

A Portable, Parallel, Real-Time Animation-System for Turbulent Fluids

Mattias Bryborn Reinhard Klein Thorsten May Sascha Schneider Andreas Weber*

Fraunhofer-Institut für Graphische Datenverarbeitung
Rundeturmstr. 6, Darmstadt, Germany
{rklein, sschneid, aweber}@igd.fhg.de

Abstract

We describe our parallelization of an unconditionally stable solution scheme of the Navier-Stokes equation that has recently been used for animation purposes. Our parallelization gives good speed-ups on current multi-processor workstations. These speed-ups close the gap that existed towards real time animation for several 3D-examples. In addition to parallelizing the solver software the integration of a fast 3D-volume renderer into the parallel framework has been an important step to achieve this goal. Our parallelization is portable on workstations and PC's running under the UNIX or the WIN32 operating systems.

Keywords: Parallel/distributed algorithms; performance evaluation and measurements; fluid simulation; volume rendering

1 Introduction

Some of the most fascinating observations one can make in nature can be explained as effects of turbulent fluids or gases. So simulating turbulent fluids is not only a major topic in engineering but *computational fluid dynamics* (CFD) has also become a topic in computer graphics and animation [1, 2, 3, 4, 5, 6]. However, in an animation context the topic of real time simulations is much more important than in a general engineering setting, whereas the accuracy of the simulations can be much less in general: very often the result of the simulation has to be just “visually right”. So the introduction of an unconditionally stable solution scheme of the Navier-Stokes equations into the area of computer graphics [6] was seen to be an important step by the graphics community.¹ Using this scheme an animation system that allows the real time simulation of 2D-turbulent fluids was implemented. For 3D-examples the described animation system was about a factor of 4 slower than real time.

In this paper we show that this stable solution scheme is well suited for parallelization and we describe a parallel implementation that is portable on workstations and PC's running under the UNIX or the WIN32 operating systems. We achieve good speed-ups closing the gap that existed towards real time

animation for several 3D-examples. In addition to parallelizing the solver software the integration of a fast 3D-volume renderer into the parallel framework has been another important step to achieve this goal.

Although there is a lot of work on parallelizing programs for computational fluid dynamics we are not aware of any prior attempt in this direction for the stable solution scheme and its specific components described in [6].

2 Preliminaries

There is a vast literature on computational fluid dynamics. For some recent textbooks we refer to [9, 10, 11, 12, 13]. For the specifics of the sequential algorithm that is the basis of our work we refer to [6].

Since for real time animations a parallelization over a network has a too high latency, we focus on shared-memory architectures, which are nowadays available in the form of relatively low cost multi-processor PCs or workstations. The general programming paradigm that is available for these platforms is the one of multi-threaded programming. Although quite similar from a programming point of view the APIs that are offered by the WIN32 operating systems and the various UNIX systems differ. However, it is possible to provide abstraction classes with little performance overhead that allow a platform independent access to the thread systems. The *Adaptive Communication Environment* (ACE) [14] and the OMNI thread package [15] are two such packages. Currently, we use the OMNI thread package as an platform independent abstraction for the threads of the WIN32 operating systems and for POSIX threads, but switching to another one like ACE would be possible with a moderate programming effort.

The windowing classes that we use are completely written in Qt [16] in order to allow a platform independent implementation. The graphical context of the window is filled with rendered OpenGL graphics [17]. These two together allow our program to run on all Operating Systems which support Qt and OpenGL.

Thus our system in particular runs under the WIN32 operating system and most of the UNIX systems.

*Corresponding author. Fax: +49 (6151) 155-139. Phone: +49 (6151) 155-639.

¹The “almost sufficiency” of staying in the stability region of a solution scheme for animation purposes has also been used in other contexts, cf. [7, 8].

3 The Sequential Algorithm

3.1 Simulation Program

The simulation program solves the Navier-Stokes equations

$$\nabla \cdot \mathbf{u} = 0 \quad (1)$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f} \quad (2)$$

for an incompressible fluid, where \mathbf{u} is the velocity vector field, ν is the viscosity, d the density, p the scalar pressure field and \mathbf{f} the external forces acting on the fluid, e. g. buoyancy.

In an incompressible fluid the role of the pressure is to allow continuity to be satisfied. It is therefore possible to write eqn. 1 and eqn. 2 as just one equation using an operator P , which orthogonally projects a vector field on to the set of divergence free fields. Using this operator the equations 1 and 2 can be written as

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}(-(\mathbf{u} \cdot \nabla)\mathbf{u} - \nu\nabla^2\mathbf{u} + \mathbf{f}) \quad (3)$$

It is this equation that is solved by the simulation program.

A fractional step method consisting of the following parts is used to solve the equation. The method we use for the solution is the described in [6], to which we refer for more details and a theoretical base for this method. We just sketch the main steps as a reference for the parallelization.

$$\begin{array}{c} \mathbf{w}_0(\mathbf{x}) \\ \xrightarrow{\text{project}} \\ \mathbf{w}_4(\mathbf{x}) \end{array} \xrightarrow{\text{addforce}} \mathbf{w}_1(\mathbf{x}) \xrightarrow{\text{advect}} \mathbf{w}_2(\mathbf{x}) \xrightarrow{\text{diffuse}} \mathbf{w}_3(\mathbf{x}) \quad (4)$$

3.1.1 Add force

The effect of the external forces is solved using an ordinary explicit Euler scheme.

$$\mathbf{w}_1(\mathbf{x}) = \mathbf{w}_0(\mathbf{x}) + \Delta t \mathbf{f}(\mathbf{x}, t) \quad (5)$$

3.1.2 Transport

Using a technique based on the method of characteristics the nonlinear transport (advection) part of the equation is solved. This method has several advantages. Two of them are ease of implementation and—as will be shown later—ease of adaptation to parallel computing. A point \mathbf{x} is back-traced along a streamline in the old vector field to time $t = -\Delta t$. The velocity at \mathbf{x} in the new vector field is then set to the value at the back-traced point. Velocity vectors not directly on a grid-point are defined using trilinear interpolation from the adjacent grid-points. Using a function $\mathbf{p}(\mathbf{x}, t)$ defined as the streamline passing through point \mathbf{x} at $t = 0$ the step can be written as

$$\mathbf{w}_2(\mathbf{x}) = \mathbf{w}_1(\mathbf{p}(\mathbf{x}, -\Delta t)) \quad (6)$$

A particle tracer can easily be implemented using any standard ODE solver. The choice for the program described in this paper is a Runge-Kutta solver of the fourth-order.

3.1.3 Diffuse

The third step deals with the effect of viscosity. This step calculates the solution of a standard diffusion equation for each Cartesian component of the vector field. These equations are solved using an implicit Euler scheme, so the resulting system is

$$(\mathbf{I} - \nu\Delta t\nabla^2)\mathbf{w}_3(\mathbf{x}) = \mathbf{w}_2(\mathbf{x}) \quad , \quad (7)$$

where the ∇^2 operator is approximated using finite differences. Several efficient methods for solving such equations exist. Specifically in the implementation described in this paper the `pois3d`-solver from the FISHPAK library is used.²

Although this routine has a complexity of about $O(n \log n)$ and this equation can be solved theoretically in $O(n)$ using a multi-grid method [18], the practical performance of `pois3d` is much better on the currently used grids: For grids consisting of about 4000 cells the `pois3d` routine is about 10 times faster than the best of the multi-grid methods that we have tested, for grids consisting of about 125000 cells it is still about a factor of 9 faster.³

If the viscosity is 0, an approximation that might be made for air, the diffusion step can be omitted.

3.1.4 Project

The vector field produced by the above steps are not ensured to satisfy the continuity equation. The final step is therefore a projection step which makes the field divergence free. The step can be written as

$$\nabla^2 q = \nabla \cdot \mathbf{w}_3 \quad (8)$$

$$\mathbf{w}_4 = \mathbf{w}_3 - \nabla q \quad (9)$$

This Poisson equation is solved using the same subroutine that is used in the diffuse step.

3.1.5 Computing scalar quantities

The evolution of a scalar quantity, e. g. temperature or smoke density in the fluid is computed using a method very similar to the one for the vector field described above. The equation that describes the behaviour of the scalar field is

$$\frac{ds}{dt} = -\mathbf{u} \cdot \nabla a + \kappa\nabla^2 s - \alpha s + S \quad , \quad (10)$$

where κ is the diffusion constant, α the dissipation rate and S a source term. All terms are solved using the same steps that are used for the vector field except of *project*, which is not needed. The dissipation term not present in the Navier-Stokes equations is solved using the following equation

$$(1 + \Delta\alpha)s(\mathbf{x}, t + \Delta t) = s(\mathbf{x}, t) \quad .$$

²FISHPAK is available from <http://www.netlib.org>.

³The computation time of `pois3d` strongly depends on the largest prime factor of the grid dimension (plus one), e. g. the computation time on a cube of $(32-1)^3$ -cells is much smaller than the one on a cube of $(31-1)^3$ cells! The given factors are the ones for the “good” grid sizes. For “bad” grid sizes the factor is about 5.

3.2 Volume Renderer

For efficient volume rendering a splatting algorithm [19] is used. Instead of the accumulation of the volume data along rays as in a standard ray casting algorithm, volume elements are projected onto the image plane. To make use of current graphics hardware the volume elements are constructed as plane geometric primitives, e. g. a triangle fan around a vertex. The possibility to use transparent objects is used to model the thickness of the volume elements, see [19] for details. The plane volume elements are rotated depending on the viewpoint in such a way that the plane normal points towards the viewer.

These volume elements (e. g. OpenGL “QUADS” or “TRIANGLE FANS”) are mapped with a texture. On newer systems already supporting OpenGL 1.2, we can use 3D textures to draw onto the primitives. This gives a fast way to sample down the rendering space into small pieces and to use the abilities of inexpensive modern graphic cards (e. g. the GeForcetm). In this space we gain speed out of the fast triangle rendering GPUs on average PCs and Workstations, cf. Fig. 1.

If the system only supports OpenGL 1.1 or lower, we can only make use of 2D textures. In that case we have to implement the handling of 3D textures by hand or to limit ourselves to 2D textures in a special way. However, in the second case we can take advantage of another nice feature of OpenGL: We can define several 2D textures by ordinary images and stack one above the other to form a 3D space. If we then decide to fix our viewpoint to the front of the stack, we can render the whole volume just by drawing planes (or geometric primitives) beginning with the plane in the greatest distance up to the nearest plane.

It is also necessary to include opacity and transparency values of the material to be rendered. This is also very simple. All we have to do is to set the correct alpha values for the OpenGL textures. The OpenGL system supports the rendering of alpha weighted textures and so we obtain easy support even for materials with varying opacities.

This is the main idea behind the renderer. The volume-rendering subsystem has to work together with the solver thread of the CFD simulator to allow animations of moving fluids. By using geometric primitives and applying texture coordinates to them as is explained above we are able to draw optically high-resoluted fluids even if the CFD calculations are performed on time conserving low-resoluted grids. The texture coordinates, which are defined at the edges of the primitives, are then moved by the interpolated velocities in the CFD grid.

By using this techniques it is possible to animate high-resoluted fluids by evaluating the CFD equations only in a much rougher grid and then to perform a fast volume rendering of the solution.

The method of using textures for optical refinement of a flow has also been used in [6, 20].

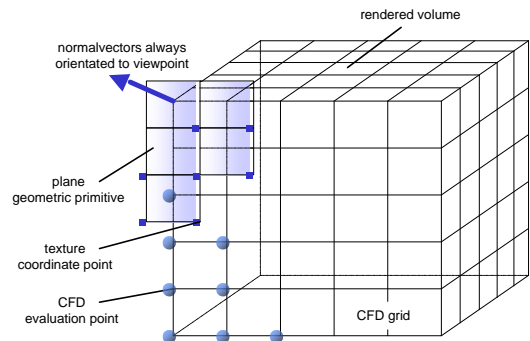


Figure 1: Volume renderer

4 Parallel Architecture

The sequential components described above can be split into various parallel components. In the following we will give a “top down” description of the parallelized components: First, we give the top-level parallel architecture between the renderer and solver components giving then the details on the parallelization of the solver components itself, starting again from the higher levels.

4.1 Communication and synchronization between renderer and solver components

In our program the renderer (OpenGL/QT window) interacts with the CFD solverthread using a double-buffer. The part of the double-buffer that is used by the renderer is called “current renderer buffer”. It contains the current velocity field and entities used for rendering, e. g. the current texture coordinates. After the solver thread has completed one calculation-step and has written its results in its “current solver buffer” (the other part of the double-buffer), it (passively) waits for the renderer to finish the current frame and then switches the pointers to “current renderer buffer” and “current solver buffer”. Thus only a switch of pointers has to be locked. A schematic view is given in Fig. 2.

4.2 Parallelization of simulation program

We compute the vector field of velocities and the scalar quantities at discrete time-steps. The simulation of a the scalar quantities at time t_n requires the knowledge of the vector field at this time step. However, in a simulation loop the simulation of the vector field for time step t_{n+1} (using the values of the vector field at time step t_n) can be done in parallel to the simulation of the time step of the scalar field at time step t_n (using

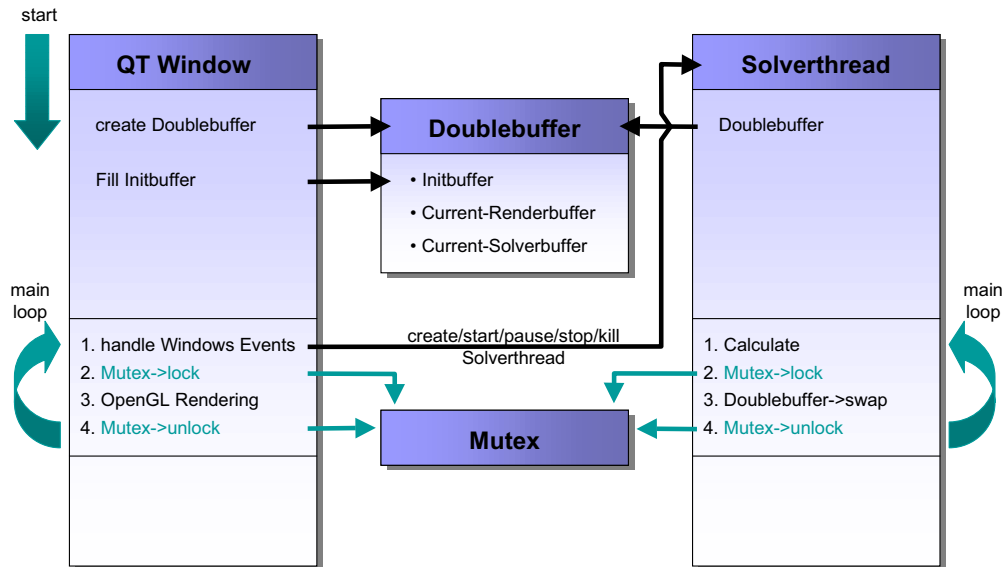


Figure 2: Communication and synchronization between renderer and solver components

the value of the scalar field t_{n-1} and the vector field at time step t_n).

Any simulation step of the vector field requires the computation of the sequence *add force* \rightarrow *transport* \rightarrow *diffuse* \rightarrow *project* (and similarly for the scalar quantities). These steps have to be taken sequentially.

The *add force* step takes little time but the addition of any force to a cell can be done in parallel. In the sequential program the *transport* step requires a major part of the computation time of the vector field simulation (about 40 % for the examples given below). This step can be well parallelized without synchronization requirements: the computation for any grid cell is independent of each other. Using n threads, where n is about four times the number of available processors to reach saturation, we divide the cells in n parts and do the transport computation in the n threads, each responsible for one part. As the computation of transport step roughly takes the same time for any grid cell these threads can be expected to have the same run-time.

The *diffusion* step requires a solution of the corresponding equations in 3 dimensions, which are independent of each other. So we can use 3 threads to do these computations in parallel, which require no synchronization and which have nearly the same run-time. Parallelizing the diffusion computation for each dimension themselves is possible in principle but cannot be done without synchronization. The fast *pois3d* routines themselves are relatively hard to parallelize but because of their much better performance we leave them as unparallelized components, which run much faster than other parallelizable components, cf. Sec. 2. As the performance mea-

surements given in Table 1 show we already have very good speed ups even with these components unparallelized.

The *project* step requires the solution of one equation using *pois3d* and is thus currently unparallelized for the same reasons discussed above in the diffuse step. It will become a sequential bottleneck for a higher number of processors; for a smaller number of processors there is still good parallelism due to the parallel execution of renderer and the scalar field computations.

The corresponding parallelization scheme for the vector field solver is given in Fig. 3, the one for the scalar field solver and the other quantities used for visualization (particles, texture coordinates) are given in Fig. 4.

5 Performance Measurements

In Table 1 the absolute computation times for the transport and diffuse steps and their speed-ups on a 4 processor SGI Onyx and a 4 processor SUN ULTRA E 450 are given. For the transport step a theoretical speed-up of 4 is possible, for the diffuse step the theoretical speed-up is 3. The given absolute times are wall clock times and the averages of 10 runs. For the measurements only the vector field computations have been performed.⁴

⁴The results of several animations (using 6750 and 4096 cells) can be found at <http://www.igd.fhg.de/igd-a3/projects/physically-based-simulation-and-animation/cfd/>.

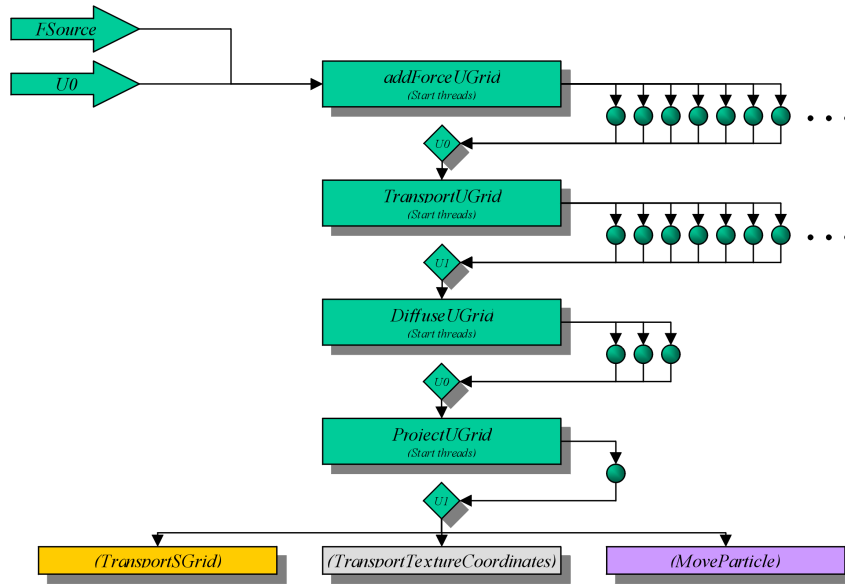


Figure 3: Parallelization of vector field solver

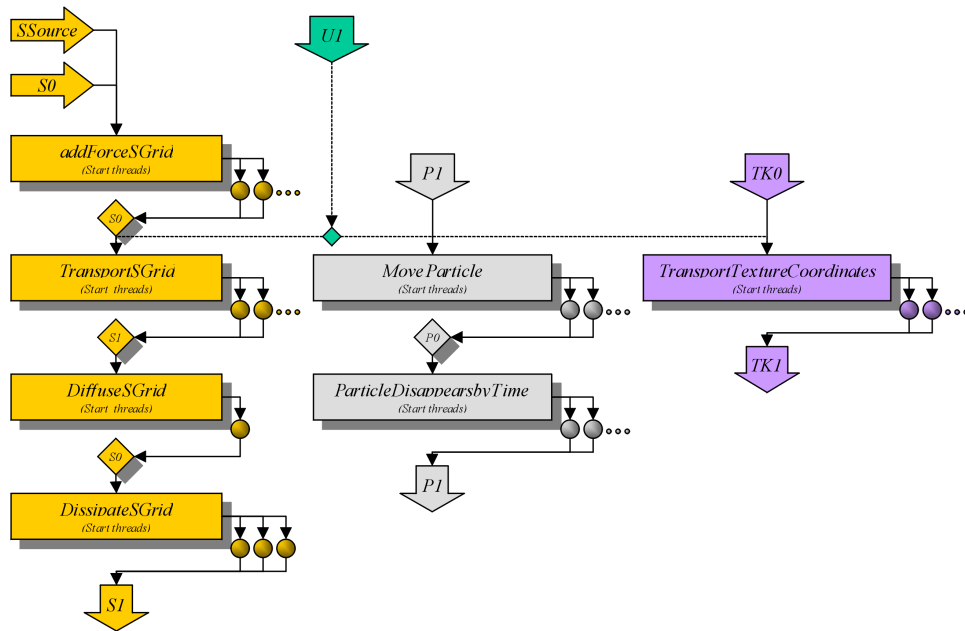


Figure 4: Parallelization of scalar field solver

Grid size (# cells)	machine	<i>transport</i>			<i>diffuse</i>		
		time (msec)			time (msec)		
		seq.	par.	speed-up	seq.	par.	speed-up
3375	SGI	24	12	2.0	29	16	1.8
3375	SUN	23	8	2.8	51	21	2.4
4096	SGI	28	13	2.1	23	15	1.5
4096	SUN	28	10	2.7	39	16	2.4
6750	SGI	46	20	2.3	43	27	1.6
6750	SUN	48	16	3.0	87	36	2.4
29791	SGI	221	70	3.1	171	99	1.7
29791	SUN	232	77	3.0	304	138	2.2
117649	SGI	1060	258	3.8	912	465	1.9
117649	SUN	927	275	3.4	1275	559	2.3
125000	SGI	1052	275	3.8	1136	662	1.7
125000	SUN	1049	278	3.7	2180	902	2.4

SGI: 4 processor Onyx 2

SUN: 4 processor ULTRA E 450

seq.: sequential algorithm

par.: parallel algorithm on 4 processors

Computation times are averaged wall clock timings in milliseconds for one simulation step.

Table 1: Speed-ups on 4 processor workstations

6 Conclusion

We have shown that the unconditionally stable solution scheme of the Navier-Stokes equation that has recently been used for animation purposes can be parallelized with good speed-ups on current multi-processor workstations. These speed-ups close the gap that existed towards real time animation for several 3D-examples. In addition to parallelizing the solver software the integration of a fast 3D-volume renderer into the parallel framework has been an important step to achieve this goal.

Our parallelization is portable on workstations and PC's running under the UNIX or the WIN32 operating systems. Because of the improving performance of relatively low cost multiprocessor PC's our system will also allow real time fluid animations on this rapidly growing hardware segment.

Acknowledgment. We are grateful to P. Borodin and F. Birra for their help in implementing parts of the system, and to W. Küchlin for the possibility to test our program on a SUN ULTRA E 450.

References

- [1] J. Stam, E. Fiume, Turbulent wind fields for gaseous phenomena, in: *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Anaheim, CA, USA, 1993, pp. 369–376.
- [2] J. Stam, E. Fiume, Depicting fire and other gaseous phenomena using diffusion processes, in: *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Los Angeles, CA, USA, 1995, pp. 129–136.
- [3] N. Foster, D. Metaxas, Modeling the motion of a hot, turbulent gas, in: *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Los Angeles, CA, USA, 1997, pp. 181–188.
- [4] J. X. Chen, X. Fu, E. J. Wegman, Real-time simulation of dust behavior generated by a fast traveling vehicle, *ACM Transactions on Modeling and Computer Simulation* 9, 1999, 81–104.
- [5] H. Weimer, J. Warren, Subdivision schemes for fluid flow, in: *SIGGRAPH 99 Conference Proceedings* [21].
- [6] J. Stam, Stable fluids, in: *SIGGRAPH 99 Conference Proceedings* [21], pp. 121–128.
- [7] D. Baraff, A. Witkin, Large steps in cloth simulation, in: M. Cohen (Ed.), *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Orlando, FL, USA, 1998, pp. 43–54.
- [8] M. Desbrun, P. Schröder, A. Barr, Interactive animation of structured deformable objects, in: *Graphics Interface '99*, Kingston, Canada, 1999, <http://www.multires.caltech.edu/pubs/GI99.pdf>.
- [9] B. R. M. Donald F. Young, T. H. Okiishi, *A Brief Introduction To Fluid Dynamics*, John Wiley & Sons, 1997.
- [10] F. M. White, *Fluid Dynamics*, 3rd Edition, McGraw-Hill, 1994.

- [11] J. H. Ferziger, M. Peric, *Computational Methods For Fluid Dynamics*, 2nd Edition, Springer Verlag, 1999.
- [12] M. B. Abbott, *Computational Fluid Dynamics : An Introduction for Engineers*, Wiley, 1989.
- [13] A. J. C. J. E. Marsden, *A Mathematical Introduction to Fluid Mechanics*, 2nd Edition, Vol. 4 of Texts in Applied Mathematics, Springer-Verlag, 1990.
- [14] D. C. Schmidt, The ADAPTIVE Communication Environment: An object-oriented network programming toolkit for developing communication software., <http://www.cs.wustl.edu/~schmidt/ACE-papers.html> (1993).
- [15] Tristan Richardson, The OMNI Thread abstraction, <http://www.lfpt.rwth-aachen.de/Links/GNU/gnu/omniorb/omnithread/omnithr\%ead.html>, available as part of omniORB [22] (1997).
- [16] Trolltech, Qt, <http://www.trolltech.com/> (1999).
- [17] M. Woo, J. Neider, T. Davis, D. Shreiner, *OpenGL Programming Guide*, 3rd Edition, Addison-Wesley, 1999.
- [18] W. Hackbusch, *Multi-grid Methods and Applications*, Springer-Verlag, 1985.
- [19] L. Westover, Footprint evaluation for volume rendering, in: *SIGGRAPH 90 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Dallas, TX, USA, 1990, pp. 367–376.
- [20] N. Max, R. Crawfis, D. Williams, Visualizing wind velocities by advecting cloud textures, in: *Proceedings of Visualization 92*, IEEE, Los Alamitos, CA, USA, 1992, pp. 179–183.
- [21] ACM SIGGRAPH, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series.
- [22] AT&T Laboratories Cambridge, omniORB, <http://www.uk.research.att.com/omniORB/> (1999).

A Results of Simulations

Below the simulation sequence of a smoking chimney is given.

