud more translucent and higher makes the rendering thicker and only values closer to the surface of the cloud are visible. The square bracket keys, '[]', adjust the 'brightness' of the display. The keys on the next row down, ';'', adjust the 'transferUpperBound', which is essentially the top edge of the colormap. The next row down from there, the ',.' keys adjust the 'transferLowerBound'. This is similarly the bottom edge of the colormap. The 'i' key inverts the coloring of the display to a black box on a white background. As the keys adjust the settings, the visualizer displays the adjusted parameters interwoven with normal output from the infrastructure. Once a good view has been determined, the options can then be fed back into the operation's parameters for future runs. The output of holding the '-' is shown below with some additional whitespace for clarity while a similar line is produced by the mouse adjustments as well.

```
Iteration 1747: Time=4.367510e-06 dt=2.500000e-09 [Wall Clock:477.743864]
density = 0.07, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
density = 0.06, brightness = 2.10, transferUpperBound = 3.45e+05,
       transferLowerBound = 1.39e+04, invert = F
density = 0.05, brightness = 2.10, transferUpperBound = 3.45e+05,
           transferLowerBound = 1.39e+04, invert = F
NORM: 4.386121e-04
Iteration 1748: Time=4.370010e-06 dt=2.500000e-09 [Wall Clock:478.055719]
```

The last additional operations are a commented version of the LogicalFieldWriteVTKOp which writes the field data to output files rather than relying on the realtime visualization. This is necessary for running the tutorial on systems that do not include NVIDIA GPU's that are compatible with the direct OpenGL interface used by the volume renderer. The options are similar to those of the heatbath tutorial. The last operation is a final NextStageOp to tell the code to advance to the next stage, or in this case, loop back to the first stage.

```

# DEFINE OPERATION
# TYPE = LogicalFieldWriteVTKOp
# DATA_NAME = FieldData
# FILE_HEAD = plt/plt_
# FIELD_NAME = Ne- CNe- Np+ CNp+ phi rho
# SKIP = 5
# # FORMAT = BINARY # Won't open!
# HELP = TRUE
# END OPERATION


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

END STAGE PLOT
############################################################################
```

4. RESULTS

The example in this tutorial was originally developed to verify TURF functionality with respect to the ICEPIC code. Using as similar parameters as possible between the two codes, the example was run and visualized in ParaView as shown in Figure 2. The setup was nearly identical to what was outlined above except more particles were used to provide smoother output. In particular, a FILL_RATIO of 4.0 was used with the e-DST, and the default value of 0.5 was used for the p+DST to ensure a similar number of particles were used in TURF as in ICEPIC. For the realtime 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 much more noisy. 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. The SPDistBoxICOp available in the TURF-DEV development package would provide a more directly comparable initialization with constant particle weights, but it was added after the original verification runs were performed and will not be included in the infrastructure core until the next revision.

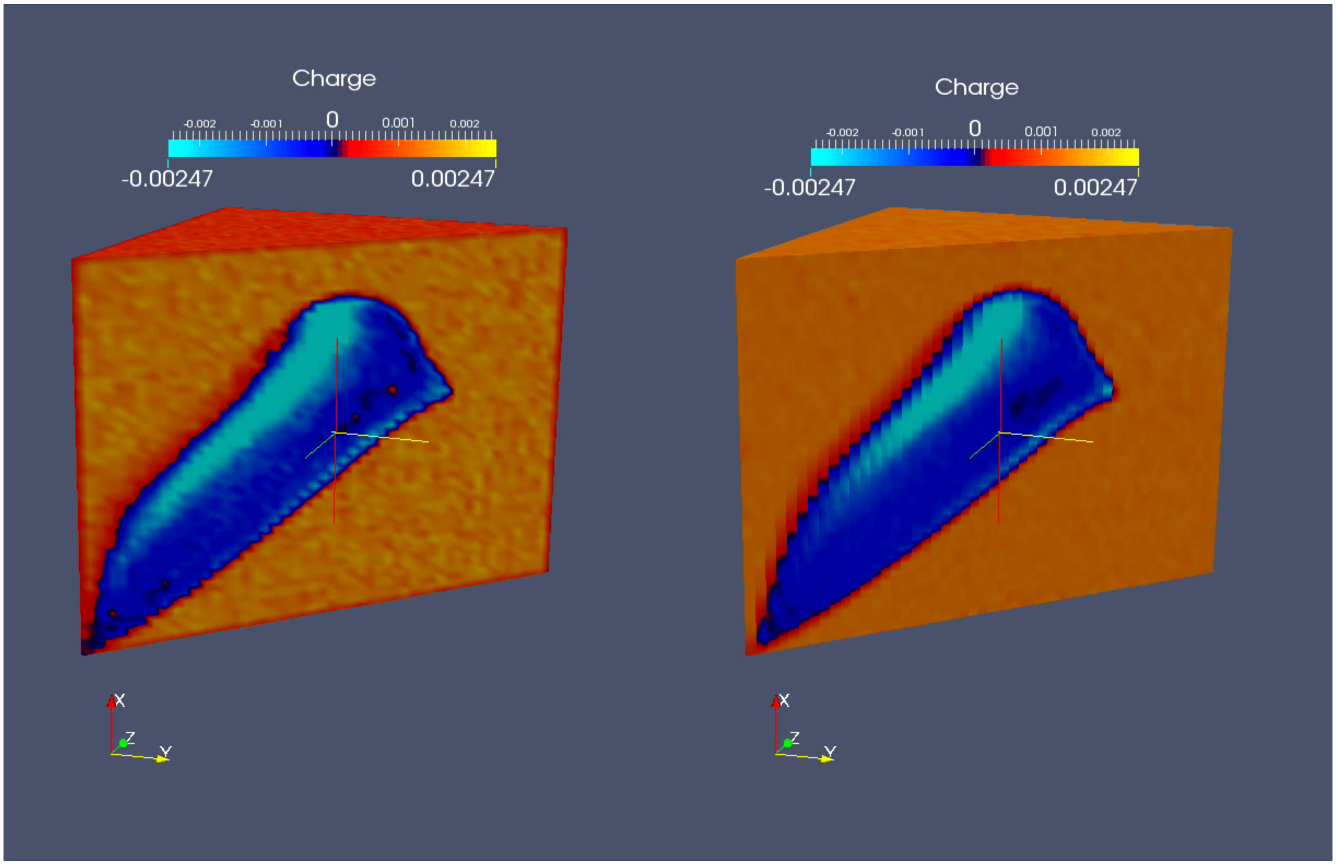


Fig. 2. Comparison of ICEPIC (left) and TURF (right) grounded box results

5. Appendix

Table 1. Summary of operations listed in `operations.list`.

Stage	Operation	Description
INITIALIZE	`SPDistConstantIC`	Initial particle distribution
	`SPDistCellIDOp`	Flag cell in which particle resides
	`SPDistSortOp`	Sort particles in cells by CellID
	`LogicalFieldSetOp`	Set field values to constant
	`SPDistDensityToFieldOp`	Sum real and computational particles/cell to field
	`SPDistToFieldOp`	Sum particle charges to field entry
	`NextStageOp`	Continue to next stage
SOLVE	`LogicalBCConstantOp`	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
	`SPDistESPushOp`	Electrostatic particle push using node electric field
	`SPdistSplitOp`	Split particle distribution by CellID flag
PLOT	`VolumeRenderOp`	Single cubic domain realtime volume rendering
	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots

References

[1] Matthew Bettencourt and Andrew 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.

TURF HEATBATH EXAMPLE - PART 1: INITIALIZATION

JONATHAN TRAN, ERC INC, AFRL/RQRS

Contents

1. Introduction 1
2. world.list 2
2.1. operations.null.list 4
2.2. World-Rank.html 5
3. Adding Particles 7
3.1. operations.addparticles.list 7
3.2. operations.writeoutput.list 8
4. Appendix 9
References 9

1. Introduction

This tutorial is the first of two parts which give an overview of the heatbath example in TURF. Prior to running this example you should already have installed TURF and verified that it was installed properly. For information on this please read THE_INSTALL_GUIDE. All relevant files can be located in `dat-SMMURF/tutorial/heatbath`. You should see several files with the `.list` extension, which act as the scripting files for TURF. In addition to the TURF software you will also need a scientific visualization software that can read VTK files such as Paraview[1] or VisIt[2]. Also useful is a text file comparison utility such as Meld[3] or diff.

The heatbath example studies particles undergoing thermal expansion confined within a box. As mentioned before, this tutorial is the first of two parts. We will discuss setting up a coordinate system, logical domain and other necessary databases required for a TURF simulation in Section 2. It also explains the use of World-Rank.html for visualizing the simulation space. Lastly, section 3 details the addition of particles to the domain and the operations necessary for TURF to output data compatible with VisIt[1]. In TURF Heatbath Example: Part 2 we will discuss the details of moving particles around the domain and having these particles specularly reflect off the boundary of the domain. Lastly we will construct the same simulation using multiple domains.

Date: 11/25/2014.

[1] `www.paraview.org`

[2] `https://visit.llnl.gov`

[3] `http://meldmerge.org`

2. WORLD.LIST

Running the TURF executable in the working directory will have TURF search for the default script file, `world.list`, and parses it automatically. In this example, `world.list` is a symbolic link to the file `world.heatbath.list`. We can take a look at the script by opening either file.

```
DEFINE WORLD
      NAME = Heatbath-Example
##########################################################
### Initially all turned off - Code does not advance ###
### ###
##########################################################
### Timestep advaces but code does nothing ###
### op_file = operations.null.list ###
##########################################################
### Add a a particle distribution object with some ###
### particles in a box ###
### op_file = operations.addparticles.list ###
##########################################################
### Write vtk field output files periodically ###
### op_file = operations.writeoutput.list ###
##########################################################
```

We begin by defining the world and giving it a name. This name is arbitrary and can be anything. We then have a block of commented lines which are calls to different operation files. Each `operation.list` file when uncommented will run a different example, building on itself and adding functionality. Over the course of this tutorial we would like to elaborate on the commands used to achieve this functionality.

We then define the world coordinate system, time step, field names and stages used in the example. TURF is written to assume all units are in MKS. With this in mind, the cell size is $100\mu m$ and our time step is $1ns$. The length of the simulation is defined by the `end_time` of $250ns$. Dividing `end_time` by `start_dt` will give us the number of iterations in the simulation, 250. Defining the various fields ensures there is memory to save the number of helium per cell and the computational number of helium per cell. The heat bath example has two stages named `INITIALIZE` and `MOVE`. A stage is a communication synchronization point after which all of the domains within the simulation can vote on whether to proceed to the next stage or repeat the current stage. This synchronization is important because different processes may finish operating on their domains before other processes do. Failure to properly synchronize may cause the simulation to produce incorrect results. Note that the names for fields and stages are only labels similarly to the world name and do not refer to any existing information in the code. However, if used elsewhere in the code it is important to reference the same name. Here we have a field named `NHe` which stores the physical number of Helium particles per cell and `CNHe` which contains the computational number of Helium particles per cell.

```
      coordinates = cartesian
      origin = (0.0,0.0,0.0)
      delta = (100.0e-6,100.0e-6,100.0e-6)
      end_time = 250.0e-9
      start_dt = 1.0e-9
# Names Should not be fully contained in an earlier name for Plotting
      fields = [NHe, CNHe]
      stages = [INITIALIZE, MOVE]
END WORLD
```

Lastly we define our domain. The location of the domain is relative to the origin of the world coordinate system. The mesh spacing is global to the coordinate system and must be the same across all domains. In this example our domain is a cube with a length of $3.2mm$ with a mesh spacing of $0.1mm$ in all directions.

```
############################################################################
## Domain Geometry
############################################################################
DEFINE DOMAIN DOM000
      bound_lo = (0.0,0.0,0.0)
      bound_hi = (3.2e-3,3.2e-3,3.2e-3)
END DOMAIN
```

2.1. **operations.null.list.** The first example details the creation of a domain, followed by 250 iterations of nothing. To run this case, we uncomment the following line in the `world.list` file.

```
##########################################################
### Timestep advaces but code does nothing ###
         op_file = operations.null.list ###
##########################################################
```

By doing so we call the `operations.null.list` file which defines the different operations used within the stages of the simulation. You will notice that for both the initialize stage and the move stage, there exist only one operation of the type `NextStageOp`. This operation simply tells the code to continue onto the next stage.

```
####################################################################
## Initial Particle Distibutions and Ghost/Exchange Distributions ##
####################################################################
DEFINE STAGE INITIALIZE
DEFINE OPERATION
# Default Criteria to Proceed to the Next Stage
      TYPE = NextStageOp
END OPERATION
END STAGE INITIALIZE
############################################################################

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

It is important to remember that the stage names initialize and move are just that, only names. Despite being named initialize, this stage is called every iteration and the GPU cores must sync before moving onto the move stage. As we will see in later examples, it is hard coded in the operations whether it is an initial operation or one that occurs iteratively.

2.2. **World-Rank.html.** When TURF is run, a html file named `World-Rank.html` is automatically generated. When opened, the user can view the object hierarchy of the example. At the base of the tree is the logical world, which was named Heatbath-Example. The branches include `GSObject` named `GSMemberVector` which have the functionality of a vector and can be used by the GPU, there is 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 which are automatically generated. Another useful feature is the visualization of the simulation environment. For this current example there exist only a single domain as shown by the blue cube. The surrounding gray region is a layer of three ghost cells which are automatically generated when the domain is formed.

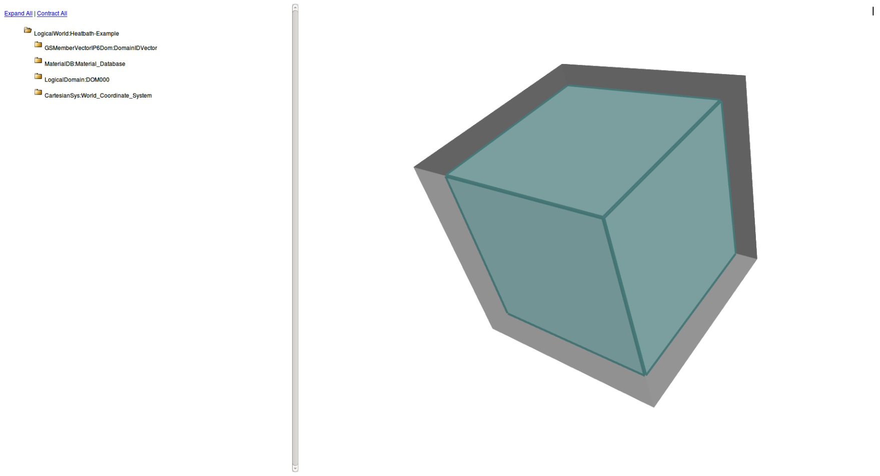

Expand All | Contract All
LogicalWorld:Heatbath-Example
GSMemberVectorIP6Dom:DomainIDVector
MaterialDB:Material_Database
LogicalDomain:DOM000
CartesianSys:World_Coordinate_System

FIG. 1. World-Rank(0).html with contracted database hierarchy.

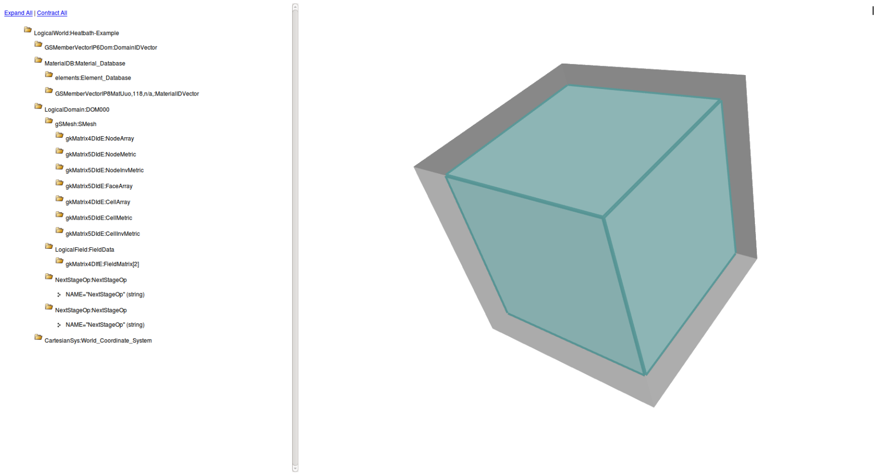

Expand All | Contract All
LogicalWorld:Heatbath-Example
GSMemberVectorIP6Dom:DomainIDVector
MaterialDB:Material_Database
elements:Element_Database
GSMemberVectorIP8MatUuo,118,n/a,:MaterialIDVector
LogicalDomain:DOM000
gSMesh:SMesh
gkMatrix4DIdE:NodeArray
gkMatrix5DIdE:NodeMetric
gkMatrix5DIdE:NodeInvMetric
gkMatrix5DIdE:FaceArray
gkMatrix4DIdE:CellArray
gkMatrix5DIdE:CellMetric
gkMatrix5DIdE:CellInvMetric
LogicalField:FieldData
gkMatrix4DIfE:FieldMatrix[2]
NextStageOp:NextStageOp
NAME="NextStageOp" (string)
NextStageOp:NextStageOp
NAME="NextStageOp" (string)
CartesianSys:World_Coordinate_System

FIG. 2. World-Rank(0).html with expanded database hierarchy.

By selecting on the visualization and pressing the 'm' key, we can cycle through volume view, line view and point view. This is useful for visualizing objects within the domain as we will see in the later examples.

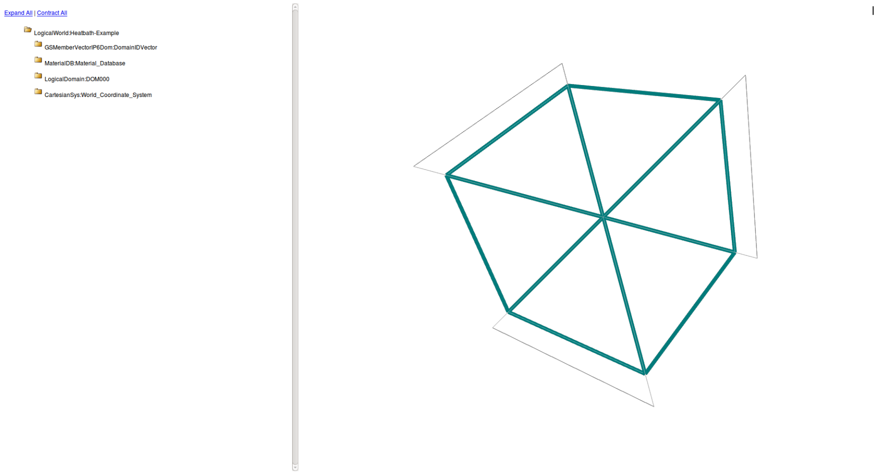

Expand All | Contract All
LogicalWorld:Heatbath-Example
GSMemberVectorIP6Dom:DomainIDVector
MaterialDB:Material_Database
LogicalDomain:DOM000
CartesianSys:World_Coordinate_System

Fig. 3. World-Rank(0).html in line view.

The usefulness of the `World.Rank` output will become evident as we add functionality to example. The object hierarchy and visualization will allow us to directly see the changes we have made in the course of this tutorial.

3. Adding Particles

The following example creates a distribution of helium particles to fill a portion of the domain. We will then observe the changes made in the `World.Rank` file and learn how to output the data in VTK format so it can be studied using a visualization software such as VisIt.

3.1. operations.addparticles.list.

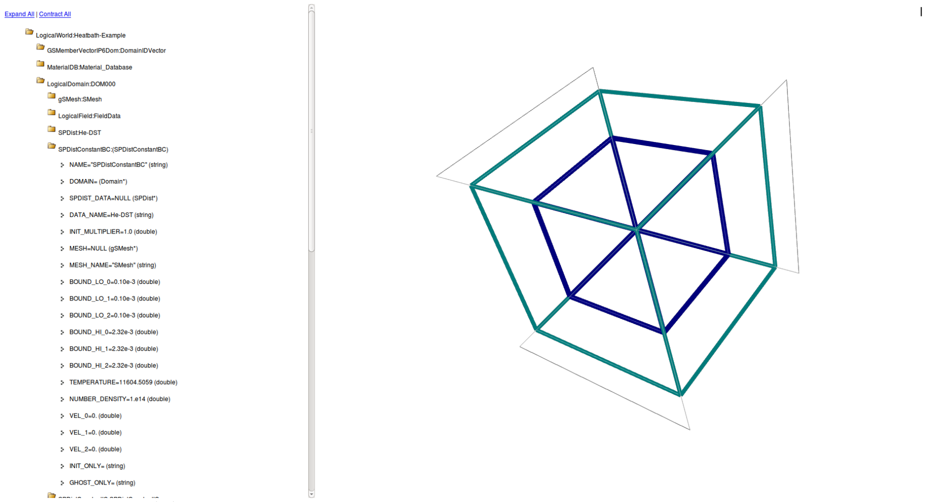
To change the example we simply call a different `operation.list` file. We will do this by commenting out the previous op_file `operations.null.list` and also uncommenting the next line named `operations.addparticles.list` as shown below.

```
##########################################################
### Timestep advaces but code does nothing ###
### op_file = operations.null.list ###
##########################################################
### Add a a particle distribution object with some ###
### particles in a box ###
         op_file = operations.addparticles.list ###
##########################################################
```

If we open both operations files with meld, we can directly compare changes between these two files, we can see that there is one additional operation defined of the type `SPDistConstantIC` shown below. This operation does is define a box with upper and lower boundary, and distribute particles with a given number density and temperature. The particles themselves will have a given charge, mass (in units on proton mass) and drift velocity. This specific operation is an initial condition so it will only do something when it is first read.

```
DEFINE OPERATION
      TYPE = SPDistConstantIC
      DATA_NAME = He-DST
      MAX_NP = 1280000
      FILL_RATIO = 0.25
      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 # 1ev
      number_density = 1.e14
      Z = 0
      Mass = 4.0 Mp
      vel = (0.,0.,0.)
END OPERATION
```

Since there are no push operations, the particles will remain at their locations for the length of the simulation. Looking at the `World-Rank.html` we can see the bounding box for which the particles will be distributed within.



Expand All | Contract All

```
LogicalWorld:Heatbath-Example
  GSMemberVectorIP6Dom:DomainIDVector
  MaterialDB:Material_Database
  LogicalDomain:DOM000
    gSMesh:SMesh
    LogicalField:FieldData
    SPDist:He-DST
    SPDistConstantBC:(SPDistConstantBC)
      NAME="SPDistConstantBC" (string)
      DOMAIN= (Domain*)
      SPDIST_DATA=NULL (SPDist*)
      DATA_NAME=He-DST (string)
      INIT_MULTIPLIER=1.0 (double)
      MESH=NULL (gSMesh*)
      MESH_NAME="SMesh" (string)
      BOUND_LO_0=0.10e-3 (double)
      BOUND_LO_1=0.10e-3 (double)
      BOUND_LO_2=0.10e-3 (double)
      BOUND_HI_0=2.32e-3 (double)
      BOUND_HI_1=2.32e-3 (double)
      BOUND_HI_2=2.32e-3 (double)
      TEMPERATURE=11604.5059 (double)
      NUMBER_DENSITY=1.e14 (double)
      VEL_0=0. (double)
      VEL_1=0. (double)
      VEL_2=0. (double)
      INIT_ONLY= (string)
      GHOST_ONLY= (string)
```

FIG. 4. World-Rank(0).html visualizing both the domain and region in which particles will be distributed. You may notice the code automatically generated objects when SPDistConstantIC is called.

3.2. **operations.writeoutput.list.** To run the next example, open the world.list file again. Comment the line `op_file = operations.addparticles.list` and uncomment the line `op_file = operations.writeoutput.list`, similarly as before. This simulation is exactly the same as the previous except now we will output the particle density distribution in a VTK format compatible with visualization software such as VisIt. This output is generated every five iterations. To do so we will use the following operations in the move stage:

```
##############################
## Sum to Fields for Output ##
##############################
DEFINE OPERATION
      TYPE = LogicalFieldSetOp
      DATA_NAME = FieldData
      FIELD_NAME = CNHe
      VALUE = 0.0
END OPERATION
DEFINE OPERATION
      TYPE = LogicalFieldSetOp
      DATA_NAME = FieldData
      FIELD_NAME = NHe
      VALUE = 0.0
END OPERATION
DEFINE OPERATION
      TYPE = SPDistDensityToFieldOp
      FIELD_DATA_NAME = FieldData
      SPDIST_DATA_NAME = He-DST
      PSORT_NAME = Sort_He-DST
      FIELD_NAME = NHe CNHe # Computational and Physical Number per Cell
END OPERATION
######################
## Write VTK Output ##
######################
DEFINE OPERATION
```

```
      TYPE = LogicalFieldWriteVTKOp
      DATA_NAME = FieldData
      FILE_HEAD = heatbath_1/plt_
      FIELD_NAME = NHe CNHe
      SKIP = 5
      HELP = TRUE
END OPERATION
```

The operation `LogicalFieldSetOp` sets the value of the field to zero for the given field name, in this case we have an operation for CNHe and another for NHe. The operation `SPDistDensityToFieldOp` sums the quantity of helium for each cell and stores it into CNHe and NHe. The last operation `LogicalFIeldWriteVTKOp` outputs the field data for CNHe and NHe in VTK format every five iterations.

Opening the VTK files with visualization software such as VisIt we notice TURF has created a cube with a uniform density of helium particles just as we expected. As the simulation progresses in time, the particles remain unchanged. In the next tutorial TURF Heatbath Example: Part 2 we will discuss how to push particles through the domain and how to handle particles that leave the specified domain.

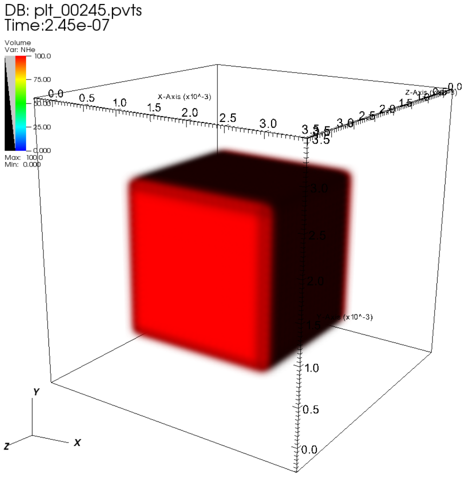


FIG. 5. The final state of the helium particles visualized in VisIt.

4. Appendix

TABLE 1. Summary of operations listed in `operations.dsmc1Dshock.list`.

Stage	Operation	Description
INITIALIZE	`SPDistConstantIC`	Initial particle distribution
MOVEOP	`NextStageOp`	Continue to next stage
	`SPDistDensityToFieldOp`	Sum particles per cell for field entry
	`SPDistSortOp`	Sort particles according to cell ID
	`LogicalFieldSetOp`	Initialize the field parameters
POSTOP	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots

References

[1] Brad Whitlock. *Getting Data Into VisIt.* Lawrence Livermore National Laboratory, Version 2.0.0 edition, July 2010. LLNL-SM-446033.

TURF HEATBATH EXAMPLE - PART 2: EVOLUTION

JONATHAN TRAN, ERC INC, AFRL/RQRS

Contents

1. Introduction 1
2. Particle Pushing 2
2.1. operations.push-untrimmed.list 2
2.2. operations.push.list 2
3. Particle Heatbath 4
3.1. operations.heatbath.list 4
3.2. Multiple Domain Case 5
4. Appendix 8
References 8

1. Introduction

This tutorial is the second of two parts which give an overview of the heatbath example in TURF. If you have not yet gone over the first heatbath tutorial, it is advised that be done prior to running this example. You should already have installed TURF and verified that it was installed properly. For information on this please read THE_INSTALL_GUIDE. All relevant files can be located in `dat-SMMURF/tutorial/heatbath`. You should see several files with the `.list` extension, which act as the scripting files for TURF. In addition to the TURF software you will also need scientific visualization software that can read VTK files such as Paraview[1] or VisIt[2][1]. Also useful is a text file comparison utility such as Meld[3] or diff.

In the first part of the heatbath tutorial we discussed the basics of constructing a world coordinate system, domain and the operations needed to add particles to the simulation. In this tutorial we plan on expanding our simulation by adding a time dependence. Section 2 we will discuss how to thermally expand particles and how to handle particles which have left the domain. In Section 3 we will impose boundary conditions which specularly reflect incoming particles, thus completing the heatbath example. Lastly, we would like to construct the same simulation with multiple domains requiring changes to both the world.list and operations.list files.

Date: 11/25/2014.

[1] www.paraview.org

[2] https://visit.llnl.gov

[3] http://meldmerge.org

Distribution A: Approved for public release; unlimited distribution; PA (Public Affairs) Clearance Number TBD#XXXXX.

2. Particle Pushing

In TURF Heatbath Example: Part 1 our final example had particles which remained stationary for the length of the simulation. The next logical step is to allow the particles to thermally expand.

2.1. operations.push-untrimmed.list.

Pushing particles is quite simple, requiring one additional operation. We begin by running the `operations.push-untrimmed.list` file the same way as before. In this case after initially distributing the particles in a cube, the operation `SPDistMoveOp` will thermally expand the particle distribution over time.

```
DEFINE OPERATION
      TYPE = SPDistMoveOp
      SPDIST_DATA_NAME = He-DST
END OPERATION
```

You may notice the total number of particles in the simulation decreasing over time. This is due to a lack of boundary conditions for our domain; particles continue on their trajectory beyond the bounds of the simulation.

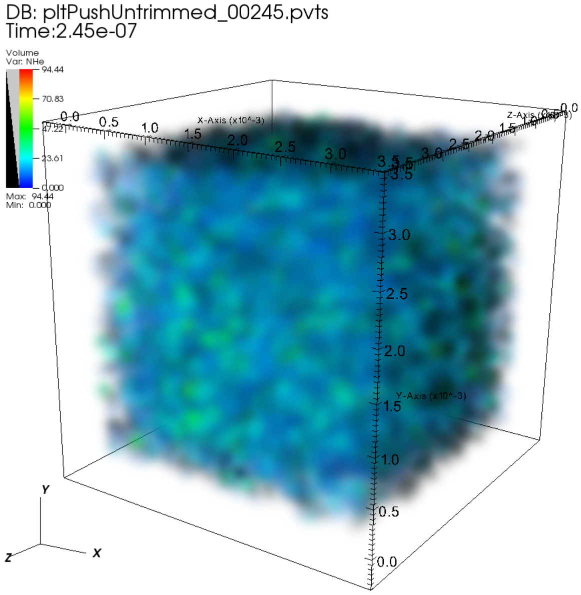


Fig. 1. The final state of the helium particles visualized in VisIt for the push-untrimmed example.

2.2. operations.push.list.

Running this example shows an output identical to the push-untrimmed case however, it now has functionality to sort through the particle list and explicitly remove those that have moved into the ghost cells as opposed to the previous example where these particles would be ignored and continue on their trajectory. It does so using the following operations:

```
DEFINE OPERATION
      TYPE = SPDistConstantIC
      DATA_NAME = He-GST
      MAX_NP = 1280000
END OPERATION
###################################################
## Initial Sort Removes Particles Outside Domain ##
###################################################
DEFINE OPERATION
      TYPE = SPDistCellIDOp
      DATA_NAME = He-DST
END OPERATION
DEFINE OPERATION
      TYPE = SPDistSortOp
      NAME = Sort_He-DST
      SRC_NAME = He-DST
      DST_NAME = He-GST
END OPERATION
```

From the previous example we have added a distribution for helium named He-GST. The operation `SPDistCellIDOp` determines what cell every particle is in and the operation `SPDistSortOp` moves particles in ghost cells from He-DST to He-GST.

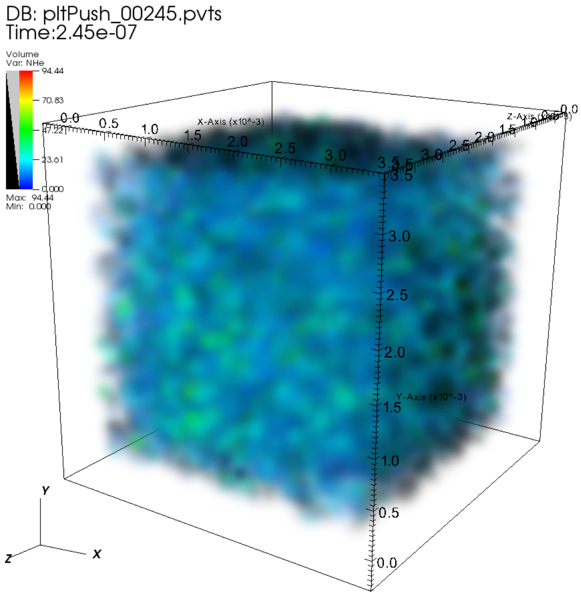


FIG. 2. The final state of the helium particles visualized in VisIt for the push example.

3. Particle Heatbath

3.1. **operations.heatbath.list.** The final example is the particle heatbath, thermally expanding in a box. To do so, we impose boundary conditions and have particles specularly reflect off the walls of the domain. By running `operations.heatbath.list`, we use an operation named `SPDistBCSpecOp` which creates a region that share a surface with the domain. These regions will reflect incoming particles in a given direction. An example of the use of this operation is shown below. The code requires us to write this operation six times, one for every surface of the cubic domain. A visualization of these regions can be seen below in the .html file.

```
###############################
## 1-Walls reflect particles ##
###############################
DEFINE OPERATION
      TYPE = SPDistBCSpecOp
      DATA_NAME = He-DST
      DIRECTION = xm
      BOUND_LO = (-100.e-4,-100.e-4,-100.e-4)
      BOUND_HI = ( 0.00001e-4, 132.e-4, 132.e-4)
END OPERATION
```

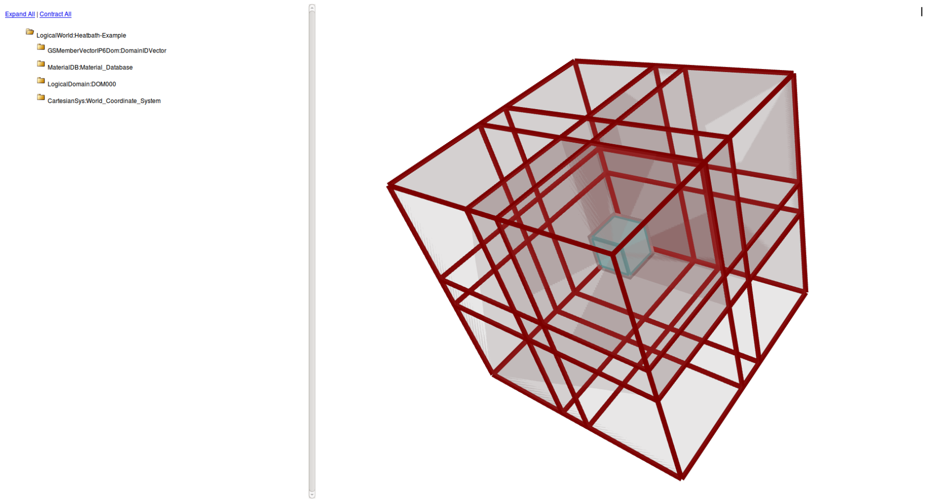

Expand All | Contract All
LogicalWorld:Heatbath-Example
GSMemberVectorIP6Dom:DomainIDVector
MaterialDB:Material_Database
LogicalDomain:DOM000
CartesianSys:World_Coordinate_System

Fig. 3. `World-Rank(0).html` visualizing both the domain and boundary condition region which specularly reflects incoming particles.

Taking a look at the output in VisIt we notice the total number of particles remains unchanged. If the simulation is run longer, it will eventually approach steady state.

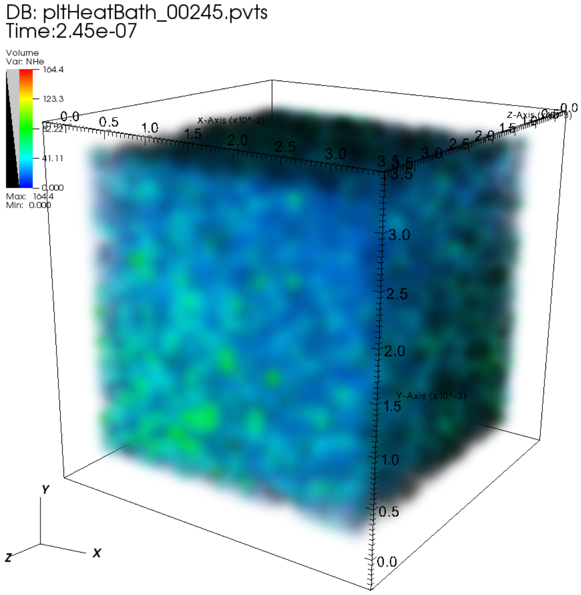


FIG. 4. The final state of the helium particles visualized in VisIt for the heatbath example.

3.2. **Multiple Domain Case.** The final example demonstrates the ability to use multiple domains. Doing this requires us to change the domain geometry in the world.list file. Luckily for us we already have a file we can change the pointer to named world.heatbathx2.list. We first remove the previous pointer and create a new pointer with the same name to `world.heatbathx2.list`.

```
user@comp:~/SMMURF$ projects/dat-SMMURF/> rm world.list
user@comp:~/SMMURF$ projects/dat-SMMURF/> ln -s world.heatbathx2.list world.list
```

Comparing the two world.list files the domain geometry is the only modification.

```
##############################################################################
## Domain Geometry
##############################################################################
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 = (3.2e-3,3.2e-3,3.2e-3)
END DOMAIN
```

When using multiple domains, it must be possible to exchange particles between the different domains. Looking at the operations.list file we notice two significant differences between the single domain and multiple domain cases. The first of which handles the exchange of particles from one domain to the other using a distribution named He-EXC. The operation `SPDistCombineOp` unifies the particles from He-EXC distribution with He-DST at the beginning of every loop.

```
DEFINE OPERATION
      TYPE = SPDistConstantIC
      DATA_NAME = He-EXC
      MAX_NP = 1280
```

```
END OPERATION
##########################################################
## Combine EXC into DST from Patch at End of Move Stage ##
##########################################################
DEFINE OPERATION
      TYPE = SPDistCombineOp
      SRC_DATA_NAME = He-EXC
      DST_DATA_NAME = He-DST
# VERBOSE = TRUE
END OPERATION
```

If the particle moves between the two domains, it is temporarily removed from He-DST placed into the distribution He-EXC until the beginning of the next iteration which we saw in the previous block of code.

```
###########################################################
## Split Particles Still Outside Active Domain for Patch ##
###########################################################
DEFINE OPERATION
      TYPE = SPDistCellIDOp
      DATA_NAME = He-DST
END OPERATION
DEFINE OPERATION
      TYPE = SPDistSplitOp
      SRC_NAME = He-DST
      DST_NAME = He-EXC
END OPERATION
DEFINE OPERATION
      TYPE = SPDistPatchOp
      SRC_NAME = He-EXC
      DST_NAME = He-EXC
END OPERATION
DEFINE OPERATION
      TYPE = NextStageOp
END OPERATION
```

Running the simulation we see that the output is similar to the single domain heatbath case as we would expect it to be, do note however in the `world.list` file the overlapping ghost cells in the volume domain which are required for the exchange of particles between domains.

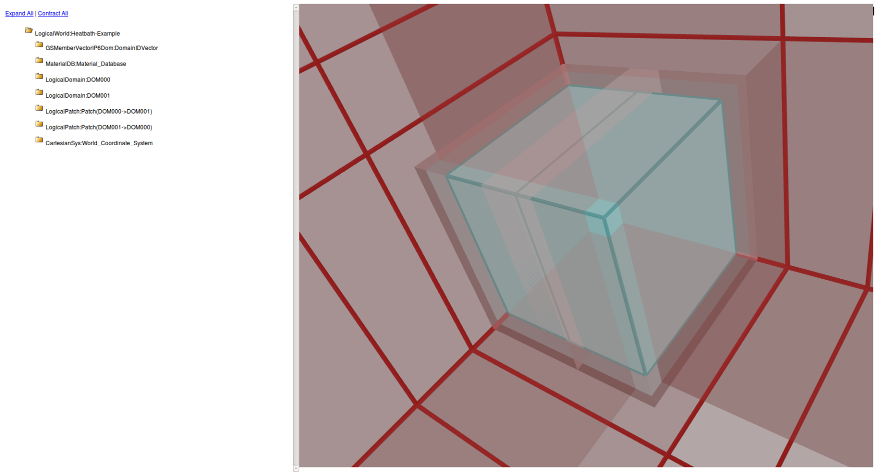

Expand All | Contract All
LogicalWorld:Heatbath-Example
GSMemberVectorIP6Dom:DomainIDVector
MaterialDB:Material_Database
LogicalDomain:DOM000
LogicalDomain:DOM001
LogicalPatch:Patch(DOM000->DOM001)
LogicalPatch:Patch(DOM001->DOM000)
CartesianSys:World_Coordinate_System

FIG. 5. Volume domain split into two pieces.

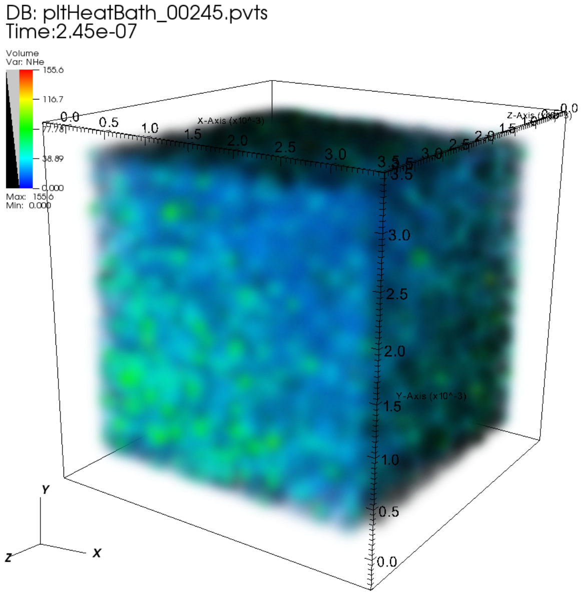


FIG. 6.

This completes the TURF Heatbath Example. If you have any questions or concerns, please direct them to Jonathan Tran[4] at ARFL/RQRS.

[4] jonathan.tran.3.ctr@us.af.mil

4. Appendix

Table 1. Summary of operations listed in `operations.dsmc1Dshock.list`.

Stage	Operation	Description
INITIALIZE	`SPDistConstantIC`	Initial particle distribution
MOVEOP	`NextStageOp`	Continue to next stage
	`SPDistMoveOp`	Advancement of particles
	`SPDistDensityToFieldOp`	Sum particles per cell for field entry
	`SPDistSortOp`	Sort particles according to cell ID
	`LogicalFieldSetOp`	Initialize the field parameters
	`SPDistBCSpecOp`	Specularly reflecting boundary condition
	`SPDistCombineOp`	Unifies the particles from different distributions
	`SPDistCellIDOp`	Marks the cell ID in which particles reside
	`SPDistSplitOp`	Splits particle distribution into two by cell ID
	`SPDistPatchOp`	Transfers particles between domains
POSTOP	`LogicalFieldWriteVTKOp`	Write to output files for 3D plots

References

[1] Brad Whitlock. *Getting Data Into VisIt.* Lawrence Livermore National Laboratory, Version 2.0.0 edition, July 2010. LLNL-SM-446033.