LBMHexMesh: an OpenFOAM based grid generator for the Lattice Boltzmann Method (LBM)

Conference Paper · January 2013

CITATION
1

READS
207

4 authors:

Andrea Pasquali
FluiDyna GmbH
5 PUBLICATIONS **24** CITATIONS
SEE PROFILE

Martin Schönherr
Technische Universität Braunschweig
8 PUBLICATIONS **70** CITATIONS
SEE PROFILE

Martin Geier
Technische Universität Braunschweig
29 PUBLICATIONS **278** CITATIONS
SEE PROFILE

Manfred Krafczyk
Technische Universität Braunschweig
158 PUBLICATIONS **3,447** CITATIONS
SEE PROFILE

LBMHexMesh: an OpenFOAM® based grid generator for the Lattice Boltzmann Method (LBM)

Andrea Pasquali, Martin Schönherr, Martin Geier, Manfred Krafczyk
Institut für Rechnergestützte Modellierung im Bauingenieurwesen (iRMB)
Technische Universität Braunschweig
Pockelstr. 3, 38106 Braunschweig
Tel: +49 531 391 94531
Email: {pasquali,schoenherr,geier}@irmb.tu-bs.de

Abstract

Although much progress has been made in the last decades, grid generation is still an important topic in Computational Fluid Dynamics (CFD), especially when it comes to adaptive meshes in massively parallel computations. Grid generation for three-dimensional flow simulations in arbitrarily complex boundaries is still very expensive in terms of hardware and manpower resources.

In the last three decades the Lattice Boltzmann Method (LBM) has emerged as an effective modelling and simulation approach to CFD. A typical statement found in LBM-related literature is that grid generation is trivial as the LBM approach is based on Cartesian grids which allows a relatively simple meshing. Yet, this only holds for first order accurate meshes which cannot be considered state-of-the-art in CFD. For LBM meshes including second-order accurate boundary representations, 3D-grid generation is far from trivial. The situation becomes worse if LBM-specific grid refinement approaches have to be included. They require overlapping the coarse and fine grid with specific overlay patterns between the grid patches for both coarse-to-fine and fine-to-coarse interfaces. The boundary surfaces are defined by grid node related sub-grid-distance vectors, which represent the distance between the fluid lattice node and the boundary.

In order to provide a generic and powerful grid generator for LBM, we based our framework on the open source OpenFOAM® grid generator snappyHexMesh. We introduce a new code module to build overlapping interfaces between patches of different resolution and the computation of node-based sub-grid-distance vectors.

Introduction to LBM

The Lattice Boltzmann Method is a numerical method for solving the weakly compressible Navier Stokes equation that is conceptually different from Finite Volume, Finite Element, and Finite Difference Methods. It is motivated by the Boltzmann transport equation and deals with the momentum density function in seven dimensions, i.e. three spatial dimensions, three dimensions of momentum and one dimension of time. The hydrodynamic variables density, velocity and pressure are the moments of the momentum density function and can be computed locally. Using the momentum density function as primary variables allows us to split the evolution of the flow field into two simple steps: a streaming step propagates the distributions according to their respective momentum

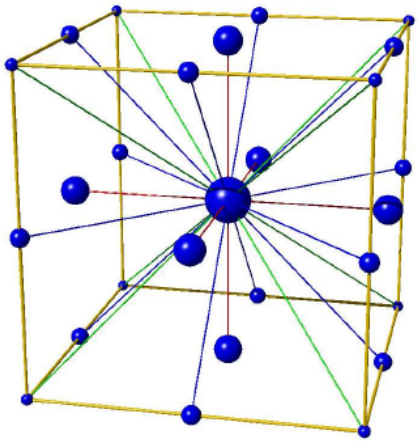

Figure 1: The D3Q27 lattice

direction from node to node on a Cartesian grid. A subsequent collision step rearranges the local distributions on each node. In order for the streaming to be exact the discretization of the momentum space is chosen to match the Cartesian grid. In this study we are using a grid in three spatial dimensions with 27 discrete speeds denoted by D3Q27 lattice (Figure 1). The speeds are chosen such that all 26 neighbouring and the source point are reached in a single time step. Due to this duality of the spacial discretization and the discretization of the momentum space the Lattice Boltzmann Method fullfills all chosen conservation laws, i.e. conservation of mass, momentum and angular momentum, exactly. Interpolation is never required in the standard Lattice Boltzmann Method and finite differences are never explicitly computed.

The very simple computational setup provided by the Lattice Boltzmann Method is not free of problems when it comes to real world engineering applications of Computational Fluid Dynamics. In principle, the Lattice Boltzmann Method is fundamentally limited to be used with Cartesian grids. The aspect ratio of the grid has to be exactly one and the grid cannot be aligned to curved or inclined boundaries. However, methods have been developed to change the resolution of the simulation by coupling several grids together and to place boundaries off-grid. Using these methods Lattice Boltzmann can be as flexible geometrically as any Finite Volume scheme. However, the requirements for the grid generation are quite different from those for other methods: all grids must be Cartesian, nodes close to boundaries require data on the distance to the boundary to facilitate the application of off-grid boundary conditions and grids of different resolution must overlap to facilitate the interpolation between the grids.

In this contribution we present the first general purpose grid generator that complies with the special requirements of the Lattice Boltzmann Method.

Lattice Boltzmann data structure: EsoTwist

The simplicity of the Lattice Boltzmann stream and collide algorithm has led to the advent of highly optimized data structures with a special focus on the execution on massively parallel hardware such like GPGPUs (General Purpose Graphics Processing Units). The grid generator should comply with the data structure to avoid new bottle necks. The data structure used in our Lattice Boltzmann application is the so-called

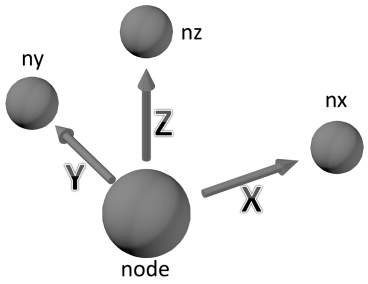

Figure 2: The X-, Y- and Z-neighbours of the respective node

sparse EsoTwist method. It consists of a sparse matrix containing the data associated with the nodes (its 27 distributions). Nodes will be processed concurrently on parallel hardware. Early implementations of the Lattice Boltzmann Method used two data arrays for streaming, one for reading and one for writing. Streaming and collision were separated steps. The EsoTwist data structure allows to combine streaming and collision into a single step and requires only one set of variables for both reading and writing. This is accomplished by using one array for each lattice velocity. To localize valid neighbours, it is necessary to store the X, Y and Z neighbour indices in separate arrays (Figure 2).

The notation of the distributions are east (e) and west (w) for the positive and negative X-Directions, north (n) and south (s) for the positive and negative Y-Directions and top (t) and bottom (b) for the positive and negative Z-Directions. The diagonal distributions are combinations of these six. Distributions moving in positive X, Y and Z directions are stored at the index of the respective node while a distribution moving in negative X direction is stored at the index of the neighbouring node in positive X direction (Figure 3). The same holds for distributions moving in negative Y and Z directions. Distributions having two or three negative components are stored in the respective neighbour in opposite diagonal direction. For example, the distribution "bsw" is stored at the node in top-north-east direction (Figure 3).

Streaming is accomplished by swapping pairs of distributions with antiparallel directions after the collision steps. All distributions are written back to the place were the distribution moving in opposite direction has been read. After this has been done for all

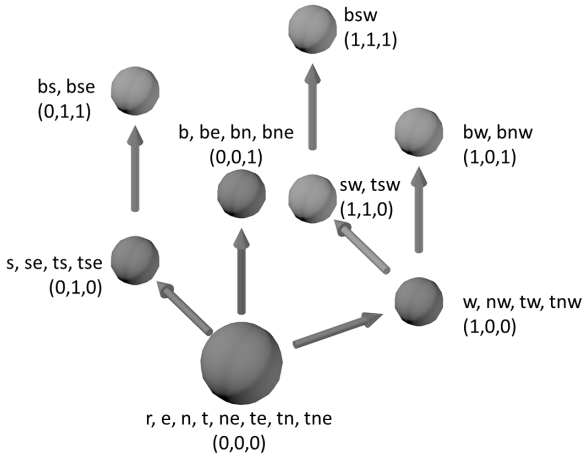

Figure 3: Location of the distributions

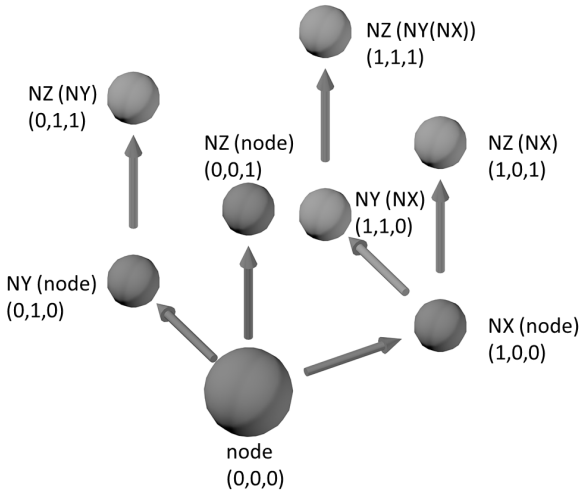

Figure 4: Functionality of the pointer chasing

nodes, pointers to the arrays for distributions moving in positive and negative directions respectively are exchanged such that the original ordering of the distributions has been restored. Since EsoTwist accesses only neighbours in positive directions only links to seven neighbours have to be provided. This number is further reduced to three by the application of pointer chasing (Figure 4).

That means that the neighbour in the diagonal positive X and positive Y direction is accessed by taking the neighbour in Y direction of the neighbour in X direction. These details are essential for the grid generation since each node has to have all the required neighbour information available. Since some distributions associated with a node are always stored at the neighbours we have to make sure that these neighbours exist in the data structure even if they are not part of the fluid domain.

Lattice Boltzmann grid generator: LBMHexMesh

The LBMHexMesh grid generator is based on the open source OpenFOAM® grid generator snappyHexMesh. The snappyHexMesh utility generates 3-dimensional meshes containing hexahedra and split-hexahedra automatically, starting from triangulated surface geometries in Stereolithography (STL) format and a background mesh. The background mesh must consist purely of hexahedra and fill the entire region within the external boundary. The mesh approximately conforms to the surface by iteratively refining the background mesh and morphing the resulting split-hexahedra mesh to the surface. An optional phase will shrink back the resulting mesh and insert cell layers between the mesh and the surface [1]. The snappyHexMesh stages could be summarized as:

- castellated mesh: cell refining and cell removal
- snapping to surface: moving cell vertex points onto the surface geometry
- layer addition: introducing additional layers of hexahedral cells aligned to the boundary surface

Since LBM needs a Cartesian grid, the first stage of snappyHexMesh is suitable for our purpose.

In order to have perfectly cubic cells as required for a D3Q27 lattice, a background mesh of hexahedral cells with cell aspect ratio exactly equal to one must be provided. In this way the castellated mesh stage will produce a grid that is still composed of only hexahedral cubic cells. The refinement is done according to the user specifications. A section of a 3-dimensional castellated mesh for a sphere is shown in Figure 5a. The bigger cells represent the background mesh composed of cubic cells (coarse grid, refinement level 0). The refinement is done using a refinement level 1 for all the cells surrounding the sphere. The refinement procedure splits the selected cells in order to refine the grid in all directions X, Y and Z; obtaining the fine grid still composed of cubic cells (smaller cells). This procedure ends with the removal of the cells that are outside of the fluid domain, i.e. all the cells inside the sphere.

This mesh, as obtained from the castellated mesh stage, is the starting point to generate the grid for LBM. We have introduced a new module that, after the castellated mesh stage, processes the cells and the nodes of the grid and builds a new mesh according to the LBM grid requirements. The new utility is named LBMHexMesh. A LBM grid presents mainly these features:

- overlapping cells between the coarse and the fine grid
- neighbour information for each fluid-node in each positive direction

Contrary to a Finite Volume mesh, a LBM grid requires an overlap between the coarse and fine grid nodes. This is needed because the lattice nodes in the interface regions are filled with distributions obtained by second order compact interpolation between the four enclosing lattice nodes. The compact interpolation method utilizes the discrete moments of the distribution functions of the four enclosing lattice nodes to compute distributions

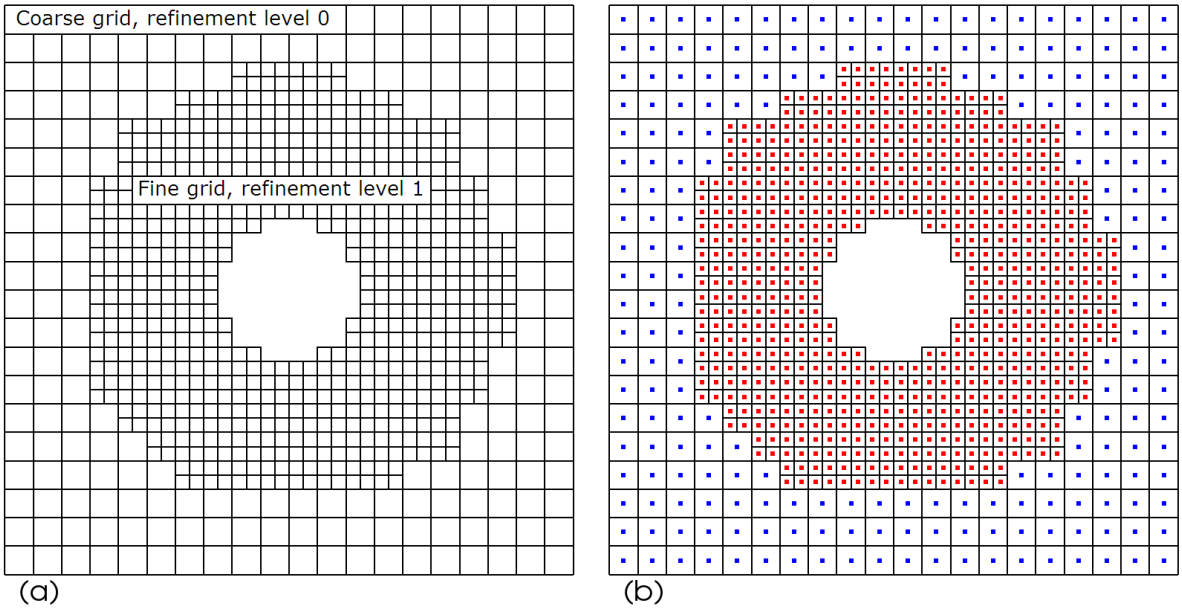

Figure 5: Section of a 3-dimensional castellated mesh for a sphere (a) and its cell centres selection used for overlapping the coarse and the fine grid (b)

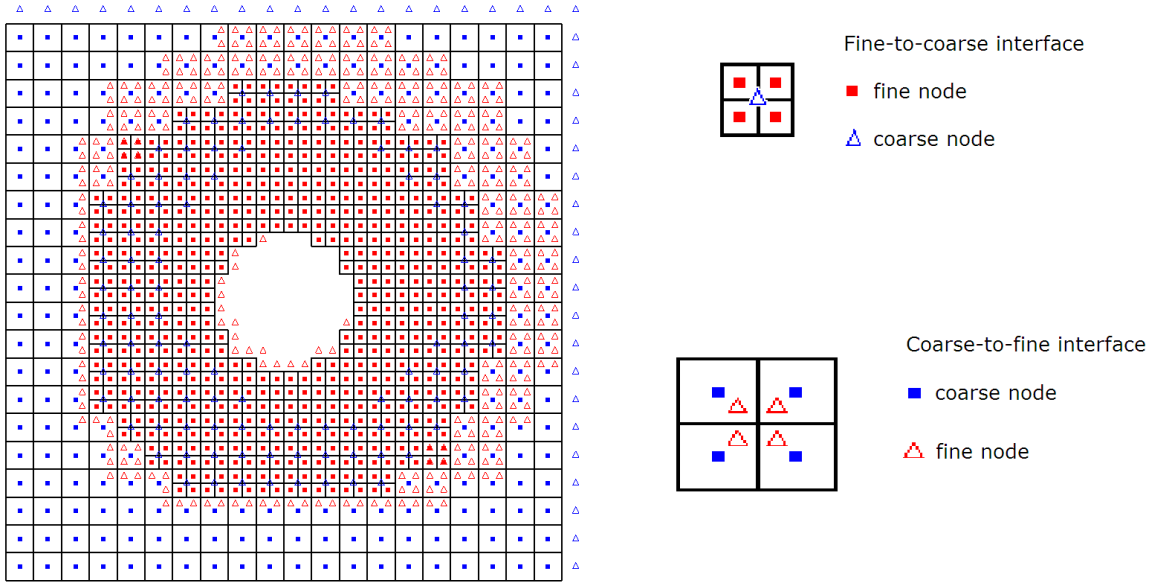

Figure 6: Building of the interface nodes (triangles) for interpolation between the coarse and the fine grid

at the internal nodes [2]. Therefore, due to the LBM code data structure used in our LBM code, each node in the fluid domain must have a target-node where to store the information. This target-node has to exist in each positive direction (+X, +Y and +Z) and also in the diagonal direction (+X+Y+Z). On the contrary, a node that is in the solid domain does not need neighbours in the positive directions.

In order to obtain an overlapping grid from the castellated mesh, we take the cell center of each Finite Volume cell (Figure 5b). The nodes that now represent the coarse grid are the cell centres of the coarse Finite Volume mesh. They are no longer aligned with the nodes that now represent the fine grid, that are the cell centres of the fine Finite Volume mesh. Hence, the two grids coarse and fine are now overlapping and separated.

To exchange correctly the information between the coarse and the fine grid, an interface patch between the fluid nodes of the separated grids is needed. This interface has to be defined for both sides, coarse-to-fine and fine-to-coarse. Figure 6 shows how this interface is built. The nodes that are added to create the interface are represented with triangles. For the coarse-to-fine interface, the eight cell nodes of the fine grid are evaluated by the interpolation with the outer eight cell nodes of the coarse grid. For the fine-to-coarse interface, only one coarse node is evaluated by the interpolation with the outer eight cell nodes of the fine grid. Each set of coarse-to-fine and fine-to-coarse cells are identified by the lower corner point of each cell. When one of the nodes required for the interpolation is missing, for example when one of the nodes is not in the fluid but in the solid, another information is needed. This information is an offset vector that defines where to look for a complete sequence of fluid nodes that have a proper interface. In Figure 6, the nodes that are the last in the positive directions +X, +Y and +Z and in the diagonal direction +Y+X+Z are always solid; the others are fluid. The fluid nodes must have a neighbour in each positive direction.

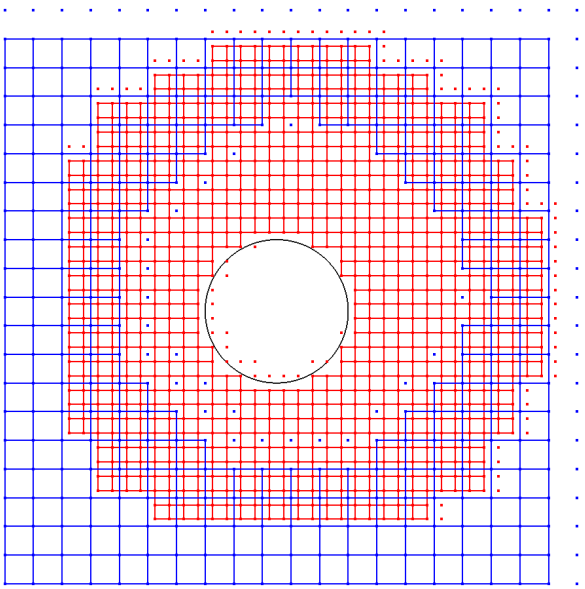

Figure 7: Section of a 3-dimensional LBM grid for a sphere generated with LBMHexMesh. The circle is the section of the sphere (STL format). The points connected by lines are fluid, the points not connected by lines are solid

To complete the description of the mesh, information on the boundaries has to be provided. Because of the Cartesian grid, the representation of the surface is not sufficient to simulate flows around complex curved geometries. For this reason, in order to have a better approximation of the surface boundary, we have to provide a number that tells us how far we are from the surface. This distance is a fraction of the grid distance ΔX_l which is the distance between the last fluid node close to the surface and the surface itself. It is stored in a vector with 27 components, because of the D3Q27 lattice used, and it is evaluated using a ray-triangle intersection algorithm, since we are using triangulated geometries (STL format).

Finally, the information needed for a LBM grid can be summarized as follow:

- coordinates (in lattice units)
- fluid and solid nodes
- neighbours
- interface patch for both coarse-to-fine and fine-to-coarse sides
- offset vectors for the interface patch
- boundaries (sub-grid-distance)

This information is generated by the new module that we have introduced. The grid is written separately for each refinement level from the coarsest to the finest refinement level. The format we use to write the files is particularly suitable to run fast on GPGPUs architectures.

In Figure 7 we show a slice cut of the 3-dimensional LBM grid for a sphere created with LBMHexMesh starting from the castellated mesh of Figure 5a. In Figure 8 we show

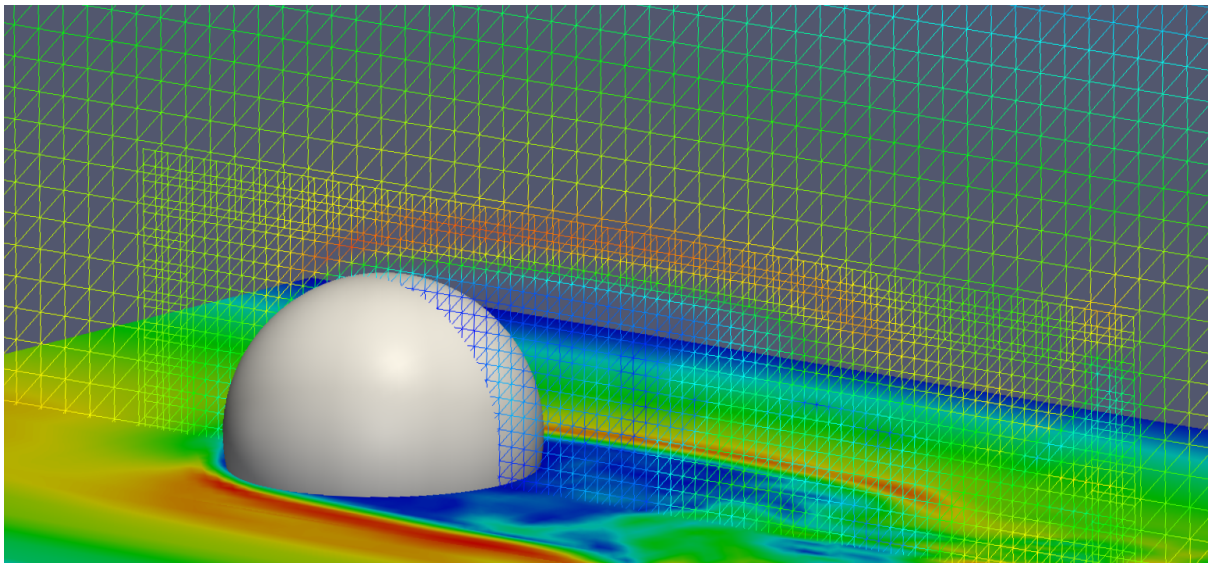

Figure 8: Velocity contour map for flow around a sphere inside a channel

the velocity contour map for flow around a sphere inside a channel. The grid is created with LBMHexMesh and presents two refinement levels. The finer grid is done using a refinement box surrounding the sphere. A velocity parabolic profile is used as boundary condition at the inlet and pressure uniformly equal to zero is set as boundary at the outlet. All the others boundaries are treated as no-slip wall.

Conclusion and outlook

In this work we described a new implementation within OpenFOAM® to generate a grid for non-uniform Lattice Boltzmann Methods. LBMHexMesh uses the powerful and fast refinement stage of snappyHexMesh, with the possibility to model very complex geometries with arbitrary refinement methods, such as refinement boxes and surface distance refinement. The LBM grid requirements are satisfied by the new utility, providing the proper grid and information for LBM applications.

A long term goal of our work could be to lay a basis for a complete LBM branch within OpenFOAM® including a state-of-the-art LBM kernel.

References

[1] OpenFOAM User Guide, Version 2.2.0, 22nd February 2013

[2] M. Schönherr, et al., Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs, Computers and Mathematics with Applications (2011)