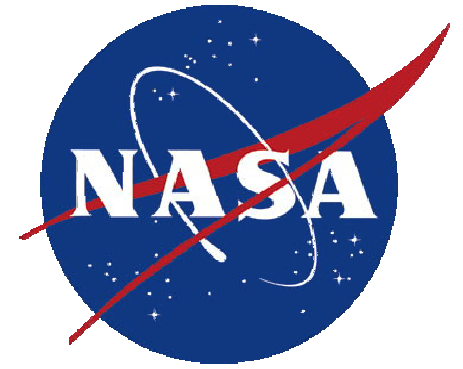




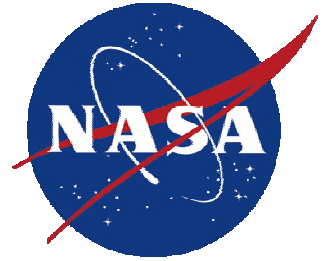
Development of an Object-Oriented DSMC Algorithm

Derek S. Liechty

Derek.S.Liechty@nasa.gov

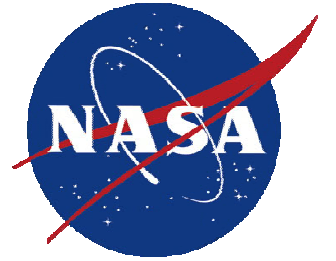
757-864-4387

Aerothermodynamics Branch
NASA Langley Research Center
Hampton, VA 23681-2199



Motivation

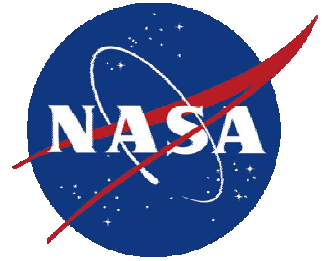
- Improved phenomenological models and updated algorithms
 - Chemistry/Internal Energy/Physics Modeling
 - Based on Bird's DS(n)V codes
 - Reduced time to solution
 - Higher density hence lower altitude simulations
- NASA standard DSMC code DAC
 - Well-established using 1990's Bird algorithms
 - Written in Fortran 77
 - Challenging to add updated simulation improvements
- Improved DSMC code needed
 - C++ object oriented programming (OOP) language chosen



Objectives

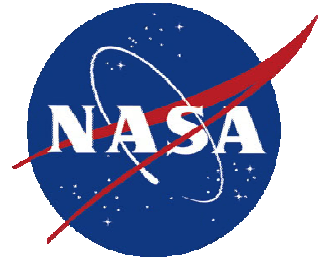
- Produce solutions more quickly
- Expand the useful range of DSMC in both velocity and altitude
- Provide expanded physical realism (modeling, physics, etc.)
- Explore benefits/disadvantages of Object-Oriented Programming (OOP) in DSMC.
- Compare performance with current NASA standard (DAC)

General Concepts of Object-Oriented Programming



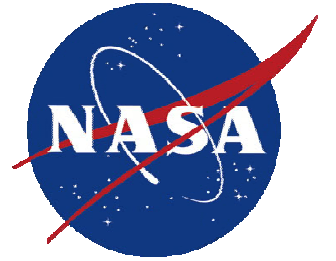
- Object-Oriented Programming (OOP) has the following advantages over conventional approaches:
 - OOP provides a clear modular structure for programs which makes it good for defining abstract datatypes where implementation details are hidden and the unit has a clearly defined interface.
 - OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
 - OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer.

Specific Concepts of Object-Oriented Programming (1)



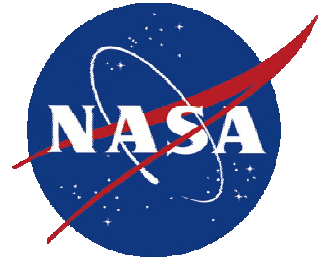
- Objects
 - Objects are the basic run-time entities in an object-oriented system. Programming problem is analyzed in terms of objects and nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data or code.
- Classes
 - A class is a collection of objects of similar type. Once a class is defined, any number of objects can be created which belong to that class.

Specific Concepts of Object-Oriented Programming (2)



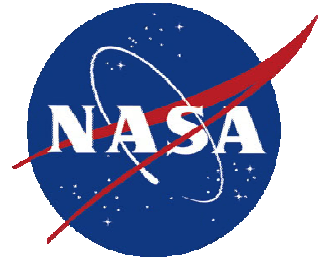
- Data Abstraction and Encapsulation
 - Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes. Storing data and functions in a single unit (class) is encapsulation. Data cannot be accessible to the outside world and only those functions which are stored in the class can access it.
- Inheritance
 - Inheritance is the process by which objects can acquire the properties of objects of other classes. In OOP, inheritance provides reusability, like adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both the classes.

Specific Concepts of Object-Oriented Programming (3)

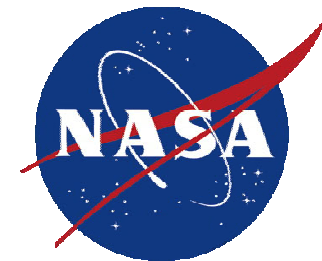


- Polymorphism
 - Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends on the data types used in the operation. Polymorphism is extensively used in implementing Inheritance.

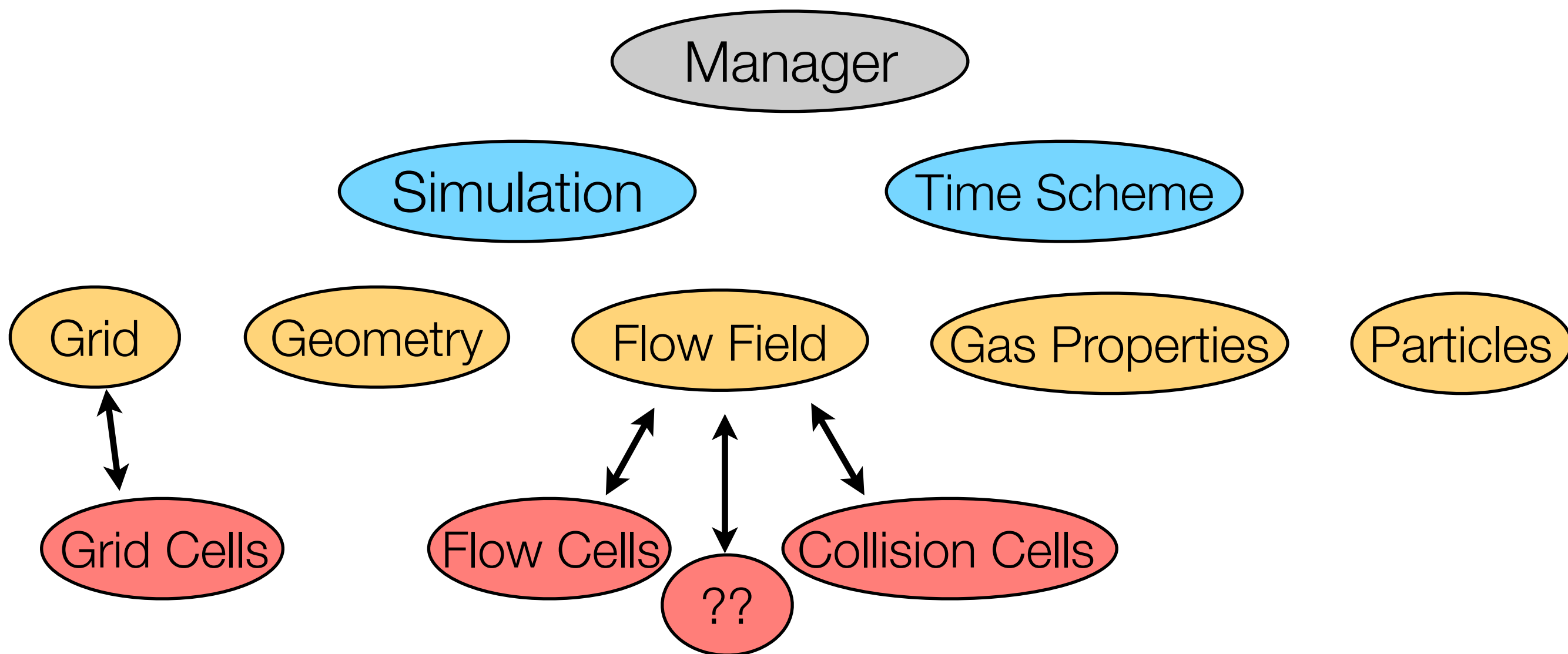
General Comments for Object Oriented Programming Applied to DSMC

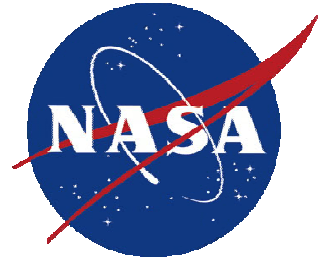


- Derived data types streamline memory and communication overheads. Create a base class for every object so derived types can be created.
 - For example, bParticle.cpp → cParticle_r.cpp adds rotational level information.
- Since derived data types will be used and each type will require different data, each class needs to be able to perform I/O for itself (or a group of the same type).
- Make class function calls general enough to take advantage of inheritance in the future.



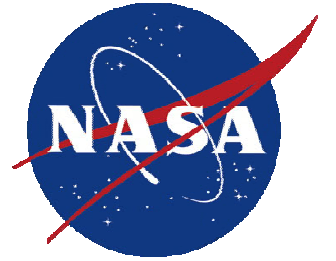
Organization of Code





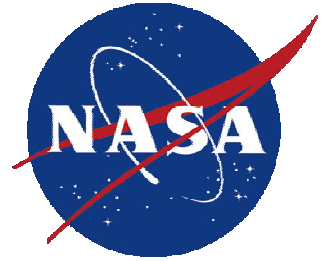
Manager

- Main interface
- Sets up and handles simulation type
- Organizes I/O operations
- Implementation allows for future development and interaction of various simulation types



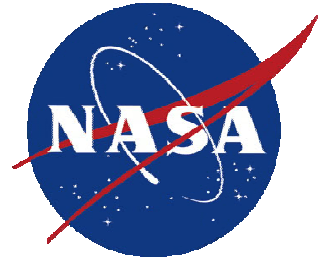
Simulation

- Currently, the only type of Simulation is strictly a DSMC simulation.
- Of particular importance in the Simulation is the `advanceTimeStep()` function that can be redefined for derived classes (say, if a system of equations needs to be solved, that can be added later).
- Also handles timing variables.



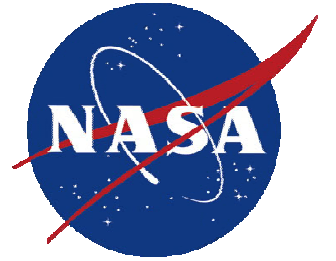
Time Scheme

- Current Time Scheme classes (derived from a base Time Scheme class):
 - DAC
 - Follows DAC time stepping scheme
 - All particles are moved at each time step (time step varies from cell to cell)
 - Currently not fully implemented.
 - Bird:
 - Follows the scheme that Bird's DS(n)V codes implement
 - Each particle has it's own time and time step, as do the cells, and the Simulation time is advanced by the minimum flow field time step
 - The particles are moved such that, on average, the particles in a flow cell are at the same time as the flow cell
 - Collisions are performed in a flow cell when the flow cell time falls half of the flow cell time step behind the Simulation time.



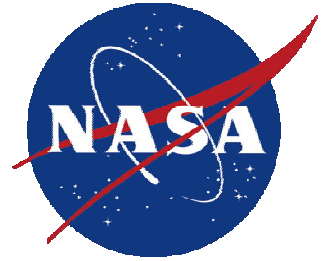
Time Scheme (cont.)

- Laux:
 - Taken from AIAA 1998-2670 (M. Laux).
 - Similar to Bird's time scheme, however each flow cell is assigned a "time zone" that is a multiple of the minimum flow field time step.
 - Flow cells are then treated when their "time zone" is up.



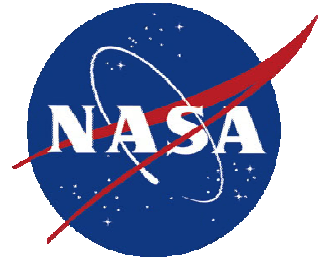
Gas Properties

- The Gas Properties class contains all data pertaining to the simulation gas.
 - Bulk flow properties (velocity, number density, temperature, mfp, mcr).
 - Fnum and RWF (for axisymmetric).
 - Species information.
 - Interaction parameters.
 - Reaction information.



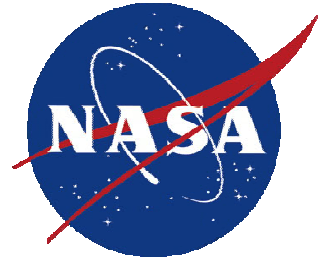
Gas Properties: Interaction

- Describes how the particles interact with each other (HS, VHS, VSS, etc.)
- Parameters that depend on the interaction model are calculated here
 - Mean free path
 - Mean collision rate
 - Deflection angles
 - Collision cross section
 - Particle relaxation



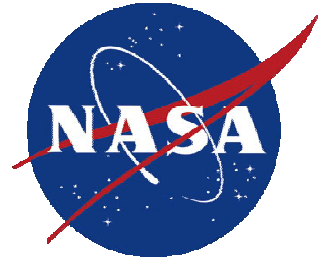
Gas Properties: Reaction

- Class to define reaction parameters and the probability of a reaction occurring.
- Currently have TCE and QK defined as reaction types.



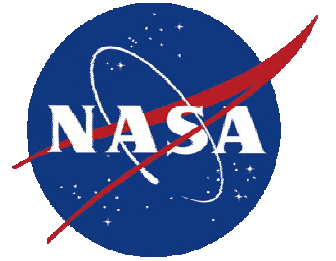
Gas Properties: Species

- The base Species class stores the species number, charge, diameter, mass, and free stream number fraction.
- It also stores pointers to the rotational, vibrational, and electronic relaxation schemes as well as recombination energy and ionization energy where applicable.



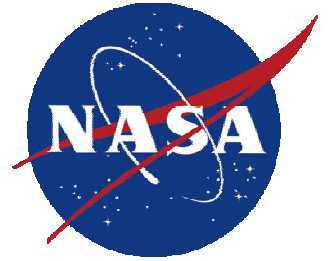
Grid/Geometry

- The Grid and Geometry classes are very much in the flavor of DAC. The major difference is that they are now organized in terms of OOP.
- Geometry
 - Triangulated surface where each surface element is an object containing all pertinent information.
- Grid
 - The grid is a 2-Level Cartesian grid (3-Level has been implemented but not included in this work) made up of Grid Cell objects that can be divided up into $I \times J \times K$ children Grid Cells.
 - The Grid Cells contain information about flow type, node number, limiting coordinates, pointers to children cells (if LI), pointers to Flow Cells (if LII), and any surfaces contained in the Grid Cell.
 - The Grid is also sent particles to be moved from the Flow Cells so that the Flow Field does not have to know what type of grid is being used.



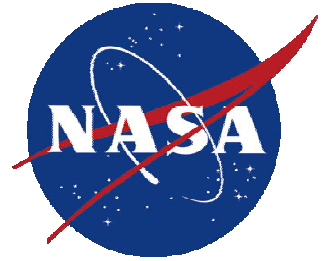
Flow Field

- The Flow Field contains a list of pointers to Flow Cells.
- Flow Cells
 - Flow Cells organize the particles and buffered particles across the Grid Cells and retain sampled information. The Flow Cells have been separated from the Grid Cells because of the possibility of multiple cut cells in a Grid Cell.
 - The base Flow Cell class only allocates memory for the basic geometric information and samples for velocity. Also contains pointers to particles and buffered particles.
 - Derived classes then add sampling for rotation, vibration, electronic energies and then also for electron energy for ionization (electron temperature).
- Collision Cells?
 - Collision Cells may be created separately from Flow Cells because of the possibility of a Flow Cell from a cut Grid Cell containing a very low number of particles. All collision parameters will be kept in the Collision Cells.



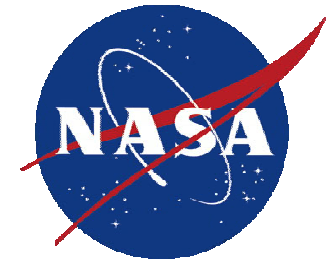
Particles

- Particles
 - Particles were originally stored as lists of pointers in the Flow Cells. Determined that pointer of arrays is better than array of pointers for speed.
 - The base Particle class stores the position, velocity, time, time step, species, particle number and last collision partner.
 - Derived Particle classes are defined for combinations of rotational, vibrational and electronic energy. Charged particles include linked electron velocity.
- Buffered Particles
 - Buffered Particles are kept in the Flow Cells now instead of using a global Buffered Particle list. This is to hopefully improve the time accuracy of the simulation.



Comparisons...

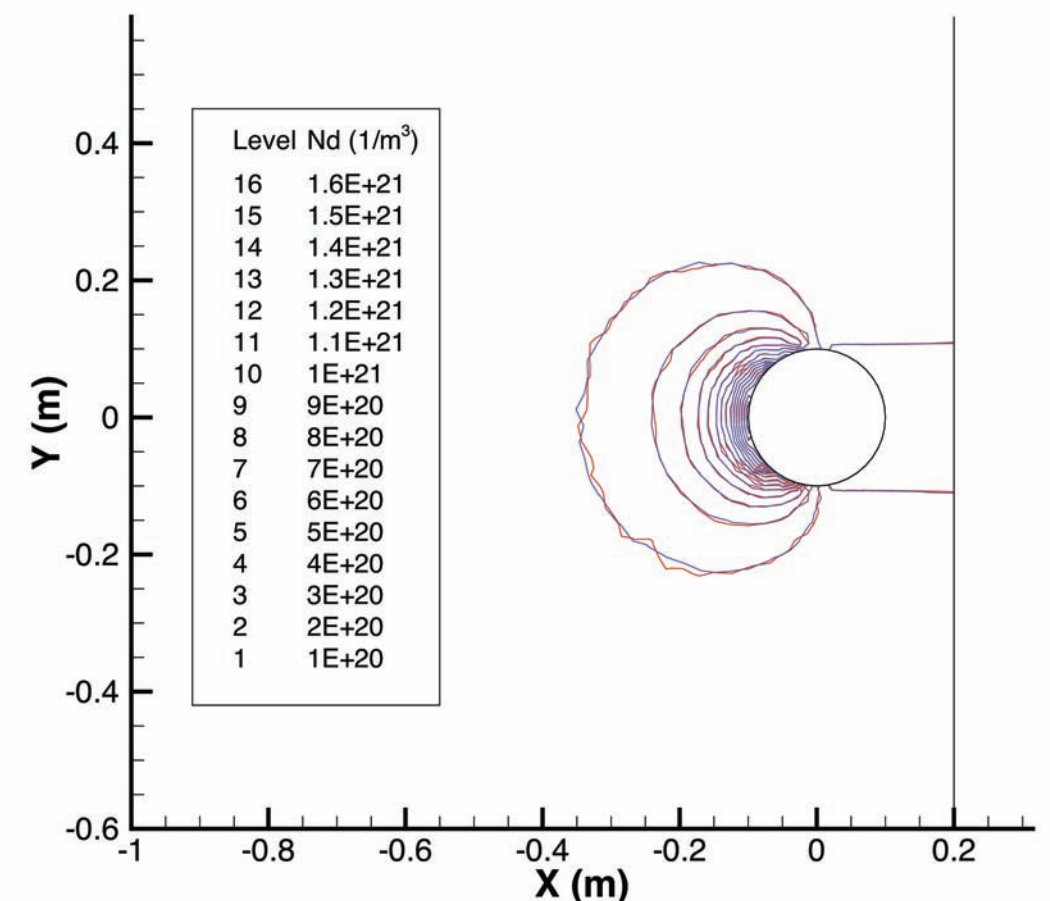
- Since DAC is the standard DSMC algorithm currently used at NASA, it is used as the benchmark.
- Test Case:
 - 0.2 m diameter sphere in Argon.
 - Single processor/single core: MacBook Pro, 2.7 GHz Intel Core i7
 - Collision subroutine off and on.
 - Single unadapted run.

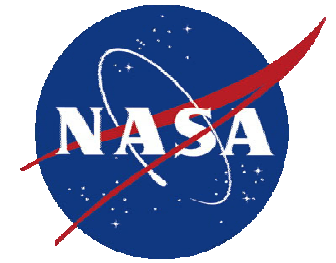


Collisionless Flow

- Percent difference is less at lower time steps
 - Only flow cells that have particles are processed in sampling.
- Time step comparison at 10,000 steps
 - 0.12 (DAC) vs. 0.155 (MAP) -> 29.2%
- Memory requirements for MAP around twice that for DAC (250 Mb vs. 500 Mb)

Step	DAC (s)	MAP (s)	%Diff
1000	116.6	116.8	0.1%
5000	609.8	685.8	12.5%
10000	1215.2	1458.0	20.0%

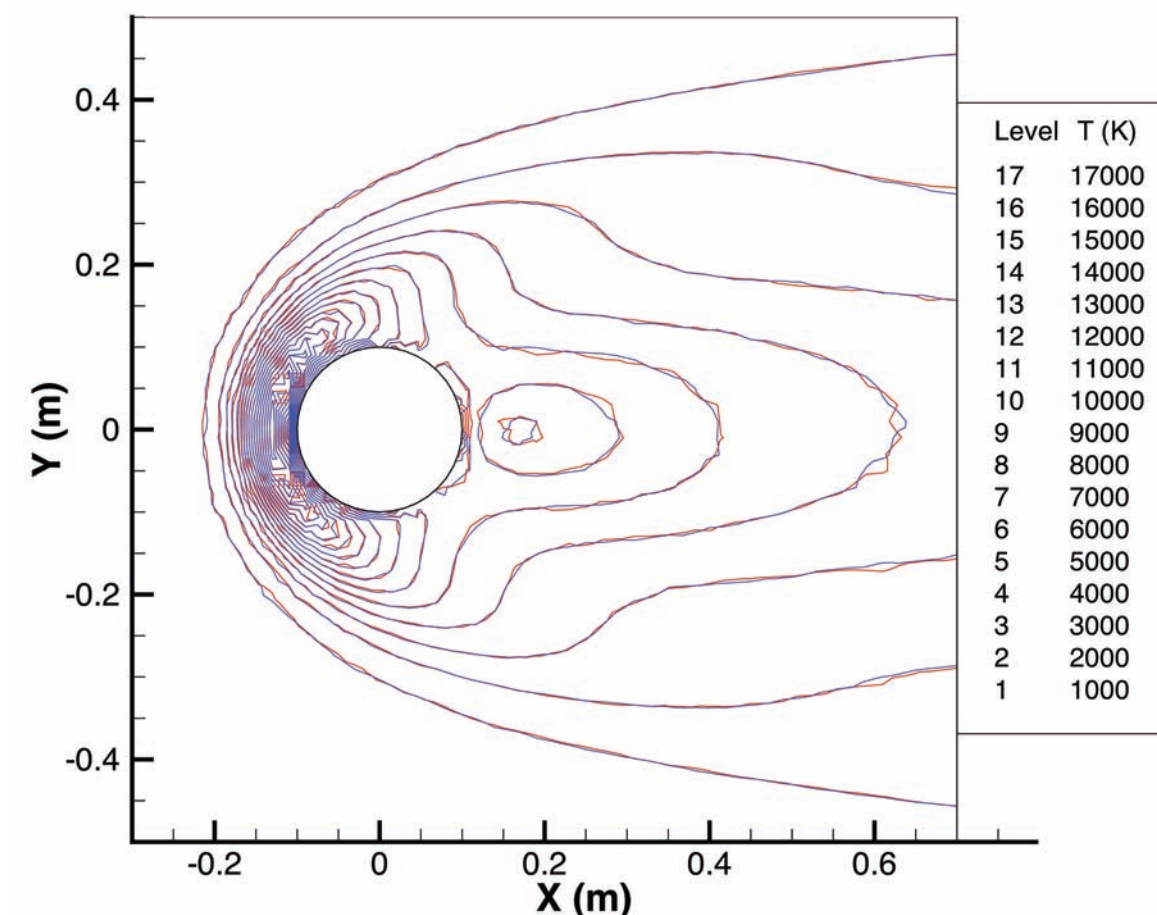


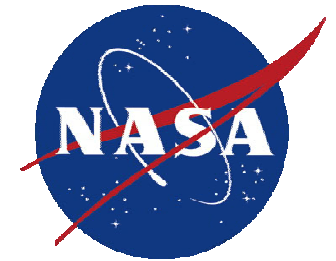


Collisional Flow

Step	DAC (s)	MAP (s)	%Diff
1000	159.6	179.5	12.5%
5000	755.7	1014.0	34.2%
10000	1498.1	2103.3	40.4%

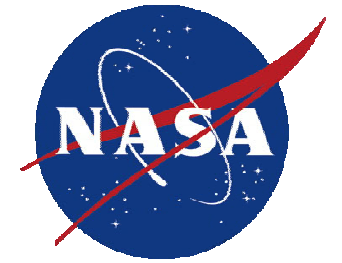
- Time step comparison at 10,000 steps
 - 0.145 (DAC) vs. 0.23 (MAP) -> 58.6%
- Comparison of flow field parameters very good.





Conclusion

- C++ Object-Oriented Programming practices have been used to develop a distinct DSMC algorithm.
 - Use of derived types tailors simulations such that only relevant parameters are allocated.
 - Because of polymorphism, separate particle species can be defined in a single simulation for memory and communication reduction.
 - New classes can be derived for simplified development and research into additional physics modules.
- While this is most probably not the way the “finished” product will look like, the code is flexible enough to evolve.



Future Work

- Look more closely at arrangement of data structures to improve speed.
- Why does MAP require more memory?
- Investigate methods of improving time to solution.
 - Adaptive Mesh Refinement
 - How do adaptive time steps affect time to solution?
- Continue code development
 - Addition of physics
 - Full implementation of MPI

