From a Product Model to Visualization: Simulation of Indoor Flows with Lattice-Boltzmann Methods

Siegfried Kühner, Bernd Crouse & Ernst Rank*
Lehrstuhl für Bauinformatik, Technische Universität München, Arcisstr. 21, 80290 München, Germany

&

Jonas Tölke & Manfred Krafczyk
Inst. für Computeranwendungen im Bauingenieurwesen, Technische Universität Braunschweig, Pockelstr. 3, 38106 Braunschweig, Germany

Abstract: After a short discussion of different types of models being involved in the process of numerical simulation in civil engineering, we focus on the analysis of indoor air flow through complex geometries. All models are derived from a product data model based on Industry Foundation Classes. Concepts of the Lattice-Boltzmann method are described, being used as the numerical kernel of our simulation system. We take advantage of spacetrees as a central data structure for all geometry related objects. Finally, we describe some advanced postprocessing and visualization techniques allowing to efficiently analyze huge amounts of simulation data.

1 INTRODUCTION

The building industry is characterized by the cooperation of a large number of partners having very different views of their product part. These range from the architects view on design and functionality of a building over the structural engineers view on loads and stability of the construction to the view of the HVAC-engineer on problems of ventilation, air conditioning, calculation of heat loss, and cooling demands in the context of energy-efficient building design. Despite this high differentiation of disciplines involved in the design and construction process, many fundamental decisions are made at a very early stage of the design, often without a profound knowledge of the consequences of these decisions in a later stage of the construction process. On the other hand, any wrong decision being detected only at a late stage of design may cause enormous extra cost and time delay during the construction process. It is estimated that up to 20% of the total cost of buildings result from late corrections of wrong decisions in early design stages. Most experts in civil and building industries agree that information, communication, and simulation technologies could play a more prominent role in making the design and construction process more transparent and reveal the consequences of decisions at an early stage. Yet, a notorious difficulty in the cooperation of different disciplines is the lack of common models for a building. It should be emphasized that this is not mainly a question of common data formats for an electronic exchange but much more a question of the different model semantics being determined by the tasks as well as by the methods available to perform these tasks. To give an example, an architect's model to set up and visualize the geometry of a building is strongly different from a finite element model for thermal or structural analysis and this is again different from a model for air flow simulation of the same building. It is our opinion that it is neither possible nor
desirable to identify a “hyper model” incorporating all aspects of building design. To better support cooperation, it is yet necessary to identify common submodels being usable by as many partners as possible, to provide means to map models onto each other, and, wherever possible, to derive models from more basic ones being a common ground for different tasks. Although focusing on a model sequence for the simulation of indoor air flow in this article, we have been applying this general concept to structural engineering and fluid-structure interaction alike (Rank et al., 2001; Halfmann, 2002).

The simulation-based prediction of internal flows in a building has only recently started to find a broader place in practical civil engineering. The most important reasons for this situation are:

- mapping of CAD data to computational grids or meshes in three dimensions is usually very time-consuming and tedious;
- the prediction of especially transient features of turbulent flow is difficult and CPU-intensive;
- efficient postprocessing of corresponding three-dimensional simulation data, is typically decoupled from the CAD environment and therefore only loosely integrated into the design process.

This article will address all three items and suggest a sequence of models and computational methods being strongly connected to spacetrees.

In Section 2, a model chain starting from a product data model and showing how the geometric part of this building description can be mapped completely automatically to a computational grid for the indoor airflow simulation will be discussed. The numerical technique being used in the computational kernel, the Lattice-Boltzmann (LB) method will be described in Section 3. Some advantages of this method for the simulation of turbulent flows in and around buildings, namely a straightforward integration of turbulence models and its applicability for arbitrarily complex geometries will be highlighted. The problem of postprocessing huge and complex three-dimensional data sets is addressed in Section 4. We will again take advantage of octrees and demonstrate an example where air flow simulation results are efficiently visualized within a model inheriting its geometric properties from the original product data model.

2 MODELS IN NUMERICAL SIMULATIONS

2.1 Chain of models

Numerical simulation always consists of several steps being associated to a chain of interdependent models, all being related to topology and geometry of a structure. A grid has to be defined or a finite element mesh must be setup in a first step. Properties like material parameters and boundary conditions are often directly assigned to these models before the numerical analysis is started. Another geometrical model is also the basis for the last step, that is, the visualization of the results (Figure 1). Considering yet different domains of simulation in civil engineering, like structural analysis, computational fluid dynamics, or even fluid-structure interaction, one immediately recognizes that often completely different types of geometric descriptions are used, depending on the chosen numerical technique to solve the problem. The numerical model for a fluid-flow simulation in a building, for example, a three-dimensional Cartesian grid for a Navier-Stokes solver strongly differs from a numerical model for a simulation of heat transfer or structural analysis of the surrounding walls, which may be a mesh of tetrahedral or hexahedral finite elements.

If the building geometry together with its related simulation attributes is defined using the numerical model itself, cooperation between different disciplines is significantly obstructed. In this case, any modification of geometric properties requires a redefinition of all corresponding models. To better support cooperation and

![Diagram](image)

Fig. 1. Classical simulation models and data flow.
to enhance consistent, multidisciplinary analyses, the geometrical and topological description of a building should instead be given independently of the special type of numerical simulation.

Therefore, we suggest a framework identifying four levels of models with different, yet interdependent tasks: the geometrical level, the physical level, the numerical level, and the presentation/analysis level. Figure 2 shows two simulation chains in this scheme together with their interconnections, a fluid-flow chain and a structural analysis chain. The geometrical level starts from a building product model based on the Industry Foundation Classes (IFC). Before a numerical model is derived, all necessary data for the flow simulation are defined on the physical level in a "numerical wind tunnel." This step includes imbedding the building model into a geometrical model with additional definitions of the building's environment and appropriate flow boundary conditions, material data, and so on. These definitions, still being independent of any numerical technique to solve the flow problem, are implemented in a tool based on AutoCAD Architectural Desktop Release 2.0™, which is also the modeling framework of the second chain depicted in Figure 2, the structural simulation chain. In this latter part, material properties, loads, and boundary conditions are set up for the structure, for example, for an elasticity analysis or for a fluid-structure interaction problem.

Note, that we did not focus on an efficient management of the complex data being generated within the physical level. A recent publication by O'Bara et al. (2002) offers a very flexible system, also being applicable for our framework, especially to handle the linkage to the geometrical level.

Only after the complete definition of the physical problem, numerical models are derived accounting for the possibly different requirements of the individual numerical analysis programs. This strict separation of physical definitions from numerical techniques facilitates the exchange of numerical tools; the derivation of both models from the (common) geometrical model simplifies cooperation, as a modification, for example, of the geometry is transferred to both simulation chains.

As on the next (numerical) level, different geometrical models have to be used in general, an exchange of information is only possible by mapping data between two different types of meshes in our framework. This is necessary, for example, in fluid-structure interaction, where our concepts are discussed by Halfmann (Halfmann et al., 2000; Halfmann, 2002). Because of the fact, that our concept is based on a partitioned solution approach, it is obvious, that a permanent data flow has to be performed between the two numerical methods on the numerical level. We will focus in the following on a transition from the physical level to the numerical level of the fluid-flow simulation chain only, and derive for a LB method (Krafczyk, 2001) a grid from the common B-rep model of the physical level, inheriting all geometry-oriented attributes.
2.2 Transition between the levels of the simulation: Octree data structures

A brief summary of spacetree data structures and extensions for an application in the fluid simulation process is given. For a detailed description of spacetrees and various applications to simulation and visualization, we refer to Frank (2000) and the references therein and to Breitling et al. (1999).

Spacetrees subdivide the space recursively into smaller parts as shown in Figure 3. It is a tree data structure based on cells with one parent and $2^D$ children cells, where $D$ is the number of dimensions of space.

Each cell represents a cube in physical space. The 2D representation is known as a quadtree, while its 3D version is referred to as an octree. The subdivision used here is uniform, so that the ratio of the edge length of a child cell to its corresponding parent cell is always given by 1:2. The root cell is the origin of the tree and has no parent. Its tree level is 0 by definition. A top-down traversal of the hierarchy of the tree will stop at the so-called leaf cells, which have no child cells. The set of leaf cells typically defines the cells of the computational grid. Each cell within the tree can be considered as a root-cell of a subtree, being a spacetree itself. This property allows the use of recursive algorithms.

A computational grid is derived from the B-rep model of the physical level in three major steps using the octree data structure:

(1) A facet model of the surface of the B-rep model is generated first, inheriting all physical data from it. The resulting model does not depend on a specific CAD system or product model any longer. Furthermore, the following geometric operations for facets are significantly less time consuming than corresponding operations on generally curved surfaces.

(2) Boundary “voxels,” leaf cells of a predefined level of the spacetree having nonempty intersection with the surface facets, are generated. The attributes “interior,” “exterior,” or “boundary” become assigned to each leaf cell.

(3) The final computational grid for the LB method, typically mapped onto a 1D array, is derived from the nonuniform octree and the coarser cells are uniformly filled.

2.3 Modifications of the octree data structure

To improve the computational efficiency and capability of the spacetrees, extending more attributes to the cells and especially to the leaves might become necessary. The search for adjacent cells forces each spacetree-based algorithm, using only a directional linkage (top-down) between the cells, to a possibly very time-consuming tree traversal. Therefore, in addition to the hierarchical linkage of cells, an additional flat linkage (links between leaf cells) is proposed. Once the adjacent cell has been determined by, for example, tree traversal and a pointer is locally stored, all following algorithms are able to access the neighbor directly and very fast (Figure 4). This leads to a graph data structure having properties of the basic hierarchical structure as well as those of a surface mesh,

Fig. 3. Octree data structure—Hierarchical spatial subdivision.

Fig. 4. Hierarchical and flat linkage.
which can, for example, be used to efficiently propagate information.

In other words, this data structure can be considered being a cache version of a standard octree data structure to speed up future query of neighboring cells.

During the generation of the octree, leaf cells are first marked by a flag, if they intersect a facet, that is, if they are “boundary” cells. So far, the position of the leaves with respect to the solid boundary “interior” or “exterior” is not yet determined. An efficient solution minimizing the number of intersection operations and using the properties of a surface mesh, is the following flooding algorithm.

Figure 5 illustrates the mechanism. Once the position of only one cell in the interior of a closed domain has been chosen, this cell acts as a seeding point for the flooding algorithm. In each propagation step, the information is shared with the neighboring cells. The break off criterion is the attribute flag “boundary.”

Performing this algorithm, the interior and exterior of a certain geometric scene will be flooded by the corresponding information. The efficiency of a flat cell linkage is advantageous here, and for many other algorithms requiring frequent neighborhood information, like algorithms for local grid refinement and coarsening, or for data reduction (see Section 4.2).

The numerical simulation being described in the following section is currently performed using a uniform Cartesian grid. However, we intend to use octrees for the simulation kernel as well, despite the following drawbacks:

- high memory usage;
- slow access of discrete data being stored at the nodes of the octree;
- demanding implementation of a vectorized simulation code.

On the other hand, we will benefit from the two major advantages of a discretization based on octree data structures:

- The numerical kernel will allow an adaptive modification of the grid during run-time to numerically resolve physical local effects (see Section 6). Thus, we will be able to more efficiently simulate problem types, despite the slow access of the nodes of the octree mentioned above.

- Straightforward implementation of a parallelized simulation code.

3 THE LB METHOD

In the last decade, LB models have been developed and used as a complementary approach (to classical techniques directly discretizing the Navier–Stokes equations) for simulating a variety of fluid-flow problems (Succi, 2001).

This section describes the basic ingredients of the method required to motivate its special suitability for the simulations described in this work. For detailed information the interested reader is referred to Succi (2001), Krafczyk (2001), and the references therein.

In addition to the numerical solution of the Navier–Stokes equations, many fluid-flow problems can be simulated by solving the so-called LB equation:

\[
f_i(t + \Delta t, \vec{x} + \vec{e}_i \Delta t) - f_i(t, \vec{x}) = \frac{\partial \Omega_i(\vec{x}, t)}{\partial t}
\]

which can be interpreted as a discrete evolution equation for particle density distributions originating from a lattice–gas system or a first-order discretization of a Boltzmann equation with a discretized microscopic velocity phase space, where the collision operator (right side) is simplified by a BGK approximation (Bhatnagar et al., 1954).

\[
\Omega_i(\vec{x}, t) = \frac{\Delta t}{\tau} (f_i(t, \vec{x}) - f_i^{(0)}(t, \vec{x}))
\]

In either case, Equation (1) describes the spatial temporal evolution of a particle density distribution function moving with discrete velocities \(\vec{e}_i\), where the set of \(\{\vec{e}_i\}\) spans a unit cell of a cartesian grid. Typically, \(\Delta t\) and \(\Delta \vec{x} = \Delta t |\vec{e}_i|\) are set to unity, which is adopted in the following.

It can be shown, that the first moments of the particle distributions \(f_i\) solving Equation (1) defined as

\[
\rho = \sum_i f_i
\]

\[
\vec{u} = \frac{1}{\rho} \sum_i \vec{e}_i f_i
\]
under certain conditions and for suitable equilibrium functions \( f_i^{(0)} \) are also solutions of the continuity equation

\[
\frac{\partial \rho}{\partial t} + \nabla (\rho \vec{u}) = 0
\]

(5)

and the Navier–Stokes equations

\[
\frac{\partial (\rho \vec{u})}{\partial t} + \vec{u} \cdot \nabla (\rho \vec{u}) = -\frac{\rho}{\rho_0} \nabla P_\infty + \mu \nabla^2 \vec{u}
\]

(6)

for corresponding initial and boundary conditions, \( P_\infty \) is the incompressible pressure and \( \rho_0 \) is the conserved initial mean density, supplemented by an equation of state

\[
\rho_\infty = c_s^2 \rho
\]

(7)

where \( c_s^2 \) is a tunable model constant (speed of sound). Theoretical analysis shows, that the kinematic viscosity \( \nu \) depends linearly on the microscopic relaxation time \( \tau \)

\[
\nu = \frac{2\tau - 1}{6}
\]

(8)

which establishes a direct connection between Equations (1) and (6). The use of generalized LB models where the distributions are relaxed in momentum space using a set of carefully selected relaxation times increases stability and minimizes anisotropic hyperviscosities (d’Humières et al., 2002).

Evidently, Equation (1) is a finite difference scheme based on a local stencil. Thus a common approach to implement Equation (1) is to map the discretized flow domain onto Cartesian grids. The grid generation then becomes trivial: A flag indicates the type of node with respect to being a part of the flow field or the solid domain or a certain type of boundary condition.

For systems under consideration, the flow is usually turbulent. It turns out that LB models can be readily extended for turbulent flows by borrowing ideas from Navier–Stokes–based turbulence models, for example, an algebraic Smagorinsky model allowing LB models to be used for large-eddy simulations (Hou et al., 1996). Then the influence of the unresolved small-scale fluid motion is modeled by an additional eddy viscosity \( \nu_T \) so that the total viscosity is composed of \( \nu_T \) and the material related kinematic viscosity \( \nu_0 \):

\[
\nu_{\text{total}} = \nu_0 + \nu_T = \frac{2\tau_0 - 1}{6} + \frac{\tau_T}{3}
\]

(9)

Smagorinsky’s Ansatz for the eddy viscosity is given by

\[
\nu_T = \frac{c_s^2}{5} e_{\text{LB}}^{1/2}
\]

(10)

where the strain rate tensor is defined as

\[
e_{\text{LB}}^{1/2} = \frac{1}{2} \frac{1}{\rho(\tau_0 + \tau_T)} S_{\text{LB}}^{1/2}
\]

(11)

based on a stress tensor

\[
S_{\text{LB}}^{1/2} = \left( \frac{1}{2\tau_0} - 1 \right) \sum_i e_{ia} e_{ia} (f_i - f_i^{(0)})
\]

(12)

The local “turbulent” relaxation time is then obtained from

\[
\tau_T = \sqrt{\frac{2}{\nu_0^2} + \frac{18c_s^2}{\nu_0^2} Q^{1/2}} - \tau_0
\]

(13)

where

\[
Q = \left| e_{\text{LB}}^{1/2} \right|
\]

It should be emphasized, that the stress tensor \( S_{\text{LB}}^{1/2} \) is a nodal (local) quantity that is not computed by numerical approximations of spatial derivatives of the velocity field. This has considerable numerical and algorithmic advantages. The strong locality in terms of data dependence for this large-eddy LB model allows the efficient use of vectorizing hardware and parallel computers.

Several extensions for thermal flows have been developed. In our research implementation, we use an approach proposed in Filippova and Hänel (2000) to additionally solve the energy transport equation and couple the temperature field to the particle distribution densities.

The implementation of inflow, outflow, and nonslip boundary conditions can be done in the LB context (Succi, 2001). Because we solve a separate energy equation discretized by finite differences, thermal boundary conditions can be implemented in a straightforward manner (Filippova and Hänel, 2000).

4 DATA VISUALIZATION AND ANALYSIS

4.1 Overview

Typical simulations are performed using grids of \( O(10^6 \text{ to } 10^7) \) discrete points. The corresponding field values of these grid nodes are usually mapped onto graphical devices as large sets of lines (e.g., streamlines mapping the discrete velocity data with a set of discrete lines, see Figure 14), triangles or polygons (see Figures 11–13).

The algorithmic chain from the discrete simulation results to the final computer graphic contains the following steps:

1. Data input is usually based on classical file interfaces, but importing the data directly from the simulation via interprocess communication will become more important in the future to tighten the coupling of the complete simulation.
Filtering: A subset of the input data is extracted, for example, considering a desired data component (e.g., pressure) or extracting data points within a given bounding volume or simply subsampling the data set.

Computing the data representation items (mapping—see Section 4.3).

Reduction of mapped data using (error controlled) polygon reduction.

Creation of graphical objects for OpenGL-based rendering.

Today, typical postprocessing software tools (e.g., AVS/Express™) provide hardware accelerated rendering capabilities and a visual programming environment. The user is enabled to quickly modify the above presented chain and thus can explore different versions of data representations.

The most challenging part of postprocessing is the step of mapping the simulation data. In contrast to reading and filtering data, it is repeated each time a user requests a new representation of the data set. This mapping is necessary, if for example a user explores the simulation results by moving an orthogonal slice through the data set from the minimum to the maximum extend.

Our approach to improve the efficiency of the visualization chain is to profit from the spacetree (octrees in 3D) data structures: An octree data structure is used to reduce the simulation data before the creation of the data representations (see Section 4.2). The corresponding algorithms for visualization are then implemented on the basis of tree data structures (see Section 4.3).

4.2 Data reduction based on octree data structures

Efficient analysis of the resulting data depends significantly on the possibility to explore different types of data representation in a very short time. Consider the following situation: An investigating engineer is studying the flow field using particle tracing techniques (see Section 4.3). Therefore, he defines a line describing the starting positions of the particle traces. For large data sets of \( O(10^6) \) to \( O(10^7) \) discrete grid points, the computation time of the particle traces is up to several seconds on a standard PC, if standard visualization procedures based on uniform grids are used. To get a detailed idea of the flow, he might take a look at a multitude of particle traces, which will be a very tedious process.

To overcome this problem, we have developed a method to reduce the simulation data set (Kühner et al., 2001) by mapping the discrete vector field onto a spacetree data structure (an octree in the 3D case, see Section 2.2). The basic idea is to benefit from the hierarchic structure of the octree and to locally coarsen it in ranges satisfying user-controlled reduction criteria. The application of the data reduction is typically separated from the postprocessing software used so as to be independent of the limited computing resources of the graphics workstation.

Figure 6 illustrates the reduction of a cell of an octree. The cell is typically coarsened eliminating its eight children, all having to be leaves at the present iteration. The discrete data points of the finite difference-based simulation kernel are stored at the corners of the cells (vertex associated data) of the octree, while a cell including its children has 27 corner vertices. The eight corners of the reduced cell will be kept, while the other 19 vertices (1 vertex in the center of the cell, 6 vertices in the center of the faces of the cell and 12 vertices at the midpoint of the edges of the cell). As a reduction criterion, we choose the curvature (Blom and Verwer, 1996) of the investigated vector field (see Figure 7). Reduction is only allowed, if a user-defined value is not exceeded (see Equation (14)) at all the 19 vertices to be considered.

The curvature of the vector field is described by the derivative \( \nabla u_j(x_k) \), being numerically approximated using finite difference schemes of second-order accuracy. Therefore, information from neighboring cells becomes necessary, which are efficiently accessed using the flat linkage described in Section 2.3.

The curvature monitor is then scaled with the maximum \( \nabla u_j, \text{max} \) of the considered component \( j \) of the vector field and finally compared to a user defined value \( \varepsilon \) as stated above:

\[
\frac{\nabla u_j(x_k)}{\nabla u_j, \text{max}} < \varepsilon,_{\text{max}} \tag{14}
\]

The method was also implemented for parallel processing, to efficiently handle very large data sets resulting from parallel computations of the LB kernel.

4.3 Computing data representation with octrees

The resulting coarsened grid is again an octree. Of course, it is possible to use visualization modules being
designed for unstructured meshes. For performance reasons, however, the following data mapping algorithms were adapted, being most important for visualization of fluid flows:

- streamlines and particle traces (for transient flows);
- isosurfaces using a modified “marching cubes” (Lorensen and Cline, 1987) algorithm;
- colored planes cutting the flow field;
- vector glyphs (arrows).

In principal, the core features of these algorithms have the same properties as in the case of block structured Cartesian grids. Yet, one main algorithmic difference between block structured meshes and spacestrees can be found when searching a cell containing a given point, being an important operation in visualization. In this context, the implementation benefits from the modified octree data structure described in Section 2.3.

The spatial organization of the octree also helps to reduce the computational effort for generating an isosurface. The “marching cubes algorithm” (Lorensen and Cline, 1987) computes isosurfaces composed of a set of discrete triangles. To find these facets, each cell of a grid is scanned and it is tested, if an intersection with the isosurface exists, where all possible intersections are stored in a precomputed lookup table. The corresponding index of the current intersection form is determined by a comparison of the iso value under consideration with the eight vertices of a cell. To tune this procedure, two additional attributes are stored for each cell: The minimum and maximum value of all vertices being located below the cell in the hierarchy of the tree. Thus, the number of cells being examined for an intersection with the isosurface can be reduced in a preselection step: The tree is traversed recursively, testing for a cell, if the given iso value is within its range of values. If this is not the case, the whole cell (including all following children) will be excluded from the intersection test described above and the recursion is terminated for this cell. Otherwise, this test is recursively repeated for all children of the cell.

5 EXAMPLE: AN OPEN-PLAN OFFICE

Indoor air flow in an open-plan office was simulated and different seating arrangements were compared. The complete chain of the simulation is presented for one selected placement of desks.

As described in Section 2.2, first a facet model is derived from the CAD model, which is used as the (approximate) input geometry for the octree generation (Figures 8 and 9).

The boundary conditions being derived from the air-conditioning system can be identified in Figure 10. Inflow
openings are located at the ceiling and at the upper part of the wall, while the other openings are outlets.

The LB simulation was performed on a high-performance parallel vector computer, the Hitachi SR8000-F1 (http://www.lrz-muenchen.de). Applying a large-eddy turbulence model, the 3D indoor air flow was computed at a Reynolds number of 50,000.

The Cartesian lattice of the simulation consists of $302 \times 78 \times 252$ grid points. Data reduction (Section 4.2) to approximately 10% of the original grid size is applied for postprocessing. Figures 11–13 present a slice plane mapping the modulus of the flow velocity for the original and the coarsened grid. These figures illustrate that almost no visible loss of accuracy can be identified.

Figure 14 shows a screenshot of our postprocessing software allowing a combined visualization of CAD data and simulation results.

6 CONCLUSIONS AND FUTURE WORK

The complete process of simulating fluid-flow phenomena around complex building structures, starting from a product data model and using LB methods has been presented in this article.

Octree data structures are used to closely couple the three main components of the simulation (preprocessing, computation, and postprocessing). The most important goal was to shorten the design cycle, especially the time being spent for the tedious processes of problem definition (grid generation) and data analysis.

Three-dimensional, turbulent indoor air flow problems at $Re$ of up to $10^5$ can be performed with acceptable simulation times using a large-eddy turbulence model. Investigation of flow problems beyond this Reynolds number exceeds, however, the capabilities of simple grids. Especially the resolution of local effects, such as
boundary layers or details of recirculation zones require locally refined grids. A prototype of an adaptive LB solver (2D) is presented in Crouse et al. (2003).

Our current research focuses on integrating all components of the simulation into one computational steering environment via interprocess communication (Mulder et al., 1999). The feasibility of this interactive concept has already been demonstrated in a prototype implementation (Kühner et al., 2003).

ACKNOWLEDGMENTS

The authors gratefully acknowledge financial support by Siemens AG, Munich and SoFiSTiK AG, Munich.

REFERENCES


