Sound Wave Propagation Applied in Games

Marcelo Zamith, Erick Passos, Diego Brandão,
Anselmo Montenegro, Esteban Clua, Mauricio Kischinhevsky, Regina C.P. Leal-Toledo

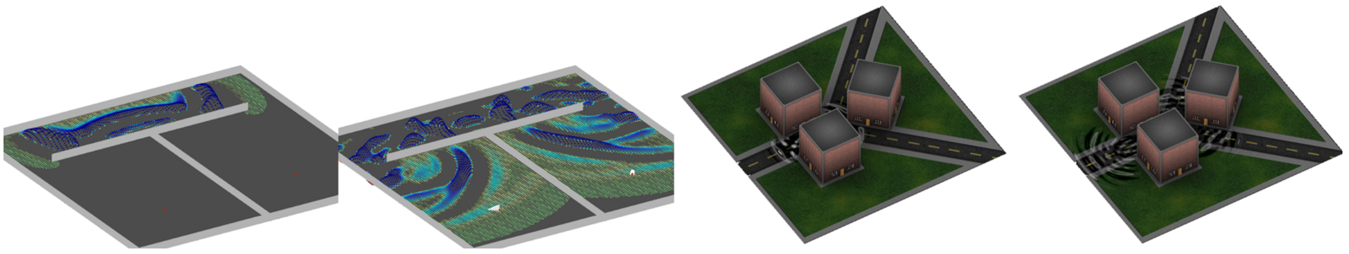

Figure 1: *Sound wave propagation*

Abstract

Many games and other interactive virtual environments are known for their focus in rendering natural phenomena, such as accurate visuals and physics, in the most believable manner. Several advances in the aforementioned fields took place during the last decade but, unfortunately, this effort has not been reflected in libraries for spatial audio. These libraries traditionally do not accurately simulate sound wave propagation through the virtual environment, never taking into consideration the speed of sound, reflection and absorbency by scene geometry, phenomena whose simulation could be used to render many interesting effects in real time. In this paper, we propose the use of a sound wave propagation simulation based on the finite difference method, running on the GPU, that can be used to compute how a sound pulse spreads through a virtual environment. In the prototypes implemented, the simulation data is interactively used to determine the perceived direction of a sound source in a closed building, and rendering a mimic of a shock-wave in an open scene.

Keywords:: Sound Wave Propagation, Finite Difference Method, Real-time Effects, GPGPU

Author's Contact:

{mzamith, epassos, dbrandao
anselmo, esteban, kisch, leal}@ic.uff.br

1 Introduction

Many electronic games, specially those that are classified as simulation-games, rely on the modeling of natural phenomena in order to immerse the player into a suspension of disbelief state. However, simulating sound wave propagation has not been considered a priority, partially due to its computational complexity. It is true that positional audio libraries are a valuable effort in this field, but the focus of the industry, and also of many academic previous and current research, has always been on providing for more realistic graphics and, to some extent, rigid-body physics.

Although positional audio libraries are now a commodity, current implementations focus only on calculating intensity and pan, based on the relative position and orientation of the sound source and the virtual listener. These libraries do not consider important issues, such as the time sound takes to travel over the environment (i.e. air) or reflection and absorbency of the objects present at the scene. Taking scene geometry into account could greatly increase the immersion experience, because it affects the way sound travels through the virtual environment, possibly changing the perceived direction from which the listener hears the generated sound. For instance, a stealth game, such as the Metal Gear franchise [Kojima], could use such simulation to adjust the perceived source position of the original sound.

Besides the aforementioned example, there are many other interesting applications of more accurate methods for sound wave propagation simulation in games. Possible uses include, but are not restricted to: mimicking an earthquake propagation on volume geometry, possibly generating force vectors for the physics engine; providing animation data for particle-based effects, such as a shock-wave; and many others.

Despite the obvious benefits, simulation methods for such phenomena are computationally intensive, and running them in real-time, concurrently with all other expensive tasks, has not been a viable option. However, current GPU technology has the computing power to run an implementation of a simulation like this in real-time, without compromising the overall performance, i.e., the GPU works as mathematics coprocessor. Considering the fact that GPUs are faster than CPU, the proposed work does not aim at showing the speed up between the technology.

In this paper, we describe how we implemented and optimized a sound wave propagation kernel in CUDA, based on a Finite Difference Method, and used it to simulate how a sound pulse, such as a gunshot or an explosion, propagates through a complex scene geometry. The related works are analyzed in Section 2, the method and details of the GPU implementation are described in Section 3. The basic data-structures are lattices to represent the velocity field and the sound wave amplitude of the environment.

This simulation kernel was developed to be completely independent of the application itself, and by monitoring the amplitude data in real time, we chose two game related problems to demonstrated the applicability of the technique:

- **Computing the perceived direction of a sound**: a closed indoor environment with two rooms and a corridor connecting them. In this example. we use the simulation data to compute the perceived direction of a sound, testing different positions for the source and listener. The result can be used to adjust the location of the sound source in the positional audio library, or using this orientation as input data to a NPC AI. The results from this example are analyzed in Section 4.

- **Mimicking a shock-wave**: an outdoor scene with some buildings, where we render a shock-wave effect from an explosion, taking into account reflection by the scene geometry. Section 5 shows more details of this application and discusses other uses of this mimicking approach.

The performance results we obtained are very encouraging, and show that our implementation is suitable for real time, even when running in low cost hardware. We also highlight the fact that no known previous research or game was found showing an implementation or proposing neither the use of accurate sound wave propagation simulation nor the methods and approach presented in this paper for real-time effects in games. Finally, the concludes are presented in Section 6 as well as some future work are discussed.

2 Related Work

Applying sound wave propagation simulation for games is mainly classified into one of the two categories, implementing a real physical sound wave propagation, using any suitable numeric method; and the second technique is making use of ray-casting techniques for sound purposes. Previous works based on numeric methods are in general too computationally expensive for real-time use. At the same time, approaches based on other techniques, such as ray-casting, do not accurately represent the physics of sound.

Many related works based on heavy numerical simulation have been developed for running on GPUs recently, but most of them are not designed for real-time. Highlights in the set of problems that have been solved with the help of GPUs include Protein Structure Prediction [Langdon and W.Banzhaf 2008], Solution of Linear Equation Systems [Bolz et al. 2003], Options Pricing [Abbas-Turki and Lapeyre 2009], Flow Simulation [Rozen et al. 2008], Wave Propagation [Balevic et al. 2008], [Michea and D.Komatitsch 2010] and others. These works try to simulate real physics phenomena through the use of more or less accurate numeric methods. Although using the fast stream processors of GPUs, none of these works is designed for real-time.

In [Mazarak et al. 1999], the authors modeled a particle system through voxels to represent of a blast wave impact on surrounding objects. The approach of this work adopts a fracture algorithm capable of accounting for the damage of multiple explosions. At another relevant work [Röber et al. 2007], an analysis of a sound wave propagation was proposed and developed, based on the computation of acoustic energy, also highlighting the possibilities to map these concepts to radiometry and graphics rendering equations. Although the authors concentrate on ray-based techniques, they also consider wave based sound propagation effects. These works are based on another family of numeric method other then Finite Difference, and the applications and performance results are not comparable to ours.

The work presented at [Nikunj et al. 2008] uses a ray-tracing approach and describe a technique to model sound propagation accurately for an arbitrary 3D scene by numerically integrating the wave equation. The performance is guaranteed by precomputing a data-structure, using eigenvalues from a refined mesh. A domain decomposition approach is done, because performing a modal analysis on the complete scene is usually not feasible. Although the proposed technique has observed up to an order of magnitude speedup compared to a standard Finite Difference technique, the processing is not completely done in real-time, which makes impossible the use with variable geometry or locations for the objects. With our approach, updating the velocity field of the lattice can be done locally for each moving obstacle.

In [van den Doel et al. 2001], an algorithm to synthesize realistic sound effects in real-time is proposed and applied for interactive simulations, such as games and animations. The sound waves are produced by the user and the proposed modal models are driven by contact forces modeled at audio rates, which are much higher than the graphics frame rate. The contact forces can be computed from simulations as well as custom designed. Finally, the work also presents the effectiveness of the technique by showing complex realistic simulations. This method is also not based on accurate modeling of the actual phenomenon.

To this date, no previous work was found that used a proper sound wave simulation method to compute the perceived relative direction of a sound source, one of the proposed applications of this paper. The use of such physically accurate methods is also not common for real-time purposes, specially in interactive applications. Most examples found in game programming literature are actually based on culling techniques, such as Potentially Audible Sets [Dickheiser 2006]. We believe that our prototypes have shown that using such accurate simulations are now possible and useful for this category of applications.

3 Sound Wave Propagation Simulation on GPU

The wave equation is a second order linear differential equation which describes the behavior of sound waves over time, amongst other types of waves, where all of them describe a medium perturbation. The acoustic wave field is described by $P(x, y, z, t)$ and $u(x, y, z, t)$, where P is the pressure field in the medium and u is the particle's displacement. The relation between pressure and particle's displacement is given by $P(x, y, z, t) = -k\nabla u(x, y, z, t)$ with k representing the volumetric compression module. One of the hypotheses of the model considered here is that the pressure field is constant along the z-axis, which implies that the partial derivative with respect to z is zero. Thus, the (2-D) wave equation with a constant volumetric compression is given by:

$$\frac{\partial^2 P}{\partial t^2} = c^2(x,y)\left[\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2}\right] + f(x,y,t) \qquad (1)$$

where, $P = P(x, y, t)$, x and y are Cartesian coordinates, t is time, c is the velocity of the acoustic wave in the medium (for instance, 300m/s in the air) and $f(x, y, t)$ is the source term (e.g., an explosion, a shotgun, etc).

The analytical solution for the problem described in Eq. 1 shall be costly or impracticable when the propagation occurs in a non-homogeneous medium. Approximation techniques are usually employed to deal with this issue. The most common family of approximation techniques is that of Finite Difference Methods (FDM), whose implementation is easier than, for example, that for the Finite Element Method. Thus, since efficiency is a key issue herein, the FDM was chosen for our simulation kernel.

3.1 Simulating sound wave propagation with FDM

Finite Difference Methods (FDM) represent the discretization of an evolutionary differential equation on a continuous region by choosing a finite set of representative points. In order to choose coefficients to weigh the importance of neighboring cells in the discrete description of the continuous problem at each point, as well as the number of neighboring points, Taylor series expansions in space and time variables are computed. Coefficients are chosen in order to provide a previously specified local truncation error. The result is a difference formula that is applicable to every point of the discretization inside the space-time domain. Such formula relates values of the unknowns on neighboring points, in space and time. Thus, the formula relates successive time steps, in order to account for the evolutionary nature of the problem.

In FDM, when one value, at some time instant, is computed through some average of the already available values at its neighboring points, the method is said to be explicit. When the calculation of the value requires the simultaneous calculation of the values at its neighboring points, the method is said to be implicit. Thus, implicit methods are often more expensive than explicit ones because they require the solution of a linear or non-linear systems of equations at each time step. For this reason, an explicit method to simulate acoustic wave propagation was chosen in this work.

A central-difference approximation can be derived from Taylor series [Mitchell 1969], and a second order approximation for space and time, assuming $h = \Delta x = \Delta y$ and $t = n\Delta t$, Eq.(1) can be written as:

$$\begin{aligned} P^{n+1}_{(i,j)} &= 2P^{n}_{(i,j)} - P^{n-1}_{(i,j)} + A \times \\ &\left[P^{n}_{(i-1,j)} + P^{n}_{(i+1,j)} - 4P^{n}_{(i,j)} + P^{n}_{(i,j-1)} + P^{n}_{(i,j+1)}\right] \end{aligned} \qquad (2)$$

where, $A = \left(\frac{c(x,y)\Delta t}{h}\right)^2$ and $n = 1, ...$ represents slice time.

In the classical approach, changes in the velocity field $c(x, y)$ reflect fluctuations of the medium. This gives rise to both reflection and diffraction of waves. Another noticeable feature of this scheme, is how open scenes can be represented, that is, non-finite domains. When employing a computational model, the computational domain size and the associated number of variables must be kept as low as possible. In order to restrict the computational domain, and thus keeping memory requirements at a minimum, while preserving a realistic description of the phenomena, artificial boundaries are introduced. This artificial chopping of the domain usually gives rise to wave reflection at the borders of domain.

Aiming to simulate semi-infinite domains, this work considers the boundary conditions proposed by [Reynolds 1978]. Such conditions are determined by decomposing the one dimensional scalar wave equation, generating the product of two terms, each representing the spread of the wavefront in one direction. Equation 3 represents this condition when the wave propagation occurs horizontally to the right. The boundary conditions for waves moving horizontally to the left can be obtained analogously [Reynolds 1978]. The conditions at the top and bottom (maximum and minimum vertical values) of the model are given by Dirichlet conditions [Golub and Ortega 1991].

$$\frac{\partial P}{\partial \vec{n}} = \frac{1}{c}\frac{\partial P}{\partial t} \tag{3}$$

3.2 Details of the chosen method

In this work, the discretized model is given by Eq. 2, where the neighborhood used in computing a value at some point in time instant $t+1$ is composed of seven elements, six from the previous time instant (t) and the last one is given by time instant $(t-1)$. Therefore, the value at a point in time $t+1$ is more influenced by its neighboring previous instant, while the time instant $t-1$ contributes with only one value. Accuracy is associated with method's order. In this case the scheme is $O(h^2, \Delta t^2)$. This means that the error approaches zero when h and Δt go to zero. More accurate methods can be used, but they require more information about the neighborhood. The approach chosen met this works' goals, providing enough accuracy while keeping computational cost low.

The complexity of the method is also defined by the discretization and neighborhood features. Thus, the lower the Δh, the greater is the number of unknows for the matrices involved. In the case of a 2D domain, for instance, given a domain of 100×100m and $\Delta h = 1$m, then matrices of size (number of rows and columns) 101×101 are involved in the solution of linear or non-linear systems. Finally, using Eq. 1, the complexity is $O(m \times n \times 6)$, where m and n are the Width and Height of the domain (that generates a lattice), respectively, and the constant 6 is the number of neighboring points for each point of the discretization.

The explicit scheme described above is less expensive than implicit methods. It is, however, conditionally stable. The stability of the explicit FDM for second order 2D wave propagation is given by the CFL (CourantFriedrichsLewy) condition. The CFL condition requires that the domain of dependence of the PDE must remain within the domain of dependence of the finite difference scheme for each mesh point of an explicit finite difference scheme. Thus, the CFL condition that guarantees the stability in the case of an acoustic wave equation is [Mitchell 1969]:

$$\sqrt{A} = \left(\frac{c\Delta t}{h}\right) \leq \frac{1}{\sqrt{s}} \tag{4}$$

where s is the dimensionality. In the present case, $s = 2$. On the other hand, being the CFL condition broken, the simulation fail and produce wildly incorrect results.

3.3 GPU Implementation

The Finite Difference Method chosen for our simulation is very well suited for a GPU implementation, since it is strongly based on data-locality (lattice based velocity and amplitude fields). Besides this, this work also made use of some features of the CUDA programming model, such as shared memory, to achieve maximum performance from the underlying hardware. This section describes some details of our GPU implementation of the aforementioned FDM.

Current GPU architectures favor arrays as data structures, as they are able to store 1D, 2D, 3D or n-dimensional versions. In this work, one only needs to use 2D arrays, because our simulation was run for matrices, and the data-structures used consist only of the amplitude field for two time instants, and a velocity field to represent the medium. These matrices must be previously allocated in the GPU dedicated memory, and must not be resized during the simulation. The corresponding memory space can be released only when the simulation ends. For instance, given the domain for a specific experiment or application, the discretization granularity must be defined before invoking the kernel, and all memory must be allocated and filled for the velocity field. The amplitude field can be initialized with zeros at the first simulation step.

The developed kernel makes heavy use of shared memory to help minimize the processing time. Hence, the kernel is divided in two steps: The first copies the pressure values in the current time for each discretized point $P^n_{(i,j)}$ from global to shared memory; the second one uses the Eq.2 to calculate the values for the next time step at each point $P^{n+1}_{(i,j)}$. Besides, in accordance to the same equation, only points in the current time instant t need to be loaded in shared memory.

The approach used maps the computation of each new value $P^{n+1}_{(i,j)}$ to a single thread. Hence, the 2D domain is divided in blocks (and threads per blocks) observing the constrains described in [NVIDIA 2010]. For instance, the maximum number of blocks that can be allocated is $65,535$, with 512 threads at each one, given a total of $33,553,920$ possible points on a grid with our approach.

The kernel configuration chosen uses a 2D block with 32 threads in one dimension and 16 threads in the other. Besides, shared memorys size is based on the number of threads plus a region called buffer border whose size is two times the neighborhood size, corresponding to the 2D stencils size. One should be aware that there is a hardware limit in the size of the shared memory per block, $16KB$ in the hardware used to run the tests described in this work.

The size of the shared memory is based on the number of threads plus one times the neighborhood size, in order to guarantee the optimization of the time access to memory. Therefore, each thread accesses the shared memory aligning its thread's unique number with shared memory positions, where these positions are neighbors of other threads. The boundaries considered for the shared memory must go beyond the number of threads to guarantee that all of them are able to access data from time instant $t-1$ from shared memory. Thus, the edges threads copy and access data from the global memory aligned with the neighboring edge threads of other blocks.

Each thread copies its data in the current time from the global memory to the shared memory. Besides, the threads at the border of a block also copy data corresponding to the data of its own neighbors which belong to a neighboring block. By doing this, the shared memory structure has all the data necessary to compute new values of the points inside a block without any additional access to the global memory. In other words, it guarantees that all threads are able to access the memory addresses corresponding to instant t from shared memory.

The second part of the kernel computes two difference equations: the first one, Eq. 2, is used for the whole domain except for the edges. The second equation is used for absorbing the wave amplitude at the edges, by using Reynods Boundary condition [Reynolds 1978].

The method is explicit in time since it provides decoupling of data, i.e., each value is defined for current (t) and previous $(t-1)$ time instants. On the other hand, this method is conditionally stable, in that the following condition must be true in order for the method to converge: $c(x,y) \times \frac{\Delta t}{\Delta h} \leq \frac{1}{\sqrt{2}}$, where v is the wave's velocity, and Δt and Δh are the discretization intervals in time and space, respectively. This stability condition was proposed in an analysis developed by [Lines et al. 1999]. Within the family of implicit

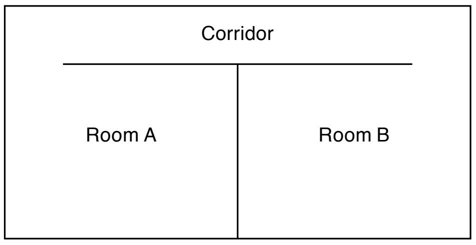

Figure 2: *Blueprint of the indoor environment*

methods, several schemes are not subjected to any constraint, thus they are unconditionally stable. In other words, any values for Δt and Δh can be used. Although implicit methods shall be stability condition free, they require a higher computational cost, because a linear system must be solved to compute the new value for each point in time step $(t + 1)$.

Nowadays, the consoles as well as home computers have multi-core architectures, which are able to process different tasks simultaneously. In this context, the GPU can be used such mathematics co-processor combined with a parallel game loop architecture [Zamith et al. 2008]. Thus, the game tasks such as AI, load data from the secondary memory and many others can be done as long as GPU processes the physics simulation.

4 Indoor Experiment

Computing the perceived direction vector of a sound is an interesting feature, specially because it can be used to augment the game mechanics, providing an extra level of accuracy for stealth games, such as the Metal Gear [Kojima] and Splinter Cell [Ubisoft] series. In this genre of games, the player often must get through complex buildings without being noticed by the enemy AI, both by avoiding eye contact and not making noises. Traditionally, checking if the AI can hear a player sound is achieved by a simple volume threshold test based on distance.

Other techniques, such as Potentially Audible Sets [Dickheiser 2006], can be used to increase the realism, but are an adaptation of a culling technique and not a proper simulation of how sound is propagated through a closed environment such as a building. Our simulation kernel is based on a proper method for computing sound propagation and reflections.

4.1 Experiment Description

In this indoor scene experiment, whose schematic view is shown in Figure 2, we represent two rooms connected by a side corridor, where the walls are sound-proof (propagation velocity equals zero), completely reflecting the sound waves.

We chose to have one sound source and two listeners at each test setup. The sound source is a position in the lattice where we generate a sound pulse at the beginning of the simulation. The two listeners are positioned at different locations at each test setup. Either source or listeners locations could be dynamically updated based on data extracted from a game engine. In our experiments, however, we chose fixed locations for the simplicity of implementation.

Each listener is represented by four connected cells of the lattice, which can be computed based on its actual position on the visual scene. To determine the relative direction from which this listener perceives the sound coming, once the sound pulse is started we monitor the amplitude value of all four cells at each time-step of the simulation. Whenever the amplitude passes a threshold level in one or two of these four cells, it is possible to classify the perceived direction of the sound source from eight (8) possible angles. Figure 3 shows the listener representation and some examples of the possibly computed angles.

We ran the tests with three different setups for the relative positions of the source and listener, as shown in Figure 4. After running

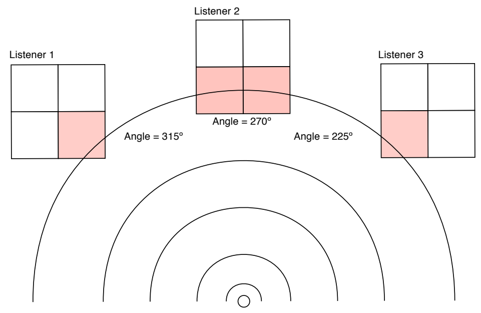

Figure 3: *Listeners*

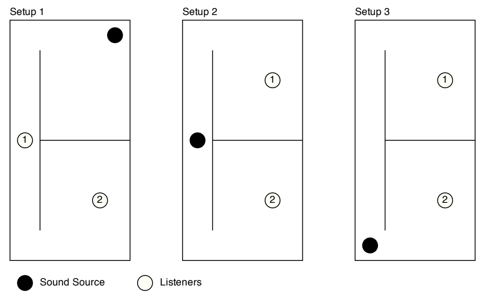

Figure 4: *Test setup for experiment 1*

some initial experiments, we realized that our algorithm was suitable even for low-end video cards, so we decided to run all tests reported in this paper in a consumer Macbook White, based on an Intel Core 2 duo and a NVidia 9400GM, a GPU with only 32 stream processors and 256MB of shared DDR3 memory. The parameters used for indoor experiments are: $\Delta h = 0.25$ meters , $\Delta t = 0.0033$ seconds, domain of 64×32 points, representing a floor of 8×4 meters. In Table 1, the column (**Exp.**) identifies the experiment number, the column (**S. P.**) represents the source position, (**P. L. 1**) and (**P. L. 2**) are the position of listener 1 and 2, respectively, while (**A. L. 1**)and (**A. L. 2**) show the perceived hearing angles for each listener, as computed by the simulation.

Exp.	S. P.	P. L. 1	A. L. 1	P. L. 2	A. L. 2
1	(4, 4)	(31,27)	0	(48,4)	135
2	(32,31)	(16,4)	45	(48,4)	135
3	(63, 31)	(16,4)	45	(48,4)	135

Table 1: *Indoor experiments*

4.2 Result Analysis

We will first describe the results in terms of the data computed for the perceived direction angles. Figure 5 shows a sequence of screenshots taken from the debug visualization of one of the test instances. One can see how the sound-wave propagates through the closed environment, finally reaching the listener. In the final screenshot, a cone shows the computed direction vector of the sound source.

Our simulation kernel is deterministic, so there was no need to perform more than one execution instance for each test setup. However, we ran and monitored several instances of the experiment, with different hardware, in order to obtain some performance data. The results showed that event for the low end card used (NVidia 9400GM) the time needed to compute each step was less than 1ms, not impacting overall performance.

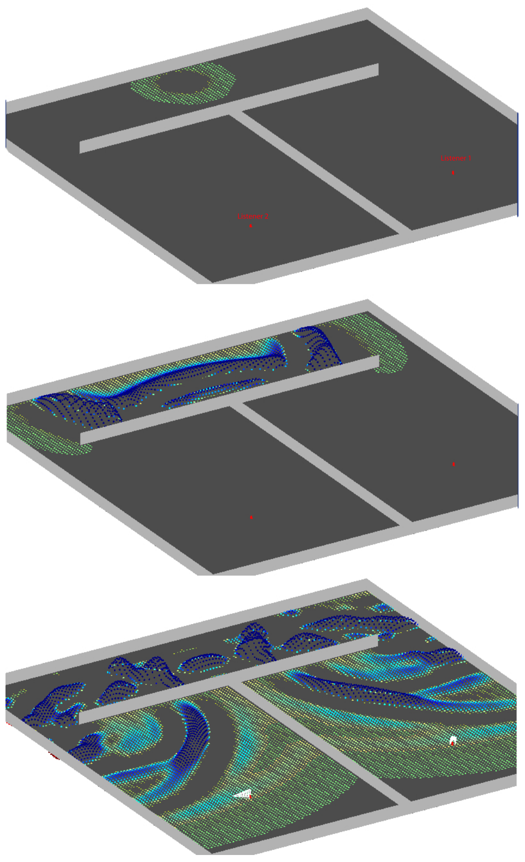

Figure 5: *Indoor experiment*

5 Outdoor Experiment

With the outdoor experiment, we wanted to mimic how a shock-wave from an explosion travels through an environment with large obstacles, such as a city block with tall buildings. The resulting amplitude field was used to render the propagation of a shock-wave-like effect at each frame-step. While not exactly simulating the physical phenomenon, a sound wave propagation is an acceptable approximation for rendering a visual effect for gaming purposes.

The outdoor environment was represented by a lattice with larger cells, and using Reynolds boundary condition [Reynolds 1978] to simulate domain continuity. The buildings were represented by zeroing the velocity of the cells that intercepted the visual models at the ground level. We included both perpendicular and diagonal buildings, as illustrated in Figure 6, to show that the chosen granularity of the lattice was enough to achieve a good visual result. The kernel parameters for this experiment are: $\Delta h = 1.0$ meter, $\Delta t = 0.0033$ seconds, domain of 128×128 points, representing a block of 128×128 meters.

We chose to run this experiment both with and without representing the buildings in the velocity field. We did this to show how the simulated wave would propagate if it did not take scene geometry into consideration. Figure 7 shows the rendered effect with the buildings geometry influencing the propagation, while Figure 8 shows how the same simulation is rendered by ignoring the walls.

As the indoor experiment, the performance results are similar, since we used data-structures of almost identical sizes.

6 Conclusion

In this paper we have shown how a wave propagation simulation can be used to augment both the mechanics and the set of visual effects available to a game designer, specially in games where better physical accuracy is a concern. We described our simulation kernel, based on a Finite Difference Method, its implementation for a CUDA-based GPU, and demonstrated its applicability with two

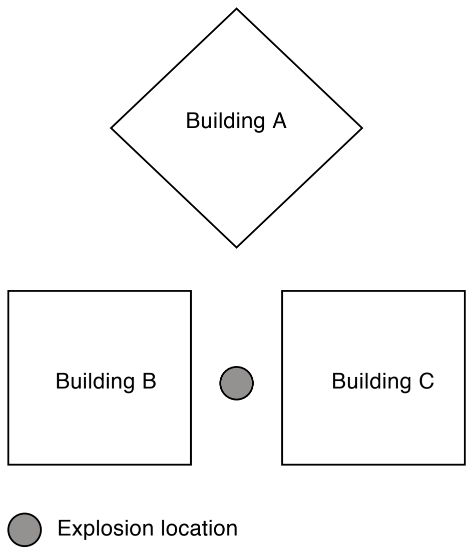

Figure 6: *Open environment representation*

real-time experiments.

In our indoor experiment, a simple algorithm was used, in conjunction with the simulation kernel, to compute an approximation of the perceived direction of a sound source. This technique can be used to augment traditional positional audio libraries, providing for a better suspension of disbelief to the player, and/or more accurate input for the enemy AI in stealth games.

Our outdoor experiment has shown how a simple mimic of a shock-wave, possibly originated by an explosion, propagating through a set of buildings compares to the same effect done without considering the reflection of the waves by the buildings. We ran both experiments with a consumer grade GPU, and the results obtained show that our technique is indeed suitable for real-time, without compromising the performance of the underling application.

While developing these applications, many other possibilities emerged, such as simulating an earthquake, to show the visual shock-wave and also computing vertical force vectors to the rigid-body physics engine. Developing a more accurate mimic of this effect is our current aim, possibly coupled with a 3D version of the simulation kernel, which will provide for much more complete amplitude data and propagation.

A weakness of our the chosen simulation kernel is that it conserves energy, so that there is no attenuation of the sound waves, resulting in a high entropy state, specially during the tests with our indoor experiment. We solved this by zeroing the amplitude of the whole field once the perceived direction vector was computed for both listeners. On the outdoor experiment, the boundaries simulated the continuation of the domain, absorbing energy and thus diminishing this undesired side-effect. To better solve this issue, we plan to apply an attenuation factor in future versions of the simulated equation.

We consider that our goals where achieved with our current implementation, as we could systematically reproduce the tests in different hardware and experiment setups, always getting the expected results.

Acknowledgements

The authors gratefully acknowledge CNPq, CAPES and FAPERJ for the financial support of this work.

References

ABBAS-TURKI, L. A., AND LAPEYRE, B. 2009. American op-

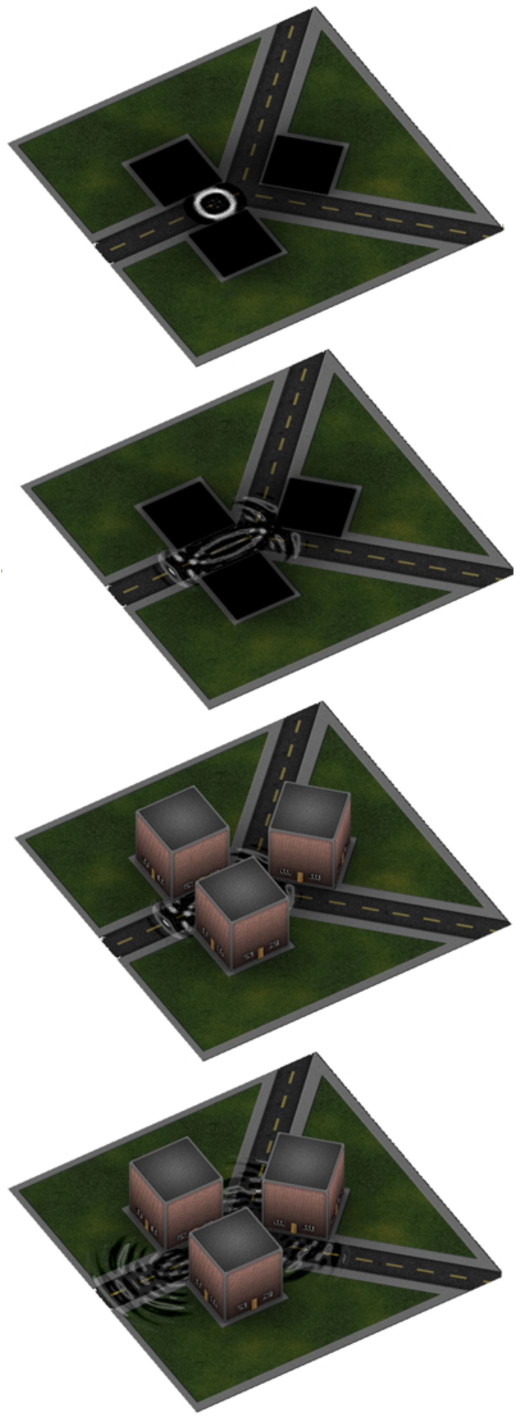
Figure 7: *Outdoor - Building with reflective walls*

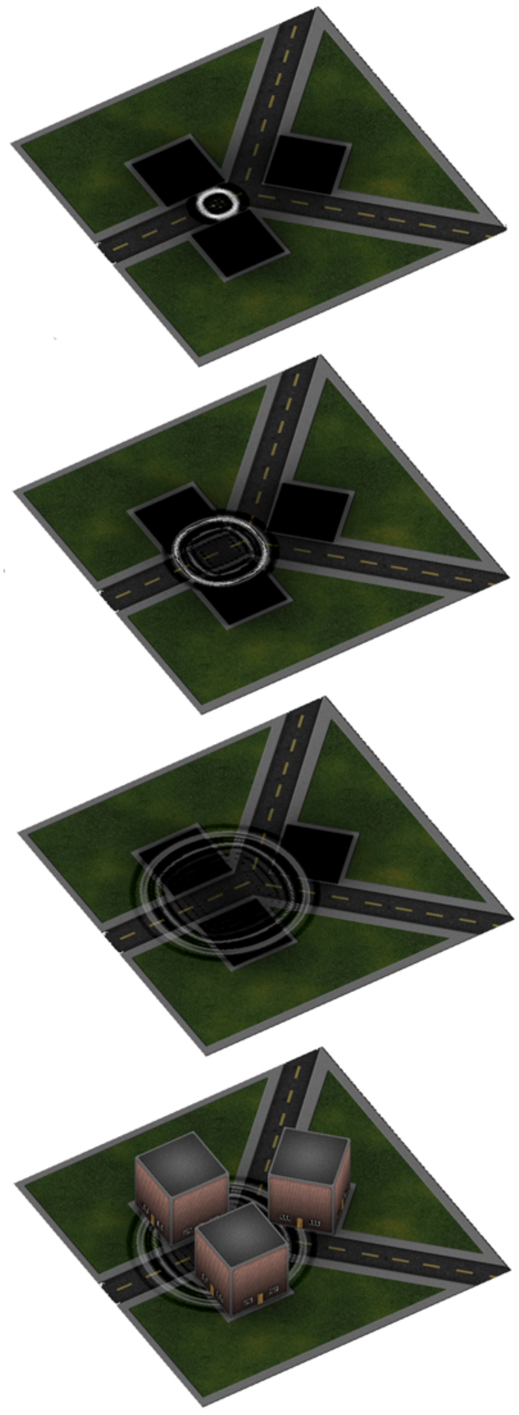
Figure 8: *Outdoor - Building without reflective walls*

tions pricing on multi-core graphic cards. *International Conference on Business Intelligence and Financial Engineering 0*, 307–311.

BALEVIC, A., ROCKSTROH, L., TAUSENDFREUND, A., PATZELT, S., GOCH, G., AND SIMON, S. 2008. Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose gpus. *Computational Science and Engineering, IEEE International Conference on 0*, 327–334.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖODER, P. 2003. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In *ACM Transactions on Graphics: Proceedings of ACM SIGGRAPH*, 917–924.

DICKHEISER, M. 2006. *C++ For Game Programmers*. Charles River Media, Inc., Rockland, MA, USA.

GOLUB, G., AND ORTEGA, J. 1991. *Scientific Computing and Differential Equations: an Introduction to Numerical Methods*, 1 ed. Academic Press.

KOJIMA, H. Metal gear solid series website. http://www.konami.jp/kojima_pro/english/index.html, accessed in: 07-19-2010.

LANGDON, W. B., AND W.BANZHAF. 2008. A simd interpreter for genetic programming on gpu graphics cards. In *Lecture Notes in Computer Science: Genetic Programming*. Springer Berlin-Heidelberg, 73–85.

LINES, L. R., SLAWINSKI, R., AND BORDING, R. P. 1999. A recipe for stability of finite-difference wave-equation computation. *Geophysics 64*, 967–969.

MAZARAK, O., MARTINS, C., AND AMANATIDES, J. 1999. Animating exploding objects. In *In Proceedings of Graphics Interface*, Morgan Kaufmann Publishers Inc, 211-218, Ed.

MICHEA, D., AND D.KOMATITSCH. 2010. Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards. *Geophysical Journal International 182*, 389–402.

MITCHELL, A. 1969. *Computational Methods in Partial Differential Equations*. John Wiley and Sons.

NIKUNJ, R., NICO, G., AND L., M. C. 2008. Accelerated wave-based acoustics simulation. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, ACM, vol. 91-102.

NVIDIA. 2010. *NVDIA - CUDA Programming Guide*. NVIDIA.

REYNOLDS, A. 1978. Boundary condition for the numerical solution of wave propagation problems. *Geophysics 43*, 1, 1099–1110.

RÖBER, N., KAMINSKI, U., AND MASUCH, M. 2007. Ray acoustics using computer graphics technology. *Proc. of the 10th Int. Conference on Digital Audio Effects* (september), DAFx–01–08.

ROZEN, T., BORYCZKO, K., AND ALDA, W. 2008. A gpu-based method for approximate real-time fluid flow simulation. *Machine Graphics and Vision International Journal 17*, 3, 267–278.

UBISOFT. Splinter cell video game website. http://splintercell.uk.ubi.com/conviction/, accessed in: 07-19-2010.

VAN DEN DOEL, K., KRY, P. G., AND PAI, D. K. 2001. Foleyautomatic: physically-based sound effects for interactive simulation and animation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 537–544.

ZAMITH, M. P. M., CLUA, E. W. G., CONCI, A., MONTENEGRO, A., LEAL-TOLEDO, R. C. P., PAGLIOSA, P. A., VALENTE, L., AND FEIJ, B. 2008. A game loop architecture for the gpu used as a math coprocessor in real-time applications. *Comput. Entertain. 6*, 3, 1–19.