Visual Comput (2007)
DOI 10.1007/s00371-007-0191-y

ORIGINAL ARTICLE

Ye Zhao

Lattice Boltzmann based PDE solver on the GPU

Y. Zhao (✉)
Kent State University,
Department of Computer Science,
Kent, OH 44242, USA
zhao@cs.kent.edu

Abstract In this paper, we propose a hardware-accelerated PDE (partial differential equation) solver based on the lattice Boltzmann model (LBM). The LBM is initially designed to solve fluid dynamics by constructing simplified microscopic kinetic models. As an explicit numerical scheme with only local operations, it has the advantage of being easy to implement and especially suitable for graphics hardware (GPU) acceleration. Beyond the Navier–Stokes equation of fluid mechanics, a typical LBM can be modified to solve the parabolic diffusion equation, which is further used to solve the elliptic Laplace and Poisson equations with a diffusion process. These PDEs are widely used in modeling and manipulating images, surfaces and volumetric data sets. Therefore, the LBM scheme can be used as an GPU-based numerical solver to provide a fast and convenient alternative to traditional implicit iterative solvers. We apply this method to several examples in volume smoothing, surface fairing and image editing, achieving outstanding performance on contemporary graphics hardware. It has the great potential to be used as a general GPU computing framework for efficiently solving PDEs in image processing, computer graphics and visualization.

Keywords Lattice Boltzmann model · Diffusion · Laplace and Poisson equation · Volume smoothing · Surface fairing · Image editing

1 Introduction

A variety of applications require solving partial differential equations (PDEs) to model, manipulate and visualize images, surfaces and volumes, such as Laplace equation in image denoising and surface fairing [8], Poisson equation in image editing [25] and mesh editing [8], Navier–Stokes equations in fluid simulation [30], etc. Numerical simulation of the PDEs usually requires high-intensity computation and large consumption of computational resources.

With the increasing high performance and programmability of the contemporary graphics processing units (GPUs), these commodity chips have been used in many numerical computations other than specific graphics applications for which they were designed [24]. The most attractive feature of the GPU is its inherent parallelism, which comes from multiple SIMD processing units. As a streaming processing engine, the GPU is ideally suited for explicit, local and lattice-based computations, since its computation kernel, which represents the lattice update rule, uses local data from the input stream to generate the output data stream.

Most previous work uses implicit schemes to discretize the PDEs that lead to solving a sparse linear system. Many numerical methods, such as Jacobi, Gaussian–Seidel, conjugate gradient, and multigrid, have been applied. However, these iterative methods cannot be directly mapped to the GPU. Many researchers endeavor to overcome this problem. Several ingenious methods [4, 12, 13, 17, 28] have been proposed to map unstructured sparse matrix solvers on the GPU and achieve good performance. They generally require deliberately designed algorithms of sparse data compression, indirect indexing and optimized matrix vector multiplications for handling matrix computations on the GPU.

Explicit finite difference methods were regarded as too expensive, due to its small time steps and high resolution of discrete grid, to compete with well-developed implicit solvers. The dramatic growth of the computing power is now changing this [5]. The explicit methods are considered again with their ease of use and gentle learning curve. This is even more true as the high-performance GPU, the commodity parallel machine, becomes available to numerical computations.

We propose an explicit approach that is consistent with GPU's local and parallel computational scheme; therefore, it can be naturally implemented on the GPU. Our solver is based on the lattice Boltzmann model (LBM), which is a relatively new approach in computational fluid dynamics (CFD), with a simple and parallelizable grid-based numerical scheme [31]. Using the LBM, one can construct simplified kinetic models that incorporate the essential physics of microscopic processes such that the macroscopic averaged properties obey the desired macroscopic Navier–Stokes equations. It has the advantage of being easy to implement and, in addition, it is especially suitable for GPU acceleration. The LBM has achieved success in the world of computational physics and has also been used in graphics and visualization for simulating a variety of fluid phenomena with complex boundary conditions [7, 33, 36, 39, 40]. It has also been accelerated on a single GPU [19] or a GPU cluster [10] with MPI. Besides GPUs, LUDWIG [9], a parallel LBM code for fluids, uses MPI to achieve full portability and good efficiency on both massively parallel processors (MPP) and symmetric multiprocessing (SMP) systems. With OpenMP [1], the LBM has been optimized to implement on multiple CPUs with shared-memory parallel programming [2, 21].

Beyond the fluid dynamics solver as it was initially designed, the LBM has been used to model diffusion process [34, 38] with a simplified collision function. This is quite straightforward since diffusion is actually one term of the Navier–Stokes equations. This LBM-based diffusion has been used in image processing [15], where anisotropic 2D image denoising is implemented on the CPU. In this paper, our main contribution is to provide a hardware-accelerated PDE solver with easy code implementation and straightforward GPU mapping, achieving excellent computation speed for large data sets. We extend the LBM scheme that simulates the diffusion process to 3D computations on the GPU, and become the first to use it as a numerical solver for Laplace and Poisson equations, which are widely used in image, volume and surface modeling and manipulation. We use this framework on image editing, volume smoothing and surface fairing to show the benefits. The method has the great potential to be further extended to various applications in computer vision, graphics and visualization.

2 LBM-based solver

LBM models Boltzmann particle dynamics on a 2D or 3D lattice [31]. It is a microscopically inspired method designed to solve macroscopic fluid dynamics problems. It lives at the interface between the microscopic (molecular) and macroscopic (continuum) worlds, hopefully capturing the best of the two. The Boltzmann equation expresses the variation of the average number of microscopic flow particles moving with a given velocity between each pair of neighboring sites. Such variation is caused by inter-particle interactions and ballistic motion of the particles. The variables associated with each lattice site are the particle distributions that represent the probability of particle presence with a given velocity. Particles stream synchronously along links from each site to its neighbors in discrete time steps and perform collision between consecutive streaming steps. The LBM is second-order accurate both in time and space, and in the limit of zero time step and lattice spacing, it yields the Navier–Stokes equations for an incompressible fluid [14].

2.1 LBM computation

Figure 1a shows a typical D2Q9 LBM lattice structure, in which each cell in a 2D lattice has 8 links with its neighbors and the cell itself. In a 3D lattice, each cell has at most 27 links with its immediate neighbors. Figure 1b shows a typical D3Q19 (3D with 18 links to neighbors and the cell itself) LBM lattice structure. Each link has its velocity vector $\boldsymbol{e}_i(\boldsymbol{x}, t)$ and the particle distribution $f_i(\boldsymbol{x}, t)$ that

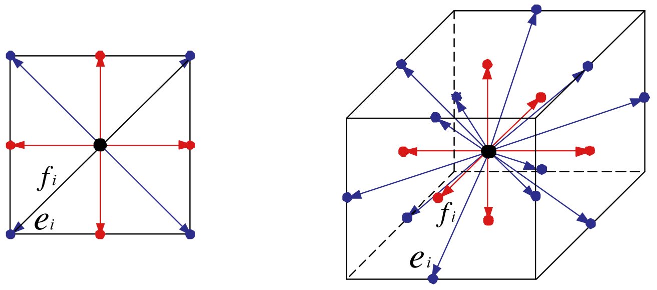

Fig. 1a,b. D2Q9 and D3Q19 LBM lattice. $\boldsymbol{e}_i$ is the velocity vector and f_i is the particle distribution moving along the vector. **a** D2Q9: 9 links for a cell in a 2D lattice including 4 axial links, 4 diagonal links and the zero link (itself); **b** D3Q19: 19 links for a cell in a 3D lattice including 6 axial links, 12 minor-diagonal links and the zero link (itself) moves along this link, where $\boldsymbol{x}$ is the position of the cell and t is the time. The macroscopic fluid density $\rho(\boldsymbol{x}, t)$, and velocity $\boldsymbol{u}(\boldsymbol{x}, t)$, are computed from the particle distributions as

$$\rho = \sum_i f_i, \quad \boldsymbol{u} = \frac{1}{\rho} \sum_i f_i \boldsymbol{e}_i. \tag{1}$$

The collision-streaming process of the LBM can be represented as

$$f_i(\boldsymbol{x}+\boldsymbol{e}_i, t+1) - f_i(\boldsymbol{x}, t) = -\frac{1}{\tau}\big(f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\big), \tag{2}$$

where the local equilibrium particle distribution, f_i^{eq}, is usually given by the Bhatnager, Gross, Krook (BGK) model [3] to model collisions as a statistical redistribution of momentum which locally drives the system toward equilibrium while conserving mass and momentum:

$$f_i^{\mathrm{eq}}(\rho, \boldsymbol{u}) = \rho\big(A_i + B_i(\boldsymbol{e}_i \cdot \boldsymbol{u}) + C_i(\boldsymbol{e}_i \cdot \boldsymbol{u})^2 + D_i \boldsymbol{u}^2\big). \tag{3}$$

Here, A_i to D_i are constant scalar coefficients specific to the chosen lattice geometry. The constant τ represents the relaxation time determining the kinematic viscosity ν of the fluid by

$$\nu = \frac{1}{3}\left(\tau - \frac{1}{2}\right). \tag{4}$$

The equilibrium distribution depends only on conserved quantities – mass ρ and momentum $\rho\boldsymbol{u}$. The calculation of Eqs. 1 and 2 requires only local and neighboring properties. Therefore, all the LBM computations can be efficiently accelerated on the GPU.

Body forces can be added to the LBM as an external input to control the flow behavior [6]. It is included in the LBM after the collision computation as

$$f_i \longleftarrow f_i + \frac{(2\tau-1)}{2\tau} B_i(\boldsymbol{G} \cdot \boldsymbol{e}_i), \tag{5}$$

where $\boldsymbol{G}$ is the body force.

The LBM is a numerical technique from a perspective of statistical physics. The incompressible Navier–Stokes equations:

$$\nabla \cdot \boldsymbol{u} = 0, \tag{6}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = \nu \Delta \boldsymbol{u} + \boldsymbol{F}. \tag{7}$$

can be derived through the Chapman–Enskog analysis, which is a procedure to solve the Boltzmann equation by means of perturbation technique [14]. Here ∇ defines the gradient operator $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$ and Δ is the Laplacian $\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$. $\boldsymbol{F}$ represents the combined effects of pressure, $-\nabla P$, and external body force, $\boldsymbol{G}$.

2.2 LBM for diffusion

LBM scheme can be used for diffusion computation. The equilibrium function (Eq. 3) in the traditional LBM is modified by removing momentum dependency as

$$f_i^{\mathrm{eq}}(\rho) = A_i \rho. \tag{8}$$

Next, we show how the modified LBM scheme recovers the macroscopic parabolic diffusion equation.

The Chapman–Enskog analysis is essentially a formal multi-scaling expansion:

$$\begin{aligned} f_i &= f_i^0 + \varepsilon f_i^1 + \varepsilon^2 f_i^2 + \cdots, \\ \frac{\partial}{\partial t} &= \frac{\partial}{\partial t_0} + \varepsilon \frac{\partial}{\partial t_1} + \varepsilon^2 \frac{\partial}{\partial t_2} + \cdots, \\ \frac{\partial}{\partial x} &= \frac{\partial}{\partial x_0} + \varepsilon \frac{\partial}{\partial x_1} + \cdots, \end{aligned} \tag{9}$$

where the small expansion parameter ε is the Knudsen number, defined as the ratio of the molecular mean free path length to a representative macroscopic length scale. First, $f_i(\boldsymbol{x}+\boldsymbol{e}_i, t+1)$ in Eq. 2 can be Taylor expanded on two variables t and $\boldsymbol{x}$. Second, $f_i(\boldsymbol{x}, t)$ is rewritten following the Chapman–Enskog expansion. Thus, comparing both sides of the resulting equation in the consecutive order of the parameter ε, we get the following:

$$f_i^0 = f_i^{\mathrm{eq}} \;\;:\;\; O(\varepsilon^0) \tag{10}$$

$$\left(\frac{\partial}{\partial t_0} + \boldsymbol{e}_i \cdot \nabla\right) f_i^0 = -\frac{1}{\tau} f_i^1 \;\;:\;\; O(\varepsilon^1) \tag{11}$$

$$\frac{\partial f_i^0}{\partial t_1} + \left(\frac{2\tau-1}{2\tau}\right)\left(\frac{\partial}{\partial t_0} + \boldsymbol{e}_i \cdot \nabla\right) f_i^1 = -\frac{1}{\tau} f_i^2 \;\;:\;\; O(\varepsilon^2). \tag{12}$$

The distribution function f_i is constrained by

$$\sum_i f_i^0 = \rho, \tag{13}$$

$$\sum_i f_i^k = 0, \quad k = 1 \; or \; 2. \tag{14}$$

Summing Eq. 12 over i yields

$$\sum_i \left(\frac{\partial f_i^0}{\partial t_1}\right) + \left(\frac{2\tau-1}{2\tau}\right) \sum_i (\boldsymbol{e}_i \cdot \nabla) f_i^1 = 0, \tag{15}$$

where we use the constraints in Eq. 14. Combining Eqs. 15 and 11, and using the constraints and Eq. 8, we obtain

$$\frac{\partial \rho}{\partial t} + (\nabla_\alpha \cdot \Pi_{\alpha\beta} \cdot \nabla_\beta)\rho = 0, \tag{16}$$

where α and β represent coordinate directions, and $\Pi_{\alpha\beta}$ is the diffusion tensor

$$\Pi_{\alpha\beta} = \sum_i \left(\left(\frac{1-2\tau}{2} \right) \boldsymbol{e}_{i\alpha} \boldsymbol{e}_{i\beta} A_i \right). \tag{17}$$

If τ is a constant, this diffusion tensor is defined by the lattice structure. Then, we recover the parabolic diffusion equation

$$\frac{\partial \rho}{\partial t} = \nu \Delta \rho, \tag{18}$$

where ν now represents the diffusion coefficient.

In D2Q9 model, $A_i = \frac{4}{9}$ for the zero link, $A_i = \frac{1}{9}$ for the axial links and $A_i = \frac{1}{36}$ for the diagonal links. Therefore, the diffusion coefficient

$$\nu = \frac{2}{9}(2\tau - 1). \tag{19}$$

In D3Q19 model, $A_i = \frac{1}{3}$ for the zero link, $A_i = \frac{1}{18}$ for the axial links and $A_i = \frac{1}{36}$ for the minor-diagonal links. Therefore, the diffusion coefficient

$$\nu = \frac{4}{9}(2\tau - 1). \tag{20}$$

When using the LBM to simulate the Navier–Stokes equations for solving fluid dynamics, it is conditionally stable as an explicit solver [31], provided the relaxation parameter τ obeys the inequality

$$|1 - 1/\tau| < 1. \tag{21}$$

In diffusion simulation, by using the diffusion coefficients defined in Eqs. 19 and 20 for the D2Q9 and D3Q19 lattice, we simply obtain

$$0 < \nu < \infty \tag{22}$$

from this inequality. This shows that theoretically the diffusion coefficient, ν, can be chosen from any value larger than zero. Thus, our diffusion LBM computation is very easy to achieve stability. In practice, we are able to choose a wide range of diffusion coefficients used for various applications.

2.3 Laplace and Poisson equations

Laplace equation is widely used in graphical applications:

$$\Delta X = 0, \tag{23}$$

where X represents an image, a polygon mesh or a 3D volume. This elliptic PDE can be numerically solved by a time-dependent diffusion process:

$$\frac{\partial X}{\partial t} = \nu \Delta X, \tag{24}$$

since if $\frac{\partial X}{\partial t} \to 0$, then $\Delta X \to 0$. The diffusion solution approximates the solution of Laplace equation. Thus the LBM scheme can be easily used to solve Laplace equation.

If the right-hand side of Eq. 23 is specified as a given function, h, then we get another widely used Poisson equation

$$\Delta X = h. \tag{25}$$

In this case, our LBM solver need to solve

$$\frac{\partial X}{\partial t} = \nu \Delta X - h. \tag{26}$$

Comparing this equation with Eq. 7, one can incorporate the function h into the LBM diffusion scheme as an external force term (Eq. 5). Note that the advection term, $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$, in the Navier–Stokes equations has been removed by modifying the equilibrium function (Eq. 8). Thus, if $\frac{\partial X}{\partial t} \to 0$, the diffusion LBM solution approaches the solution of Poisson equation.

3 GPU acceleration

There are three powerful reasons to employ the GPU for non-graphics computations: speed, cost, and its anticipated evolution. We have shown that the LBM computation scheme can be used to solve the diffusion, Laplace and Poisson PDEs. The key for the success of our method is to achieve fast speed by implementing the non-graphics computation on modern GPUs.

Mapping LBM computation onto the GPU is quite straightforward. After packing packet distributions in 19 links (3D) or nine links (2D) into four-channel textures, the collision and streaming (Eq. 2) can be performed with only local neighboring operations. We extend the GPU implementation by Li et al. [19] for the LBM computation. In our implementation, the major difference between the diffusion LBM and the typical LBM for fluids is the simplified equilibrium distribution function (Eq. 8). This leads to a simpler Cg program of the equilibrium computation in the collision step. In our experiments, the diffusion LBM runs faster than the typical LBM computation because of this simplification.

Besides the modification of equilibrium computation, the diffusion LBM method initializes the density values from particular data sets: original images or volumes, while a typical LBM usually sets the initial values as $\rho = 1.0$ and $\boldsymbol{u} = (0, 0, 0)$. For example, each voxel density of a volume is scaled to the range of [0..1], and we use *glTexSubImage* to write the values to a density texture for LBM computation.

In the typical D3Q19 implementation [19], 19 f_is of each lattice site are packed into 5 packet distribution textures. The velocity components $\boldsymbol{u}_x$, $\boldsymbol{u}_y$, $\boldsymbol{u}_z$ and the density ρ are stored in the R,G,B and alpha channels of a density–velocity texture, respectively. In total, six textures are used. For our modified LBM scheme, the fluid velocity is useless and no longer computed in order to improve the speed. For 3D models, we keep the density–velocity texture, while only the alpha channel is used for storing the density. Although we could pack the density together with f_is to use only five textures totally, this requires more complicated Cg programs. Because it involves read–write operations on the same texture at the same time when computing densities by Eq. 1, which is not well supported on current GPUs. In our experiment, this method is not as efficient as keeping six textures as before.

In 2D image processing for color images, we use the three channels in the density texture to store the density values of red, green and blue components of image pixels, respectively. Then, our GPU program handles the three components of a color image independently and simultaneously, which improves the performance. For the nine packet distributions of the three color components, we use three textures for each component. In Cg programs, we assign a four-tuple parameter, *nChannel*, to choose the red, green or blue density from each texel, $den = (d_{\text{red}}, d_{\text{green}}, d_{\text{blue}}, 0)$, in the density texture. For example, red density is acquired by dot product as $d_{\text{red}} = dot(nChannel, den)$ with $nChannel = (1, 0, 0, 0)$.

Our LBM-based solver on the GPU is mapped directly from the CPU code. It is easy for a graduate student to understand and duplicate in a short time. We have achieved outstanding performance, which is reported and discussed in Sect. 5.

4 Applications

We have shown that the GPU-accelerated LBM method can be used to solve the diffusion, Laplace and Poisson PDEs. These equations play very important role in 2D and 3D modeling and visualization. Next, we apply this method to several applications.

4.1 Volume smoothing

Volumetric data sets used in many visualization applications are usually acquired from scan devices, such as CT, MRI or ultrasound. These devices introduce noise during the scanning process, which greatly affects the quality of subsequent processing and visualization. Convolution-based filtering [35] was used to compute the weighted-average in a local neighborhood to smooth the noised data for volume rendering. A feature-preserving filter [23] minimized a global error function for smoothing volume data sets globally. Distance fields based methods [11, 37] were also used in volume smoothing. Rodgman and Chen [27] discussed a number of methods for denoising volume dataset. They compared Gaussian filter and diffusion-based smoothing methods with a well-designed quality metric.

Volume smoothing can be implemented by a diffusion process that filters binary or gray scale volumes, making all the iso-surfaces smoother at the same time. Equation 24 describes an isotropic volume smoothing procedure, if we assign X to represent the voxel densities of a volumetric data set.

Figure 2 shows the results of our LBM smoothing on a $64 \times 64 \times 64$ volumetric cube data set, whose density value varies from 0 to 1.0. All the results are rendered with OpenGL after applying the Marching Cubes algorithm [20] to generate iso-surfaces. In the following, we use the marching cubes method and OpenGL for rendering if no particular methods are mentioned. Figure 2a is the original noised cube. Noise is added to 30% of all the voxels with a random density variation (up to 10% of its origin value). Figure 2b–d are the smoothed results of the cube after ten LBM steps, where different iso-values (0.2, 0.35 and 0.5) are used to show that our simulation smoothes the whole volume data set.

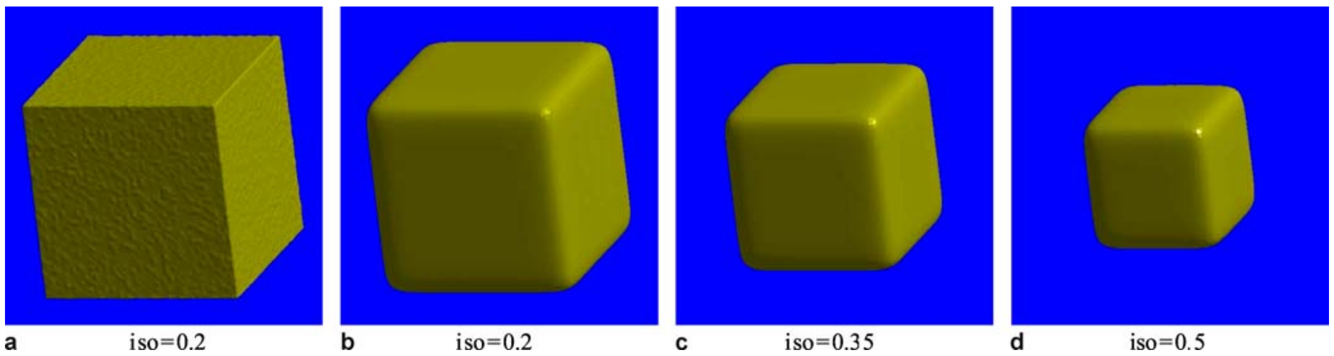

Fig. 2a–d. Results of smoothing a noised cube, rendered by OpenGL after applying the Marching-Cubes method to generate iso-surfaces. **a** Original volumetric cube with noise; **b–d** Smoothed results with different iso-values after 10 LBM steps

Our LBM-based scheme handles anisotropic diffusion. In Eq. 17, if the relaxation coefficient τ is not a constant but a function of the normal and curvature, then Eq. 16 actually represents the anisotropic diffusion equation

$$\frac{\partial X}{\partial t} = \nabla \cdot (D \nabla X), \tag{27}$$

where D represents the anisotropic diffusion tensor. We set τ as a function of the curvature κ for our volume smoothing examples,

$$\tau = (1 + C/(1+\kappa))/2, \tag{28}$$

to enable feature preserving smoothing, where C is a constant parameter determining the diffusion coefficient and controlling the diffusion behavior. The curvature referred here is the mean curvature and is computed on the GPU by the method proposed in [18]. In Fig. 3, we compare the isotropic diffusion with $\kappa = 0$ (Fig. 3a) and anisotropic diffusion results (Fig. 3b) of the cube after ten LBM steps, with $C = 10$.

Figure 4 shows the result of smoothing a heavily-noised vase in eight LBM simulation steps with $C = 6$. All the voxels are degraded by up to 30% of their density values. It illustrates that our method can handle heavily noised volume data sets very well.

Figure 5 shows the rendering result of smoothing a noised foot with 14 LBM diffusion steps. Instead of the Marching Cubes method, we use the ray casting method for direct volume rendering. All the voxels are degraded by up to 15% of their density values. The same transfer functions are used to render the smoothed volume and the noised volume for comparison.

4.2 Surface fairing

Surface fairing methods are important to improve the mesh quality by removing undesired noises or rough features. Most techniques propose energy functions defined by the normal or curvature properties, and perform the constrained energy minimization to achieve the surface smoothing. Such minimization results in solving a Laplace equation of the mesh. An explicit integration method [32]

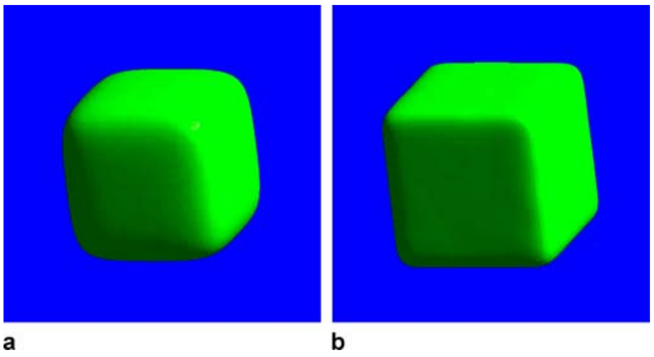
Fig. 3a,b. Results of smoothing a noised cube **a** with isotropic diffusion; **b** with anisotropic diffusion

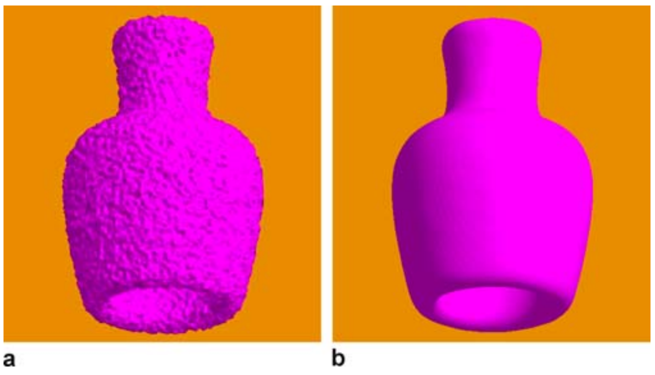
Fig. 4a,b. Result of smoothing a heavily-noised volumetric vase **a** noised vase **b** smoothed vase

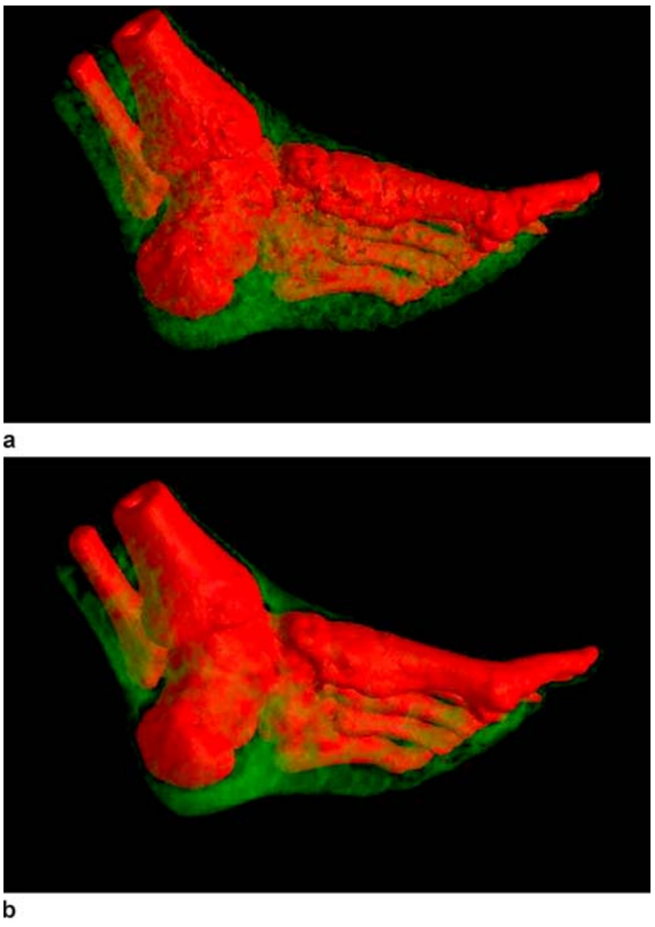
Fig. 5a,b. Volume rendering result of smoothing a noised volumetric foot **a** noised foot **b** smoothed foot

was used with linear time and memory consumption. This method did not perform well for very large mesh. Desbrun et al. [8] proposed implicit fairing that used bi-conjugated gradient to solve the linear system. The method improved a simple umbrella operator of Laplacian on the mesh by adopting a scale-dependent umbrella operator and its linear approximation. They also proposed a curvature flow to remove noise with an approximation of local curvature normal on the mesh. The methods rely on the mesh quality for their effectiveness and correctness due to the mesh-based PDE discretization and approximation.

Museth et al. [22] applied a level set method to surface editing, where a deformable implicit representation based on the distance field of a surface was controlled to implement various editing operations. Level set methods provide another direction for surface manipulation, where the computation is performed on volumetric representations to achieve more accurate results.

We utilize this strategy by first executing a mesh voxelization [29] to generate a volume data set of a surface model, and then applying our volume smoothing method directly. The voxelization is actually implemented by assigning density values for the lattice sites in the volume, according to the distance to the original mesh, which is similar, though not identical, to the definition of the level set function. After such preprocessing, we could apply our GPU-based explicit scheme with fast computation speed to surface fairing.

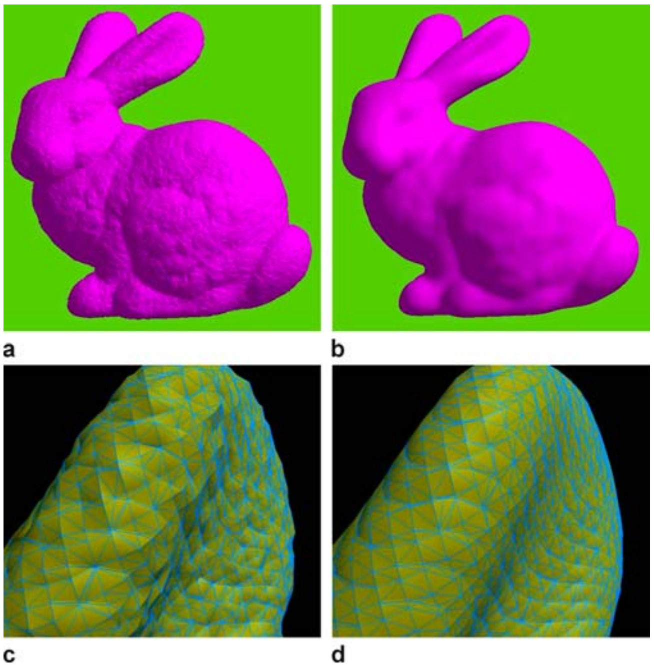
Fig. 6a–d. Results of smoothing a noised bunny **a** bunny model with noise **b** smoothed bunny **c** noised mesh **d** smoothed mesh

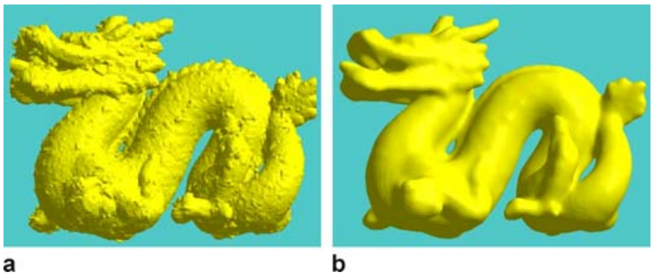
Fig. 7a,b. Result of smoothing a noised dragon **a** dragon model with noise **b** smoothed dragon

Figure 6 shows the results of smoothing a noised bunny model. The bunny is voxelized to generate a $128 \times 128 \times 128$ volumetric model. Then we apply the diffusion LBM to smooth the volume by setting control parameter $C = 1$. After four LBM steps, the noises are removed and the reconstructed surface by the Marching Cubes method is drawn to show the result. Figure 6a is the noised bunny and Fig. 6c is the noised triangle mesh on the ear of the bunny. Figure 6b shows the smoothed bunny and Fig. 6d is the smoothed triangle mesh on the ear of the bunny. Furthermore, Fig. 7 shows the result of smoothing a noised dragon model after six LBM steps, using a $128 \times 128 \times 128$ volume.

A surface is usually voxelized to a band region surrounding the surface in a 3D volume. Therefore, the smoothing can be completed in only several LBM simulation steps. This method also works for point-based shape representation. Currently, we perform our GPU LBM computations on the whole 3D volume and achieve very good performance, where each step is accomplished in around 20 milliseconds for a $128 \times 128 \times 128$ volume on a high-end GPU (see Table 1).

In pre-processing, the voxelization on CPU costs about 8.61 seconds to generate a 128^3 volume for the bunny model (69 451 triangles), and 6.18 seconds for the dragon model (47 794 triangles). In comparison, the implicit fairing method [8] used a preconditioned bi-conjugate gradient to iteratively solve a linear system created directly from the polygonal mesh. This implicit method achieved 2.98 to 18.82 seconds, depending on the time step size, for the dragon and 4.53 to 21.34 seconds for the bunny. This implies that the voxelization cost may compromise our speed achievement of the LBM fairing on GPU, which runs at about 19.6 milliseconds (see Table 1).

In the future, we will implement the voxelization algorithm on the GPU. Moreover, we will adapt the LBM scheme to directly manipulate polygonal meshes through the operations on distance field. Furthermore, we will implement an “intelligent” LBM algorithm on the GPU, similar to the narrow band algorithm of the level set method, which executes the operations only in the volumetric neighborhood of the original surface by indirectly packing them into textures. Thus, the voxelization will

Table 1. Performance results: Per LBM step simulation speed (in milliseconds) for the CPU and GPU, the GPU/CPU speedup factor, and the texture memory size (in MByte)

Applications	Examples	LBM lattice size	CPU speed (millisec)	GPU speed (millisec)	GPU speedup factor	Texture memory used (MByte)
Volume smoothing	Volumetric cube	$64 \times 64 \times 64$	406	3.1	131	50.3
Volume smoothing	Volumetric foot	$128 \times 128 \times 128$	3734	21.1	177	402.7
Volume smoothing	Volumetric vase	$115 \times 101 \times 75$	1390	8.6	161	167.3
Surface fairing	Voxelized bunny	$128 \times 128 \times 128$	3610	19.6	184	402.7
Surface fairing	Voxelized dragon	$128 \times 128 \times 128$	3609	19.6	184	402.7
Image denoising	Blood cell image	432×386	63	2.5	25	32
Poisson image editing	Sky-bird image	230×230	86	0.8	107	10.1

only need to be performed in the neighborhood region. This will decrease the consumption of computational resources, however, increase the complexity and computation time for hardware implementation.

4.3 Image editing

Diffusion has been extensively used in image processing [26]. LBM has been used in the image diffusion with a CPU implementation [15]. We extend the 2D diffusion LBM to the implementation on the GPU. Figure 8 illustrates the smoothing results of a noised blood cell image with isotropic and anisotropic diffusion. Figure 8a is a blood cell image with 20% degraded pixels. Figure 8b shows the smoothing result with isotropic diffusion with a diffusion coefficient $\tau = 1$. Following the work in [15, 26], Fig. 8c illustrates the anisotropic smoothing results after fourteen LBM steps, by setting the relaxation time τ as

$$\tau = 1 + S/(1 + |\nabla G_\sigma * X|), \tag{29}$$

where G_σ is the Gaussian filter. X here is the image intensity and $S = 6$ is a constant.

Besides diffusion, we have been the first to extend the LBM computation to solve Poisson equation on the GPU. Poisson equation with Dirichlet boundary condition has been used as a generic framework for seamless cloning and other image editing tasks [16, 25]. In Sect. 2.3, we have shown that our algorithm implements Poisson solver by incorporating a pseudo external force term in the diffusion LBM computations. Therefore, our method can be used for fast computation in the applications requiring a Poisson solver. Actually, this utilizes one of the advantages of the LBM method as a fluid solver: it does not need to iteratively solve the large linear system produced by the Poisson equation of the fluid pressure, which is solved by its microscopic computation process instead.

We implement the seamless image cloning as an example [25]. The following Poisson equation is solved by our GPU-accelerated LBM method:

$$\Delta p = \text{div}\, \boldsymbol{v} \quad \mathit{over}\ \Omega, \tag{30}$$

where *div* represents the diverge operator, p is the resulting image on a region of interest, Ω, and $\boldsymbol{v} = \nabla g$ is a given guidance field computed from the source image g. The

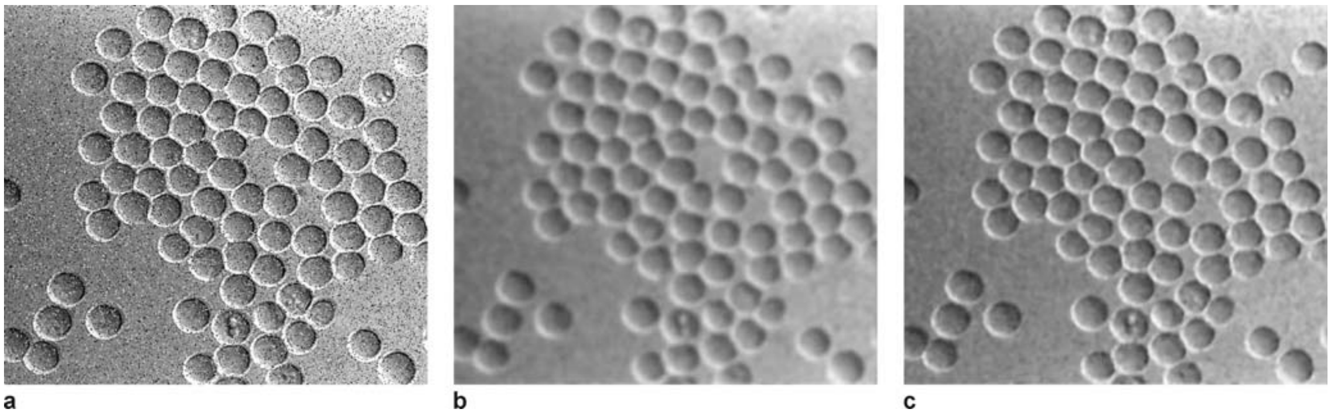

Fig. 8a–c. Image denoising results of a blood cell image **a** blood cells with noise **b** isotropic smoothing result **c** anisotropic smoothing result

Dirichlet boundary condition is defined on the boundary, $\partial\Omega$, as

$$p|_{\partial\Omega} = p^*|_{\partial\Omega}, \qquad (31)$$

where p^* is the target image.

In our GPU implementation, the rectangular bounding box of the region of interest is used to define the texture size of the LBM simulation. We also store a "status marker" for each texel to identify whether it is inside the region of interest, Ω. The texels belong to Ω have an *active* status and their initial values are set as 0. The other texels in the texture, including the boundary, $\partial\Omega$, have an *inactive* status and their initial values are set by the target image, p^*. In Cg programs, the collision and streaming operations modify the values of the whole texture following the LBM equation. After this, we restore the original values of the *inactive* texels. This operation implements the Dirichlet boundary condition on the GPU.

Moreover, a pre-computed vector field, $\boldsymbol{v}$, initializes a body force texture, which is used after collision computation by Eq. 5. For color images, there are three body force textures and each LBM computation step is performed to the red, green and blue components independently.

Figure 9 illustrates our image editing results after 350 LBM steps. It requires more simulation steps than smoothing because of the color propagation procedure from the Dirichlet boundary to the whole region of interest, which is selected here by a simple circle instead of a lasso selection. The editing operation can be finished in around 1 second in a high-end GPU (Table 1). Figure 9a and b are a sky image (target) and a bird image (source), respectively. Figure 9c shows the result of a simple cut-and-paste operation that moves several birds to the sky (from the source image to the target image). Figure 9d is the image cloning result of our LBM-based Poisson solver. The cloning operation can be further improved by optimizing the boundaries [16], which is not included in this example.

In the pioneer work from Perez et al. [25], it costs 0.4 s to iteratively solve a linear system by the V-cycle multigrid method on CPU for a disk-shaped region of 60 000 pixels. In our simulation, we spend about 32 s for a similar disk-shaped region on CPU. This shows that the explicit LBM method requires more iteration steps for strong diffusion in image editing. By leveraging the benefits of parallel computing, our method achieves the speed at about 0.33 s on GPU for all three color channels, which can catch up with the multigrid simulation speed.

5 Performance

We have shown that our LBM-based PDE solver can be used in many applications in computer graphics, image processing and visualization. The key feature of our method is its easy implementation and fast computation speed by using the contemporary GPUs.

Implicit PDE solvers that are commonly used in computer graphics usually involve complex iterative computations with conjugate-gradient, multi-grid or other advanced numerical methods. Mapping these computations onto graphics hardware requires specific "hacking" techniques to adapt the methods to efficiently use texture memory and GPU computing pipeline. In contrast, the LBM is very easy for programming following the simple collision-streaming process. With only a few lines (< 100) of code and in a very short time, the core LBM algorithm can be implemented. Graphics hardware mapping is also straightforward for a knowledgable GPU programmer.

We have implemented both CPU and GPU versions of the aforementioned examples for comparison, while both are not highly optimized. For the CPU version, our simulations are timed on one 2.0 GHz Intel Core2 T7200 CPU with 2 GB memory. Meanwhile, we have accelerated all the examples on an Nvidia Geforce 8800GTX with 768 MB texture memory. With 128 stream processors inside, the GPU is a new-generation graphics hardware released at November 2006. It represents a complete shift in Nvidia's GPU architecture with fully unified shader core, which allocates processing power to geometry, vertex, or fragment shading operations. This CUDA (compute unified device architecture) abandons the previous graphics

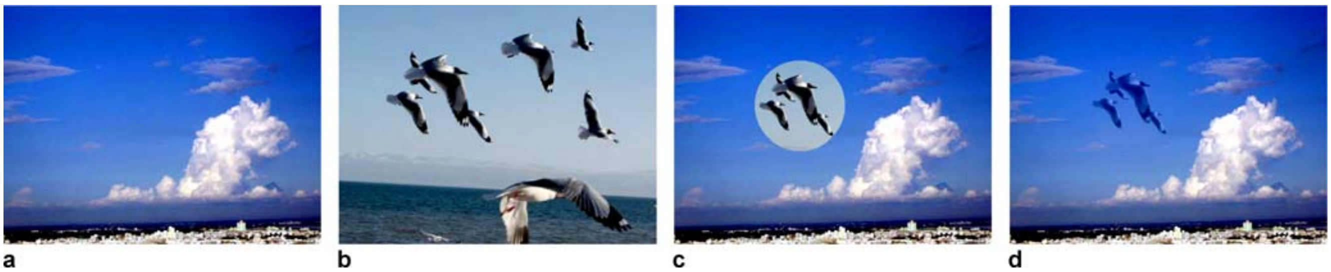

Fig. 9a–d. Poisson image editing. **a** Target: a sky image; **b** Source: a bird image; **c** Cut-and-paste several birds to the sky; **d** Cloning result by our LBM simulation

pipeline with vertex shading followed by fragment shading. Thus, our LBM computation, which only involves fragment processing, achieves superior speed improvement compared with the CPU version.

For diffusion computation, the LBM's equilibrium function is simplified by removing momentum dependency, which greatly increases the computational speed compared to classic LBM 3D fluid solver, because it removes many vector multiplications (Eq. 3) at each lattice site. For comparison, a 3D LBM fluid solver on the same GPU computes each step at around 150 milliseconds.

Usually our algorithm only requires a few steps of LBM simulation to achieve satisfying results in most of the aforementioned applications. Table 1 reports the per step performance of our simulation for different examples. It includes the LBM lattice size, the texture memory used, the computation speed on the CPU and GPU, and the GPU/CPU speedup factor per step. From the performance results, we find the following:

1. Our LBM-based PDE solver accelerated on a new-generation GPU achieves outstanding computation speed. The powerful GPU (Geforce 8800GTX) with CUDA architecture, which executes the LBM on a $128 \times 128 \times 128$ volume data, accomplishes one simulation step at only around 21 millisecond. Thus, the volume smoothing can be completed in about 0.3 seconds using 10 to 15 LBM steps.
2. The surface fairing of a noised mesh is efficiently implemented by our explicit LBM-based solver after voxelizing the mesh into a volumetric lattice. The computation speed mainly depends on the lattice size, instead of the number of polygons. The bunny and dragon models are simulated in the same speed, given the same lattice size.
3. The speed of the 2D simulations are accelerated less than that of the 3D simulations on the GPU. The GPU/CPU speedup factors of large-size 3D volumes is larger than those of small-size volumes. These two observations show that our GPU-based method works even better for modeling and manipulating large data sets.
4. The Poisson image editing uses small LBM lattice size than the image denoising. However, it manipulates the color image with three independent color components and uses more complicated operations; therefore, it runs slower than the image denoising of the gray scale image.
5. Texture memory size of an GPU is a decisive factor for the usability and effectiveness of GPU algorithms. A $128 \times 128 \times 128$ lattice consumes approximately 402.7 MB texture memory, achieving great performance improvement on the Geforce 8800GTX GPU with 768 MB texture memory. We will work on optimizing the memory consumption for large data sets in next step. For extremely large data sets, multiple GPUs could be adopted.

6 Conclusion

We have shown that a fluid dynamics methodology, lattice Boltzmann model, can be modified and extended to solve PDEs in various non-fluid applications. Based on LBM's explicit and local operations, this scheme is directly implemented on the GPU; thus, it provides very fast computation speed. Besides the volume smoothing, surface fairing and image editing examples presented in this paper, it can be applied to more applications involving diffusion, Laplace and Poisson equations. In the future, we will endeavor to improve the method with optimized consumption of computational resources, direct operation on surfaces without voxelization, and innovate its usage in solving more PDEs (e.g., level set equation), so that it can be used as an GPU-based PDE simulation engine for computer vision, computer graphics and visualization.

Acknowledgement The bunny and the dragon model are courtesy of the Computer Graphics Laboratory, Stanford University. The volume foot is courtesy of the National Library of Medicine. The cube and vase model are courtesy of the Vislab, Stony Brook University.

References

1. http://www.openmp.org
2. Bella, G., Filippone, S., Rossi, N., Ubertini, S.: Using openMP on a hydrodynamic lattice-Boltzmann code. In: Proceedings of the Fourth European Workshop on OpenMP, Roma (2002)
3. Bhatnagar, P.L., Gross, E.P., Krook, M.: A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. Phys. Rev. **94**(3), 511–525 (1954)
4. Bolz, J., Farmer, I., Grinspun, E., Schröder, P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Trans. Graph. **22**(3), 917–924 (2003)
5. Boris, J.: New directions in computational fluid dynamics. Ann. Rev. Fluid Mechanics **21**, 695 (1987)
6. Buick, J., Greated, C.: Gravity in a lattice Boltzmann model. Phys. Rev. E **51**(5), 5307–5319 (2000)
7. Chu, N., Tai, C.: Moxi: real-time ink dispersion in absorbent paper. ACM Trans. Graph. **24**(3), 504–511 (2005)
8. Desbrun, M., Meyer, M., Schröder, P., Barr, A.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of SIGGRAPH, pp. 317–324. ACM Press, Los Angeles, CA (1999)
9. Desplat, J.C., Pagonabarraga, I., Bladon, P.: LUDWIG: A parallel Lattice–Boltzmann code for complex fluids. Comput. Phys. Commun. **134**(3), 273–290 (2001)
10. Fan, Z., Qiu, F., Kaufman, A., Yoakum-Stover, S.: GPU cluster for high performance computing. In: Proceedings of ACM/IEEE Supercomputing Conference,

pp. 47–59. IEEE Computer Society Press, Pittsburgh, PA (2004)
11. Gibson, S.: Using distance maps for accurate surface representation in sampled volumes. In: Proceedings of the IEEE Symposium on Volume Visualization, pp. 23–30. IEEE Computer Society Press, Research Triangle Park, NC (1998)
12. Goodnight, N., Woolley, C., Lewin, G., Luebke, D., Humphreys, G.: A multigrid solver for boundary value problems using programmable graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 102–111. Eurographics Association, San Diego, CA (2003)
13. Harris, M., Baxter, W., Scheuermann, T., Lastra, A.: Simulation of cloud dynamics on graphics harware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 92–101. Eurographics Association, San Diego, CA (2003)
14. He, X., Luo, L.: Lattice Boltzmann model for the incompressible Navier-Stokes equation. J. Stat. Phys. **88**(3/4), 927–944 (1997)
15. Jawerth, B., Lin, P., Sinzinger, E.: Lattice Boltzmann models for anisotropic diffusion of images. J. Math. Imaging Vis. **11**, 231–237 (1999)
16. Jia, J., Sun, J., Tang, C., Shum, H.: Drag-and-drop pasting. In: Proceedings of SIGGRAPH, pp. 631–637. ACM Press, Boston, MA (2006)
17. Krüger, J., Westermann, R.: Linear algebra operators for GPU implementation of numerical algorithms. ACM Trans. Graph. **22**(3), 908–916 (2003)
18. Lefohn, A., Whitaker, R.: A GPU-based, three-dimensional level set solver with curvature flow. University of Utah technical report UUCS-02-017 (2002)
19. Li, W., Fan, Z., Wei, X., Kaufman, A.: Flow Simulation with Complex Boundaries, chap. 47, pp. 747–764. GPU Gems 2. Addison-Wesley, Boston, MA (2005)
20. Lorensen, W., Cline, H.: Marching cubes: A high resolution 3d surface construction algorithm. In: Proceedings of SIGGRAPH, pp. 163–169. ACM Press, Anaheim, CA (1987)
21. Massaioli, F., Amati, G.: Optimization and scaling of an open MP LBM code on IBM SP nodes. Scicomp06 Talk (2002)
22. Museth, K., Breen, D., Whitaker, R., Barr, A.: Level set surface editing operators. In: Proceedings of SIGGRAPH, pp. 330–338. ACM Press, San Antonio, TX (2002)
23. Neumann, L., Csebfálvi, B., Viola, I., Mlejnek, M., Gröller, E.: Feature-preserving volume filtering. In: Proceedings of the symposium on Data Visualisation 2002, pp. 105–114. Eurographics Association, Barcelona (2002)
24. Owens, J., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A., Purcell, T.: A survey of general-purpose computation on graphics hardware. In: Eurographics State of the Art Reports, pp. 21–51. Eurographics Association, Dublin (2005)
25. Perez, P., Gangnet, M., Blake, A.: Poisson image editing. In: Proceedings of SIGGRAPH, pp. 313–318. ACM Press, San Diego, CA (2003)
26. Perona, P., Malik, J.: Scale-space and edge detection using anisotropic diffusion. IEEE Trans. Pattern Anal. Machine Intell. **12**(7), 629–639 (1990)
27. Rodgman, D., Chen, M.: Volume Denoising for visualizing refraction. In: Scientific Visualization: The Visual Extraction of Knowledge from Data, pp. 163–185. Springer, Berlin (2006)
28. Rumpf, M., Strzodka, R.: Nonlinear diffusion in graphics hardware. In: Proceedings of Eurographics/IEEE TCVG Symposium on Visualization, pp. 75–84 (2001)
29. Sramek, M., Kaufman, A.: Alias-free voxelization of geometric objects. IEEE Trans. Vis. Comput. Graph. **5**(3), 251–267 (1999)
30. Stam, J.: Stable fluids. In: Proceedings of SIGGRAPH, pp. 121–128. ACM Press, Los Angeles, CA (1999)
31. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Numerical Mathematics and Scientific Computation. Oxford University Press, New York (2001)
32. Taubin, G.: A signal processing approach to fair surface design. In: Proceedings of SIGGRAPH, pp. 351–358. ACM Press, Los Angeles, CA (1995)
33. Thürey, N., Rüde, U.: Free surface lattice-Boltzmann fluid simulations with and without level sets. In: Proceedings of Workshop on Vision, Modeling, and Visualization (Stanford, CA), pp. 199–208. IOS Press, Amsterdam (2004)
34. van der Sman, R., Ernst, M.: Diffusion lattice Boltzmann scheme on orthorhombic lattice. J. Stat. Phys. **94**(1–2), 203–216 (1999)
35. Wang, S., Kaufman, A.: Volume-sampled 3D modeling. IEEE Comput. Graph. Appl. **14**(5), 26–32 (1994)
36. Wei, X., Zhao, Y., Fan, Z., Li, W., Qiu, F., Yoakum-Stover, S., Kaufman, A.: Lattice-based flow field modeling. IEEE Trans. Vis. Comput. Graph. **10**(6), 719–729 (2004)
37. Whitaker, R.: Reducing aliasing artifacts in iso-surface of binary volumes. In: Proceedings of the IEEE Symposium on Volume Visualization, pp. 23–32. IEEE Computer Society Press, Salt Lake City, UT (2000)
38. Wolf-Gladrow, D.: A lattice Boltzmann equation for diffusion. J. Stat. Phys. **79**(5–6), 1023–1032 (1995)
39. Zhao, Y., Han, Y., Fan, Z., Qiu, F., Kuo, Y.C., Kaufman, A., Mueller, K.: Visual simulation of heat shimmering and mirage. IEEE Trans. Vis. Comput. Graph. **13**(1), 179–189 (2007)
40. Zhao, Y., Qiu, F., Fan, Z., Kaufman, A.: Flow simulation with locally-refined LBM. In: Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 181–188. ACM Press, Seattle, WA (2007)

YE ZHAO is currently an assistant professor in the Department of Computer Science at the Kent State University. He received his BE and MS degrees in computer science from the Tsinghua University of China in 1997 and 2000. He further received his PhD degree in computer science from the Stony Brook University in 2006. His research interests include physically based modeling and visualization, GPGPU, image and geometric processing, and volume graphics.