Eurographics Symposium on Parallel Graphics and Visualization (2009)
J. Comba, K. Debattista, and D. Weiskopf (Editors)

Dynamic Grid Refinement for Fluid Simulations on Parallel Graphics Architectures

Marco Ament[1] Wolfgang Straßer[1]

[1]WSI/GRIS, Universität Tübingen, Germany

Abstract

We present a physically-based fluid simulation with dynamic grid refinement on parallel SIMD graphics hardware. The irregular and dynamic structure of an adaptive grid requires sophisticated memory access patterns as well as a decomposition of the problem for parallel processing and the distribution of tasks to multiple threads. In this paper, we focus on the representation and management of the dynamic grid on the graphics device for an efficient parallelization of the advection step and the iterative solving of the Poisson equation. In order to achieve high performance, we utilize the hardware's capabilities like fast cache access and trilinear filtering. Furthermore, expensive data transfer between host and device is minimized to avoid a major bottleneck. We report results on the inherent overhead of the dynamic grid compared to an equivalent Cartesian grid. In addition, a visual simulation of smoke is presented with radiosity-based illumination and volume ray casting at interactive frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

In computer graphics animation and special effects industry, fine-scaled fluid simulations like swirling smoke or spraying water remain a challenging task which originates from the complex and turbulent motion of these phenomena. Realistic animations require rich visual details like high-resolved vortices and small droplets as well as adequate interactions with the environment and convincing illumination techniques for a plausible impression. However, the computational complexity of current methods is very high, especially for the desired small features which encouraged the development of adaptive techniques to address this issue.

In the recent past, a new class of parallel high performance graphics devices emerged with attractive capabilities for general purpose computations. However, these architectures differ significantly from current CPU hardware and require specific patterns in the processing to achieve high peak rates. To the best of our knowledge, all current fluid simulations on GPUs rely either on pure particle-based methods like SPH or on regular grids which are well-suited for fast processing on these architectures. A major drawback of regular grids is the waste of computation time in empty or uninteresting regions of the domain. This holds true in particular for computer graphics because the main focus lies on visual appearance, not on rigorous physical correctness. We propose to utilize the power of current GPUs in floating point calculations and memory bandwidth in conjunction with adaptive Eulerian grid methods.

One of the main problems with adaptive grids is their dynamic and irregular structure which is contrary to the design of SIMD graphics hardware. In this paper, we present a problem decomposition that takes advantage of the specific properties of the hardware while reducing expensive hierarchy traversals. The data representations of the fluid's quantities greatly affect the overall performance. We suggest to distinguish between the storage of the velocity for the advection step and the storage of the pressure for the Poisson solver to account for the different memory access patterns.

Our work results in a flexible smoke and nebular simulation that handles complex scenes and adapts itself dynamically according to local refinement conditions. The integration in our radiosity-based visualization module offers photorealistic rendering and interactive frame rates while walking through the scene.

DOI: 10.2312/EGPGV/EGPGV09/009-015

2. Related Work

In the computer graphics context, Foster et al. [FM96] were the first who simulated the full three-dimensional Navier-Stokes equations with finite differences. The seminal work of Stam [Sta99] laid the ground for unconditionally stable fluid simulations with the Semi-Lagrangian advection scheme. Selle et al. [SFK*08] achieved second-order accuracy with a modified MacCormack method and BFECC. Fedkiw et al. [FSJ01] introduced vorticity confinement to tackle the problem of numerical diffusion and preserve fine-scaled details. Numerous authors used hybrid approaches with particles like [FF01] and [EMF02] for level-sets. Selle et al. [SRF05] presented a vortex particle method for smoke and water simulations. In CFD, adaptive mesh refinement (AMR) was introduced by Berger et al. [BO83], [BC89] for compressible flows with shock waves by utilizing overlapping grids of various sizes. Lossaso et al. [LGF04] developed an adaptive discretization on an octree data structure in the graphics context while keeping a symmetric positive definite linear system for fast PCG solvers. Another approach to fluid simulations are pure particle-based methods like SPH [MCG03], for example.

Previous works of fluid simulations on GPUs include SPH methods like [KC05] and [ZSP08]. Simulations with regular Eulerian grids include [WLL04] and [LLW04], for example. Harris et al. [HBSL03] introduced flat 3D textures to simulate cloud dynamics on a regular Cartesian grid and presented another implementation in [Har04]. A real-time simulation of smoke and water was presented by Crane et al. [CLT07].

Other GPU-related works include linear algebra operations by Krüger et al. [KW05] and iterative solvers for systems of linear equations like the Multigrid and the Conjugate Gradient (CG) method by Bolz et al. [BFGS03]. Adaptive GPU data structures were studied by Lefohn et al. [LSK*06].

3. Physically-based Fluid Simulation

3.1. Equations of Fluid Dynamics

For the physical model, we rely on the inviscid, incompressible Navier-Stokes equations for the conservation of mass and momentum:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{f}$$
$$\nabla\cdot\mathbf{u} = 0$$

The PDE describes the motion of fluids in a continuum where $\mathbf{u}$ denotes the velocity field, ρ the density, p the pressure field and $\mathbf{f}$ external forces.

3.2. Advection

The momentum equation is splitted into an advection step and a pressure solving procedure. At first, an intermediate velocity $\mathbf{u}^*$ is calculated that only accounts for the non-linear advection. We use the modified MacCormack method from Selle et al. [SFK*08] with back and forth error compensation and correction (BFECC). The method can be employed to an adaptive grid in a straightforward manner as long as the destinations of the particle trajectories can be determined and trilinear interpolation is handled correctly at resolution borders which is shown in section 4.2.

3.3. The Discrete Pressure Equations

The second step accounts for the pressure gradient. The Helmholz-Hodge decomposition leads to:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t\frac{1}{\rho}\nabla p$$

The incompressibility constraint $\nabla\cdot\mathbf{u}^{n+1} = 0$ yields the following Poisson equation:

$$\Delta p = \frac{\rho}{\Delta t}\left(\nabla\cdot\mathbf{u}^*\right)$$

We follow Losasso et al. [LGF04] with their discretization of the divergence and the pressure gradient on an octree data structure. Invoking Gauss Divergence Theorem on the integral form of the divergence yields:

$$V_{\text{cell}}\nabla\cdot\mathbf{u}^* = \sum_{\text{faces}}\left(\mathbf{u}^*_{\text{face}}\cdot\mathbf{n}\right)A_{\text{face}}$$

where $\mathbf{n}$ is the outward unit normal, A_{face} the area of a cell face and V_{cell} the volume of a cell. In the same way, the theorem is applied to the Laplacian of the pressure which actually is a $div(grad(p))$ term:

$$V_{\text{cell}}\frac{\Delta t}{\rho}\nabla\cdot(\nabla p) = \frac{\Delta t}{\rho}\sum_{\text{faces}}\left((\nabla p)_{\text{face}}\cdot\mathbf{n}\right)A_{\text{face}}$$

The remaining discretization of the pressure gradient $(\nabla p)_{\text{face}}$ is carried out in such a manner that the resulting matrix is symmetric. It was shown that the system still yields a consistent approximation when the gradients are calculated with standard central differences applied to the direct neighbour cells as long as the perturbation in the pressure location is $\mathcal{O}(\Delta x)$.

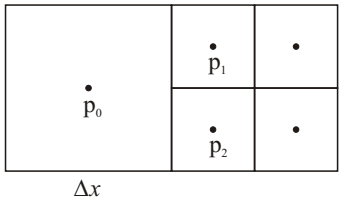

Figure 1: *Pressure discretization on octree*

Figure 1 shows a 2D example. The pressure gradient of the large cell is:

$$\sum_{\text{faces}}((\nabla p)\cdot\mathbf{n})A_{\text{face}} = \left(\frac{p_1 - p_0}{\Delta x} + \frac{p_2 - p_0}{\Delta x}\right)\frac{1}{2}\Delta x = \bar{p} - p_0$$

In this notation, $\bar{p}$ is the arithmetic average of p_1 and p_2 which comes in handy later. The discretization yields a large and sparse linear system with an equation for every cell of the grid. Fast iterative solvers like the CG-method [She94], [BFGS03] can be applied due to the symmetric positive definite matrix.

4. Parallel Simulation on SIMD Graphics Hardware

In this section, we describe our parallel implementation on graphics hardware with CUDA [NVI08].

4.1. Problem Decomposition

In order to decompose the problem appropriately for parallel processing on current graphics hardware, we suggest to employ a hierarchical grid with subdivisions of 4^3 cells instead of a classical octree with 2^3. In this way, regions of constant resolution are more likely and the hierarchy depth is usually smaller than with octrees which reduces expensive memory traversals. An octree implementation also suffers from either poor occupancy when a thread block is assigned to only 8 nodes or from a complex task distribution scheme with space filling curves, for example. With our decomposition structure, a block of 64 threads is assigned to a single sub-grid. For each grid cell one thread computes the intermediate velocity $\mathbf{u}^*$ in the advection and the pressure p in the projection step in a SIMD manner for the next time step. The parent cells that contain a sub-grid are also simulated to avoid expensive branching operations. The overhead of this procedure is only about 1.5%. Note that the above discretization of the divergence and the pressure gradient is also applicable for such a grid.

4.2. Hierarchical Grid Structure for the Advection

The advection step with the MacCormack or the Semi-Lagrangian method implies several requirements to the data structure of the hierarchical grid for efficient processing. The dominant operation is random-like read access within a spatially local area in the surrounding of each grid cell for the destinations of the particle trajectories. The second operation is trilinear interpolation of the velocity at arbitrary locations in the grid. These observations encourage a texture-based storage of velocities because read access benefits from fast texture cache as long as the access is within a local area. In addition, the texture unit also supports fast trilinear interpolation.

The basic idea is to use a pyramid of 3D texture volumes to map the grid hierarchy to memory. In contrast to a 3D mipmap only those texels that correspond to a cell in the grid domain contain valid information and are currently simulated. Figure 2 depicts a 2D example of a hierarchical grid with a staggered arrangement and the corresponding texture layout. Besides the three velocity components, a type field is stored in the alpha channel that provides auxiliary information about a grid cell, e.g. sub-grid containments or inflow and solid boundary conditions.

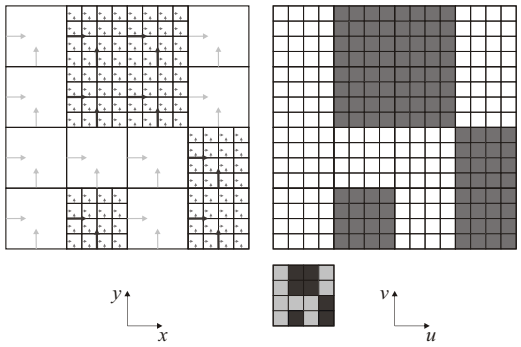

Figure 2: *Left: Hierarchical grid. Right: Corresponding texture layout.*

block	u	v	level	block	u	v	level
0	0	0	0	4	4	8	1
1	4	0	1	5	8	8	1
2	12	0	1	6	4	12	1
3	12	4	1	7	8	12	1

Table 1: *Dynamic topology table*

The light and medium grey texels represent the actual grid whereas the dark grey shaded elements describe the averaged values in the parent nodes. The arrangement of the active grid blocks within the texture pyramid is listed in a dynamic topology table (DTT). There is a row for every block of the grid with the corresponding texture coordinates of the first cell and the refinement level. The grid topology is completely defined with this table and is managed by the CPU which also takes care of distributing tasks to GPU thread blocks. The assignment of grid cells to the individual threads is achieved by reading the texture coordinates from the table entry and by adding the thread-id as an offset. The white texels in the above figure are not used and represent memory overhead but allow a fast rearrangement of the topolgy without (de)allocating memory at run time. Furthermore they are partially used to correctly handle interpolation at resolution borders on the fine level.

Trilinear interpolation requires an additional surrounding of one grid cell in areas of changing resolution levels as in the 2D example of figure 3. In the case of the coarse cell, we use the averaged velocity from the fine block which is shaded in dark grey. In this way, the interpolation is carried out solely on the coarse level. For the fine border cells, a hierarchical update is necessary. The thread block of the coarse grid fills the child-cells with interpolated values from the parent-cell which is indicated with the black dashed lines.

This procedure allows the use of fast trilinear interpolation by the texture unit in a natural manner because the correct adjacency is guaranteed on both levels.

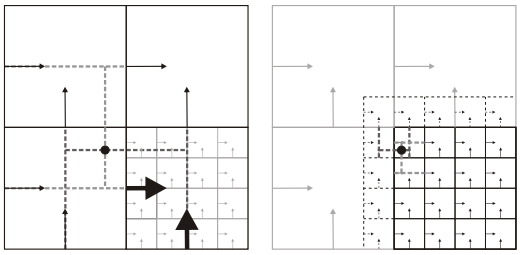

Figure 3: *Bilinear interpolation near resolution borders. Due to the staggered arrangement, a component-wise interpolation is necessary. Left: On coarse cell. The blue values are the averaged velocities from the fine cells Right: On fine cell. The dashed values are the interpolated velocities from the coarse cell.*

The MacCormack method requires the determination of the appropriate resolution level at the destinations of the particles. Lefohn et al. [LSK*06] proposed a page table that maps an arbitrary position from the virtual domain to the physical domain. We have found that it can be faster to assume that the resolution level at the destination is the same as at the starting point. As long as the hierarchy depth is not too high, e.g. only 2-3 levels, and heterogeneous areas are rare it is likely that the assumption is true in many cases and amortizes potential hierarchy traversals when the assumption fails. Furthermore, there is no need to keep the spatial layout of the page table up to date.

4.3. Hierarchical Grid Structure for the Pressure

The drawback of a texture-based data storage is the lack of a direct write access which necessitates a device-to-device copy operation of the whole grid. In the advection step, the advantage of fast read operations amortizes this problem. For example, the MacCormack method in conjunction with a second-order Runge-Kutta implies a total number of five read operations per grid cell but only one write operation. In the case of iterative solvers, write operations carry more weight. Experiments with a non-adaptive Cartesian grid have shown that a texture-based approach is significantly slower than the direct access to global device memory with coalesced read and write patterns.

However, there are several constraints which must be kept in order to achieve high performance, e.g. data elements must be accessed in sequence by the threads. For this reason, we map the hierarchical grid to an array structure and store the pressure values in it as shown in figure 4.

With a discretization as described above, almost all elements of a row in the matrix are zero except at the locations of the direct neighbour cells. Typical iterative solvers require these non-zero elements to be read from memory. For blocks of constant resolution, pressure values can be read coalesced into shared memory for further processing but the regular pattern is broken at resolution borders. Figure 5 depicts the two possibilities. Due to the discretization, the pressure of the large cell is needed in the four adjacent fine cells (medium grey) and vice versa. The left image shows the access to the coarse cell by replicating the value on the finer level (light grey) and reading it in a coalesced manner. The replication is carried out by the thread block of the coarse cell in a cooperative and coalesced manner. The right image depicts the access to the fine cells which requires a different approach. To reduce excessive non-coalesced access, we calculate the means of the pressure values (dark grey) on each cell face and store them in an additional array as can be seen in figure 4. We use them instead of reading the pressure values from the fine cells. This procedure goes perfectly along with the discretization of the pressure gradient.

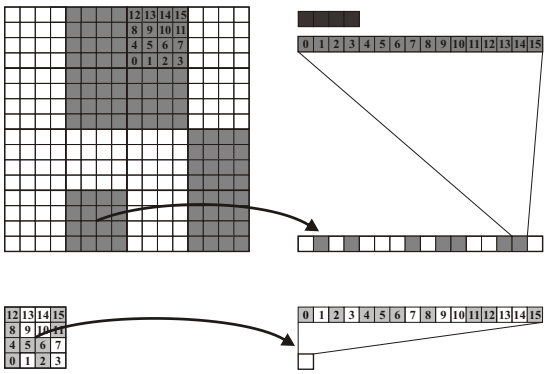

Figure 4: *Mapping of the hierarchical grid to an array structure. Pressure values are stored in sequence for every block. We calculate the averages of the pressure values on each face and store them in an additional array (dark grey)*

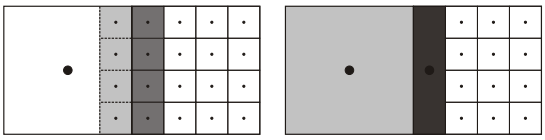

Figure 5: *Left: Access to coarse cell with replication (light grey). Right: Access to fine cells with averaged pressure (dark grey)*

We have implemented these access patterns in conjunction with the Jacobi method to solve the system for the pressure but more sophisticated solvers like the CG-method would also benefit from this procedure in the matrix-vector products.

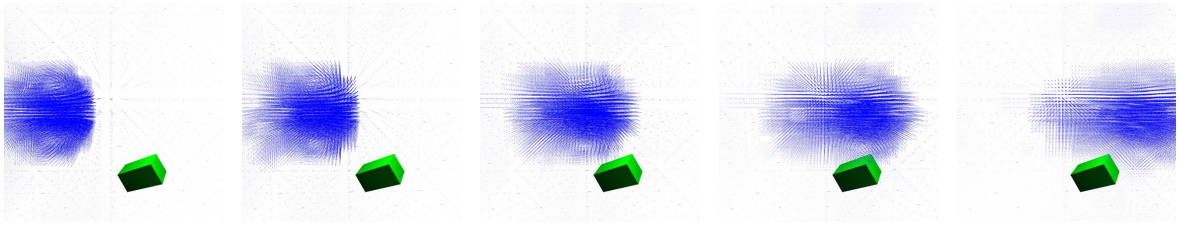

Figure 6: *Tracking a vortex with dynamic grid refinement in real-time*

4.4. Dynamic Grid Refinement

We used three common criteria [LGF04] to decide whether a cell is refined or a block is coarsened in the next time step. At solid obstacles, the grid is fully refined to capture the geometry of the scene and reduce visual artifacts when the fluid interacts with the objects. The second criterion concerns high-vorticity areas. When the magnitude reaches a certain threshold, the grid is refined. In addition, the third rule refines the grid within in a band of smoke density. The lower threshold excludes non-visible density values from high refined areas whereas the upper threshold accounts for very dense regions which are totally opaque and contribute only few visual details. Figure 6 shows an example of a vortex that is tracked with dynamic grid refinement.

The refinement and coarsening procedures are performed in parallel but as the GPU cannot administrate its own resources, data must be transferred to the CPU in order to manage the grid. For this reason, every thread block informs the CPU with the decision which cells are refined or which blocks are coarsened. With this data, the CPU is able to reorganize the dynamic topology table (DTT) and distribute tasks to thread blocks in the next time step. The DTT and the result from the decision is the only data which is transferred between host and device within one time step. Our measurements with grid sizes of about 128^3 showed that the resulting overhead is negligible (<1%) compared to a regular Cartesian grid which does not need these transfers.

5. Simulation of Smoke

We integrated the adaptive fluid simulation in our visualization module to demonstrate its application in a realistic environment. Global illumination is calculated with radiosity which allows photorealistic scenes to be walked through in real-time. In this context, we focus on smoke and nebular simulations in conjunction with realistic lighting conditions like self-shadows and indirect light while keeping interactive frame rates.

We use a standard approach with a scalar density field $_s$ to represent the smoke particles. The motion of smoke is modelled with the linear advection equation:

$$\frac{\partial \ _s}{\partial t} + (\mathbf{u} \cdot \nabla) \ _s = 0$$

The density domain is discretized with the same dynamic grid structure as the velocity and the advection is solved again with the MacCormack method. In addition, vorticity confinement [FSJ01] is added to the velocity field.

5.1. Preprocessing with Radiosity

The illumination of the density field with radiosity is carried out in a preprocessing step on a regular Cartesian grid with a resolution equal to the highest refinement level of the dynamic simulation grid to cover all possible configurations. Every grid cell gathers radiosity by aligning virtual normals in the direction of the current light source during the progressive refinement procedure. Occlusion is handled with ray tracing which offers static and volumetric shadows in the later smoke rendering process. To account for self-shadowing in the next section, we store the radiosity of the k strongest light sources separately for each voxel. The remaining radiosity is stored as a sum and is later used as an ambient-like term. The gathered radiosity is uploaded on the device before the simulation begins.

5.2. Dynamic Self-Shadowing

Self-shadowing produces rich contrast in the density distribution and is mandatory for photorealistic rendering. The self-shadowing changes dynamically with the motion of the smoke and depends on the integrated density along the shadow ray, hence a recalculation is necessary after each time step. For this reason, the method proposed by Fedkiw et al. [FSJ01] is applied to the gathered radiosity on the GPU.

The radiosity grid is ray-traced from the k brightest light sources towards each grid cell in parallel. Initially, the transparency of each ray is set to $T_r = 1$. Along the ray, samples are taken from the density texture and a local transparency is calculated with $T_s = 1 - \ _s$. For every sample, the transparency of the ray is updated to $T_r = T_r \cdot T_s$. When the ray reaches the grid cell, the gathered radiosity of the corresponding light source is reduced to:

$$B_{\text{new}} = \Omega \cdot T_r \cdot \ _s \cdot B_{\text{orig}}$$

where Ω denotes the albedo. This approach handles self-shadowing as the transparency of the ray diminishes on the way through the density field and thereby reduces the

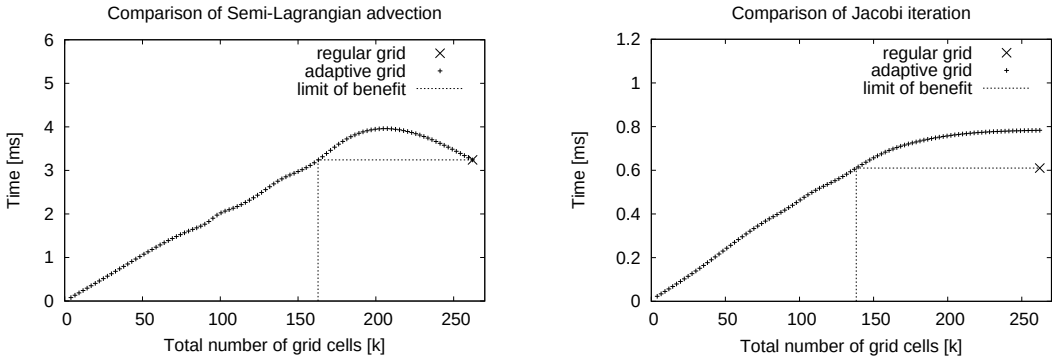

Figure 7: *Results from an adaptive grid with an effective resolution of* 64^3 *and a hierarchy depth of 2. Left: Semi-Lagrangian advection. The MacCormack method implies 2x Semi-Lagrangian steps. Right: Jacobi iteration*

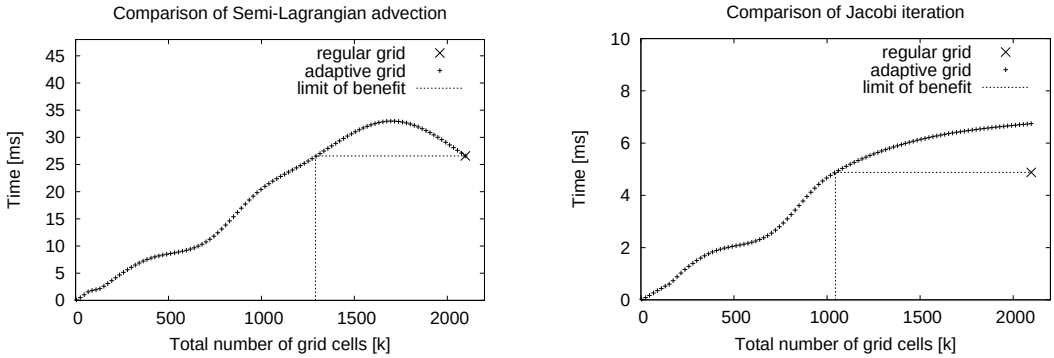

Figure 8: *Results from an adaptive grid with an effective resolution of* 128^3 *and a hierarchy depth of 2. Left: Semi-Lagrangian advection. The MacCormack method implies 2x Semi-Lagrangian steps. Right: Jacobi iteration*

amount of light that reaches the grid cell. The final radiosity is the sum of the shadowed values and the radiosity from the rest of the patches. Finally, the density and the color values are stored in an additional 3D texture which is used for rendering in the next section.

5.3. Rendering

We use a standard ray casting procedure on the GPU to render the 3D volume of the density and the lighting information. In addition, we provide simple shadows by tracing secondary rays from the intersection points of the eye rays with the geometry of the scene to the *k* most powerful light sources and integrate the density. The intersection points are calculated with the depth buffer to avoid expensive geometry processing. The final pixel color is gained by composing the ray-casted image with the frame buffer of the scene.

6. Performance Measurements

We evaluate the performance by comparing the dynamic grid with an optimized implementation for an equivalently resolved Cartesian grid. All results were obtained on a common workstation with an Intel Core2 Quad processor running at 2.4 GHz and a NVIDIA GeForce 8800 GTX with 768MB of memory.

The results from the advection step (left images in fig. 7 and 8) show an almost linear behaviour with a growing total number of cells. At about $62-65\%$ of the fully refined grid, the benefit from the adaptive grid is compensated. The oscillations in the left image of figure 8 originate from a more complex distribution of the blocks as the simulation and the dynamic grid evolves which depends on the scene and the initial condition, not on the resolution. The lowering at the end of the curve originates from the assumption of the refinement level in the particle tracing. When the resolution of the adaptive grid comes close to the fully refined grid, all assumptions are true which results in the same behaviour as the Cartesian grid.

The results from the pressure solving (right images in fig. 7 and 8) yield a similar curve. The benefit is limited at about $51-55\%$ of the fully-refined Cartesian grid. In contrast to the advection procedure, the pressure solving does not profit

from any assumptions at the end of the curve. However, there is also a lowering which originates from fewer non-coalesced read operations concerning the averaged pressure (see figure 5) when the total number of cells comes close to the fully refined Cartesian grid.

Depending on the scene, we achieved practical speedups of 1-3 compared to an equivalent Cartesian grid, especially for effective resolutions of 128^3 without loosing too much details due to a too strict refinement policy.

7. Conclusion

In this paper, we presented a fluid simulation on parallel SIMD graphics hardware with dynamic grid refinement. Our approach reduces irregularity compared to an octree implementation to accommodate for the hardware's performance constraints. We suggested data structures that cope with the individual demands of the advection step and the pressure solving routine. We integrated the simulation into our radiosity-based visualization module and achieved interactive frame rates in conjunction with photorealistic rendering of smoke. Comparisons with equivalent Cartesian grids showed that overhead is inevitable but speedups are possible if the refinement policy is well-chosen. This is not directly related to our method but is a general issue of adaptive grids. A future work could be to extend our work to multi-GPU systems like a HPC graphics cluster.

Acknowledgements

We would like to thank Ralf Sonntag for letting us use his visualization module RadioLab.

References

[BC89] BERGER M. J., COLELLA P.: Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys. 82*, 1 (1989), 64–84.

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖODER P.: Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM, pp. 917–924.

[BO83] BERGER M. J., OLIGER J. E.: *Adaptive mesh refinement for hyperbolic partial differential equations*. Tech. rep., Stanford, CA, USA, 1983.

[CLT07] CRANE K., LLAMAS I., TARIQ S.: Real-time simulation and rendering of 3d fluids. In *GPUGems 3* (August 2007), Addison-Wesley Professional, pp. 633–675.

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3 (2002), 736–744.

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 23–30.

[FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. *Graph. Models Image Process. 58*, 5 (1996), 471–483.

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 15–22.

[Har04] HARRIS M.: Fast fluid dynamics simulation on the gpu. In *GPU Gems* (2004), Addison-Wesley Professional, pp. 637–665.

[HBSL03] HARRIS M. J., BAXTER W. V., SCHEUERMANN T., LASTRA A.: Simulation of cloud dynamics on graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 92–101.

[KC05] KOLB A., CUNTZ N.: Dynamic particle coupling for gpu-based fluid simulation. *Proc. 18th Symposium on Simulation Technique* (2005), 722–727.

[KW05] KRÜGER J., WESTERMANN R.: Linear algebra operators for gpu implementation of numerical algorithms. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), ACM Press.

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 457–462.

[LLW04] LIU Y., LIU X., WU E.: Real-time 3d fluid simulation on gpu with complex obstacles. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 247–256.

[LSK*06] LEFOHN A. E., SENGUPTA S., KNISS J., STRZODKA R., OWENS J. D.: Glift: Generic, efficient, random-access gpu data structures. *ACM Trans. Graph. 25*, 1 (2006), 60–99.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 154–159.

[NVI08] NVIDIA: Cuda. Website, 2008. `http://www.nvidia.com/object/cuda_home.html#`.

[SFK*08] SELLE A., FEDKIW R., KIM B., LIU Y., ROSSIGNAC J.: An unconditionally stable maccormack method. *J. Sci. Comput. 35*, 2-3 (2008), 350–371.

[She94] SHEWCHUK J. R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep., Pittsburgh, PA, USA, 1994.

[SRF05] SELLE A., RASMUSSEN N., FEDKIW R.: A vortex particle method for smoke, water and explosions. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM, pp. 910–914.

[Sta99] STAM J.: Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 121–128.

[WLL04] WU E., LIU Y., LIU X.: An improved study of real-time fluid simulation on gpu: Research articles. *Comput. Animat. Virtual Worlds 15*, 3-4 (2004), 139–146.

[ZSP08] ZHANG Y., SOLENTHALER B., PAJAROLA R.: Adaptive sampling and rendering of fluids on the gpu. In *Proceedings Symposium on Point-Based Graphics* (2008), pp. 137–146.