14.2 Using Video Gaming Technology to Achieve Low-cost Speed up of Emergency Response Urban Dispersion Simulations

E.R. Pardyjak[1,*], B. Singh[1], A. Norgren[2] and P. Willemsen[2]
[1]University of Utah, [2]University of Minnesota Duluth

1. INTRODUCTION

As a result of the demand for high performance graphics capabilities driven by the computer video game industry, the processing performance of video cards is rapidly evolving. Recent trends in computing have shifted toward multi-core processors and programmable graphics processors equipped with highly parallel data paths for processing geometry and pixels. Multi-core machines are now readily available with 2 cores, but machines with 4, 8, and even 16 or more cores are projected for the near future. Data parallelism in modern graphics cards is also increasing with raw performance of graphics processing units (GPUs) surpassing performance of central processing units (CPUs). While initially specialized for processing computer graphics, GPUs can be programmed for general-purpose computations. As a result, GPUs have become useful computational tools providing inexpensive highly parallel data paths to accelerate a wide range of scientific and simulation applications.

One area of simulation that could greatly benefit from inexpensive parallelization is emergency response transport and dispersion modeling in urban areas. In a previous paper (Willemsen et al. 2007), we implemented a simple Lagrangian dispersion model based on the Quick Urban and Industrial Complex (QUIC) Dispersion Modeling System (Pardyjak and Brown 2001;Williams et al. 2002; Nelson et al. 2006) on the GPU for a simple continuous point source release in a uniform flow. The GPU simulations outperformed the CPU simulations by nearly three orders of magnitude. For the present paper, we again utilize the QUIC dispersion modeling framework to extend our GPU simulations to an urbanized domain with explicitly resolved buildings. We compare GPU simulation results to the standard QUIC CPU results, highlight performance gains and discuss challenges associated with implementation of

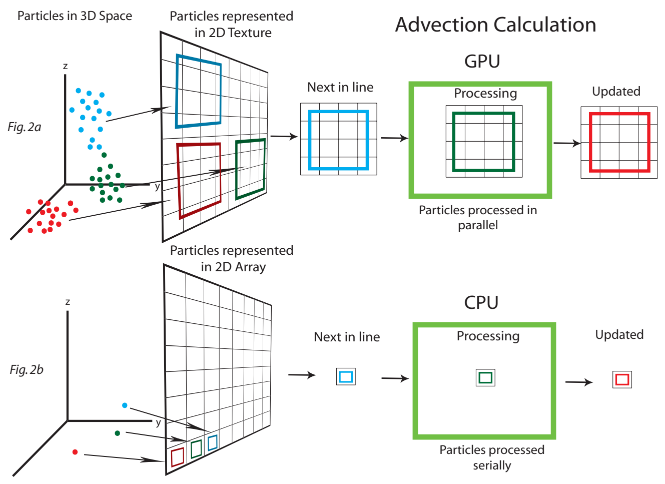
Particles in 3D Space
Particles represented in 2D Texture
Advection Calculation
GPU
Fig. 2a
z
y
Next in line
Processing
Updated
Particles processed in parallel
Particles represented in 2D Array
CPU
Fig. 2b
z
y
Next in line
Processing
Updated
Particles processed serially

Figure 1: Illustration contrasting a Lagrangian particle model's advection calculation on a standard CPU and on a GPU.

Lagrangian dispersion models onto the GPU.

In addition to performance benefits associated with this approach, the methodology provides a natural mechanism for real-time visualization of particle dispersion. Because the domain data are already loaded into the GPU memory, rendering these data to the screen after each simulation step is a fairly trivial process and affords a convenient means for visualizing the dispersion field. This is particularly important for virtual environment applications where a physical system must be integrated with a dispersion model running in near real-time for applications such as emergency response training exercises. In fact, the authors and colleagues are using the methodology developed here to model environmental flows in virtual environments (e.g., Hollerbach et al. 2005.; Kirkman et al. 2006; Kulkarni et al. 2007).

1.1 GPU SIMULATION BACKGROUND

Recently, researchers have been investigating utilization of the computational power of the GPU to solve problems that are not associated with computer graphics or rendering. The GPU, and in particular, the pixel processing (or rasterization) component on the GPU, is a highly parallel stream processor capable of floating point computations (as of this writing only 32-

* *Corresponding author address:*
Eric R. Pardyjak, University of Utah, Department of Mechanical Engineering, Salt Lake City, UT 84112, pardyjak@eng.utah.edu

bit/single precision). Pixel processing on graphics hardware has traditionally involved only the coloring of pixels on a monitor, but is now a highly programmable component in the stream processing architectures included with current graphics hardware. These stream processors are often well suited for memory bound and compute bound applications in which the computations can be transformed into SIMD (single-instruction, multiple-data) stream computations to be executed on the graphics hardware. The SIMD computations these stream processors are designed for provide data level parallelism and are the primary component used to parallelize our dispersion simulation.

Current graphics hardware (such as the NVIDIA 8000 series or the AMD/ATI R580), have up to 128 parallel stream processors responsible for the calculations usually performed on vertices, geometry, and pixels to produce graphical output. These processing elements can be programmed for SIMD operations and can operate in a highly parallel manner on large sets of data. Examples of using a GPU's stream processing functionality are numerous in the literature and include computing and applying FFT to images (Moreland & Angel 2003), numerically solving the Navier-Stokes equations (Scheidegger et al. 2005), solving multigrid problems (Bolz et al. 2003; Goodnight et al. 2003), solving dense linear systems (Galoppo et al. 2005), and cloud dynamics (Harris et al. 2003). Simple particle simulations involving one million particles have been run at interactive frame rates using the GPU (Kipfer et al. 2004). The common point of these examples is that algorithm performance on the GPU can outperform the equivalent CPU computations. However, much effort generally goes into keeping data and computation on the graphics hardware (and hence off of the CPU) to maximize performance. Our efforts build on these techniques and work towards creating a viable solution for real-time simulation and visualization of atmospheric flows.

2. GPU SIMULATION METHODOLOGY

2.1 GENERAL DISPERSION METHODOLOGY

Computing particle dispersion on the GPU requires that the simulation be constructed to fit within the constraints of the GPU architecture to take advantage of the highly parallel stream processing elements on the GPU. Program organization on the GPU is different from writing software for a CPU. The two biggest differences between traditional CPU based programs and GPU programs are how the stream processors are utilized and how memory is managed. Figure 1 illustrates the stream processing parallelism in the GPU as compared with a single processor CPU. A GPU-based particle dispersion simulation will perform the advection step on sets of particles simultaneously by utilizing the stream processors on the GPU, whereas on the CPU, only a single particle is advected at a time. In the illustration, multiple particles' positions are being update simultaneously on the GPU whereas in the CPU a single particle position is updated. After completion of one simulation time step, all particles in the 2D array will have been advected. Performing advection on the GPU requires that the advection computations or any other per-particle processing be coded into a

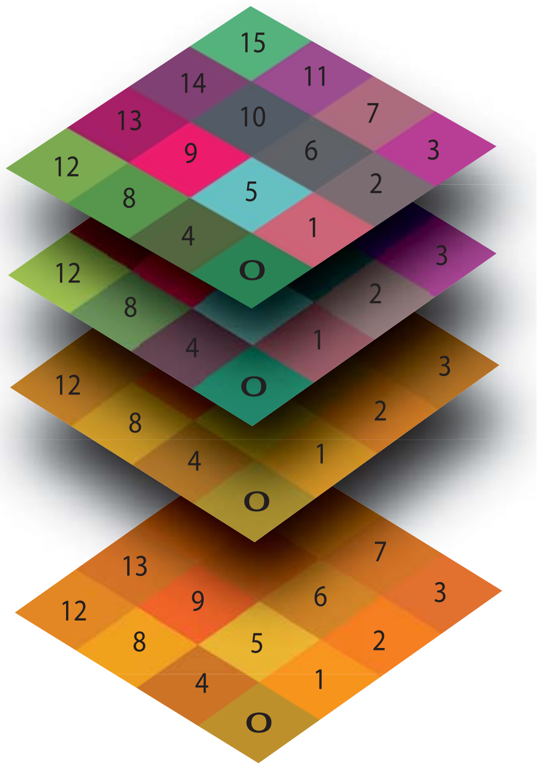

Figure 2: Illustration of the use of textures as data memory on the GPU. The rectangles shown are 2D textures that store particle velocity and position. The 2D 4x4 textures shown each represent 16 particles. Simulations with more particles will have larger 2D textures representing position and orientation (*e.g.* 1M particles = 1000 x 1000 2D texture). Each of the elements (e.g. the colored squares) of the 2D array can hold a 4 component vector, thus representing 4 attributes of a particular particle.

small program using a specialized graphics programming language. Furthermore, all data representing the particle positions, wind fields, or other quantities must be transformed into 2D textures. Textures are the primary memory structure on the GPU normally used to represent 2D images for use in texturing geometry in video games. However, in the context of general-purpose computation, such as our dispersion simulation, textures become data sources. Our primary data source is a 2D array of particle positions. This 2D array represents all of the particles active within the simulation and is the data that the stream processors operate on to perform an advection step. For instance, as illustrated in Figure 2, if we simulated a total of 16 particles, the position and velocity texture data sources would be of size 4x4. We achieve data-parallelism by encoding the particle advection computation as if it were pixels being colored or lit by the graphic processor. Rather than computing the color of a pixel, the GPU advects a single particle. Since there are upwards of 128 stream processors on modern GPUs, hundreds of particles can be operated on simultaneously. As is the case with the CPU implementation of QUIC, there is no *for loop* that iterates over each particle. We load the advection data as textures and a set of SIMD programs (often called shaders due to relationship to graphics) onto the graphics card memory. We then instruct the GPU to execute the shader programs using the loaded data. This is analogous to having the GPU redraw a scene within a game, except nothing is drawn to the screen. After the particles have been advected, we are immediately able to visualize them and show the results to the user. Note that simulations without visualization increases performance.

Data sources such as the position and velocity arrays are directly encoded into a texture array. As shown in Fig. 3, 3D data sources, such as the wind field, are transformed from 3D to a 2D texture by splaying out the vertical slices of the wind field onto a single plane. To locate the wind field velocity at or near a particle position, we use GPU texture lookup functions to lookup the velocity in the wind field texture using the particle's position as the index. Similar transformations and operations are performed to obtain the fluctuating wind field quantities.

Our application has been programmed using C++, OpenGL, and the OpenGL Shading

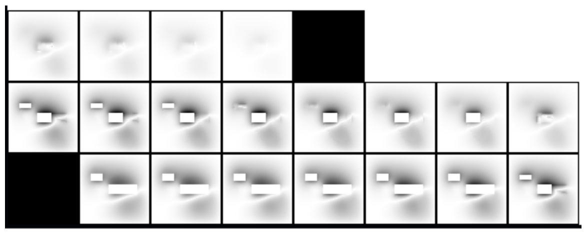
Figure 3: 3D data sources such as the wind and turbulence fields are flattened out into a 2D structure for loading into GPU texture memory.

Language. The OpenGL Shading Language is what is used to write the SIMD shader programs that run on the GPU and perform the advection steps. Our application runs on Linux, OS X, and Windows machines equipped with a modern 3D graphics card, such as those made by NVIDIA or ATI. The results reported below were run on a 2.4 Ghz Intel Core 2 Duo Processor with an NVIDIA GeForce 8800 GTS video card.

2.2 METHODOLOGY FOR IMPLEMENTING BUILDINGS

In our current implementation, we have added support for calculating particle reflection off buildings in the dispersion field. Our simulations support multiple buildings. Figure 4 shows a screen capture from our simulation with multiple buildings. GPUs naturally support reflection calculations since reflection is an extremely common graphics operation. The primary challenge with supporting building reflection is again an issue of encoding. We encode geometric descriptions (plane equations and extents of walls) of the buildings in the environment through use of a 2D texture. A dispersion field celltype texture provides us with run-time information describing whether a particle is within a structure or in free space. If a particles is in a structure, we can directly query the geometric information about the building and

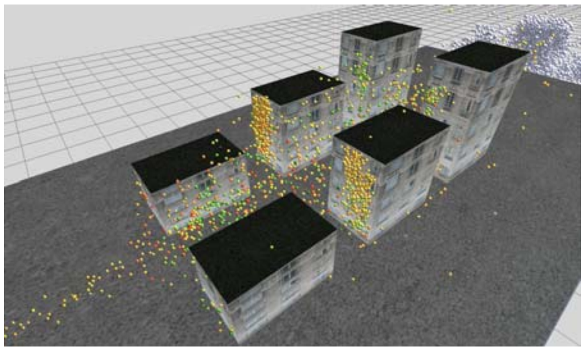
Figure 4: Graphical depiction of dispersion through multiple buildings in a realistic urban domain.

perform the appropriate reflections.

3. THE DISPERSION MODELS

The basic equations for both the QUIC GPU and CPU (Williams et al. 2002) models are shown below in Eqs. 1-5 (following *Rodean, H.C.*, 1996) where the flow has been decomposed into mean and fluctuating quantities (i.e., $u_i = U_i + u'_i$).

$$x_i = x_{p,i} + U_i \Delta t \quad + \frac{u'_{p,i} + u'_i}{2} \Delta t \qquad (1)$$

$$u'_i = u'_{p,i} + du'_i \qquad (2)$$

$$du'_1 = -\frac{C_o \varepsilon}{2}\left(\lambda_{11} u'_{p,1} + \lambda_{13} u'_{p,3}\right)dt + \left(C_o \varepsilon dt\right)^{1/2} d\xi_1 \qquad (3)$$

$$du'_2 = -\frac{C_o \varepsilon}{2}\left(\lambda_{22} u'_{p,2}\right)dt + \left(C_o \varepsilon dt\right)^{1/2} d\xi_2 \qquad (4)$$

$$du'_3 = -\frac{C_o \varepsilon}{2}\left(\lambda_{13} u'_{p,1} + \lambda_{33} u'_{p,3}\right)dt + \left(C_o \varepsilon dt\right)^{1/2} d\xi_3 \qquad (5)$$

In Eqs. 1-5, the subscript *p* indicates a particle's velocity or position at the previous time step, and $d\xi_i$ are uncorrelated, normally distributed variables with means of zero and standard deviations of 1. The tensor $\lambda_{ij} = Adj\left(\tau_{ij}\right)/\det\left(\tau_{ij}\right) = \tau_{ij}^{-1}$ is the inverse matrix of the symmetric Reynolds stress tensor τ_{ij}. To minimize differences between the two simulations, the stress tensor for the GPU was imported from the QUIC-Plume's turbulence model. Future work will include implementing a new turbulence model onto the GPU model. C_o is taken as 5.7 and the dissipation rate is given by $\varepsilon = u_*^3 / kh$, where *k* is the von Karman constant.

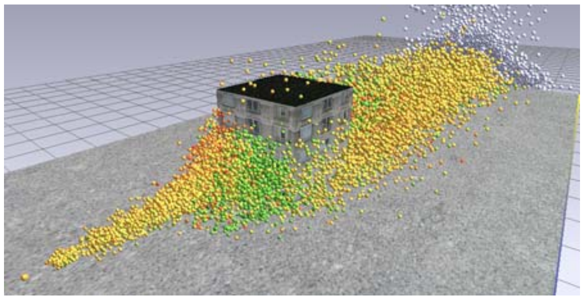
Figure 5: Graphical depiction of the single building dispersion simulation test case.

4. DESCRIPTION OF THE TEST CASE

The validation strategy used here is to compare the present GPU implementation of QUIC-Plume to the local-mixing CPU version of QUIC-Plume (v4.7). For validation, an idealized continuous point source release upstream of an isolated cube was tested.

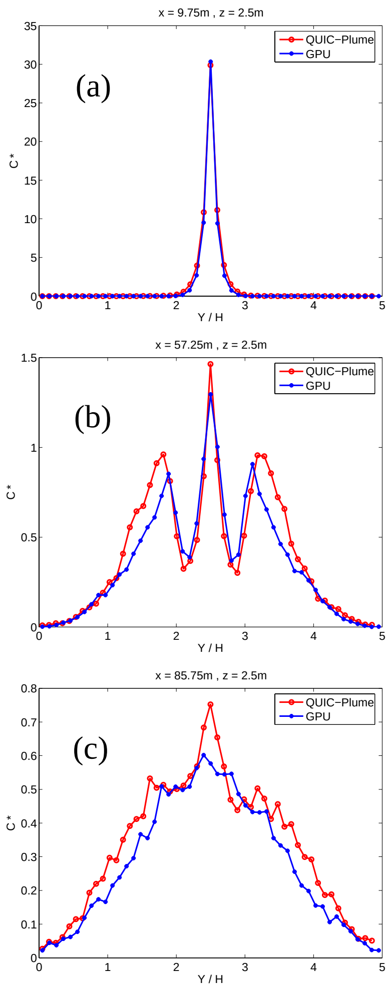

Figure 6: Dimensionless lateral concentration profile comparisons at (a) x = 9.75m, (b) x = 57.25m and x = 85.75m.

As shown in Fig. 5, a point (spherical) source of radius 0.1m was placed 20m upstream along the centerline axis of a cubical building (H = 10m) at x = 5m, y = 25m and z = 3m. The domain used for the test case was 100mx50mx20m in x, y and z directions respectively.

The velocity field for the dispersion calculations was generated using QUIC-URB. In QUIC-URB, the inlet/intial velocity profile was specified to be logarithmic with a reference velocity of 3ms^{-1} at a height of 10m. The roughness length was set to 0.1m. For this test case, 50,000 particles were released from the point source over the duration of 1000 seconds. The averaging time for the concentration estimation was 1000 seconds.

5. RESULTS AND DISCUSSION

5.1 Comparison of Results

Figure 6a shows a lateral concentration profile upwind of the building, at x = 9.75m and z = 2.5m above the ground. The GPU lateral concentration profile matches the CPU generated profile fairly well at this height. A peak concentration is seen near the source location (at y = 25m) in both GPU and QUIC produced results.

Figure 6b shows a lateral concentration profile, downwind of the building, at x = 57.25m and z = 2.5m above the ground. The peak concentration is near to the center of the building (y=25m) in line with the source location. The two counter-rotating vortices behind the building entrain the particles. The resulting concentration peak is due to this entrainment of the particles behind the building. Two small concentration dips are observed near the edges of the building (y=20m and y=30m). The GPU produced concentration profile shows less lateral dispersion, but follows the QUIC produced profile quite well at this height. We believe that the lack of lateral dispersion can be attributed to the difference in model implementation. The standard CPU implementation of QUIC utilizes a rotated co-ordinate system (Williams et al. 2004) which is not implemented in the current GPU model.

Figure 6c shows a lateral concentration profile, downwind of the building at x = 85.75m and z = 2.5m above the ground. The peak concentration is near to the center of the building (y = 25m) and in line with the source location. The GPU produced results match the CPU results well except for some small differences. These differences are due to the different random seeds used for generating random number sequences and the difference between the GPU and QUIC implementation.

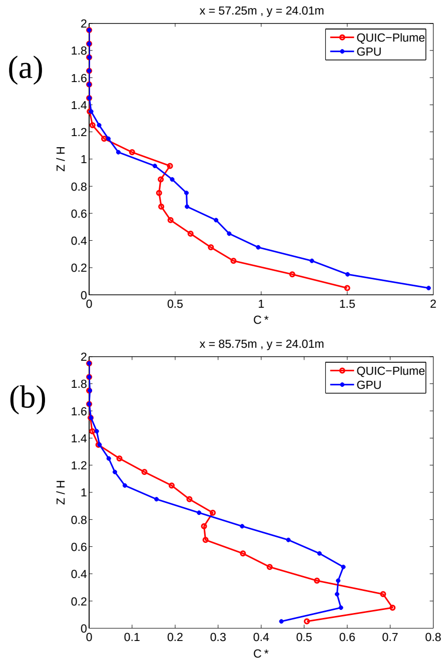

Figure 7: Dimensionless vertical concentration profile comparisons near the domain centerline at (a) x = 9.75m, (b) x = 57.25m and x = 85.75m.

Figure 7 shows vertical concentration profiles at two streamwise locations downwind of the building: at x=57.25 and y=24.01m (Fig 6a), and at x = 85.75m and y = 24.01m (Fig 6b). The concentration profiles show that GPU and QUIC produce quite similar results except for few differences. As described earlier, these differences can be attributed to the different implementations of the models.

5.2 GPU Performance Evaluation

For this work, the performance of the single processor CPU QUIC-Plume was compared to the GPU implementation for flow around a single building. The test case was an instantaneous release of N particles upwind of the buildings using the same geometry and conditions discussed in section 4. The particles were then advected for 1000 times steps. Figure 8 shows the average time required to advect N particles during the 1000 time step simulation. The GPU shows over an order of magnitude speed up over the CPU simulations. In fact for the

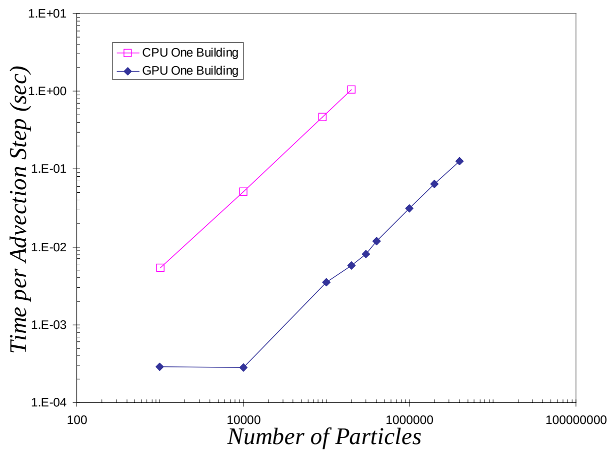

Figure 7: Average time taken to advect N particles in one time step.

simulations shown the CPU was 180 times faster than the CPU. This speed up is not as great as the increases reported by Willemsen et al. (2007) for dispersion in a uniform flow without reflection. Those simulations showed the GPU to be up to about 250 times faster than the CPU. Apparently, the additional overhead associated with additional turbulence texture storage and computations along with reflection calculations results in a performance penalty of about an order of magnitude.

5.3 Summary

Programming the GPU is currently a non-trivial task, but does provide increased performance over CPU implementations. Moreover, we are easily able to visualize the results of our simulations in real-time. Programming GPUs is likely to become easier as higher level languages are developed to access the graphics hardware. Graphics card manufacturers, such as NVIDIA, are developing additional tools to help program these cards using C APIs to access the hardware. As part of our continued work we are now investigating how NVIDIA's CUDA (Compute Unified Device Architecture) framework might compare in performance and functionality to our current implementation. For our future work, we plan to optimize our system for increased functionality, including support for generalized building structures, greater than 10 million particles, and multi-GPU configurations. Current GPUs limit a single 2D texture size to a maximum of 4096x4096 elements. This places a direct limit on the number of particles we can simulate or domain size we can handle in our simulations. To get around this memory limitation, we will need to adopt a strategy for decomposing the total number of particles into smaller sets of 2D textures (of size 4096x4096), thus advecting up to 16M particles at a time by iterating over the smaller sets. Multi-GPU arrangements may help with speeding up the advection of an increased numbers of particles since the work could be spread across the set of GPUs.

6. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the financial support of the National Science Foundation (grant IIS–0428856) and Department of Homeland Security. The authors also wish to thank Andrew Waidler for his contribution to the CUDA implementation.

7. REFERENCES

Bolz, J., I. Farmer,E. Grinspun & P. Schroder, 2003: Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. ACM Computer Graphics (SIGGRAPH 2003), 917–924.

Galoppo, N., N.K. Govindaraju, M. Henson & D. Manocha, 2005: Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware. In Proceedings of the 2005 ACM/IEEE conference on supercomputing, 1–12.

Goodnight, N., Woolley, C., Lewin, G., Luebke, D., & Humphreys, G., 2003. A multigrid solver for boundary value problems using programmable graphics hardware. In Proceedings of the ACM siggraph/eurographics conference on graphics hardware, 102–111.

Hansen, B., Singh, B., Brown, M.J. & Pardyjak, E.R., 2007. Evaluation of the QUIC-URB fast-response urban wind model for an idealized cubic building array, to be submitted to J. Wind Eng. Ind. Aero. 2007

Harris, M.J., W.V.I. Baxter, T. Scheuermann, & A. Lastra, 2003: Simulation of cloud dynamics on graphics hardware. In Proceedings of the ACM siggraph/eurographics conference on graphics hardware, 92–101.

Hollerbach, J., D. Grow, & C. Parker, 2005: Developments in locomotion interfaces, 2005

IEEE 9th International Conference on Rehabilitation Robotics, 28 June-1 July 2005, 2005, Chicago, IL, 522-525.

Kipfer, P., M. Sega & R. Westermann, 2004: Uberflow: a gpu-based particle engine. In Proceedings of the ACM siggraph/eurographics conference on graphics hardware, 115-122.

Kirkman, R., M. Deaver, E. Pardyjak & M. Metzger, 2007: Sensitivity Analysis of a Three-Dimensional Wind Tunnel Design, 2006 ASME Joint U.S.-European Fluids Engineering Summer Meeting, FEDSM 2006, July 17-20, Miami, FL, pp. 10.

Kulkarni, S.D., M.A. Minor, M.W. Deaver & E.R. Pardyjak, 2007: Output feedback control of wind display in a virtual environment, Proc. IEEE Intl. Conf. Robotics and Automation, Rome, Italy, April 10-14, 2007.

Moreland, K. & E. Angel, 2003: The FFT on a GPU. In Proceedings of ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics hardware, 112–119.

Nelson, M.A., B. Addepalli, D. Boswell, M.J. Brown, 2006: The QUIC v. 4.5 Start Guide. LA-UR-07-2799.

Pardyjak, E.R. & M.J. Brown, 2001: Evaluation of a fast-response urban wind model–comparison to single-building wind-tunnel data. in Proceedings of the 2001 International Symposium on Environmental Hydraulics. Tempe, AZ.

Pardyjak, E.R. & M. Brown, 2002: Fast response modeling of a two building urban street canyon, 4th AMS Symp. Urban Env., Norfolk, VA.

Rodean, H.C., 1996: Stochastic Lagrangian models of turbulent diffusion, The American Meteorological Society, Boston, MA, pp. 82.

Scheidegger, C., Comba, J., & Cunha, R.,: Practical CFD Simulations on the GPU using SMAC. Computer Graphics Forum, 24 (4), 715–728.

Willemsen, P., A. Norgren, B. Singh and E.R. Pardyjak, 2007: Development of a new methodology for improving urban fast response Lagrangian dispersion simulation via parallelism on the graphics processing unit. Proceedings of the 11th International Conference on Harmonisation within Atmospheric Dispersion Modelling for Regulatory Purposes, Queen’s College, University of Cambridge, United Kingdom, July 2-5, 2007.

Williams, M.D., M.J. Brown & E.R. Pardyjak, 2002: Development and testing of a dispersion model for flow around buildings. in 4th AMS Symp. Urban Env. Norfolk, VA.

Williams, M.D., M.J. Brown, B. Singh & D. Boswell, 2004: QUIC-Plume Theory Guide. LANL Report: LA-UR-04-0561.