DOE-FIU SCIENCE & TECHNOLOGY WORKFORCE DEVELOPMENT PROGRAM

STUDENT SUMMER INTERNSHIP TECHNICAL REPORT

June 4, 2012 to August 10, 2012

Development of Pre-processing Software for Lattice Boltzmann Fluid Dynamics Solver

Principal Investigators:

Eric Inclan (DOE Fellow)
Florida International University

Dr. Prashant Jain, Mentor
Oak Ridge National Laboratory

Florida International University Program Director:

Leonel Lagos, PhD, PMP®

Prepared for:

U.S. Department of Energy
Office of Environmental Management
Office of Science and Technology
Under Grant No. DE-EM0000598

ABSTRACT

In order to simulate any form of fluid flow, a computational fluid dynamics program requires the specification of boundary conditions, as well as the discretization of the fluid domain. This paper describes the development of a boundary tagging script, in conjunction with the adaptation of an open-sourced meshing algorithm called CartGen for use with a Lattice Boltzmann computational fluid dynamics program called PRATHAM, developed at Oak Ridge National Laboratory. The completion of the boundary tagging script was an important step toward enabling PRATHAM to simulate fluid flows in and around real-world geometries.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The Thermal Hydraulics and Irradiation Engineering (THIE) group at Oak Ridge National Laboratory (ORNL) has begun development of a computational fluid dynamics (CFD) package called the **P**a**RA**llel **T**hermal **H**ydraulics simulations using **A**dvanced **M**esoscopic method (PRATHAM). As described in [1], PRATHAM is a three-dimensional (3D) Lattice Boltzmann method (LBM) based parallel flow simulation software. The LBM algorithm in PRATHAM requires a uniform, coordinate system-aligned, non-body-fitted structured mesh for its computational domain. Furthermore, the LBM algorithm requires specific information that identifies the location and type of boundary conditions on or within the computational domain. These additional features, which are called pre-processing steps, will enable PRATHAM to simulate a wide range of fluid flows through real-world geometries.

The task for this internship was to develop pre-processing software for PRATHAM. This task was divided into two main categories: boundary tagging and meshing. The meshing algorithm used here was based on an open-source mesh generator, CartGen [3]. This GNU-licensed, open source code provides much of the functionality needed, i.e., it produced uniform Cartesian meshes. However, it needed to be extended for use with PRATHAM. As described in [3], CartGen was modified (i) into CAD geometry to a uniform structured Cartesian mesh, (ii) to provide a mechanism for PRATHAM to import the mesh and identify the fluid/solid domains, and (iii) to provide a mechanism to visually identify and tag the domain boundaries on which to apply different boundary conditions.

The boundary tagging portion, however, had to be developed from scratch and became the primary goal of this internship. In order to meet design and budgetary constraints, the new Boundary Tagging Script (BTS) has to (i) utilize widely available file formats, (ii) operate independently of any single Computer Aided Design (CAD) package, (iii) generate output in a commonly used format, and (iv) utilize a user-friendly Graphical User Interface (GUI) to perform the boundary tagging function. These requirements were set in order to reduce costs of implementing the preprocessor while making the preprocessor more robust (i.e., facilitate its use by other engineers that may use different CAD packages).

The final BTS was encoded with seven boundary types, and gave the user the option to enter custom boundaries if needed. It was also able to distinguish between multiple boundaries of the same type, allowing the user to specify multiple distinct boundaries of the same type, such as in a Y-channel situation.

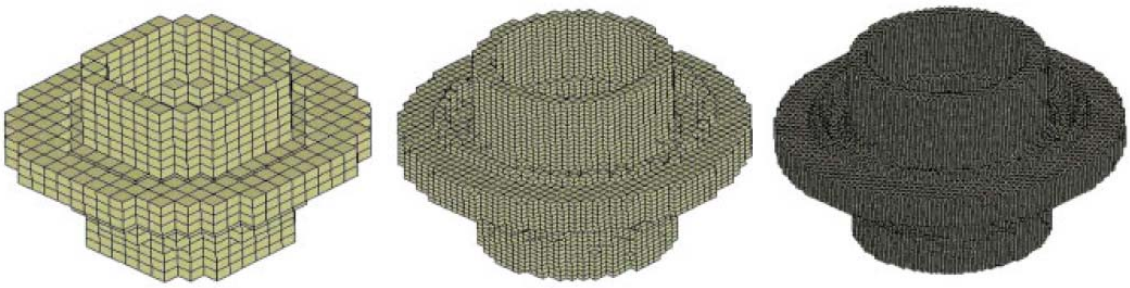
2. EXECUTIVE SUMMARY

This research work has been supported by the DOE-FIU Science & Technology Workforce Development Program, an initiative designed by the US Department of Energy's Office of Environmental Management (DOE-EM) and Florida International University's Applied Research Center (FIU-ARC) to create a "pipeline" of minority engineers and scientists specially trained and mentored to enter DOE-EM's workforce. During the summer of 2012, DOE Fellow intern, Eric Inclan, spent 10 weeks doing a summer internship at Oak Ridge National Laboratory under the supervision and guidance of Dr. Prashant Jain. Mr. Inclan's project was initiated in June 4, 2012 and continued through August 10, 2012, with the objective of developing pre-processing software for the Lattice Boltzmann Code named PRATHAM.

3. RESEARCH DESCRIPTION

3.1 CartGen

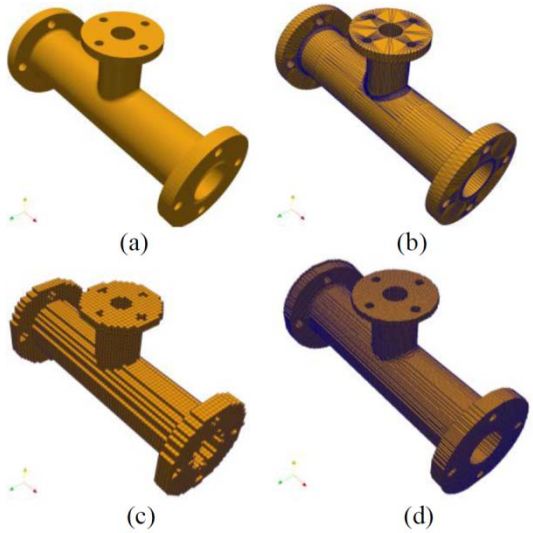
Designed to be robust and easy to use, CartGen creates Cartesian meshes of different types [3]. The type needed for PRATHAM was the uniform Cartesian mesh, which is essentially an evenly spaced 3D grid. CartGen creates this grid by creating a large box (called the "environmental box") around the geometry to be meshed. It then subdivides this box into cubes with dimensions specified by the user. It then colors the resulting cubes (called voxels) as white to signify that the voxel is empty, or black to signify that the voxel lies inside the geometry. The figure below, taken from [3], shows the results of voxelizing a geometry at increasing levels of refinement.



(a) (b) (c)

Fig. 1. Voxelization results using CartGen using (a)25x25x25 cubes (b) 50x50x50 cubes, (c) 100x100x100 cubes

The following figure, taken from [1], shows a geometry as it would appear rendered by a CAD package, and then as it would appear in stereolithography (STL) format, followed by two voxelized versions.



(a) (b) (c) (d)

Fig.2. A flange-mounted manifold: (a) 3D CAD geometry, (b) triangulated STL surface mesh, (c) coarse volume mesh, and (d) fine volume mesh

The lattice would then be created using the corners of these cubes as grid points. However, the code itself was not as robust as the original author claimed. The code made available online would not compile properly unless a specific compiler was used, and the program contained a number of logical errors that caused geometry files to be incorrectly voxelized.

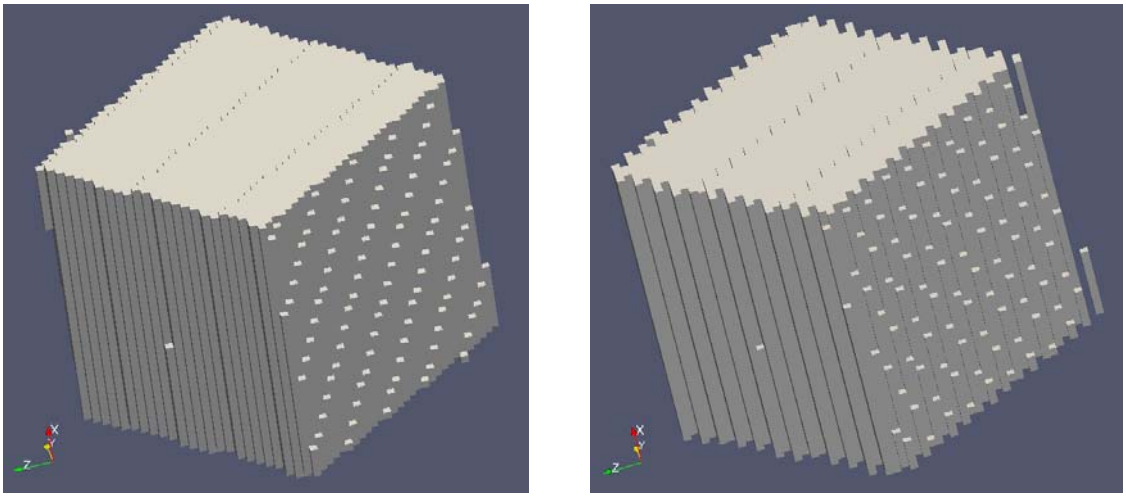


Fig. 3. Incorrectly meshed cubes

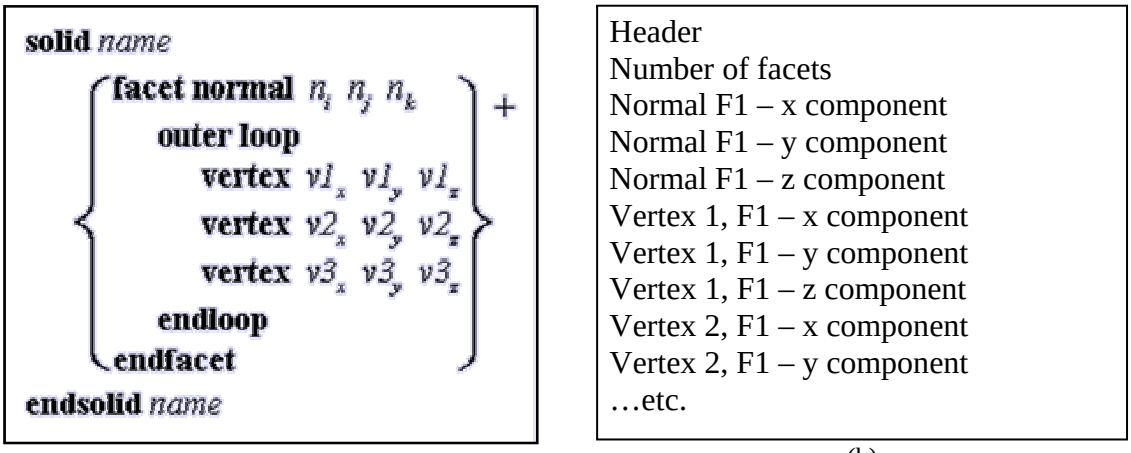
The images shown above in Fig. 3 were produced by making changes to the code and are not exactly representative of the errors found in the software distributed online. The errors actually found, however, did include gaps in sections of geometries as simple as a cube. In the case of a cube, only one face would be completely meshed, while the remainder of the cube would miss diagonal elements. If the geometry contained a hole, sometimes these cavities would be incorrectly represented, such as containing extraneous material or incomplete wall segments.

Despite these errors, however, CartGen successfully meshed most geometries quickly and efficiently, while using a minimal amount of memory. Furthermore, since the code was open-sourced, it could be modified to extend its capabilities and any problems with the algorithm could be analyzed and resolved. CartGen was designed to read in STL files containing geometry information, and write Visual Toolkit (VTK) files containing the mesh, which are two popular and simple formats, making it very practical to work with. Therefore, in spite of its shortcomings, CartGen possessed many powerful advantages and was selected to perform the meshing for the preprocessor.

3.2 The STL file format

An STL file essentially models the surface of a 3D geometry by dividing it into several triangular facets. The STL file lists the (x,y,z) coordinates of the three vertices for each triangular facet and the (x,y,z) components of the vector normal to the facet [1] using the minimum number of triangles required to accurately represent the geometry. An STL file can be saved in either ASCII or Binary format. The Binary format allows for compact

storage while the ASCII format makes the file information user-readable. The figure below shows the layout of ASCII and Binary file formats.



(a) (b)

Fig. 4. STL formats in (a) ASCII [2], and (b) Binary forms

Since some geometries require millions of facets, the binary format was preferred. All results shown in this report used binary formatted STL files. The data in the binary format is organized as follows,

- 80 bytes – header
- 4 bytes – number of facets
- 50 bytes – each facet
  - 12 bytes – normal vector
  - 36 bytes – vertices
  - 2 bytes – unused

Fig. 5. Binary STL Format

The two unused bytes of the STL format were pivotal to the design of the BTS. Since most programs ignore these two bytes they can be overwritten to store boundary tag information without increasing the file size or causing compatibility issues. CartGen could then be modified to read these two bytes and tag the mesh as it is created.

3.3 The VTK file format

Also considered for boundary tagging purposes, CartGen outputs mesh files in VTK format. This file was rejected for boundary tagging use, however, because fewer programs open this format than the STL format. Furthermore, the VTK file contains the final mesh, not the original geometry; therefore, the BTS would have to perform a mathematical mapping from real coordinates to the integer lattice. This additional step would increase the risk of errors in tagging boundaries, which would dramatically increase the errors generated during LBM simulations. Nevertheless, this format provides a compact way of representing the mesh.

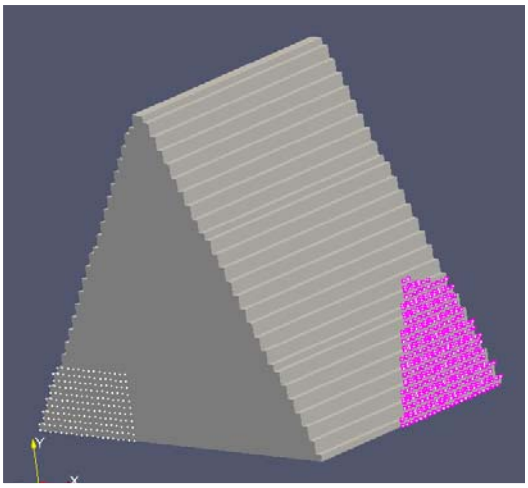
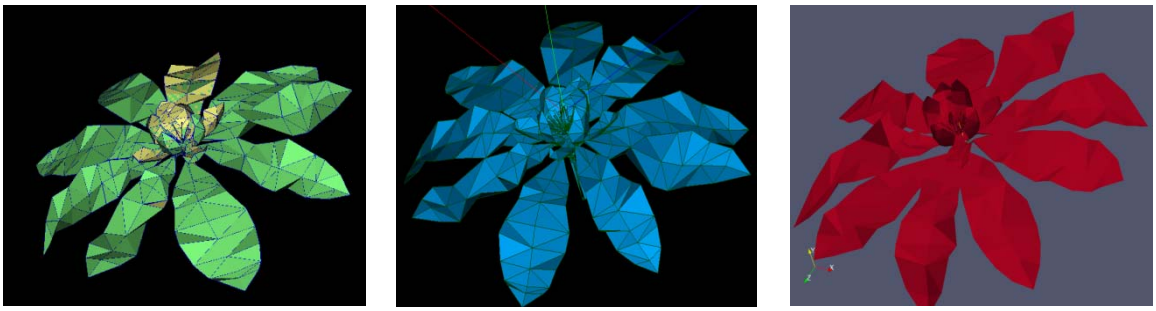


Fig. 6. VTK mesh rendered in Paraview

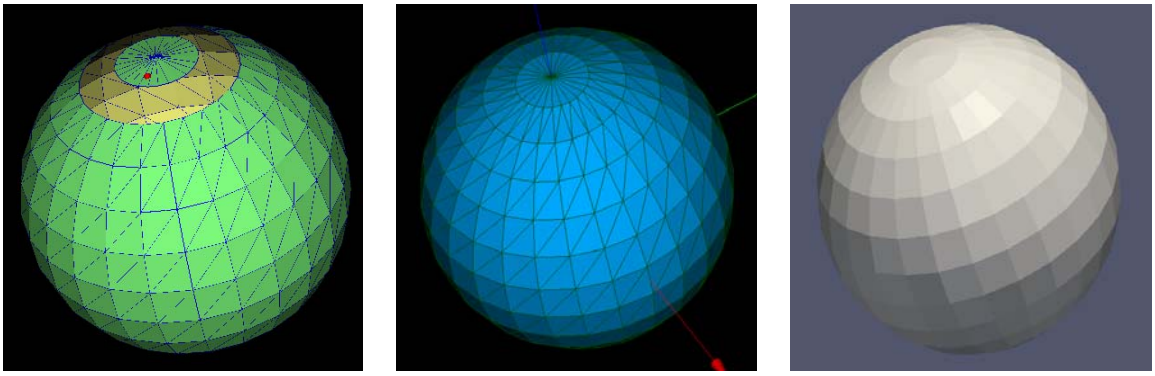
Figure 6 shows two sections of a VTK mesh, with nodes highlighted using Paraview. The mesh is rendered as a stack of solid cubes because the VTK organizes lattice nodes by the voxel with which they are associated. However, as stated before, only the nodes are needed for LBM simulations. Figure 6 also illustrates one of the difficulties of working with voxelized geometries. When using STL files viewed in a CAD package, one can usually select an entire surface together with a single click. With a voxel mesh, however, the user would have to select each individual node. For a simple face, this could potentially be done with a single click+drag command, but in the case of a complicated geometry, the user would have to perform several zooms and rotations in order to select the nodes of interest.

3.3 Other Possibilities

During the course of this internship various methods and file formats were considered for use. These included open-source or freeware programs like Salome, enGrid, Paraview and gmesh.



(a) (b) (c)



(d) (e) (f)

Fig. 7. STL files rendered using various programs (a,d) enGrid, (b,e) Salome, (c,f) Paraview

Some of these programs generated errors when rendering STL files (possibly due to configuration issues), but opened up the possibility of working with file formats such as .silo, .msh and others. In the end, it was determined that these avenues limited the user and increased the overall complexity of the pre-processing method.

4. RESULTS AND ANALYSIS

4.1 The Pre-processing Algorithm

The pre-processing algorithm created for PRATHAM utilizes a group of CAD geometry files in STL format. In order to begin the process, the user must create or import the input geometry file (containing the complete geometry relevant to the simulation) using the CAD package available to the user.

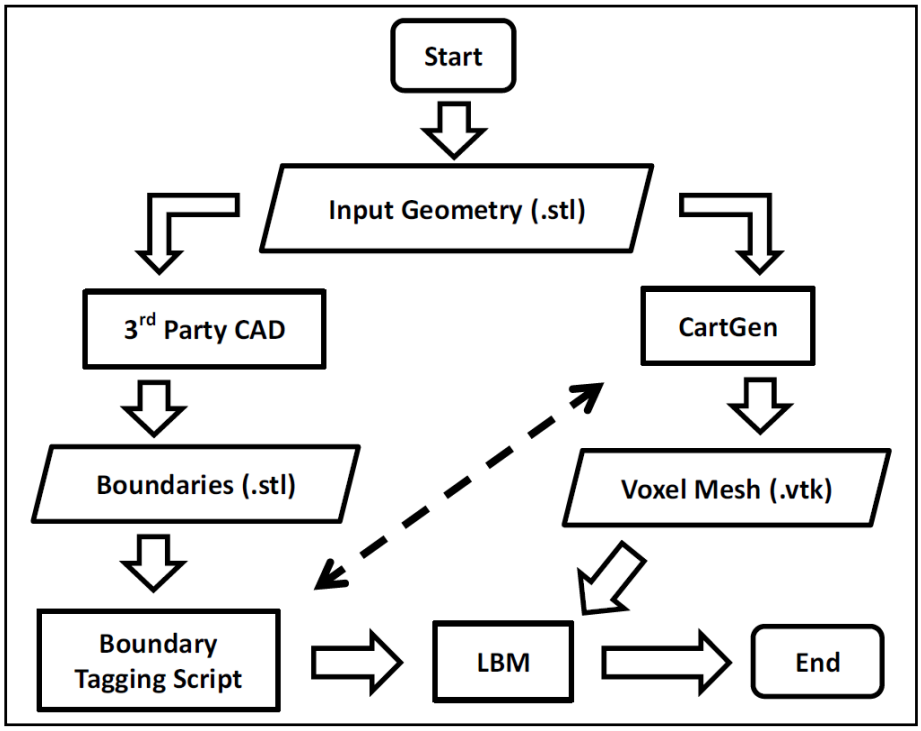


Fig. 8. The complete simulation process flow [1]

With this CAD package, the user must select the surfaces of the geometry to be used for fluid boundary conditions and each surface as a separate geometry file in STL format. After this is done, the user places all of these files (the original input geometry file along with the boundary files) in the same directory as the BTS, and executes the BTS. The BTS will then provide the tagged STL to CartGen, which will produce the voxel mesh, as shown in Figure 8.

4.2 The BTS Algorithm

The BTS algorithm itself is fairly simple. Once all the files are located within the directory of the BTS and the BTS is executed, the user will specify which of those files is the main geometry by entering its name. The BTS will assume all remaining files are boundary files.

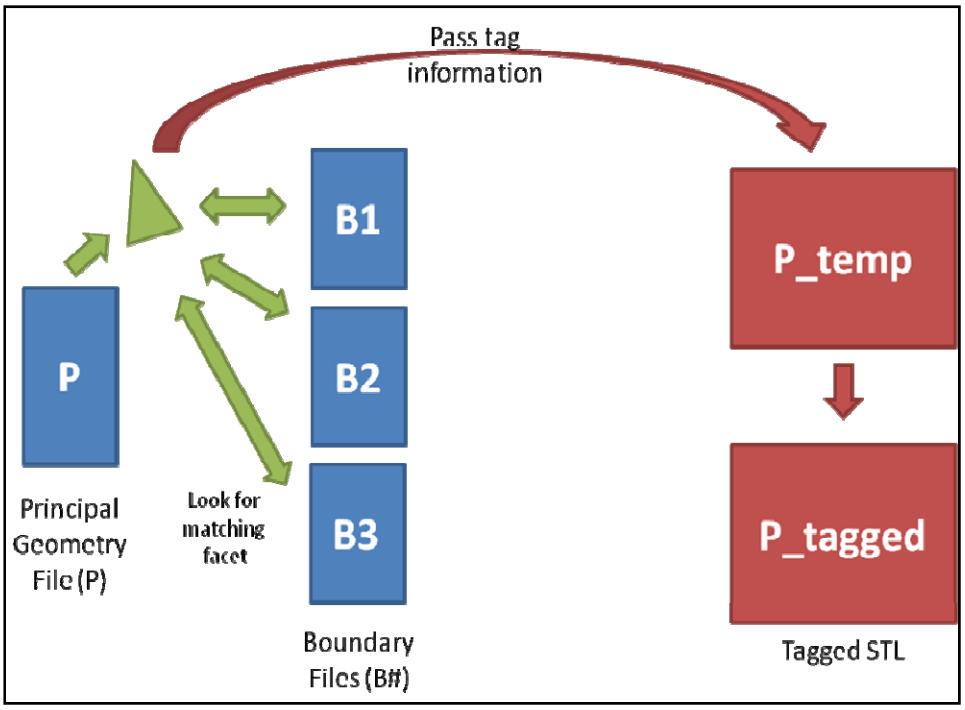


Fig. 9. The BTS Algorithm

The BTS iterates through the main geometry file (called the Principal Geometry in Figure 9) facet by facet. For each facet, the BTS reads through each boundary file and looks for an identical facet within those files. If a matching facet is found, it is tagged accordingly. If no matching facet is found, it is tagged as a wall. Once all facets in the principal geometry have been tagged, the program will produce a tagged geometry file. The user can distinguish this file from the original because it will contain the text "_tagged" at the end of the file name.

4.3 BTS User-Guide

The user should follow these steps when using the compiled BTS (the code compiles on Windows and Ubuntu systems, but may not compile on Mac OS or other Linux distributions):

- Make sure all necessary STL files are in the same directory as the executable (current version of program only).
  - That is, include the file containing the entire geometry (main geometry), and the files containing boundary geometries.
- Upon launching the program:
  - Enter the name of the main geometry file (all other stl files will be assumed to be boundaries).
  - If there are issues opening the files, the user will be prompted to check the files, and the program will exit.
  - If file names cannot be interpreted by the program, user will be prompted to enter the desired tag code (for example, the program can read pressure_inlet.stl, but will not identify press_in.stl as a pressure inlet).
- No other user input is required.

The BTS uses the following codes for boundary tagging,

- Velocity Inlet = a
- Velocity Outlet = b
- Pressure Inlet = c
- Pressure Outlet = d
- Symmetry = e
- Periodic = f
- Wall = 0

In order to add a new boundary tag to the source code, the user should follow these steps.

- The easiest way to add a tag is to include the following line of code at the end of the "Other Conditions" section in Boundary_Type().

  if (copy.find("*tag name*") != std::string::npos) return '*tag code*';

  - Make sure the tag name is written in all caps.
  - The tag code must be a single ASCII character.

The user should note that if multiple distinct boundary types are specified in the file name (e.g. velocity_pressure.stl) the program may simply tag it based on the first identifier found.

4.3 BTS Test Cases

The BTS was given a number of test cases to verify its functionality, two of which are shown below. One of the first was the STL of a sphere.

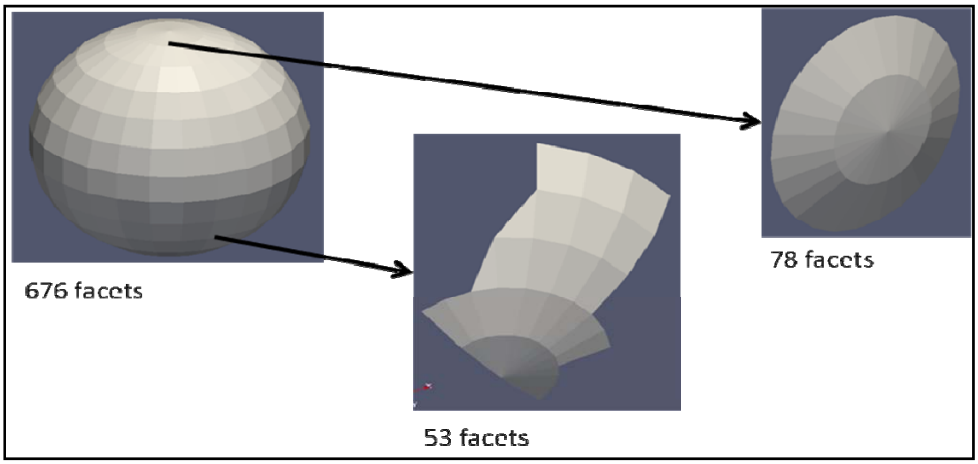


Fig. 10. BTS Test Case including the main 676 facet geometry, and two arbitrarily drawn boundaries

The BTS successfully tagged this complete geometry in less than two seconds using a powerful but fairly common desktop PC.

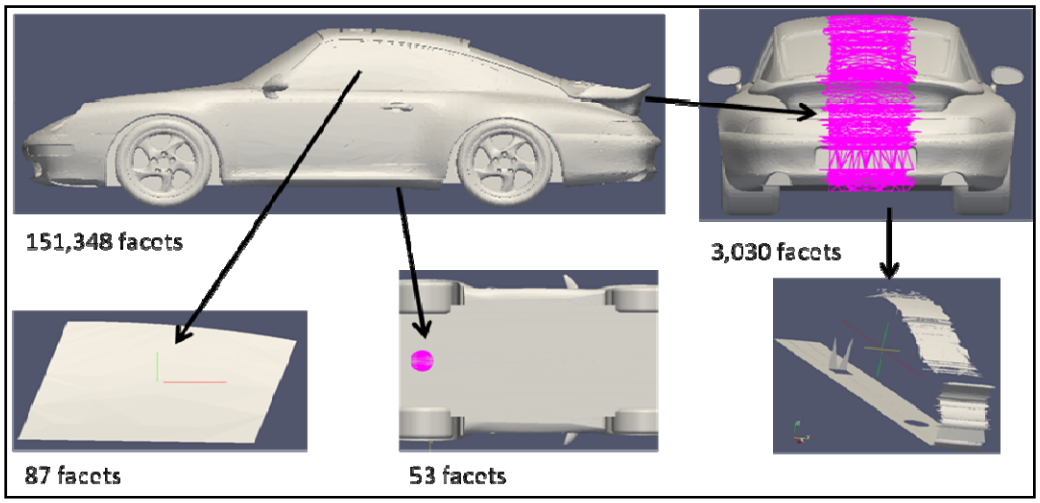


Fig. 11. BTS Test Case including the main 151,348 facet geometry, and two arbitrarily drawn boundaries

The BTS successfully tagged this complete geometry in roughly thirty minutes using the same machine. A larger geometry (~1.5 million main geometry facets and ~3,000 boundary facets) was successfully tagged in roughly four hours.

5. CONCLUSION

Over the course of this internship an efficient, low-cost, robust pre-processing algorithm was identified. Several approaches and open-source CAD packages were evaluated before the algorithm was finalized. This pre-processing algorithm includes using existing CAD software, an open-sourced meshing algorithm, and a boundary tagging script developed by DOE Fellow, Eric Inclan.

The boundary tagging script was tested on a set of STL files ranging from simple, abstract geometries to complex, real-world geometries and was shown to be effective. Furthermore, the script was developed with type-checking and other features to make it robust and safe to use from a computer and network security perspective. In the future, the script can be parallelized, and extended to use hash tables in order to improve its speed. Another improvement would be to include a subroutine within the program that checks for overlapping boundaries.

With this step of the process completed, the remaining steps are to finish debugging CartGen. CartGen has already been extended to read the tagged STLs. Now, CartGen must be extended to write a tagged mesh using this information. Once this is complete, PRATHAM will be able to simulate real-world geometries.

6. REFERENCES

[1] Cantrell, J. N., Inclan, E., Joshi, A. S., Popov, E. L., & Jain, P. K. (2012). Extending a CAD-Based Cartesian Mesh Generator for the Lattice Boltzmann Method. ANS Winter Meeting. San Diego.

[2] Ennex Corporation. (1999). The StL Format: Standard Data Format for Fabbers Reprinted from Section 6.5 of Automated Fabrication by Marshall Burns, Ph.D. Used with permission. Technical source: StereoLithography Interface Specification, 3D Systems, Inc., October 1989:. Retrieved September 1, 2012, from fabbers.com: http://www.ennex.com/~fabbers/StL.asp

[3] Tavakoli, R. (2008). CartGen: Robust, efficient and easy to implement uniform/octree/embedded boundary Cartesian grid generator. Int. J. Numer. Meth. Fluids, 57, 1753–1770.