The Lattice-Boltzmann Method for Simulating Gaseous Phenomena

Xiaoming Wei, *Student Member*, *IEEE*, Wei Li,
Klaus Mueller, and Arie E. Kaufman, *Fellow*, *IEEE*

Abstract—We present a physically-based, yet fast and simple method to simulate gaseous phenomena. In our approach, the incompressible Navier-Stokes (NS) equations governing fluid motion have been modeled in a novel way to achieve a realistic animation. We introduce the Lattice Boltzmann Model (LBM), which simulates the microscopic movement of fluid particles by linear and local rules on a grid of cells so that the macroscopic averaged properties obey the desired NS equations. The LBM is defined on a 2D or 3D discrete lattice, which is used to solve fluid animation based on different boundary conditions. The LBM simulation generates, in real-time, an accurate velocity field and can incorporate an optional temperature field to account for the buoyancy force of hot gas. Because of the linear and regular operations in each local cell of the LBM grid, we implement the computation in commodity texture hardware, further improving the simulation speed. Finally, textured splats are used to add small scale turbulent details, achieving high-quality real-time rendering. Our method can also simulate the physically correct action of stationary or mobile obstacles on gaseous phenomena in real-time, while still maintaining highly plausible visual details.

Index Terms—Lattice Boltzmann model, graphics hardware, GPU, textured splatting, gaseous phenomena modeling.

◆

1 INTRODUCTION

Gaseous phenomena, such as rolling clouds, moving dust, rising steam, and billowing smoke, play an important role in graphics simulations. A good fluid model for gaseous phenomena should not only describe the flow, but also model the interaction between the flow and the surrounding environment in a physically correct manner. In Computational Fluid Dynamics (CFD), fluid properties and behaviors have been studied for many years. However, while the goal of researchers in fluid dynamics is to obtain highly accurate fluid behavior, our goal is to achieve physically-based realistic-looking results, while maintaining fast calculation and rendering speed.

A fluid is described by several macroscopic variables, such as density, velocity, pressure, and temperature. In CFD, fluid dynamics can be described by the Navier-Stokes (NS) equations. The following formulas are the incompressible 3D NS equations:

$$\nabla \cdot u = 0 \tag{1}$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu \nabla^2 u + f, \tag{2}$$

where u is the velocity field, p is the pressure field, ν is the kinematics viscosity of the fluid, ρ is its density, f is an external force, and ∇ is the vector of spatial partial derivatives. Equation (1) describes the conservation of mass, indicating that the inflow and outflow of mass in a fluid element are balanced. Equation (2) shows the conservation of momentum. It describes the velocity changes in time, due to convection, spatial variations in pressure, viscous forces, and external forces. Many books and articles (e.g., [12], [16]) have been published on how to solve these equations numerically. The difficulty in solving the NS equations is due to their nonlinear terms. This complexity makes the solution of these equations in 3D very time-consuming.

Since our goal is to achieve real-time computation and rendering speed, we propose in this paper the use of the linear and microscopic Lattice Boltzmann Model (LBM), instead of solving the macroscopic NS equations directly. Fluid flow consists of many tiny flow particles. The collective behavior of these microscopic particles results in the macroscopic dynamics of the fluid. The LBM is similar to a cellular automata whereby individual particle packets move on a discrete lattice at discrete time steps. The calculation is performed on a regular grid. At every grid cell, there are variables indicating its status. All the cells modify their status at every time step based on the same linear and local rules. Since these interaction rules are defined in such a way that they satisfy the conservation of mass and momentum at each local grid cell, the macroscopic NS equations are satisfied globally. The LBM also lends itself to proper handling of the boundary conditions and the interaction of the gaseous phenomena with the environment in a physically correct way.

In the traditional CFD approach, the physical modeling of the fluid behavior and the numerical approximation are separate steps, while, in LBM, those two phases are combined together. The main difference between the LBM and the traditional numerical approaches of the NS equations is that, in the LBM, the physical transition rules are discrete, while, in the latter, the discretization is performed on the level of the macroscopic NS equations. In both cases, the same physical conservation laws are satisfied.

• *The authors are with the Center for Visual Computing and Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400. E-mail: {wxiaomin, liwei, mueller, ari}@cs.sunysb.edu.*

In this paper, we show the application of the LBM to model both the movement of gaseous phenomena and their behavior with stationary or moving obstacles at interactive frame rates. The final performance gap is filled by realizing that the linear and local calculations that occur at each grid cell are very amenable to acceleration on a commodity graphics processing unit (GPU), resulting in interactive modeling rates of 30 frames/sec.

Using the LBM to yield a velocity volume rich in small scale rotational and turbulent details still requires a large 3D grid. Since our goal is to achieve fluid-like effects in real-time and not a strict physical simulation, we base our work on relatively low-resolution grids. Texture hardware acceleration enables us to accomplish real-time computation speed. To provide the missing details, we augment the low-resolution velocity field with high-resolution textures of small-scale perturbations, which can also be efficiently rendered on commodity texture mapping hardware. Our two-tier approach is justified by the fact that, in most graphics applications, gaseous objects are mainly used to enhance the realism of the virtual environment—users don't require observation of highly accurate details of the interaction between gaseous phenomena and obstacles. All that is needed is that the gases are behaving in a near physically correct way. It turns out that, by combining a low-resolution velocity grid with high-detailed textures, a surprisingly realistic viewing experience can be generated. By incorporating into the LBM a number of physically accurate effects, such as the buoyancy effect of hot gases, additional realism can be provided. Our framework makes efficient use of the concept of textured splats [6], which are associated with the macroscopic particles to represent the gaseous phenomena. The textured splats form the observable "display particles," such as the smoke particles or dust particles, while the LBM deals with nonobservable microscopic particles. To distinguish the two, we use the term "packet" to stand for the microscopic particles and the term "display primitive" to stand for the smoke or gas particles. The display primitives move freely through space, advected by the velocity field simulated by the underlying LBM grid as well as to other forces, such as the temperature-induced buoyancy.

The rest of the paper is organized as follows: In the next section, we briefly discuss related work in computer graphics on gaseous phenomena modeling and manipulation. In Sections 3 and 4, we present the basic ideas of the LBM and the initial and boundary condition settings. In Section 5, we discuss the incorporation of a temperature field in the LBM. We introduce the implementation of the LBM calculation in texture hardware in Section 6. In Section 7, we show how to use textured splatting to achieve high-quality real-time rendering speed. Finally, we outline our implementation and describe several examples in Section 8.

2 PREVIOUS WORK

A common approach to simulating gaseous phenomena is procedural modeling [11], [26], [30], where fluid behaviors are described by procedural functions. This method is fast and easy to program, but it is difficult to find the proper parameter settings that achieve realistic results. The interactions between the fluid and the surrounding objects, such as flowing around obstacles, blown by the wind, are also difficult to model in a physically correct manner.

Physically-based modeling is another broadly used method; however, it usually requires a large amount of computation time and memory, especially in 3D. In 1991, Wejchert and Haumann [38] gave an analytic solution to the NS equations by using simple flow primitives. Chen and Lobo [2] solved a simplified NS equations in 2D using a finite difference approach. Later, Foster and Metaxas [14] presented a full 3D finite difference solution to simulate the turbulent rotational motion of gas. Because of the inherent instability of the finite difference method with a larger time step, the speed of this approach is limited. Stam [31] devised a fluid solver using a semi-Lagrangian advection scheme and implicit solver for the NS equations. Each term of the equations is handled in turn, starting with external force, then advection, diffusion, and finishing with a projection step. This method is unconditionally stable and produces compelling simulations of turbulent flows. It can also achieve real-time speed for a low-resolution grid. However, the numerical dissipation associated with the method causes the turbulence to decay too rapidly. Fedkiw et al. [13] further improved the result by introducing the concept of vortex confinement to the graphics field. The problem of numeric dissipation is addressed by feeding energy back into vortices through vortex confinement. Two rendering approaches were used in their work: One is a fast hardware-based renderer, the other is an expensive photon map renderer to create production quality animations.

In this paper, we demonstrate a new and pioneering direction on how real-time fluid simulation could also be approached, instead of solving the traditional NS equations. All of the above methods generate fluid-like behaviors by solving macroscopic equations either by explicit or implicit approaches. In contrast, the LBM approach considers the problem from the microscopic perspective. Since we use the LBM on a relatively low-resolution grid, one may ask why not solve the NS equations on the same grid. Although it is true that combining the NS equations with the high-resolution textures has the same effect, however, the computation of the NS equations is not as simple as that of the LBM. Also, the fact that the calculation of the LBM only consists of simple operations such as addition, subtraction, and multiplication and it is conducted locally allows us to further improve the calculation speed of LBM by employing commodity texture hardware. This achieves the desired real-time speed. In the next section, we present a complete framework for the LBM-based fluid simulation and how its calculation is implemented based on the fast growing GPU technology.

Physically-based particle models [5], [7], [22], [24], [33], [34], [35] have also been used to describe fluid behaviors. Particle systems were first introduced by Reeves [28] as a technique for modeling fuzzy objects, such as fire, clouds, smoke, and water. Tonnesen [35] used a discrete model for the heat transfer equations to describe the interaction of particles due to the thermal energy. Terzopoulos et al. [34] implemented a similar approach. Particles and springs are utilized to render a series of *blobbies*. Desbrun and Gascue [7], [8] developed a paradigm extended from the Smoothed Particle Hydrodynamics approach used by physicists for cosmological fluid simulation. This technique defines a type of particle system which uses smoothed particles as samples of mass smeared out in space. A highly inelastic fluid is simulated by computing the variations of contin-

uous functions such as mass, density, speed, pressure, or temperature over space and time. Stora et al. [33] also used smooth particles to simulate lava flow. Particles are coated with implicit surfaces to get the final rendering result. Particles in all of the above methods move and interact freely in space. They are irregular, difficult, and expensive to track at $O(N^2)$ complexity. While, in the LBM approach, particle packets move and interact on a regular grid based on linear and local rules, it is proven that the global behavior is the same as the one achieved by directly solving the heavy NS equations, which is the common approach in graphics [19].

Dobashi et al. [9] implemented a realistic animation of clouds based on a cellular automata model. Our work is fundamentally different from theirs. Dobashi et al.'s model can only be used to model clouds. In contrast, the LBM is a general method used to describe a variety of fluids. Harris et al. [18] implemented a coupled map lattice, a variation of cellular automata, on graphics hardware, which has similar motivation as this paper. To the best of our knowledge, we pioneered the introduction and use of the LBM to simulate gaseous phenomena in the graphics literature.

3   LATTICE BOLTZMANN MODEL

The LBM [19], [25] is a lattice model similar to a cellular automata. It is defined on a 2D or 3D discrete grid, where the time and the state of each cell are also discrete. Fluid behavior can be understood as a self-organizing process evolving from the microscopic collisions of atoms or molecules. Cellular automata is used to simulate these microscopic movements and collisions in order to get the continuum macroscopic equations of fluid dynamics in two and three dimensions. The class of cellular automata used for the simulation of fluid dynamics is called the *Lattice Gas Automata* (LGA) [10]. The main difference between the LGA and the traditional numerical approaches of the NS equations is that, in the LGA, the physical evolution rules are discrete, while, in the latter, the discretization is performed on the level of the macroscopic flow equations.

The first LGA, introduced by Hardy et al. [17], was called the HPP model, defined on a square lattice. In this model, microscopic particles of unit mass and unit speed move along the lattice links. Not more than one particle in a given direction can be found at a given time and node. When two microscopic particles arrive at a node from opposite directions, they immediately leave the node in the two other, previously unoccupied directions. These rules conserve mass (particle count) and momentum. The main problem with this model is that the gaseous behavior it modeled is anisotropic.

A historically important lattice gas model is the FHP model, introduced by Frisch et al. [15] in 1986. It is a 2D hexagonal lattice used to ensure macroscopic isotropy. In this model, each cell has six nearest neighbors and, consequently, six possible velocity directions. Updating the grid involves two types of rules: propagation and collision. Propagation means the microscopic particles move to the nearest neighbor along their velocity direction. Collision is the most important part. It can force particles to change direction and is decided by the collision operator. No matter how we define the collision operator, the conservation of mass and momentum must be satisfied. For instance, when
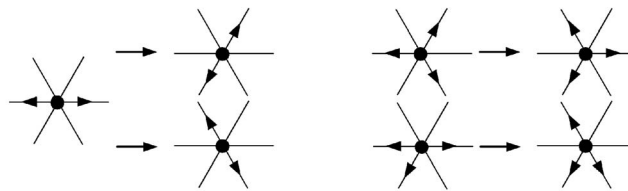

Fig. 1. Collision rules of the FHP model, defined on a triangular lattice. An arrow denotes a microscopic particle and its moving direction. When multiple postcollision states are possible, a random selection is made. The collision rules satisfy mass and momentum conservation.

two microscopic particles enter the same node with opposite velocities, both of them are deflected by 60 degrees such that the net momentum in the postcollision state remains zero. When multiple states are possible, a random selection is made. Fig. 1 represents the collision rules for the FHP model.

It can be demonstrated [15] that, by observing the propagation and collision rules, we can simulate the following macroscopic equations based on the LGA:

$$\frac{\partial u}{\partial t} = -(g(\rho)u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu\nabla^2 u, \qquad (3)$$

where ρ is the density and can be calculated as the average number of microscopic particles per cell and $g(\rho) = \frac{\rho-3}{\rho-6}$. If we renormalize the time t, the viscosity ν, and the pressure p using $t' = g(\rho)t$, $\nu' = \frac{\nu}{g(\rho)}$, $p' = \frac{p}{g(\rho)}$, the NS equations for incompressible fluids without external force can be recovered.

Although the LGA has proven to be very useful for modeling fluid behavior, one major drawback of the method is the statistical noise in the computed hydrodynamic fields. This is a direct consequence of the single particle Boolean operation. One method to smooth out the noise is to average in space and time. In practice, spatial averages can be taken over 64, 128, 256, 512, or 1,024 neighboring cells for time-dependent flow in two dimensions. This, however, dramatically limits the space resolution we can get. The problem can be solved by replacing the Booleans in the LGA model by *real-valued densities* of microscopic particles that move along each bond of the lattice, following the motion of an *average* distribution f_{qi} of microscopic particles. This gives rise to a model known as the *Lattice Boltzmann model* (LBM) [3]. In the densities f_{qi}, the index qi describes the D-dimensional sublattice defined by the permutations of $(\pm1, \ldots, \pm1, 0, \ldots 0)$, where q is the number of nonzero components and i counts the sublattice vectors. For a 2D LBM, there are three lattice models: D2Q5, D2Q7, and D2Q9. Since our goal is to simulate the 3D gaseous phenomena, we concentrate on the 3D LBM. For a 3D grid, the geometry of the model should be symmetrical to satisfy the isotropic requirement of fluid properties. To correctly recover the NS equations, it also requires sufficient lattice symmetry. In order to understand the design of 3D LBM model, the notion of *sublattice* is used in the LBM literature. Out of the 26 neighbors of an individual cell, four symmetrical sublattices are defined on a 3D grid, as described in Fig. 2a, Fig. 2b, Fig. 2c, and Fig. 2d:
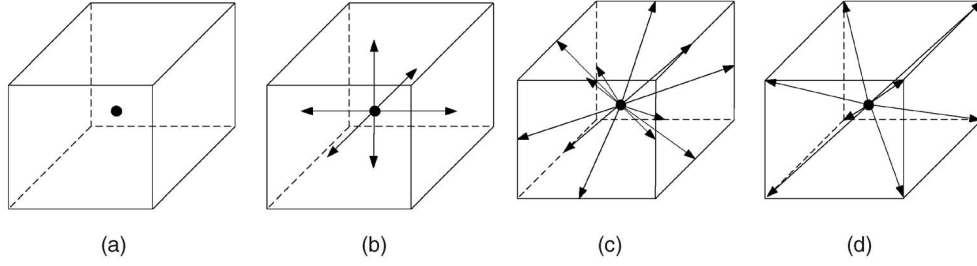
Fig. 2. The four sublattices defined on a 3D grid. The velocity vectors e_{qi} are shown as arrows in each sublattice.

1. $q = 0$, the cell $(0, 0, 0)$ has particle packets with zero velocities—Fig. 2a;
2. $q = 1$, the six nearest neighbors $(\pm1, 0, 0)$, $(0, \pm1, 0)$, $(0, 0, \pm1)$, where the particle packets move with unit velocity—Fig. 2b;
3. $q = 2$, the 12 second-nearest neighbors $(\pm1, \pm1, 0)$, $(0, \pm1, \pm1)$, $(\pm1, 0, \pm1)$, where the particle packets move with a velocity of $\sqrt{2}$—Fig. 2c; and
4. $q = 3$, the eight third-nearest neighbors $(\pm1, \pm1, \pm1)$, where the particle packets move with a velocity of $\sqrt{3}$—Fig. 2d.

To simulate a 3D flow, and for the sake of computational efficiency, we try to find the minimum necessary number of packet distributions. In practice, there are three widely used 3D LBM geometries: D3Q15, D3Q19, and D3Q27. D3Q15 includes sublattices 0, 1, and 3, shown in Fig. 2a, Fig. 2b, and Fig. 2d, having the least number of packet distribution values. However, it is the least isotropic and more prone to numerical instability. It also may induce an artificial checkerboard invariant and cause spatial oscillation when simulating turbulent flows. D3Q27 is composed of all four sublattices and, due to this complexity, it requires 27 packet distribution calculations at each fluid node. It needs too much CPU time and storage space. Our work is based on D3Q19, which represents a good compromise in terms of both computational efficiency and reliability. The D3Q19 lattice consists of three sublattices: sublattices 0, 1, and 2, shown in Fig. 2a, Fig. 2b, and Fig. 2c. In other words, at each cell of D3Q19, there are 19 possible flow directions and that $f_m(x, t)$ is the packet distribution at location x, at time t and moving in direction m, where $1 \le m \le 19$. Fig. 3 shows the D3Q19 model lattice geometry. The velocity directions of the 18 moving packet distributions are shown as arrows. The center is the packet distribution with zero velocity.

Similar to the LGA, the LBM updates the packet distribution values at each node based on two simple and local rules: collision and propagation. Collision describes the redistribution of particle packets at each local node. It is decided by the collision operator. Propagation means the particle packets move to the nearest neighbor along their velocity directions. These two rules of the LBM can be described by the following equations:

$$collision : f_{qi}^{new}(x, t) - f_{qi}(x, t) = \Omega_{qi} \qquad (4)$$

$$propagation : f_{qi}(x + e_{qi}, t + 1) = f_{qi}^{new}(x, t), \qquad (5)$$

where Ω is a general collision operator and e_{qi} is the unit vector, representing the packet velocity along the lattice link

qi. The collisions are completely local, making the LBM efficiently parallelizable. In one time step t, each packet distribution value at every node is updated based on the collision operator Ω. Then, in time step $t + 1$, the new packet distribution value propagates to the nearest node along the velocity vector e_{qi}.

The macroscopic density (mass) ρ and velocity u are calculated from the respective velocity moments of the packet distributions as follows:

$$\rho = \sum_{qi} f_{qi} \qquad (6)$$

$$u = \frac{1}{\rho} \sum_{qi} f_{qi} e_{qi}. \qquad (7)$$

Combining (4) and (5), we get

$$f_{qi}(x + e_{qi}, t + 1) - f_{qi}(x, t) = \Omega_{qi}.$$

The update of the LBM system is decided by the collision operator Ω_{qi}. It is critical to select Ω_{qi} in such a way that the mass and momentum are conserved locally. Based on the work of Chen and Doolean [3], we assume that, for each individual packet distribution f_{qi} at each cell, there is always a local equilibrium packet distribution f_{qi}^{eq}. Its value only depends on the conserved quantities ρ and u at that cell. In this way, we get a new equation, also called the *kinetic equation*:

$$f_{qi}(x + e_{qi}, t + 1) - f_{qi}(x, t) = -\frac{1}{\tau}(f_{qi}(x, t) - f_{qi}^{eq}(\rho, u)), \qquad (8)$$

where τ is the relaxation time scale and $f_{qi}^{eq}(\rho, u)$ is the equilibrium packet distribution. According to Muders's
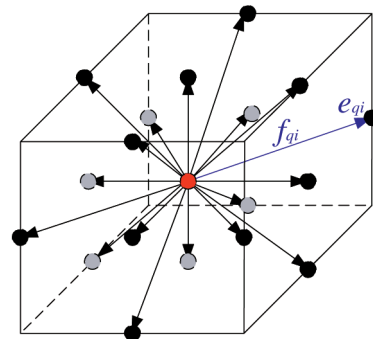

Fig. 3. The D3Q19 lattice geometry.

TABLE 1
Coefficients of the Three Sublattices in the D3Q19 Model

	Sub-lattice 0	Sub-lattice 1	Sub-lattice 2
A_q	$\frac{1}{3}$	$\frac{1}{18}$	$\frac{1}{36}$
B_q	0	$\frac{1}{6}$	$\frac{1}{12}$
C_q	0	$\frac{1}{4}$	$\frac{1}{8}$
D_q	$-\frac{1}{2}$	$-\frac{1}{12}$	$-\frac{1}{24}$

work [25], the equilibrium packet distribution can be represented by a linear formula:

$$f_{qi}^{eq} = \rho(A_q + B_q(e_{qi} \cdot u) + C_q(e_{qi} \cdot u)^2 + D_q(u)^2), \quad (9)$$

where the coefficients A_q through D_q are dependent on the employed lattice geometry. They are constant values for the specific model. In Table 1, we list the coefficients used in the D3Q19 model. Using these coefficients in conjunction with (8) and (9) ensures local conservation of mass and momentum.

The *viscosity* of a fluid is a measure of the fluid resistance to change of shape. For example, water has a higher viscosity value than gas. In LBM, the viscosity ν is decided by the parameter τ with equation $\nu = \frac{1}{3}(\tau - \frac{1}{2})$. Since the viscosity is always greater than zero, τ must be greater than $\frac{1}{2}$.

Our algorithm for calculating the LBM is as follows:

1. Set the initial conditions for all grid cells, choose proper density and velocity for inlet cells, and select a relaxation time scale τ;
2. Calculate the macroscopic variables of density and velocity for each cell using (6) and (7);
3. Compute the equilibrium packet distribution for each packet distribution by (9);
4. Plug the packet distribution and equilibrium values into the kinetic equation (8);
5. Propagate the packet distribution to all neighboring cells;
6. Modify the packet distribution locally to satisfy the boundary conditions;
7. Back to Step 2.

4   INITIAL AND BOUNDARY CONDITIONS

Initial conditions are usually specified in terms of macroscopic variables, such as densities and velocities. For the LBM, these macroscopic values are translated into the corresponding microscopic packet distribution values for each fluid node. This is done by solving the equilibrium equation (9). Hence, the initial values of the density and velocity of each node are plugged into (9) and the equilibrium particle distribution values are set as the initial packet distribution values for each node.

For a grid node near the boundary, some of its neighboring nodes lie outside the fluid domain. Therefore, the packet distribution values coming from those solid nodes are not uniquely defined. For D3Q19, as shown in Fig. 4, there are several unknown packet distribution values for the boundary nodes, for example, five unknowns for planar surface nodes and nine for concave-edge nodes. We need to generate a boundary solution to set these unknown values.
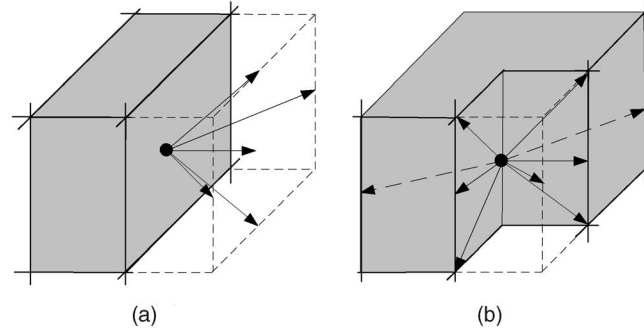

(a)   (b)

Fig. 4. Boundary geometry examples for D3Q19: (a) planar boundary surface with five unknown values, (b) concave edge boundary with nin unknown values (the invisible links shown by dashed arrow lines).

Different types of boundary conditions have been introduced in the field of Hydrodynamics for the LBM. Bounce-back is the simplest one, where boundary nodes are placed halfway between the grid points. When particle packets propagate to the boundary nodes, they just bounce back along the same link. The propagation step is changed to $f_{-qi}(x, t+1) = f_{qi}(x, t)$. Another approach is using periodic boundary conditions. Here, the departing packet distribution along outward-pointing links is allowed to reenter the lattice via corresponding inward-pointing links on opposite boundaries. Both of these approaches are easy to implement and solve the unknown packet distribution values based on the known packet distribution values of the neighboring cells. However, these two methods cannot set a unique value for the invisible links shown in Fig. 4b. Also, they can't be used to set the specific density and velocity constraints at the inlets, outlets, and walls.

The purpose of setting boundary conditions is to determine the packet distribution values for the incoming links on the boundary nodes. Most methods (e.g., [19]), such as the bounce-back and periodic boundary condition, calculate these unknown packet distribution values explicitly. In this way, we cannot determine the velocity and density value on these boundary nodes, such as a velocity value of 0.1 at the inlet boundary node. An incorrect density or velocity value at the boundary nodes can eventually cause a negative density value at local grid points. As we solve the kinetic equation (8) with τ close to $\frac{1}{2}$, we find that if $f_{qi}^{eq}(\rho, u)$ is much smaller than $f_{qi}(x, t)$, the packet distribution value $f_{qi}(x + e_{qi}, t+1)$ will be changed to a negative number, which is incorrect. This error can accumulate and eventually generate a negative density value. The problem can happen as the boundary conditions become complicated, such as a fast moving boundary object or a complex geometric structure.

In our work, we implemented the boundary conditions based on Mei et al.'s method [23] for curved boundaries. In their approach, the problem is solved in another way. Instead of directly setting the microscopic values, they calculate the macroscopic variables of density and velocity at the boundary nodes first. By modifying the equilibrium equation, one can get better results for complicated boundaries.

As shown in Fig. 5, in the propagation step, the knowledge of $f_{\overline{qi}}(x_b, t)$ of a node b at x_b on the boundary
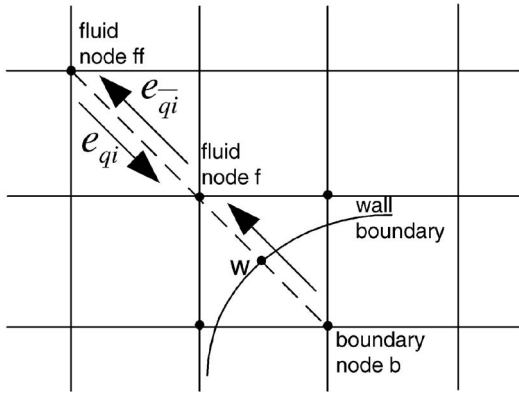
Fig. 5. The 2D projection of a regularly spaced lattice and a wall boundary. e_{qi} and $e_{\overline{qi}}$ are velocity vectors in opposite directions.

side is required in order to obtain the $f_{\overline{qi}}(x_f, t+1)$ for a node f at x_f on the fluid side. To decide the $f_{\overline{qi}}(x_b, t)$, we define

$$\delta = \frac{|x_f - x_w|}{|x_f - x_b|} \qquad (10)$$

as the fraction of an intersected link in the fluid region (it is obvious that $0 \leq \delta \leq 1$). The $f_{\overline{qi}}(x_b, t)$ is calculated based on the packet distribution values, velocity, and density values of neighboring cells at the current time step:

$$f_{\overline{qi}}(x_b, t) = (1 - \chi)f_{qi}(x_f, t) + \chi f_{qi}^*(x_b, t) + 6A_q \rho e_{\overline{qi}} \cdot u_w, \quad (11)$$

where A_q is the constant value we introduced in (9). χ is defined by δ and τ (see below) and u_w indicates the speed of the wall boundary. In the case of boundary surfaces with nonzero velocity, we modify the value of u_w. $f_{qi}^*(x_b, t)$ is the packet distribution value at the boundary node b. It is decided by a modified equilibrium distribution function:

$$f_{qi}^*(x_b, t) = \rho(A_q + B_q e_{qi} \cdot u_{bf} + C_q(e_{qi} \cdot u_f)^2 - D_q(u_f)^2), \qquad (12)$$

where A_q, B_q, C_q, and D_q are the constant coefficients defined in (9) and u_{bf} is the virtual speed at the boundary node b. It is set according to δ. When $\delta \geq 1/2$, we define $\chi = \frac{2\delta - 1}{\tau + 1/2}$ and

$$u_{bf} = \left(1 - \frac{3}{2\delta}\right)u_f + \frac{3}{2\delta}u_w. \qquad (13)$$

When $\delta < 1/2$, we have $\chi = \frac{2\delta - 1}{\tau - 2}$ and :

$$u_{bf} = u_{ff}. \qquad (14)$$

5 TEMPERATURE FIELD

In this section, we show how to incorporate a temperature field to model the buoyancy force of hot gases. Other types of external forces can also be added based on the work of Muders [25]. The temperature of smoke or gas has a direct effect on the behavior of their motion. Hot smoke tends to rise more quickly due to the buoyancy effect. The D3Q19 model used in our work is an isothermal model and it can't describe the energy transfer due to difference in temperature. There are many thermal LBMs [3] in the Hydrodynamics field. However, most of them suffer from instability problems.

To describe the effect of the buoyancy force, the evolution of temperature must also be modeled. The change of the temperature of the smoke in the air can be characterized as a combination of the convection and diffusion of heat in neighboring cells. To achieve fast speed, we use a linear equation, similar to [4], instead of an accurate differential equation, to approximate the change of temperature for the display primitives. Our model is governed by the following heat formula:

$$T_k(t) = \alpha T_k(t - 1) + \beta \sum_{j \neq k} G(d_{kj})T_j(t - 1), \qquad (15)$$

where α is the conservation coefficient, β is the transferability coefficient, d_{kj} is the distance between the display primitive k and j, and $G()$ is a function describing the thermal diffusion. We use a Gaussian filter as the function $G()$ to approximate the effect of diffusion. At $t = 0$, $T_k(0)$ is a predefined initial value, indicating the initial temperature of the display primitives.

If the temperature of a particle is lower than the ambient temperature in the air, the particle will disappear. For each particle, we add the vertical buoyancy force to the movement of the particle. This force depends on the temperature field and is governed by the following equation:

$$F_{buoyancy} = Hg(T_k - T_{ambient}), \qquad (16)$$

where g is the gravity in the vertical direction, H is the coefficient of thermal expansion, and $T_{ambient}$ is the predefined ambient temperature value.

6 MAPPING LBM TO GRAPHICS HARDWARE

We briefly review in this section the basic ideas of mapping LBM to graphics hardware, that is, a graphics processing unit (GPU). (For more details, see [21].) To compute the LBM equations on GPU, we divide the LBM grid and group the packet distributions f_{qi} into arrays according to their velocity directions. All the packet distributions with the same velocity direction are grouped into the same array, while keeping the neighboring relationship of the original model. Fig. 6 shows the division of a 2D model. We then store the arrays as 2D textures. For a 2D model, all such arrays are naturally 2D, while, for a 3D model, each array forms a volume and is stored as a stack of 2D textures. The idea of the stack of 2D textures is from 2D texture-based volume rendering, but note that we don't need three replicated copies of the data set.

In addition to the packet distributions, the density ρ, the velocity u, and the equilibrium distributions f_{qi}^{eq} are similarly stored in 2D textures. We project multiple textured rectangles with the color-encoded densities, velocities and distributions. For convenience, the rectangles are parallel to the viewing plane and are rendered orthogonally. By setting the texture interpolation to nearest-neighbor and choosing the resolution of the frame buffer and the texture coordinates properly, we create a one-to-one mapping from
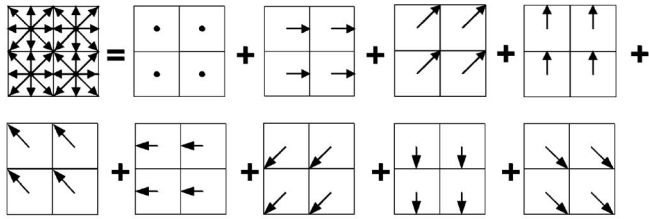
Fig. 6. Division of the D2Q9 model. Distribution values are grouped into textures according to their velocity directions.

Fig. 7. Propagation of the packet distributions along the direction of the velocity component orthogonal to the slices.

the texture space to the image space (one texel mapped to one pixel).

The textures of the packet distributions are the inputs. Density and velocity are then computed from the distribution textures. Next, the equilibrium distribution textures are obtained from the densities and the velocities. According to the propagation equation, new distributions are computed from the distributions and the equilibrium distributions. Finally, we apply the boundary conditions and update the distribution textures. The updated distribution textures are then used as inputs for the next simulation step. To reduce the overhead of switching between textures, we stitch multiple textures representing packet distributions with the same velocity direction into one larger texture. Fig. 7 shows an example in which every four slices are stitched into a larger texture.

According to (5), each packet distribution having nonzero velocity propagates to the neighboring grid every time step. Since we group packets based on their velocity directions, the propagation is accomplished by shifting distribution textures in the direction of the associated velocity. We decompose the velocity into two parts, the velocity component within the slice (*in-slice velocity*) and the velocity component orthogonal to the slice (*orthogonal velocity*). The propagation is done for the two velocity components independently. To propagate in the direction of the in-slice velocity, we simply translate the texture of distributions appropriately.

If we don't stitch multiple slices into one texture, the propagation in the direction of the orthogonal velocity is done simply by renumbering the distribution textures. Because of the stitching, we need to apply translation inside the stitched textures as well as copying subtextures to other stitched textures. Fig. 7 shows the out-of-slice propagation for stitched slices. The indexed blocks denote the slices storing packet distributions. The rectangles with thicker boundaries mark the subtextures that are propagated. For example, the subtexture composed of slices 1 to 3 is shifted down by the size of one slice in the Y dimension. Slices 4 and 8 are moved to the next textures. Note that, in time step $t + 1$, a new slice is added owing to the inlet or the boundary condition, while block 12 has moved out of the framework and is discarded.

To implement the LBM computation with fixed point graphics hardware, we need to map all the variables to the range of $[0, 1]$ or $[-1, 1]$. Special care must be taken so that all the intermediate values are also within the hardware supported range. Because various multiplications are involved, we only apply scaling to change the numerical ranges. Apparently, the scale factors should be appropriate so that no clamping error occurs and the computation exploits the full precision of the hardware.

For each function $y(x)$, we compute two scaling factors: the left-hand scalar, $y^{max} = \max_x(|y(x)|)$, and the right-hand scalar, $U(y)$, as the maximal absolute value of all inter-
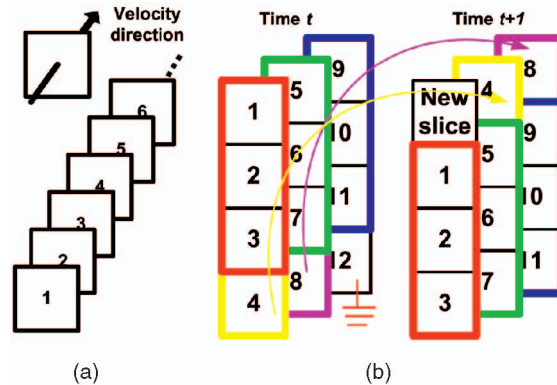
mediate results during the evaluation of $y(x)$, no matter what computation order is taken when the computation contains multiple operations. Assume f_q^{max} is the left-hand scalar of the packet distributions and the equilibrium packet distributions of sublattice q. We define the scaled distributions $\widetilde{f_{qi}}$ and the scaled density $\widetilde{\rho}$ as:

$$\widetilde{f_{qi}} = \frac{1}{f_q^{max}} f_{qi} \qquad (17)$$

$$\widetilde{\rho} = \frac{\rho}{\rho^{max}} = \sum_{qi} \frac{f_q^{max}}{\rho^{max}} \widetilde{f_{qi}}. \qquad (18)$$

Since all the f_{qi} are positive inputs, $\rho^{max} = U(\rho) = \sum_{qi} f_q^{max}$. We also define:

$$\frac{1}{\widetilde{\rho'}} = \frac{\rho^{min}}{\widetilde{\rho}\rho^{max}}, \qquad (19)$$

where ρ^{min} is the lower bound of the density and $\frac{1}{\widetilde{\rho'}} \in [0, 1]$.

We compute the right-hand scalar of the velocity $U(u)$ as:

$$U(u) = \frac{1}{\rho^{min}} \max_b \sum_{qi} f_q^{max} \{e_{qi}[b] > 0\}, \qquad (20)$$

where b is the dimension index of vector e_{qi}. Note that $U(u)$ and u^{max} are scalars rather than vectors. Then, the scaled velocity is computed as:

$$\widetilde{u} = \frac{u}{u^{max}} = \frac{U(u)}{u^{max}} \frac{1}{\widetilde{\rho'}} \sum_{qi} \left(\frac{f_q^{max}}{U(u)\rho^{min}} e_{qi} \right) \widetilde{f_{qi}}. \qquad (21)$$

With such range scaling, (8) and (9) become:

$$\widetilde{f_{qi}}(x + e_{qi}, t + 1) = \widetilde{f_{qi}}(x, t) - \frac{1}{\tau} (\widetilde{f_{qi}}(x, t) - \widetilde{f_{qi}^{eq}}) \qquad (22)$$

$$\widetilde{f_{qi}^{eq}} = \frac{U(f_q^{eq})}{f_q^{max}} \widetilde{\rho} \left(\frac{A_q \rho^{max}}{U(f_q^{eq})} + \frac{2 B_q u^{max} \rho^{max}}{U(f_q^{eq})} < \frac{e_{qi}}{2}, \widetilde{u} > \right.$$
$$+ \frac{4 C_q (u^{max})^2 \rho^{max}}{U(f_q^{eq})} < \frac{e_{qi}}{2}, \widetilde{u} >^2 + \qquad (23)$$
$$\left. \frac{4 D_q (u^{max})^2 \rho^{max}}{U(f_q^{eq})} < \frac{\widetilde{u}}{2}, \frac{\widetilde{u}}{2} > \right).$$
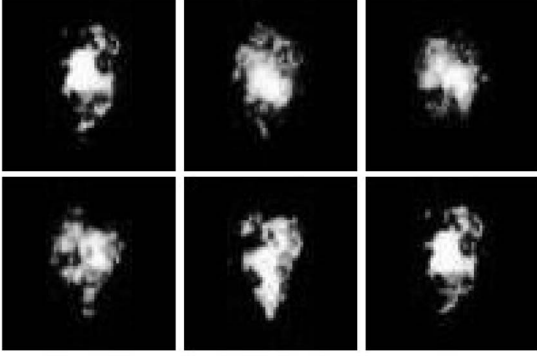
Fig. 8. A set of textures generated from real images (courtesy of Scott King).

For the D3Q19 model, $A_q \geq 0$, $B_q \geq 0$, $C_q \geq 0$, and $D_q \leq 0$, hence:

$$U(f_{qi}^{eq}) = \max((\rho^{max} A_q + 2B_q u^{max} \rho^{max} + 4C_q(u^{max})^2 \rho^{max}),$$
$$(2B_q u^{max} \rho^{max} - 4D_q(u^{max})^2 \rho^{max})).$$

$$(24)$$

Note that, in (23), we scale the vectors before the dot products. The scaling factor is chosen to be a power of two for easy implementation in hardware.

A major concern about using graphics hardware for general computation is accuracy. Most graphics hardware supports only 8 bits per color channel. Fortunately, the variables of the LBM fall into a small numerical range, which makes the range scaling effective. Besides, the property of the LBM that the macroscopic dynamics is insensitive to the underlying details of the microscopic physics [1] relaxes the requirement on the accuracy of the computation. In the new generation of GPUs, such as ATI's R300 and Nvidia's NV30, floating-point computation is available throughout the fragment shader, hence the accuracy is less of an issue. However, floating-point requires more memory and is slower than its fixed-point counterpart. Therefore, the proposed scaling is still valuable in applications where speed is more important than accuracy or the variables are restricted to a small range. Note that our range scaling makes no assumption of the precision of the hardware.

7 RENDERING WITH TEXTURED SPLATS

In this section, we discuss our approach of rendering the gaseous phenomena. When observing gaseous objects in real life, we notice a variety of rotational and turbulent movements and structures at a variety of scales. Employing the LBM and the temperature field to calculate all these

Fig. 9. Hot steam rising up from a teapot and its spout.

turbulence details would require a very large 3D grid size. Extremely time-consuming simulations would ensue, which is counter to our goal of generating a realistic-looking model at interactive frame rates. The LBM model already provides the physically correct large-scale behaviors and interactions of the gaseous phenomena, at real-time speeds. What we require now is an equally efficient way to add the small-scale turbulence details into the visual simulation and render these to the screen. One way to model the small-scale turbulence is through spectral analysis [32]. Turbulent motion is first defined in Fourier space and then it is transformed to give periodic and chaotic vector fields that can be combined with the global motions. Another approach is to take advantage of commodity texture mapping hardware, using textured splats [6] as the rendering primitive. King et al. [20] first used this technique to achieve fluid animation based on simple and local dynamics. A drawback of their model is, however, that it lacks the interaction of the fluid with environmental influences, such as wind, temperature, obstacles, and illumination, isolating the modeled gaseous phenomena from the rest of the scene.

While our LBM simulation takes care of the large-scale interactions of the fluid with the scene, our rendering approach adds the small-scale interactions and visual details on-the-fly during the interactive viewing stage. A key component of our approach is textured splats, which can be efficiently rendered on any commodity graphics hardware. Textured splats allow us to model both the visual detail of the natural phenomena itself as well as the volumetric shadows cast onto objects in the scene. Our

TABLE 2
LBM Timing Results (in ms)

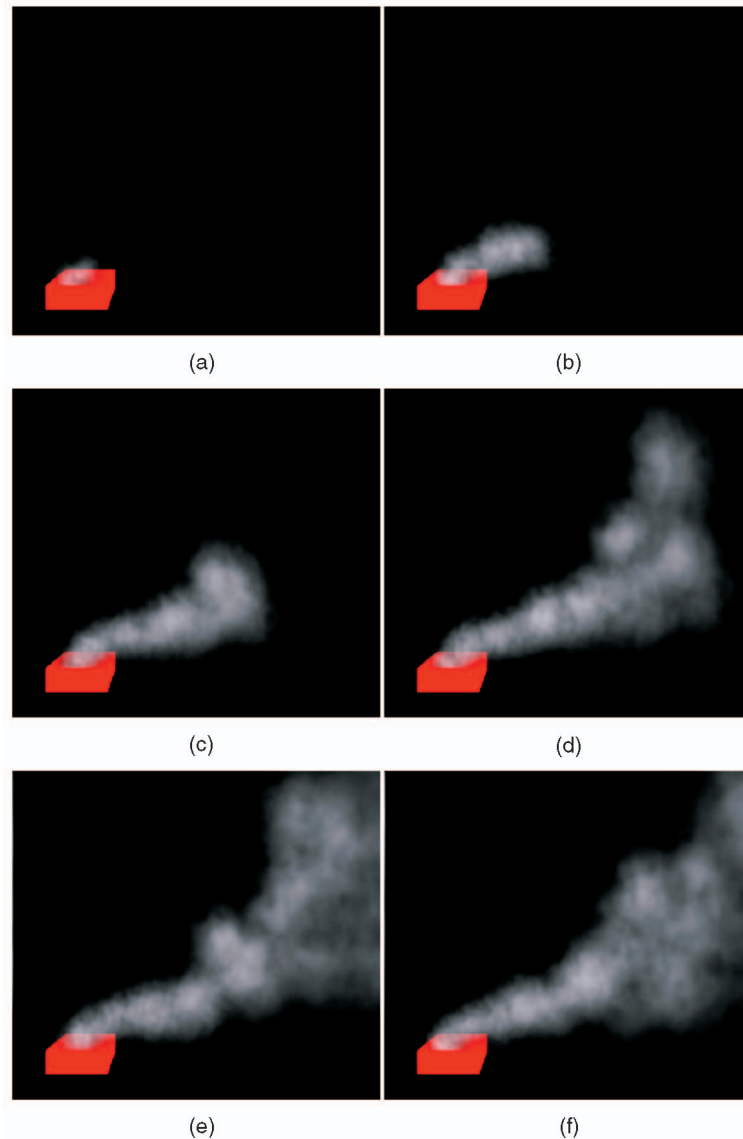| Example	Fig.	Grid Size	Software Calc.	Hardware Calc.	Rendering
Teapot	9	$30 \times 12 \times 30$	70	3.6	15
Chimney	10	$32 \times 32 \times 32$	180	3.6	15
Obstacle	11	$44 \times 20 \times 30$	155	4.1	15
Urban Canyon	12	$58 \times 40 \times 30$	460	9.1	15

Fig. 10. A sequence of images showing smoke emanating from a chimney.

rendering approach is as follows: First, we select a set of textures of turbulence details that match the phenomena we wish to model. These textures can be generated from real images or a noise function (see Fig. 8 for some example textures). The size of the textures we use is 32×32 and our texture database presently includes 32 textures. To ensure proper blending at rendering time, the textures must be weighted with a smooth function, for example, a Gaussian. When a smoke particle enters the flow field, an initial texture is selected from the texture database at random and assigned to the particle. As the particle moves along the velocity vector on the grid, one can either circulate through the texture database and select a different texture for each time step, which adds an additional degree of small-scale turbulence, or one can retain the initially assigned texture for the entire lifetime of the particle. In the former case, the assigned texture series must be somewhat continuous, else disturbing sparkling effects may occur. Each particle is also influenced by the buoyancy force and the longer the particle stays in the field, the dimmer its color, until it eventually disappears. At each time step, all the display primitives are

rendered in a back-to-front order. As the user changes the viewpoint position, the splats are aligned so that they face the user at all times. Alternatively, one can use 3D hypertextures [27] which are sliced according to the viewpoint.

The use of textured splats also allows the efficient modeling of shadows cast by the volumetric gaseous phenomena. First, all regular scene objects are rendered from the view of the light source, with the z-buffer write-enabled. The resulting z-image is saved as a (hard shadow) image. Then, the z-buffer is write-disabled but left test-enabled and the RGBA portion of the frame buffer is set to zero. Now, the textured splats are rendered from the view of the light source. The resulting color-buffer image represents the attenuation of the light due to those gaseous scene objects that are not occluded by opaque scene objects. This image is used as an amorphous shadow texture in the subsequent second rendering pass, now from the view of the camera. Here, we render the polygons of the scene objects, mapping both the amorphous shadow image and
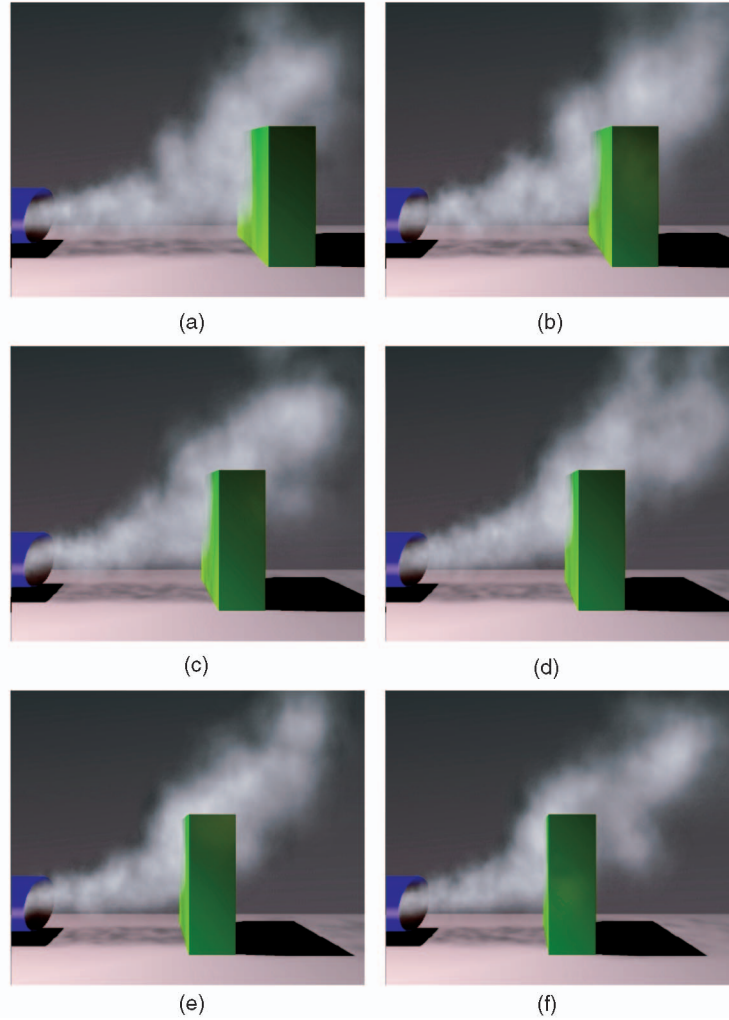
Fig. 11. A sequence of images showing smoke interacting with a green moving obstacle.

the hard shadow image onto the polygons, via projective textures [29].

8   IMPLEMENTATION AND RESULTS

We use a finite volume to represent the gaseous phenomena. The objects in the virtual environment are defined by axis-aligned bounding planes. Different types of boundary conditions are specified on the surface of the grid cells that overlap with the surface of the bounding planes. For the LBM computation, the density and velocity are defined at the center of each grid cell. Each display primitive in the volume has its own temperature and vertical velocity field.

For the initial conditions of the LBM, as mentioned before, the packet distribution f_{qi} is initialized to equal the equilibrium packet distribution value f_{qi}^{eq}, which is known in terms of density and velocity. This approximation may introduce errors into the system at the beginning of the simulation. For this reason, in our work, we discard the first few steps of the simulation.

At the outlet of the model, fluid leaves the grid. One way to achieve this is to assign a density value for the outflow, changing the problem of outlets to that of the density boundary conditions. However, we have found that it is difficult to define the outflow density before the simulation. We instead impose a zero derivative condition after the collision step, which works very well. Suppose the surface $Z = N_z$ is an outlet (where N_z is the number of lattice cells in the Z-direction), for each of the outlet nodes, we execute the equation: $f_{qi}(i, j, N_z - 1) = f_{qi}(i, j, N_z)$.

Our LBM-based system works in the following way:

1.   Inject new display primitives into the system with an initial temperature field and texture (the number of display primitives characterizes the density of the gaseous phenomenon);
2.   Update the velocity vector on the grid points according to the LBM algorithm in Section 3;
3.   Change the temperature of display primitives using (15). If its temperature is lower than the ambient temperature, remove it from the system;
4.   For each display primitive, first calculate its buoyancy force using (16). Then, compute its velocity vector by trilinear interpolation from the velocities stored at the surrounding grid vertices. Based on these forces, move all display primitives to their new locations. If a display primitive moves out of the grid, remove it from the system;
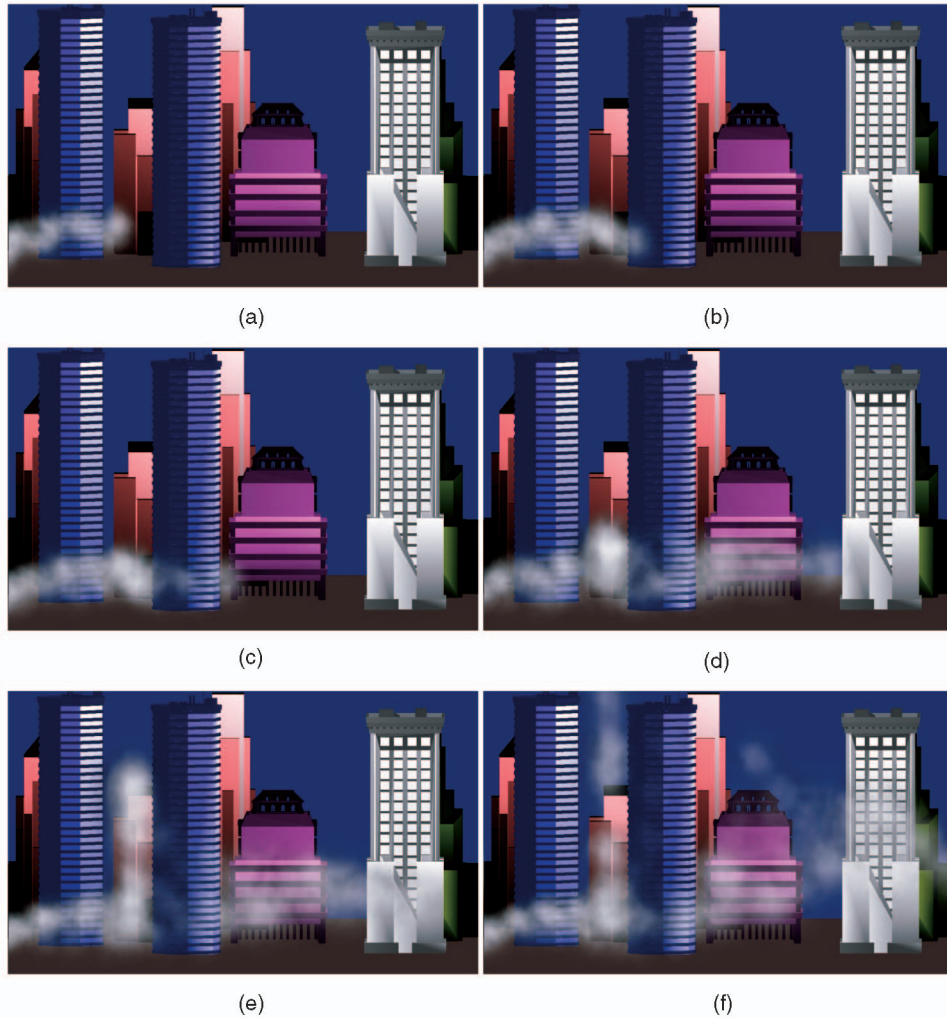
Fig. 12. A sequence of images showing smoke billowing around the buildings in an urban canyon.
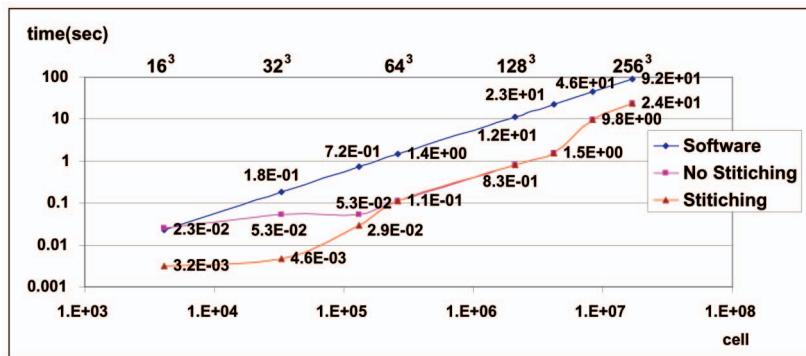

Fig. 13. Time per step of the LBM computation with graphics hardware and software.

5. Move the viewpoint to the light source and precompute the amount of light reaching each display primitive;
6. Render all the display primitives in a back-to-front order using texture splatting;
7. Go back to Step 1.

The power of our method is that we distinguish between the microscopic packets in the LBM and the macroscopic display primitives that we can see. Thus, we don't require a huge amount of display primitives, which allows the rendering to be fast. In our work, a few hundred display primitives are used to generate the results. We present several examples to demonstrate our LBM method. All the results have been generated on a P4 1.6GHz PC with Nvidia GeForce4 Ti 4600 card that has 128MB of memory. For all the following examples, we have succeeded in achieving real-time computation and animation. Table 2 shows the resolution, the calculation time of each step with and without the texture hardware acceleration, and the rendering time of the splatting algorithm for four examples.

Fig. 9 shows the result of hot steam rising up from a teapot and its spout. We model the inlets of the steam as two patches. One is 3×3, the other is 1×1. The steam exits the inlet with a speed of 0.1 and a density value of 0.42. We set the temperature near the inlet of the steam to be $100°$ C and the temperature of air to be $28°$ C.

Fig. 10 is a sequence of images modeling smoke coming out of a chimney. The inlet of the smoke is modeled as a 2×2 patch at the top of the chimney. The left side of the computation grid is modeled as a wind field, while the top and right side of the computation grid are described as outlets. Fig. 11 is another example indicating the effect of a green moving obstacle on the gas behavior. The object moves from the right side of the grid to the left, continuously changing the shape of the gas.

Fig. 12 shows smoke propagating in an urban city model. Several buildings are used as boundary objects. The inlet of the smoke is modeled as a 4×4 patch. The smoke exits the inlet with a speed of 0.07 and a density value of 0.42. The left side of the grid is assigned a speed of 0.1 along the X axis to model the effect of wind. The top and right side of the grid are modeled as open surfaces. The shape of the smoke will change as the wind speed increases or decreases. We set the temperature near the inlet of the smoke to be $40°$ C and the temperature of air to be $28°$ C.

Fig. 13 compares the time (in seconds) per step of the hardware LBM with a software implementation. The statistics does not include the time for rendering. The "Stitching" curve refers to the performance after stitching small textures into larger ones, while "No Stitching" does not. Note that the hardware accelerated technique wins in speed for any size of the model, except that for the 16^3 grid, the "No Stitching" method is not faster than software. This is because, in "No Stitching," the overhead of switching between textures becomes a bottleneck. However, simply by stitching 16 16×16 textures into four 64×16 textures gains a speedup factor of 7. Fig. 13 is in logarithmic scale for both axes. Note that stitching is very effective for grids smaller than 64^3 and the simulation can proceed more than 200 steps per second.

9 CONCLUSIONS

The two primary contributions of this paper are:

- Introducing the LBM as a solution to the computer graphics problem of animating gaseous phenomena;
- Achieving real-time, physically-based, realistic animation of fluid with turbulent effects in complex environments.

Our approach has the following advantages:

- Both the gaseous phenomenon evolution by the LBM and the temperature updates require only an efficient, linear model for the lattice computations. This allows the modeling to be achieved at near-real-time frame rates.
- Since the computations on the grid cells can be performed on commodity texture hardware, an additional acceleration factor of at least 20 can be obtained. This yields the desired real-time modeling speed.

- The modeling of the flow can be performed on an efficient low-resolution grid since high-resolution textures are used to supply the fine details required for realistic appearance. Due to this novel hybrid approach, we can model near physically correct interactions of gaseous phenomena with moving objects at interactive speed, while still attaining high visual details.
- Images can be rendered quickly and realistically by taking advantage of fast, commodity texture hardware.

In the future, we plan to investigate the result of updating textures following a pattern. Sequences of texture splats (we call them "video splats") or even 3D video kernels can be generated offline based on an accurate simulation of the velocity field with a large LBM computation grid. Then, during the rendering part, appropriate video splats can be chosen on the fly to represent and visualize small-scale turbulence and swirls. We believe this approach will improve the fluid animation. We also plan to model the behaviors of objects in the flow, such as the leaves blowing in the wind. Besides gaseous phenomena, our model can also be used to simulate liquid [36], heat in a solid, and the like, and be extended to model fire [37].
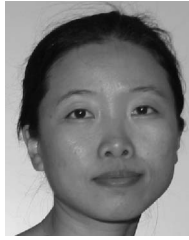
REFERENCES

[1] C. Cercignani, *Mathematical Methods in Kinetic Theory.* Plenum, 1990.
[2] J.X. Chen, N. Da, and V. Lobo, "Toward Interactive-Rate Simulation of Fluids with Moving Using Navier-Stokes Equations," *Graphical Models and Image Processing,* vol. 57, no. 2, pp. 107-116, Mar. 1995.
[3] S. Chen and G.D. Doolean, "Lattice Boltzmann Method for Fluid Flows," *Ann. Rev. Fluid Mechanics,* vol. 30, pp. 329-364, 1998.
[4] N. Chiba, K. Muraoka, H. Takahashi, and M. Miura, "Two-Dimensional Visual Simulation of Flames, Smoke and the Spread of Fire," *J. Visualization and Computer Animation,* vol. 5, pp. 37-53, 1994.
[5] N. Chiba, S. Sanakanishi, K. Yokoyama, and I. Ootawara, "Visual Simulation of Water Currents Using a Particle-Based Bahavioural Model," *J. Visualization and Computer Animation,* vol. 6, pp. 155-171, 1995.
[6] R.A. Crawfis and N. Max, "Texture Splats for 3D Scalar and Vector Field Visualization," *Proc. IEEE Visualization,* pp. 91-98, Oct. 1993.
[7] M. Desbrun and M. Gascue, "Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies," *Proc. Eurographics Workshop Animation and Simulation,* pp. 61-76, 1996.
[8] M. Desbrun and M.-P. Gascue, "Animating Soft Substances with Implicit Surfaces," *Proc. SIGGRAPH,* pp. 287-290, Aug. 1995.
[9] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita, "A Simple, Efficient Method for Realistic Animation of Clouds," *Proc. SIGGRAPH,* pp. 121-128, Aug. 2000.
[10] G.D. Doolen, *Lattice Gas Methods for Partial Differential Equations.* Addison-Wesley, 1990.
[11] D.S. Ebert and R.E. Parent, "Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-Buffer Techniques," *Proc. SIGGRAPH,* vol. 24, no. 4, pp. 357-366, 1990.

[12] S.J. Farlow, *Partial Differential Equations for Scientists and Engineers.* New York: Dover Publications, 1982.

[13] R. Fedkiw, J. Stam, and H. Jensen, "Visual Simulation of Smoke," *Proc. SIGGRAPH,* pp. 129-136, Aug. 2001.

[14] N. Foster and D. Metaxas, "Modeling the Motion of a Hot, Turbulent Gas," *Proc. SIGGRAPH,* pp. 181-188, Aug. 1997.

[15] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-Gas Automata for the Navier-Stokes Equations," *Physical Rev. Letters,* vol. 56, no. 14, pp. 1505-1508, Apr. 1986.

[16] N. Gershenfeld, *The Nature of Mathematical Modeling.* Cambridge Univ. Press, 1999.

[17] J. Hardy, O. de Pazzis, and Y. Pomeau, "Molecular Dynamics of a Classical Lattice Gas: Transport Properties and Time Correlation Functions," *Physics Rev. A,* vol. 13, pp. 1949-1961, 1976.

[18] M. Harris, G. Coombe, T. Scheuermann, and A. Lastra, "Physically-Based Visual Simulation on Graphics Hardware," *Proc. SIGGRAPH/Eurographics Workshop Graphics Hardware,* pp. 1-10, Sept. 2002.

[19] B.D. Kandhai, "Large Scale Lattice-Boltzmann Simulations," PhD thesis, Univ. of Amsterdam, Dec. 1999.

[20] S.A. King, R.A. Crawfis, and W. Reid, "Fast Volume Rendering and Animation of Amorphous Phenomena," *Volume Graphics,* pp. 229-242, 2000.

[21] W. Li, X. Wei, and A. Kaufman, "Implementing Lattice Boltzmann Computation on Graphics Hardware," *The Visual Computer,* to appear.

[22] A. Luciani, A. Habibi, A. Vapillon, and Y. Duroc, "A Physical Model of Turbulent Fluids," *Proc Eurographics Workshop Animation and Simulation,* pp. 16-29, 1995.

[23] R. Mei, L. Luo, and W. Shyy, "An Accurate Curved Boundary Treatment in the Lattice Boltzman Method," *J. Computational Physics,* vol. 155, pp. 307-330, 1999.

[24] G. Miller and A. Pearce, "Globular Dynamics: A Connected System for Animating Viscous Fluids," *Computers and Graphics,* vol. 13, no. 3, pp. 305-309, 1998.

[25] D. Muders, "Three-Dimensional Parallel Lattice Boltzmann Hydrodynamics Simulations of Turbulent Flows in Interstellar Dark Clouds," PhD thesis, Univ. of Bonn, Aug. 1995.

[26] K. Perlin, "An Image Synthesizer," *Proc. SIGGRAPH,* vol. 19, no. 3, pp. 287-296, July 1985.

[27] K. Perlin and E.M. Hoffert, "Hypertexture," *Proc. SIGGRAPH,* vol. 20, no. 3, pp. 253-262, July 1989.

[28] W.T. Reeves, "Particle System—A Technique for Modeling a Class of Fuzzy Objects," *Proc. SIGGRAPH,* vol. 17, no. 3, pp. 359-376, July 1983.

[29] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli, "Fast Shadows and Lighting Effects Using Texture Mapping," *Proc. SIGGRAPH,* pp. 249-252, Aug. 1992.

[30] K. Sims, "Particle Animation and Rendering Using Data Parallel Computation," *Proc. SIGGRAPH,* vol. 24, no. 4, pp. 405-413, Aug. 1990.

[31] J. Stam, "Stable Fluids," *Proc. SIGGRAPH,* pp. 121-128, Aug. 1999.

[32] J. Stam and E. Fiume, "Turbulent Wind Fields for Gaseous Phenomena," *Proc. SIGGRAPH,* pp. 369-376, Aug. 1993.

[33] D. Stora, P.O. Agliati, M.P. Cani, F. Neyret, and J.D. Gascuel, "Animating Lava Flows," *Proc. Graphics Interface,* pp. 203-210, June 1999.

[34] D. Terzopoulos, J. Platt, and K. Fleischer, "Heating and Melting Deformable Models (from Goop to Glop)," *Proc. Graphics Interface,* pp. 219-226, June 1989.

[35] D. Tonnesen, "Modeling Liquids and Solids Using Thermal Particles," *Proc. Graphics Interface,* pp. 255-262, 1991.

[36] X. Wei, W. Li, and A. Kaufman, "Melting and Flowing of Viscous Volumes," *Proc. Computer Animation and Social Agents,* pp. 54-59, May 2003.

[37] X. Wei, W. Li, K. Mueller, and A. Kaufman, "Simulating Fire with Texture Splats," *Proc. IEEE Visualization,* pp. 227-237, Oct. 2002.

[38] J. Wejchert and D. Haumann, "Animation Aerodynamics," *Proc. SIGGRAPH,* vol. 25, no. 4, pp. 19-22, July 1991.

Xiaoming Wei received the BSc degree from Beijing University of Aeronautics and Astronautics of China in 1995 and the MSc degree in computer science from Tsinghua University of China in 1998. She is a PhD candidate in the Computer Science Department at Stony Brook University (SBU), where she is also a research assistant in the SBU Center for Visual Computing (CVC). Her research interests include physically-based modeling, natural phenomena modeling, and computer graphics. She is a student member of the IEEE. For more information see http://www.cs.sunysb.edu/~wxiaomin.

Wei Li received the BE and ME degrees in 1992 and 1995, respectively, both in electrical engineering, from Xi'an Jiaotong University, China, and the master's degree in computer science from Stony Brook University (SBU) in 2000. He is a PhD candidate in the Computer Science Department of the SBU. His current research interests are focused on volume visualization and graphics hardware. For more information see http://www.cs.sunysb.edu/~liwei.

Klaus Mueller earned the BS degree in electrical engineering (University of Ulm, Germany, 1987), the MS degree in biomedical engineering (The Ohio State University, 1990), and the PhD degree in computer science (The Ohio State University 1998). He is currently an assistant professor in the Computer Science Department at Stony Brook University, where he also holds coappointments in the Biomedical Engineering and the Radiology Departments. His current research interests are computer graphics, visualization, augmented reality, and medical imaging. He is a recipient of a US National Science Foundation CAREER award (2000) and has served as a program cochair at various conferences, such as the Volume Graphics Workshop (2001, 2003) and the Symposium on Volume Visualization and Graphics (2002).

Arie E. Kaufman received the BS degree (1969) in mathematics and physics from the Hebrew University of Jerusalem, the MS degree (1973) in computer science from the Weizmann Institute of Science, Rehovot, Israel, and the PhD degree (1977) in computer science from the Ben-Gurion University, Israel. He is the director of the Center for Visual Computing (CVC), a leading professor and chair of the Computer Science Department, and leading professor of radiology at Stony Brook University. He was the founding Editor-in-Chief of the *IEEE Transaction on Visualization and Computer Graphics* (*TVCG*), 1995-1998. He has been the cochair for multiple Eurographics/Siggraph Graphics Hardware Workshops and Volume Graphics Workshops, the papers/program cochair for the ACM Volume Visualization Symposium and the IEEE Visualization Conferences, and the cofounder and a member of the steering committee of the IEEE Visualization Conference series. He has previously chaired and is currently a director of the IEEE Computer Society Technical Committee on Visualization and Computer Graphics. He is an IEEE fellow and the recipient of a 1995 IEEE Outstanding Contribution Award, the 1996 IEEE Computer Society's Golden Core Member, 1998 ACM Service Award, and 1999 IEEE Computer Society's Meritorious Service Award, and 2002 State of New York Entrepreneur Award. He has conducted research and consulted for more than 30 years specializing in volume visualization, graphics architectures, algorithms, and languages, virtual reality, user interfaces, and multimedia. For more information see http://www.cs.sunysb.edu/~ari.