Dispersion Simulation and Visualization for Urban Security

Feng Qiu Ye Zhao Zhe Fan Xiaoming Wei Haik Lorenz Jianning Wang
Suzanne Yoakum-Stover Arie Kaufman Klaus Mueller [*]

Center for Visual Computing and Department of Computer Science
Stony Brook University, Stony Brook, NY 11794-4400

Abstract

We present a system for simulating and visualizing the propagation of dispersive contaminants with an application to urban security. In particular, we simulate airborne contaminant propagation in open environments characterized by sky-scrapers and deep urban canyons. Our approach is based on the Multiple Relaxation Time Lattice Boltzmann Model (MRTLBM), which can efficiently handle complex boundary conditions such as buildings. In addition, we model thermal effects on the flow field using the hybrid thermal MRTLBM. Our approach can also accommodate readings from various sensors distributed in the environment and adapt the simulation accordingly. We accelerate the computation and efficiently render many buildings with small textures on the GPU. We render streamlines and the contaminant smoke with self-shadowing composited with the textured buildings.

Keywords: Lattice Boltzmann Model, GPU, Visualization

1 Introduction

According to the National Research Council report *Tracking and Predicting the Atmospheric Dispersion of Hazardous Releases*, "Our nation's capacity to respond to atmospheric C/B/N (chemical/biological/nuclear) events stands, like a three-legged stool, on the strength of three interconnecting elements: (1) dispersion models that predict the path and spread of the hazardous agent; (2) observations of the hazardous plume itself and of local meteorological conditions; and (3) interaction with emergency responders who use the information provided by the models." The simulation work we present in this paper is directly relevant to the first and third elements. The Lattice Boltzmann Model (LBM) that we use can accurately model air flow and contaminant transport and mixing in geometrically complex environments with the inclusion of thermal effects due to surface heating. By exploiting the inherent locality of the LBM and implementing the computation on the GPU, we further demonstrate that it is feasible to build large scale simulations that span a whole city. We also show the performance and visualization advantages that result from using the GPU for scientific computation. The importance of visualization stems from its ability to enhance the usefulness and accessibility of the information provided by the model. Our demonstration application illustrates how the combination of LBM modeling and GPU computation can enhance our understanding of meteorological and fluid dynamic processes governing dispersion in urban areas and also allow emergency management, law enforcement and other personnel to adequately plan for, train for, and respond to potential accidents or attacks involving toxic airborne contaminants.

[*]Email:{qfeng, yezhao, fzhe, wxiaomin, hlorenz, jianning, suzi, ari, mueller}@cs.sunysb.edu

IEEE Visualization 2004
October 10-15, Austin, Texas, USA
0-7803-8788-0/04/$20.00 ©2004 IEEE

2 Related Work

Researchers have conducted dispersion observation experiments in various environments. The urban tracer and meteorological field campaign (URBAN) conducted in Salt Lake City in 2000 investigated meteorological and fluid dynamic processes governing dispersion in urban environments. In particular, the study attempted to resolve interacting scales of atmospheric motion from the scale of individual buildings to that of whole cities and entire regions [2]. Another meteorological field campaign conducted during October 2000 in the Salt Lake Valley studied vertical transport and mixing (VTMX) processes [13]. The focus of that project was to measure, characterize and analyze VTMX processes, especially in urban areas larger than that of URBAN 2000. The data and insights resulting from these campaigns will help to build better models and evalute the performance of existing numerical simulations for dispersion in urban environments.

In terms of modeling, Pardyjak et al. [31, 30], Williams et al. [42, 43] and Boswell et al. [6] have proposed a fast-response urban dispersion modeling system that computes 3D wind patterns and dispersion of airborne contaminants in urban areas with many buildings. The wind model (QUIC-URB) uses empirical algorithms that estimate the wind fields around buildings. The Lagrangian dispersion model (QUIC-PLUME) computes the dispersion using random walk equations based on the mean wind field produced by QUIC-URB. Brown et al. [7, 8] have presented a modeling approach to compute wind fields and simulate the transport of agents in three different scales. A numerical weather prediction model called COAMPS [21, 10] computes the wind field and other meterological physical effects such as temperature at the urban scale. At the many-building scale, HIGRAD [8, 34] computes the flow field around buildings and simulates contaminants transport. This model is a second-order accurate computational fluid dynamics (CFD) model based on the Navier-Stokes equations (NSE) with finite difference approach. For single to few-building scale, another CFD model called FEM3MP [9] was used. This is a finite element model that can simulate a flow field and dispersion around individual buildings in great detail. The three models take appropriate scale-dependent physics into account and share data together.

Recently, LBM [36] has been introduced to the graphics community for modeling various flow phenomena including wind, smoke, fire, and melting [38, 39, 41, 45]. Although LBM is a relatively new CFD procedure, it has the advantages of being simple to implement, parallelizable, and can accommodate complex boundaries. It can also be extended to model thermal effects, reactive flows, and other physics with relative ease. In contrast to the flow simulation methods described above, the LBM does not model the NSE directly. Rather, it models the micro-scale Boltzmann kinetics of fluid elements streaming and collision. As a numerical scheme, it is explicit, synchronous, second-order space-time accurate with an advection limited time-step. In the limit of zero time step and lattice spacing, LBM yields the NSE for an incompressible fluid. As a kind of explicit finite difference method, LBM is consistent for flows with low Mach number (i.e., flow velocities small compared to the speed of sound) and its time step is advection limited. Although LBM is only conditionally nonlinearly stable, a subgrid model that represents small scale energy damping and the MRTLBM method that presented in this paper can be used to enhance the stability for flows with higher Reynolds numbers. The properties of LBM, such as stability, restrictions and configuration of the simulation parameters, are elucidated in [36] and [40]. Comparing with the stable solver for Navier-Stokes equations, the LBM takes more time steps as an explicit method. However, the LBM has simple operations (no need to solve the linear systems for each step) and it is able to handle complex and moving boundaries. Furthermore, its parallel nature and locality facilitate the GPU acceleration. These advantages make LBM a good choice to simulate dispersions in urban area.

GPU acceleration has been found useful for a broad range of non-graphics computations, including several physically-based simulations. Harris et al. [20] implemented the Coupled Map Lattice (CML) on GPUs, and simulated cloud dynamics using partial differential equations [19]. Gootnight et al. [18] have implemented a multigrid solver on the GPU. Krüger and Westermann [26] have presented a GPU implementation of several linear algebra operators and used them to solve the NSE. Bolz et al. [5] have developed a GPU-based multigrid solver, and presented a conjugate-gradient solver on the GPU based on a sparse matrix representation which they applied to the Navier Stokes equations. In our previous work, Li et al. [29] have accelerated single-relaxation-time LBM computation on GPU and Wei et al. [38, 39, 41] have used the accelerated LBM to model gaseous phenomena, such as fire, smoke and wind. (More examples can be found on the web site of GPGPU (general-purpose computation using graphics hardware) [1]). In this paper, we extend our previous work to speed-up the more complicated Multiple Relaxation Time LBM simulation, which is more stable.

Textures for city models are usually captured together with the geometry. For example, Wang et al. [37] have generated both geometry and texture from a large set of registered images taken automatically. On the other hand, Früh and Zakhor [17] have implemented texture capturing as a separate video based process parallel to geometry scanning. All these methods, however, generate huge amounts of texture data since every building gets its own texture. There are several approaches to reduce the number of textures. Wonka et al. [44] have done so by creating a detailed semantic based geometry using grammars, which is textured with few repeated textures. Legakis et al. [28] have concentrated on brick patterns by synthesizing textures using a cell based method. Another approach common to commercial solutions is to concentrate on landmarks, which are postprocessed by hand using CAD applications or taken from libraries. Other parts of the city model are left without texture. In contrast, our texturing is not part of the model generation process. Instead, we have used pictures of the real buildings and incorporate them into the geometry.

Many techniques have been proposed for smoke rendering. Ebert and Parent [14] and Foster and Metaxas [16] have used volumetric ray tracing to render smoke and other gaseous phenomena. Stam [35] has devised a fluid solver using semi-Lagrangian advection schemes, a projection step to ensure incompressibility, and an implicit treatment for viscosity, producing compelling simulations of turbulent flows. Fedkiw et al. [15] have used a photon mapping algorithm for participating media [22]. However, although ray tracing and photon mapping methods can render high-quality images, they are extremely slow, usually several minutes per frame. Wei et al. [39] have rendered smoke with textured splats, which was proposed by Crawfis and Max [11] and used by King et al. [23] to render fire. This method can render smoke in real-time but doesn't incorporate global illumination effects, such as self-shadowing.

3 Simulation

3.1 Multiple Relaxation Time LBM

The LBM models Boltzmann particle dynamics on a lattice. Particles stream in discrete time steps to neighboring sites. Between streaming steps, they undergo collision. Both streaming and collision steps are applied in the local neighborhood of each cell. Moreover, the complex boundaries that represent the internal objects are also treated as part of the collision step in the local neighborhood. Because of these features, compared to other CFD methods, the LBM has the advantages of being easy to implement, parallelizable, and able to handle complex and moving boundaries. A commonly used collision model is the single-relaxation-time LBM (SRTLBM) model of Bhatnagar, Gross and Krook (BGK) [4]. The BGK model represents collisions as a statistical redistribution of momentum toward equilibrium. The model has one free parameter - the relaxation time, which controls the viscosity of the fluid. We refer the readers to a book [36] and our previous works [38, 39, 41, 45] for details on SRTLBM. MRTLBM uses a more general collision model in which many of the hydrodynamic moments relax toward their equilibria independently [12]. The additional freedom afforded by the decoupled relaxation parameters, gives the model better stability and facilitates the coupling of additional physics. In this section we provide a basic introduction to the MRTLBM.

As in the SRTLBM , the set of b lattice vectors in the unit cell define the set of discrete velocities in the model, $\{\mathbf{e_i}|(i = 0, 1, \ldots, b-1)\}$. Corresponding to each lattice vector there is a velocity distribution function $\{f_i|(i = 0, 1, \ldots, b-1)\}$. The distribution function $f_i(\mathbf{r})$ represents the probability that a flow element with velocity $\mathbf{e}_i$ exists at node $\mathbf{r}$. The hydrodynamic moments, such as mass density ρ and momentum density $\mathbf{j}$, are obtained by linear combinations of these distributions. For example $\rho = \sum_i f_i$ and $\mathbf{j} = \frac{1}{\rho}\sum_i f_i \mathbf{e}_i$. We may therefore define two b-dimensional vector spaces, one called phase space, which is spanned by the f_i and the other one called moment space, which is spanned by the lowest order hydrodynamic moments.

For our simulations, we use a simple 3D LBM lattice, denoted by D3Q13. As illustrated in Figure 1, a unit cell of this lattice includes the center node with zero velocity and the twelve second-nearest neighbor links (the six nearest neighbor axial links are not used). In this lattice, we have 13 distributions, $\{f_i|(i = 0, 1, \ldots, 12)\}$ and 13 moments $\{m_i|(i = 0, 1, \ldots, 12)\}$. Each of these moments has a physical meaning. For example, m_0 is the mass density ρ, $m_{1,2,3}$ are the components of the momentum density vector $\mathbf{j}$, m_4 represents the energy, higher order moments represent components of the stress tensor, and so on.

The transformation from one representation to the other may be expressed as:

$$|m\rangle = M|f\rangle, |f\rangle = M^{-1}|m\rangle \tag{1}$$

$$|f\rangle = (f_0(\mathbf{r},t_n), f_1(\mathbf{r},t_n), \ldots, f_{12}(\mathbf{r},t_n))^T, \tag{2}$$

$$|m\rangle = (m_0(\mathbf{r},t_n), m_1(\mathbf{r},t_n), \ldots, m_{12}(\mathbf{r},t_n))^T, \tag{3}$$

where T is the transpose operator, M represents the invertible linear mapping, $\mathbf{r}$ represents each lattice point, and t_n is the time step.

In MRTLBM, the streaming is also performed on the discrete velocity space, while the collision is applied in the moment space. The MRTLBM equation is:

$$|f(\mathbf{r}+\mathbf{e_i}, t_n+1)\rangle = |f(\mathbf{r},t_n)\rangle - M^{-1}S[|m(\mathbf{r},t_n)\rangle - |m^{eq}(\mathbf{r},t_n)\rangle], \tag{4}$$

where $|m^{eq}\rangle$ is a vector whose components are the equilibria of the moments, and S is a diagonal matrix whose elements are the relaxation rates:

$$S = diag(s_0, s_1, \ldots, s_{12}). \tag{5}$$

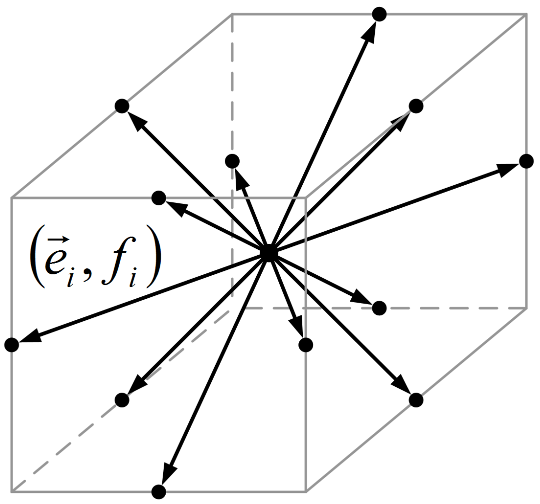


Figure 1: The D3Q13 lattice geometry. The particle distribution f_i is associated with the link corresponding to the $\mathbf{e}_i$ velocity vector.

As in SRTLBM, the mass and momentum densities are conserved, so their equilibrium values do not change. The equilibrium values of the non-conserved moments depend only on the conserved moments. However, because they are exposed in the Hybrid Thermal LBM (HTLBM) model [27] we can modify them to simulate other effects, such as body force or temperature. In Section 3.2, we describe how we use body force to accommodate sensor input.

To model thermal effects on flows in the HTLBM the equilibrium energy is modified to account for temperature as follows:

$$m_4^{eq} = \frac{39}{2}(c_{s0}^2 - \frac{8}{13})\rho + \frac{39}{4}(\frac{5}{3} - \gamma)\mathbf{j}\cdot\mathbf{j} + \frac{39}{2}q_1 T, \tag{6}$$

where T is the temperature, γ is the specific heat, q_1 is a constant coupling coefficient, and c_{s0} is the isothermal speed of sound. The temperature evolution may be modeled with a diffusion-advection equation,

$$\partial_t T + \mathbf{u}\cdot\nabla T = \kappa\Delta T + q_2(\gamma - 1)c_{s0}^2\nabla\cdot\mathbf{u}, \tag{7}$$

where κ is the thermal diffusivity of the fluid, q_2 is a constant coupling coefficient, and $\mathbf{u}$ is the velocity. Equation 7 can be solved by a finite-difference scheme. The resulting temperature field is used in Equation 6.

The values of relaxation rates s_i are determined by a linear analysis. They are related to the viscosity of the fluids as:

$$\nu = \frac{1}{2}(\frac{1}{s_6} - \frac{1}{2}), \tag{8}$$

$$\xi = (\frac{2}{3} - \gamma c_{s0}^2)(\frac{1}{s_5} - \frac{1}{2}), \tag{9}$$

where ν is the shear kinetic viscosity, and ξ is the bulk kinetic viscosity. Their values may be chosen to define the characteristics of the fluid. The remaining relaxation rates are then determined.

The boundary condition of MRTLBM does not differ from the SRTLBM, as it is also applied in phase space. We can apply bounce-back, outflow, inflow, curved, and moving boundary conditions.

3.2 Sensor Feedback

To study the behavior of smoke particles, gases, aerosols, and other plumes in an urban environment, such as New York City (NYC), the Department of Energy's Urban Atmospheric Observatory (UAO) plans an extensive sensing test bed of a dense coverage of sensors (chem. bio, lasers, radars) in an 1.6km x 1.6km area of midtown NYC, which includes Madison Square Garden. To refine and validate our LBM models, we will use the results of "pilot exercises" of a set of outdoor Perfluorocarbon tracer (FFT) studies to be conducted over the span of several days in the midtown area later in 2004 and 2005. Currently, we use a 10-block area around the Environmental Measurements Laboratory (EML) building in the West Village of NYC at the corner of Houston Street and Varick Avenue. Those sensors record the wind velocity, temperature, etc. in real-time. Currently, there are 3 sensors installed on the EML building.

The accuracy of the simulation is limited and different errors (rounding error, discretization error, etc.) may accumulate, and consequently the LBM results are different from the data read from sensors. Therefore, sensor readings are used to correct and guide the simulation. Once the live-sensor input is communicated over network links, how to incorporate the sensor data with existing simulation remains a challenge. Currently we use two approaches to adapt the simulation's numerical models to accommodate sensor data. In the first method, the effect of the sensor data is incorporated as a body force on the corresponding grid node. In the second method, we trace those boundary nodes that will affect the sensor points and modify those boundary nodes directly to match the sensor data.

Consider a simple LBM for a wind field in a complex urban environment with anemometers at various locations. To bring the simulation into sensor agreement, we read the sensor velocity at every time step and use it to modify the distributions at the nearby grid nodes either through the equilibrium distributions or via a body force. Better, we use a weighted sum of the sensor and LBM velocities such that its contribution decreases smoothly away from the sensor. A loss of momentum conservation may result – the nodes near the sensor behave as local sources or sinks in the flow depending on the difference between the sensor data and the simulation data. Also, when viewed globally, this approach would only "fix" the downstream flow. We set up a test bed with our current thermal LBM simulation to examine the effect of body force. Starting from an initial simulation with predefined boundary conditions, we recorded the velocity data at each sensor nodes for each time step (self defined). Later, by slightly changing the boundary conditions, we run another simulation. The velocity difference between the recorded data and the current velocity at the sensor points is calculated as the body force on the corresponding grid node. To smooth away any sudden change of the body force at each time step, we average over two time steps. We observe that as long as the difference between the predefined boundary condition and the new boundary condition doesn't exceed 10 percent, the method works well. For the MRTLBM used in our system, we simply add the external body force $\mathbf{F}$ to the momentum, by $\mathbf{j}' = \mathbf{j} + \mathbf{F}\delta t$ $(\delta t = 1)$, as we introduced in Section 3.1. It is understood that in order to conserve mass up to second order in the Chapman-Enskog analysis, the net effect of the force term is that the resultant momentum is equal to $\mathbf{j} + \mathbf{F}\delta t/2$ [27]. The algorithm is as follows:

1. Advection of f_i,
2. Compute moments m_i of f_i,
3. $\mathbf{j}' = \mathbf{j} + \frac{1}{2}\mathbf{F}$,
4. Relaxations of the moments (collision),
5. $\mathbf{j}'' = \mathbf{j}' + \frac{1}{2}\mathbf{F}$,
6. Compute f_i from the moments m_i.

where $\mathbf{j}'$ is used as the measured field for output.

Since a flow field is largely a result of the definition of its boundary conditions, in the second approach we use the velocity error to modify the boundary conditions used on the simulation volume. In the simplest case where the flow volume contains no obstructions, we simply modify the boundary upstream of the nodes near the sensor. For complicated geometries where flows swirl around objects and create vortices, identifying what is upstream is much more challenging. In our current work, we follow the streamlines from the nodes near the sensor back to identify the regions of boundary that are most responsible for the flow. The advantage of this approach is that the flow invariants within the simulation volume are not disturbed and the entire flow is "fixed". However, to observe at the sensor points the effect of the change in the boundary condition, we need to wait for several LBM time steps. The further away the sensor from the corresponding boundary node, the longer the wait.

3.3 Hardware Acceleration

Because of the locality, and hence parallelizability, of the LBM operations, accelerating LBM computation on today's programmable GPU is straightforward and efficient. In our previous work [29], we have accelerated SRTLBM on the GPU and we have extended our previous work here to improve the performance of MRTLBM which has a more complicated collision operation that requires matrix-vector multiplication (see Equation 1). In what follows we briefly review the approach taken for accelerating LBM on the GPU and then detail our implementation of the MRT collision operations.

Mapping non-graphics computation onto the GPU involves two main issues: (1) laying-out the data in texture memory; and (2) using graphics operations to compute the results.

- To layout the LBM data, we divide the lattice sites into several volumes. Each volume contains data associated with a given state variable and has the same resolution as the LBM lattice. For example, each of the 13 particle distributions $f_0 - f_{12}$ in the D3Q13 MRTLBM, is represented in a volume as is each of the 13 moments $m_0 - m_{12}$. To use the GPU vector operations and save storage space, we pack four volumes into a series of 2D textures (note that a fragment or a texel has 4 color components). Therefore, both the 13 particle and moment distributions are packed into 4 series of textures, respectively. The boundary link information (e.g., the flags to indicate whether the lattice links intersect with boundary surfaces) is also stored in textures, but since most links do not intersect the boundary surface, we do not store boundary information for the whole lattice. Instead, during rendering we cover the boundary region of each Z slice using multiple small rectangles. Hence, we need to store the boundary information only inside those rectangles in one 2D texture.

- The LBM operations (e.g., streaming, collision, and boundary conditions) are translated into fragment programs that can be executed in one rendering pass. For each fragment in a given pass, the fragment program fetches any required current state information from the appropriate textures, evaluates a result, and renders it to a pixel buffer. When the pass is completed, the results are copied back into another texture.

To implement matrix-vector multiplication in MRTLBM, we take advantage of the GPU vector operations which have been optimized. Note that because the GPU only supports a 4 dimensional vector space, we need to decompose our 13 dimensional matrix and vector operations as illustrated in Figure 2. With this decomposition, the multiplication of $y \leftarrow M \times x$ can be rewritten into four equations:

$$y_0 \leftarrow M_{00} \times x_0 + M_{01} \times x_1 + M_{02} \times x_2 + M_{03} \times x_3, \quad (10)$$

$$y_1 \leftarrow M_{10} \times x_0 + M_{11} \times x_1 + M_{12} \times x_2 + M_{13} \times x_3, \quad (11)$$

$$y_2 \leftarrow M_{20} \times x_0 + M_{21} \times x_1 + M_{22} \times x_2 + M_{23} \times x_3, \quad (12)$$

$$y_3 \leftarrow M_{30} \times x_0 + M_{31} \times x_1 + M_{32} \times x_2 + M_{33} \times x_3 \quad (13)$$

Each equation 10 - 13 is implemented as a single fragment program. Fragment program i takes sub-matrices M_{ij} as input, fetches the elements of vector x from the appropriate texels, evaluates the equation, and outputs the result y_i.

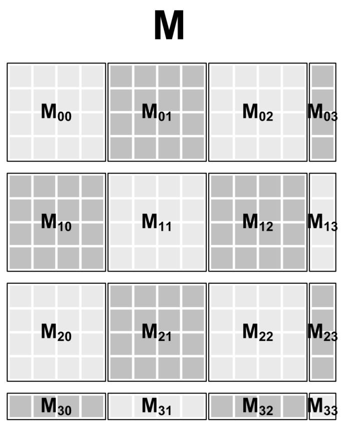


Figure 2: Decomposing the matrix and vectors.

4 Visualization

In the simulation, massive amounts of results are generated. These numbers are hard to understand by most scientists. With visualization, the user can better analyze the simulation results of flow fields through streamlines. Realistic visualization in real time can help trainees and emergency services personnel (end users) better understand the situation and make decisions in real events. The visualization has two parts. The first is to render buildings with textures. Because the simulation is executed on the GPU and most of the texture memory is used to store simulation data, there is little room to store textures of the buildings. Instead, we use noise textures and a smart shader to help texturing the buildings. The second part of the visualization is to render smoke with self-shadows in real-time.

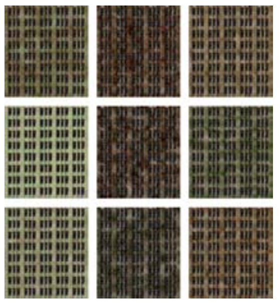
4.1 Buildings

Our city model consists of plain geometry only. To improve the visual appearance, building facade textures are used to resemble the look of the actual city. This leads to two problems: the textures themselves need to be created and rendered and the geometry has to be augmented with texture coordinates.

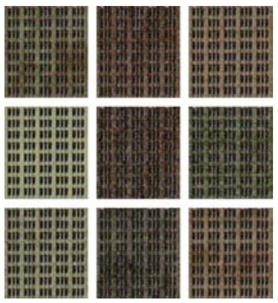
Facade textures are prepared by hand from pictures taken on site. Since texture memory is a scarce resource, we are allowed only a very small amount of actual distinct facade textures. The trade-off is between a larger number of low resolution textures and a smaller number of high resolution textures. Since we rate blurry artifacts introduced in the former case more disturbing than visual repetitiveness in the latter one, we use a few high resolution textures.

To reduce the repetitiveness of this approach, we make use of the programmable fragment shading capability of modern graphics hardware by implementing a texture-aging-and-variation shader. This shader changes the overall appearance of a facade texture by adding dirt and cracks without affecting major features such as windows. To do so, it needs an appropriate opacity map stored in the

(a) Plain facade.



(b) Sample variations using Perlin noise.



(c) Sample variations using corrosion patterns

Figure 3: Facade variation using one set of textures.

alpha channel of the facade texture. In total it uses five textures per facade, one of which is the facade texture itself.

Since texture memory is the most important constraint, the shader and its data are designed for versatile use. Thus, information about color and intensity of dirt added to the facade is split into color parameters and grey scale textures. This enables the shader to produce very different results with the same textures, and thus reduces the overall number of textures needed.

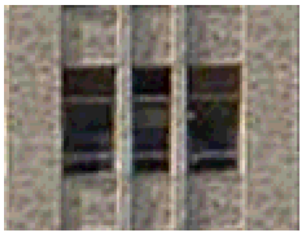
For each facade three grey scale "noise" textures are used to add three differently colored layers of dirt. The term noise is quoted since statistical or Perlin noise [32] usually does not give best results. Patterns of corrosion or erosion found in nature are more suitable, as shown in Figure 3. Dirt is added by means of color replacement. For each dirt layer the shader has a base color and a dirt color. The more the local original facade color is similar to the base color, the more the actual fragment color is dragged towards the dirt color. The similarity is attenuated by the respective noise texture.

Additionally, one more grey scale texture is used to attenuate the facade texture's intensity directly, simulating cracks. Bump maps were tested for this purpose, but results indicated that, besides the necessary three color channels instead of one, high resolution maps are needed in order to make them visually effective. Again, for versatile use a parameter attenuates the impact of the intensity texture.

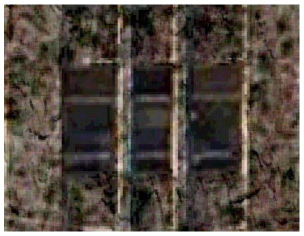
Due to the layering, the perceived final texture resolution is higher than the individual layer resolutions. Since the facade texture already has high resolution, the noise textures do not need to (see Figure 3). Additionally, due to the nature of a noise texture it can be shrunk and tiled across the whole facede without obvious artifacts. Shrinking factors up to 2 give good results. Thus, the impact of noise textures on texture memory can be kept to a minimum without sacrificing effectiveness.

The second problem we have to address is texture coordinate generation. This includes the following steps: separation of buildings into facades, choice of a facade texture, and finally the actual texture coordinate generation for all five textures per facade.

The first step has no general solution. Its implementation depends highly on the input geometry. In our model, buildings generally follow a box shape. This allows us to associate the building's triangles with a facade based on their normals. Therefore, we use k-means clustering [3] to get four groups of similarly aligned triangles. These four groups form two opposite wide facades and two narrow ones.



(a) Plain facade closeup view.



(b) Facade variation closeup view.

Figure 4: Closeup view using nearest neighbor interpolation.

Subsequently, a facade texture has to be chosen. Since we have only a limited number of original facade textures we must fit them to multiple buildings. This is done by first registering the prepared facade textures with their respective original buildings to get an estimate of the respective physical floor height h_f and window width w_f. The facade texture assigned to a building is the one that can be fitted best using only multiples of h_f and w_f. The four noise textures are chosen randomly from a given set.

Finally, texture coordinates and shader parameters have to be generated. The facade texture coordinates are computed directly from the number of floors and windows that the chosen facade yields. Noise textures on the other hand are less restricted. To increase vividness of the result, starting from the facade texture coordinates, the coordinates are transformed randomly using rotation, translation, and scaling in a given range. The shader parameters are generated randomly as well except for the base colors, which are attributes of the facade textures. Dirt colors are varied in the red and green channels only since blue colors are not found in natural dirt.

4.2 Smoke

In our approach, the LBM simulation computes the position and velocity of smoke particles with the coarse interactions of the fluid with the scene. The particles can be rendered using OpenGL points after reading back the positions from the GPU to the main memory and sorting them into slices by the CPU. Each particle is projected onto the image plane as a textured splat, which can be accomplished on graphics hardware efficiently. Textured splats add the small-scale interactions and visual details to the final image. However, the original textured splats method does not take into account the shadows among splats, although the shadows of all the splats can be cast onto other scene objects. Kniss et al. [24, 25] have proposed a shading model for volumetric shadows and translucency. Instead of slicing the volume in the view direction, this method adopts the half angle slicing technique, as shown in Figure 5. The angle between l and h and the angle between h and v are both θ. For each slice, the light map is computed. All slices are projected to the image plane in a front to back or back to front order as in texture based volume rendering, using a light map for shading. In Kniss et al.'s model, the volume is stored in a 3D texture and the 3D texture hardware can be exploited to reconstruct each slice efficiently. In our case, the volume is a series of particles and the slices are reconstructed by splatting.

Our smoke rendering algorithm works as follows. First, the view direction v and the light direction l are determined and the half direction h is computed. To reconstruct the volume, the half space coordinate system must be established. h is the z-axis. The cross product of h and v is the x-axis. The origin is the center of the bounding box of the simulation. Then, the bounding box of the vol-

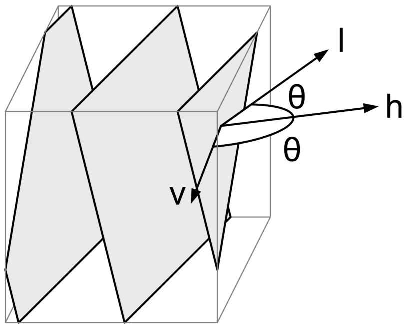


Figure 5: This figure shows the light direction l, view direction v and half direction h. The slicing planes are perpendicular to h.

ume in the half space coordinate system is computed. This bounding box is sliced into n slabs with slicing planes perpendicular to the z-axis of the half space coordinate system. Thus, each slab has a start and an end z-value. For each particle, the z coordinate in the half space is computed and used to sort it into one slab. This bucket sorting costs $O(m \log n)$ time, where m is the number of particles. In each slab, the particles are rendered using the textured splats method into the density map for the current slice. The slice is projected onto the image plane and its density map and light map are used for shading. In the half space, the light map of the next slice is computed by attenuating the current light map with the density map.

Because a particle is treated as a gaussian sphere of diameter d, the final area covered by one splat on the image plane should be a circle. Therefore, the area of one splat projected onto the slicing plane is an ellipse with minor axis of length d and major axis of length $d/\cos(\theta)$, where θ is the angle between slicing plane and viewing plane. In the half space coordinate system, the major axis is parallel to the y-axis and the minor axis is parallel to the x-axis. $\cos(\theta)$ is the dot product of the half direction and the view direction. When projected onto the light plane (plane perpendicular to the light direction), the area covered by this ellipse is a circle of diameter d. This is because the angle between the light plane and the half plane is also θ. Therefore, the light transport is correctly computed in the half space. Figure 6 shows how the gaussian reconstruction kernel for one splat is projected on the three planes.

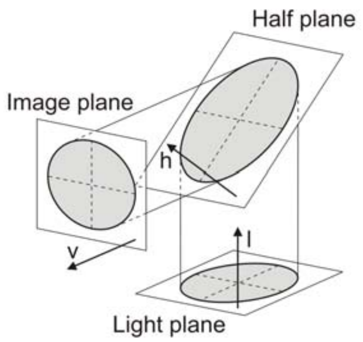


Figure 6: The projected spherical gaussian kernel on different planes.

5 Results

Compared with our method, the QUIC system is an empirical estimation of wind field and the dispersion is computed in the mean wind field. Our method is a second order space-time accurate computational fluid dynamics method. The method proposed by Brown et al. [7] called HIGRAD solves a $1.6km \times 1.5km$ area in Salt Lake City at the grid spacing of 10 meters in few hours on supercomputer or cluster to resolve turbulent eddies. Our method can simulate the West Village area of New York City (about $0.5km \times 0.5km$) with grid spacing of 3.8m at interactive speed. The high speed comes from the simplicity of our model and acceleration by graphics hardware. It can solve in detail complex boundaries such as building models.

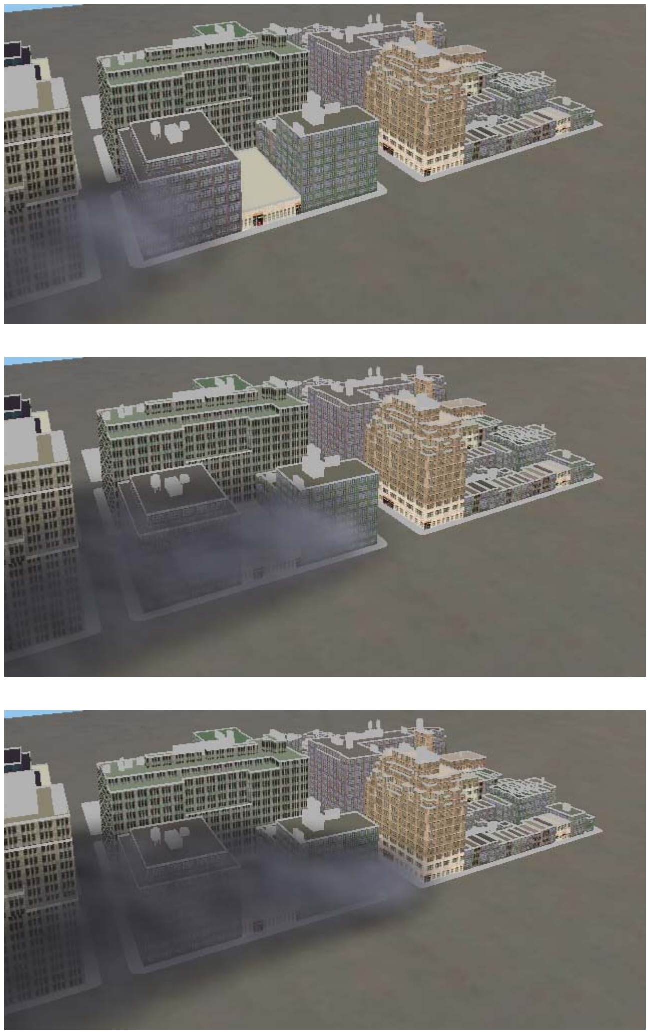


Figure 7: Snapshots of smoke dispersion simulation in West Village area of New York City at time steps 247, 288 and 319.

Figure 7 shows several snapshots of the dispersion simulation procedure. Figure 8 shows closeup views of the buildings and smoke during the simulation. Figure 10 shows the simulation results of a 10-block area rendered by our visualization program. The LBM model consists of $90 \times 30 \times 60$ lattice sites with lattice spacing of less than 5m. The building GIS models are at 1m resolution in the West Village. The smoke particles with initial temperature and velocity are generated at the upper left corner of the bounding box. The air flows from left to right. The 6 images are snapshots of the scene at 6 different time steps. For a 640×640 image, each time step, the simulation costs 81 ms, rendering the buildings costs 16 ms, and rendering smoke costs 31 ms.

We compare the performance of the software (CPU) version and the hardware (GPU accelerated) version of our MRTLBM simulation of the NYC model. The performance is measured on a computer with a 2.53 GHz Intel Pentium 4 CPU and an NVIDIA GeForce FX 5950 Ultra GPU. With a lattice size of $90 \times 30 \times 60$, our GPU implementation can run at 81ms per step, which is 8.02 times faster than the CPU implementation (650ms per step).

For texturing our program uses 4 different facade textures of size up to 512×512 consuming 2.25MB in total. Additionally 10 different noise textures of size 256×256 are used, adding 640KB. Thus, less than 3MB of texture memory are used for visualizing the buildings.

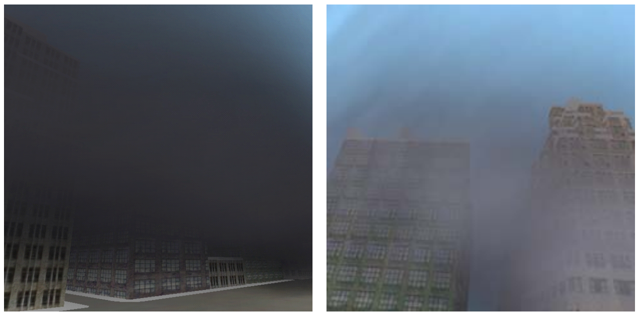


Figure 8: Closeup views of buildings and smoke.

6 Conclusions

In this paper, we describe a method of simulating airborne dispersion in urban environments. This paper demonstrates the first step in our effort to provide simulation and visualization tools of flow dispersion for urban security. The flow field is modeled and simulated using Hybrid Thermal Lattice-Boltzmann Model. The simulation takes a temperature field into account, which is an important factor in real urban environments. We implement the simulation process on graphics hardware with floating-point precision and achieve interactive speed. We use smoke particles as an example to demonstrate dispersion in urban environments, although the method can be applied to other materials or particles without much difficulty. Our visualization program renders buildings with a few textures, yet still making the appearance of buildings different from others using a fragment shader and noise functions. The smoke is rendered with self-shadows, which increases the realism. Together the simulation and visualization run at interactive speed.

In the future, we will simulate and visualize flows in much larger urban environments. The model used in this paper consists of only 10 blocks. Future models may include the entire Manhattan or New York City. With such a large model, we will exploit the capability of GPU clusters for real-time simulation and visualization. We have built a GPU cluster, the Stony Brook Visual Computing Cluster. Currently, using 30 GPU nodes we have run LBM simulations on a large lattice of size $480 \times 480 \times 80$ achiving interactive speeds (Fig. 9).

Acknowledgement

This work has been partially supported by grants from NSF CCR-0306438 and from Department of Homeland Security, Environment Measurement Laboratory.

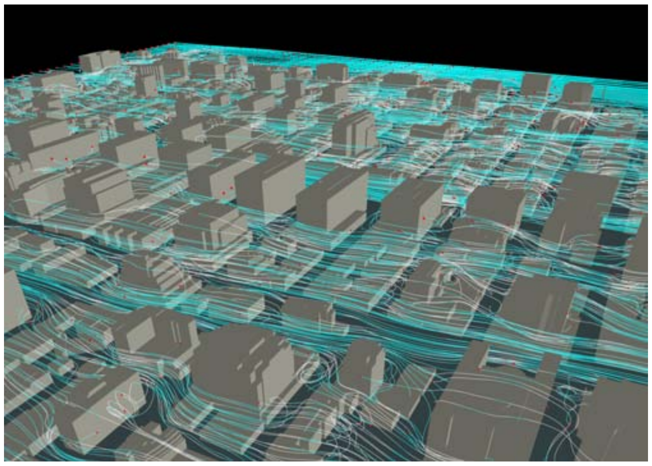


Figure 9: Streamlines in the Time Square area.

References

[1] GPGPU. *http://www.gpgpu.org*.

[2] K. J. Allwine, J. H. Shinn, G. E. Streit, K. L. Clawson, and M. Brown. Overview of URBAN 2000: A multiscale field study of dispersion through an urban environment. *Bulletin of the American Meteorological Society*, 83(4):512–536, 2002.

[3] M. R. Anderberg. *Cluster analysis for applications*. Number 19 in Probability and Mathematical Statistics. Academic Press, New York, 1973. xiii+359 pages.

[4] P. Bhatnagar, E.P. Gross, and M.K. Krook. A model of collision processes in gases. *Physical Review Letters*, 94(511), 1954.

[5] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph. (SIGGRAPH)*, 22(3):917–924, 2003.

[6] D. Boswell and M. Brown. The QUIC quick start guide. 2003.

[7] M. Brown, M. Leach, R. Calhoun, W.S. Smith, D. Stevens, J. Reisner, R. Lee, N.-H. Chin, and D. DeCroix. Multiscale modeling of air flow in salt lake city and the surrounding region. *ASCE Structures Congress*, 2001. LA-UR-01-509.

[8] M. Brown, M. Leach, J. Reisner, D. Stevens, S. Smith, H.-N. Chin, S. Chan, and B. Lee. Numerical modeling from mesoscale to urban scale to building scale. *AMS 3rd Urb. Env. Symp.*, 2000.

[9] S. T. Chan and D. E. Stevens. Evaluation of two advanced turbulence models for simulating the flow and dispersion around buildings. *The Millennium NATO/CCMS Int. Tech. Meeting on Air Pollution Modeling and its Application*, pages 355–362, May 2000.

[10] S. Chen, J.A. Cummings, J.D. Doyle, T.R. Holt, R.M. Hodur, C.S. Liou, M. Liu, A. Mirin, J. A. Ridout, J.M. Schmidt, G. Sugiyama, and W.T. Thompson. COAMPS version 3 model description, 2002. Available from the Naval Research Laboratory.

[11] R. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. *Proceedings of Visualization*, pages 261–266, 1993.

[12] D. d'Humieres, I. Ginzburg, M. Krafczyk, P. Lallemand, and L. Luo. Multiple-relaxation-time lattice boltzmann models in three-dimensions. *Philosophical Transactions of Royal Society of London*, 360(1792):437–451, 2002.

[13] J. C. Doran, J. D. Fast, and J. Horel. The VTMX 2000 campaign. *Bulletin of the American Meteorological Society*, 83(4):537–551, 2002.

[14] D. S. Ebert and Richard E. Parent. Rendering and animation of gaseous phenomena by combining fast volume and scanline a-buffer techniques. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 357–366, 1990.

[15] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. pages 15–22, 2001.

[16] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. pages 181–188, 1997.

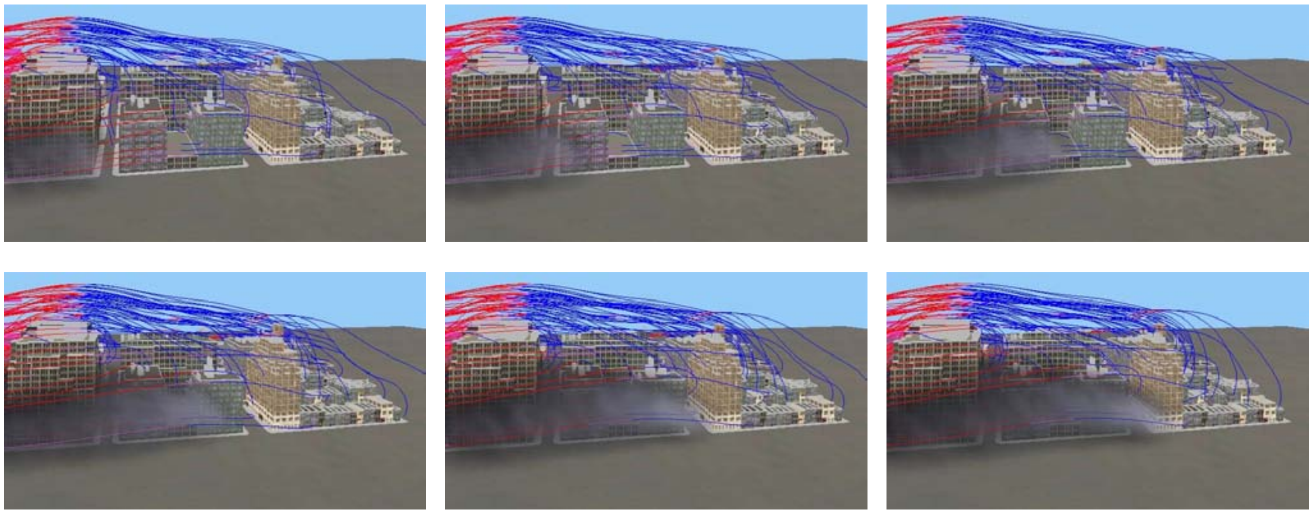


Figure 10: Smoke and streamlines representing dispersion simulation results in the West Village area of New York City. Red (blue) streamlines indicate upward (downward) streaming

[17] C. Früh and A. Zakhor. Constructing 3D city models by merging aerial and ground views. *IEEE Computer Graphics and Applications*, 23(6):52–61, November/December 2003.

[18] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 102–111, July 2003.

[19] M. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 92–101, July 2003.

[20] M. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 109–118, September 2002.

[21] R. M. Hodur. The Naval Research Laboratory's coupled ocean/atmosphere mesoscale prediction system (COAMPS). *Mon. Wea. Rev.*, (125), 1997. 1414-1430.

[22] H. W. Jensen and P. H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. pages 311–320, 1998.

[23] S. A. King, R. A. Crawfis, and W. Reid. Fast volume rendering and animation of amorphous phenomena. pages 229–242, 2000.

[24] J. Kniss, S. Premože, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling. October 2002.

[25] J. Kniss, S. Premože, C. Hansen, P. Shirley, and A. MacPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):150–162, 2003.

[26] J. Krüger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Trans. Graph. (SIGGRAPH)*, 22(3):908–916, 2003.

[27] P. Lallemand and L. Luo. Theory of the lattice Boltzmann method: Acoustic and thermal properties in two and three dimensions. *Physical Review E*, 68:036706, 2003.

[28] J. Legakis, J. Dorsey, and S. J. Gortler. Feature-based cellular texturing for architectural models. In *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 309–316. ACM Press / ACM SIGGRAPH, 2001.

[29] W. Li, X. Wei, and A. Kaufman. Implementing lattice boltzmann computation on graphics hardware. *Visual Computer*, To appear, 2003.

[30] E. Pardyjak and M. Brown. Evaluation of a fastresponse urban wind model - comparison to single building wind-tunnel data. *Int. Soc. Environ.*, 2001. LA-UR-01-4028.

[31] E. Pardyjak and M. Brown. Fast response modeling of a two building urban street canyon. *4th AMS Symp. Urban Env.*, 2002. LA-UR-02-1217.

[32] K. Perlin. An image synthesizer. pages 287–296, 1985.

[33] N. Rasmussen, D. Q. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Trans. Graph.(SIGGRAPH)*, 22(3):703–707, 2003.

[34] J. Reisner, W. Smith, J. Bossert, and J. Winterkamp. Tracer modeling in an urban environment. *2nd AMS Urb. Env. Symp.*, 1998.

[35] J. Stam. Stable fluids. *Proceedings of SIGGRAPH*, Computer Graphics Proc., Annual Conference Series:121–128, 1999.

[36] S. Succi. *The lattice Boltzmann equation for fluid dynamics and beyond*. Numerical Mathematics and Scientific Computation. Oxford University Press, 2001.

[37] X. Wang, S. Totaro, F. Taillandier, A. Hanson, and S. Teller. Recovering facade texture and microstructure from real-world images. *Proc. 2nd International Workshop on Texture Analysis and Synthesis at ECC*, pages 145–149, 2002.

[38] X. Wei, W. Li, K. Mueller, and A. Kaufman. Simulating fire with textured splats. *IEEE Visualization*, pages 227–234, October 2002.

[39] X. Wei, W. Li, K. Mueller, and A. E. Kaufman. The lattice-boltzmann method for simulating gaseous phenomena. *IEEE Trans. on Visualization and Computer Graphics*, 10(3):164–176, March/April 2004.

[40] X. Wei, Y. Zhao, Z. Fan, W. Li, F. Qiu, S. Yoakum-Stover, and A. E. Kaufman. Lattice-based flow field modeling. *IEEE Trans. on Visualization and Computer Graphics*, 10(6), November/December 2004.

[41] X. Wei, Y. Zhao, Z. Fan, W. Li, S. Yoakum-Stover, and A. Kaufman. Blowing in the wind. *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 75–85, July 2003.

[42] M. Williams, M. Brown, and E. Pardyjak. Development of a dispersion model for flow around buildings. *4th AMS Symp. Urban Env.*, 2002. LA-UR-02-0839.

[43] M. D. Williams. QUIC-PLUME user's guide, 2002. LAUR-02-375.

[44] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Transactions on Graphics*, 22(3):669–677, July 2003.

[45] Y. Zhao, X. Wei, Z. Fan, A. Kaufman, and H. Qin. Voxels on fire. *Proceedings of IEEE Visualization 2003*, pages 271–278, October 2003.