EUROGRAPHICS Workshop on ... (2005)
J. Marks and M. Alexa (Guest Editors)

GPU Simulation and Rendering of Volumetric Effects for Computer Games and Virtual Environments

Jens Krüger[†] and Rüdiger Westermann[‡]

TU-München

Abstract

As simulation and rendering capabilities continue to increase, volumetric effects like smoke, fire or explosions will be frequently encountered in computer games and virtual environments. In this paper, we present techniques for the visual simulation and rendering of such effects that keep up with the demands for frame rates imposed by such environments. This is achieved by leveraging functionality on recent graphics programming units (GPUs) in combination with a novel approach to model non physics-based, yet realistic variations in flow fields. We show how to use this mechanism for simulating effects as demonstrated in Figure 1. Physics-based simulation is performed on 2D proxy geometries, and simulation results are extruded to 3D using particle or texture based approaches. Our method allows the animator to model and to flexibly control the dynamic behavior of volumetric effects, and it can be used to create plausible animations of a variety of natural phenomena.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction and Related Work

As simulation and rendering capabilities continue to increase, volumetric real-world phenomena will be frequently encountered in computer games and virtual reality environments. Popular examples include smoke, clouds and fog, fire, or explosions. Such effects are beneficial because they can significantly increase the degree of realism and thus help to improve the immersiveness of such environments.

Due to the numerical complexity of physics-based techniques for simulating these effects, they usually cannot fulfill the demands for frame rates in interactive environments. Hence, simplified physical models in combination with 2D visual approximations like sprites, billboards or animated images have been proposed [Ree83, CMTM94, Ina94, KCR00, WLMK02, HBSL03]. Fractal approaches, on the other hand, can be used to model structures that exhibit similar statistical properties as those evolving in nature [Per85, Gar85, EMP*94, SW92, SF93]. Pre-computed density and velocity maps, on the other hand, have been introduced to explicitly specify the shape of large-scale structures in flames and wind fields [PP94, SF95]. Despite their beneficial properties, all these methods only provide restricted means for realistically modelling small-scale variations in the simulated fields. Furthermore, it is difficult when using these methods to control the dynamic evolution of structures as it appears in reality.

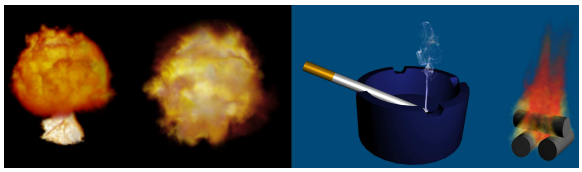

Figure 1: *Volumetric effects like explosions, fireballs, smoke and fire can be simulated and rendered in real-time on recent GPUs.*

In interactive environments, the aforementioned phenomena are usually presented to the viewer as "side effects" en-

† e-mail: jens.krueger@in.tum.de
‡ e-mail:westermann@in.tum.de

hancing the overall visual impression. The simulation is required to create plausible results, and it should be controllable either by external parameters or by the actor itself. In this paper, we address these requirements for the generation of volumetric fluid effects. Besides user control, the techniques we propose enable visual simulation and rendering of such effects at a few hundred frames per second, thus leaving sufficient time for other tasks to perform, i.e. game AI, scene graph traversal, or rendering. Procedural extrusion of space filling effects from 2D slices keeps memory requirements very low. As simulation and rendering is performed on the GPU, the effects we describe do not impose additional constraints on the communication channel, and they are thus suited for use in bandwidth-intensive applications.

The effects we address in this work can be modelled by means of the Navier-Stokes equations (NSE), which govern the motion of non-turbulent, Newtonian fluids. In computer graphics, these equations have been employed for the realistic synthesis of liquids [KM90, FM96, OH95, FF01], hot gases and explosives [FM97, YOH00, OBH02, FOA03, RNGF03] or smoke and fire [Sta99, FvJS01, NFvJ02], to name just a few.

While these techniques are effective in revealing intrinsic fluid properties, they are only of limited relevance for the simulation of volumetric effects in real-time environments. Numerical solutions to the NSE on reasonably sized 3D grids, i.e. where detailed flow structures can evolve, are not able to fulfill the demands on frame rates imposed by such environments. In regard to this observation, we propose a different strategy:

- On the GPU, physics-based simulation [HBSL03, BFGS03, KW03] is performed on a few slices in the 3D domain, and scalar as well as vector valued simulation results are extruded to 3D via spherical linear interpolation.
- To give the animator control over the phenomena and to enable realistic evolution of small-scale variations in the simulated fields, we introduce a new modelling technique – so called pressure and impulse (velocity) templates. These templates locally perturb the scalar pressure field and the velocity field used in the solution to the NSE. While the pressure field is modified such as to produce sources, sinks and small-scale vorticity structures, impulse templates produce large-scale structures in the simulated flow fields.
- The simulation engine is combined with both a particle renderer and a texture based volume renderer. Particles are advected through the reconstructed flow field, and they are then rendered in turn using functionality in Vertex Shader 3.0 [Mic04] to fetch texture values in the vertex units of the GPU. Volume rendering is utilized to visualize scalar flow quantities like density, temperature or injected dye.

In the remainder of this paper we will shortly describe the NSE as well as the numerical machinery used for computational fluid dynamics (CFD). We then introduce pressure and impulse templates as a powerful mechanism to model realistic variations in dynamic flow fields and thus to enhance the appearance of the evolving structures. Next, we describe the GPU simulation engine, which enables realistic simulation of a variety of real-world phenomena. Finally, the integration of the simulation engine into particle and texture based rendering systems is described. We conclude the paper with a discussion of future work, and we show additional results of the presented techniques.

2. Computational Fluid Dynamics

In CFD, the goal is to numerically compute a prediction of what will happen when fluids flow. Figure 2 shows such a simulation. Therefore, a physical model of fluid flow as well as a mathematical formulation of this model is required. The NSE are such a formulation – they are the fundamental differential equations describing the dynamics of fluid flow.

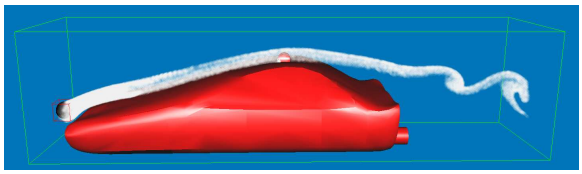

Figure 2: *A CFD simulation of the flow around a car is shown.*

We solve for the velocity $V = (u,v)^T$ governed by the NSE

$$\frac{\partial u}{\partial t} = \frac{1}{Re}\nabla^2 u - V\cdot\nabla u + f_x - \frac{\partial p}{\partial x} \quad (1)$$

$$\frac{\partial v}{\partial t} = \frac{1}{Re}\nabla^2 v - V\cdot\nabla v + f_y - \frac{\partial p}{\partial y} \quad (2)$$

$$\nabla\cdot V = 0 \quad (3)$$

in two passes. First, by ignoring the pressure term p an intermediate velocity is computed. The computation of this fields is carried out on an uniform grid. On this grid, the diffusion operator $\nabla^2 u$ is discretized by means of central differences, and, as proposed in [Sta99], the advection part $V\cdot\nabla v$ is solved for by tracing the velocity field backward in time. In the above equations, *Re* is the Reynolds number and $(f_x, f_y)^T$ is the external force vector to be considered at every grid point. To make the resulting intermediate vector field free of divergence, pressure is used as a correction term. Mass conservation of incompressible media leads to a Poisson-Equation for updating this pressure term. This equation is solved using a Conjugate-Gradient method (CG).

3. Effect Modelling with Flow Patterns

Ideally, the simulation of incompressible non-turbulent fluids using the NSE produces physically accurate results. On

the other hand, adjusting fluid parameters to achieve a particular effect can be complicated or even impossible due to restrictions of the underlying physical model. In particular, expansion and contraction effects are not seen covered by these equations. Additionally, due to the limited size of grids that can be used in real-time scenarios, highly detailed features can hardly be generated.

To overcome the described limitations one could either perturb the resulting velocity field to introduce small-scale features [SF93, RNGF03], or one could try to modify the simulation process in such a way as to produce the desired features automatically. For this purpose we introduce parameterized templates that are blended into the respective fields, and which are then considered by the simulation. Internally our system distinguishes between pressure templates – precomputed pressure masks that are continually inserted into the pressure field *during* the solution of the poisson equation – and velocity templates – vector field masks that are inserted as external forces into the velocity field.

The key observation that lends itself directly to pressure templates is, that characteristic features in the vector field are related to characteristic features in the corresponding pressure field (see Figure 3 for an example). While pressure templates provide an intuitive way to model divergence phenomena, i.e. sinks and sources, direct modification of the vector field enables the user to intuitively add large-scale structures like vortices of particular shape and size.

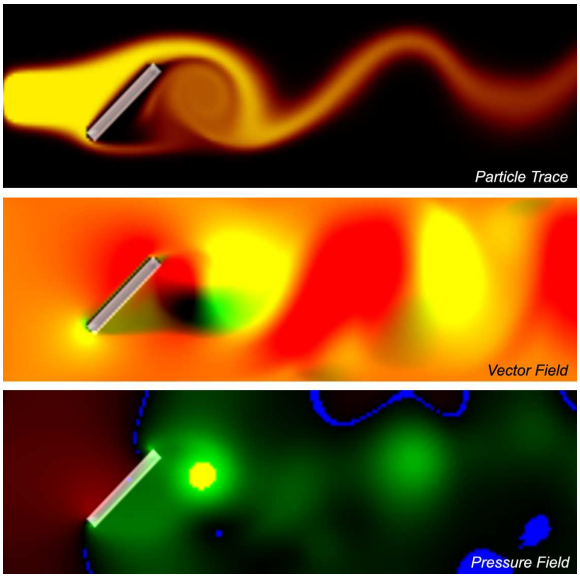

Figure 3: *A vortex in the flow field, the vector field and the corresponding pressure field (red and green indicate high and low pressure regions, respectively).*

The need for pressure templates in addition to impulse templates is motivated by the following observations: First, pressure is a comprehensible feature of space, and it is thus easy to imagine what would happen if the pressure at a certain point in space is changed – increasing pressure leads to a source in the field, whereas a decrease creates a sink. By modifying the pressure distribution within a region, a wide range of different velocity structures can be modelled (see Figure 4). Second, pressure templates allow for the creation of divergence effects that can not be efficiently and intuitively modelled using velocity templates. As pressure acts as a global correction term for the velocity field, divergence effects created by velocity templates will be destroyed by the Poisson solver. If we do not want the effect to be corrected, direct modification of the pressure term turns out to be an effective mechanism.

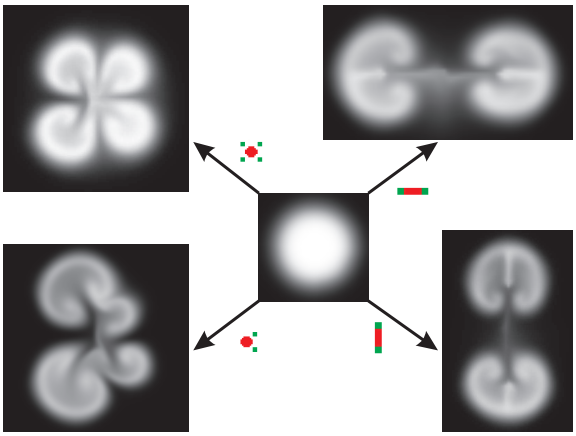

Figure 4: *Velocity structures created by different pressure templates are shown. Pressure templates are shown in red/green indicating negative/positive pressure.*

Our system provides the user a set of effect templates that have been pre-computed, or which have been captured either from the pressure or the velocity field. To the user, only iconic representations of the effects generated by a particular template are presented. The way these effects have been generated is hidden.

3.1. User Interaction

Our approach enables the user to design different effects by capturing a template from the respective fields. By adding external forces via mouse drags or by applying templates from a library, desired flow structures are produced. It is possible to switch between different views showing either the vector field visualized by dye advection or the pressure field. If a desired structure has been created, the simulation can be stopped thus allowing the template to be grabbed with a selection box. Depending on the current view, the system either captures a pressure or an impulse template. The current template then becomes active and the user can blend it into the current field to analyze the effect. Selected templates

are stored as RGBA images in the template library, and they can thus be post-processed using imaging tools.

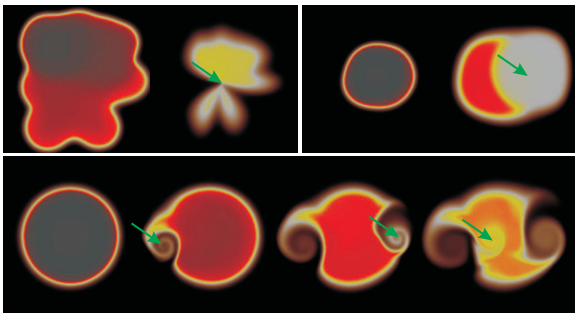

Figure 5: *Effects of interactive template insertion are shown. The upper left image shows the effect of a sink template, in the right image a source template was applied. In the lower row an impulse template producing a vortex was inserted at different positions and orientations.*

To model a particular phenomenon, the user has two possibilities: First, templates from the library can be pasted interactively into the current simulation (see Figure 5). Second, templates can be inserted automatically by the program into user-defined regions (see Figure 6). While the first approach is more adequate for prototyping and experimenting, the second approach enables the design of complex phenomena requiring only simple user interaction. In either case, templates can be arbitrarily transformed to an appropriate size and orientation, and values stored in a template can be scaled at run-time to produce more or less dominant effects. In particular, templates can be automatically aligned along pre-defined contours or to the shape of the insertion regions, e.g. to produce radially symmetric flow.

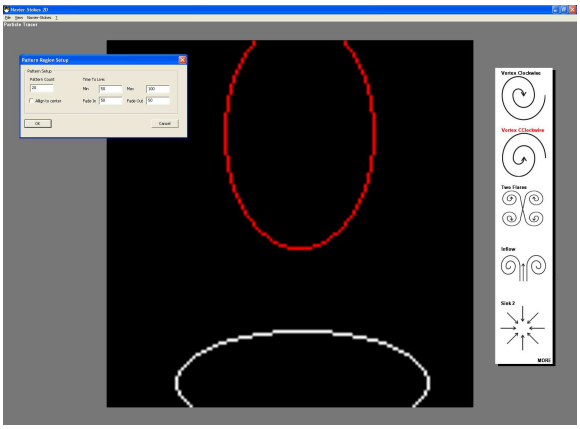

Figure 6: *The user interface to define regions for automatic insertion of templates.*

One important effect that needs to be paid attention to is the modification of inserted templates in each simulation step. If a template is only inserted once, it will be destroyed by the pressure correction term in the upcoming iteration. Consequently, templates need to be injected constantly into the field over a number of iterations. Only then will the respective flow structures be formed.

From this observation an additional parameter that controls the effect caused by a template is obtained. Templates can be kept in a certain region for a certain number of time steps. The longer the templates stay in the flow, the more dominant the resulting structures will appear and the longer their lifetime will be after they have been taken out of the flow. If multiple templates are applied simultaneously and their lifetime is stochastically varied between a minimum and a maximum lifetime, the frequency characteristics of the resulting structures in the velocity field can be varied. These effects are illustrated in Figure 7.

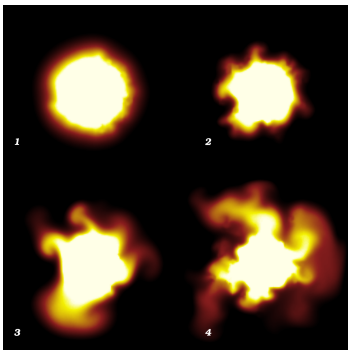

Figure 7: *In all images, templates of equal shape and number are inserted. From image 1 to 3 we see high, medium, and low frequency structures. In image 4 all frequencies are contained. These effects are achieved by using different lifetimes and scale factors.*

On the GPU, the insertion of template can be handled very efficiently. An appropriately scaled and positioned quadrilateral texture mapped with the desired template is rendered into the respective field. In the fragment stage, scalar or vector values are fetched from the texture, and these values are then alpha blended with the simulated results. To generate continuous transitions at template boundaries, a Gaussian distribution of alpha values ranging from one in the center of the template to almost zero at the boundaries is used.

4. Real-Time Fluid Effects

The governing equations for incompressible fluid flow are entirely solved on the GPU. This is accomplished by means of a GPU-based simulation engine, which provides a variety of different parameters to model the flow behavior.

4.1. GPU Simulation Engine

The GPU engine stores relevant flow quantities in 2D textures representing a uniform computational grid structure. In every iteration, simulation results are rendered to an equally sized texture render target, which becomes the container for these quantities in the next pass. Apart from the CG solver to enforce mass conservation, all numerical operations are carried out explicitly in the fragment units. By approximating partial derivatives in the NSE with finite differences, we obtain a set of difference equations to be solved for every grid point. To approximate the operators on the left hand side of the NSE and to accumulate external forces, only simple arithmetic and a few localized texture fetches are required. For the implementation of the CG solver let us refer to [BFGS03, GWL*03, KW03].

Besides physics-based simulation of fluid flow at high frame rates (see Figure 8 for some results), running the simulation on the GPU has another important advantage: As the simulation results already exist on the GPU, they can be rendered immediately. Any data transfer between the CPU and the GPU is avoided.

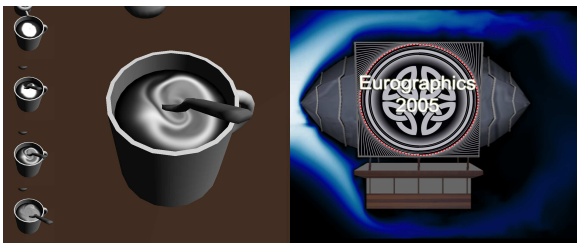

Figure 8: *Images of an incompressible fluid are shown. On a* 128×128 *grid, simulation and rendering is performed at about 450 fps on recent GPUs.*

To realistically simulate the large-scale behavior of fluid effects, the simulation engine enables flexible control of flow conditions. Inflow regions and characteristics can be specified, the user can place obstacles exhibiting special boundary conditions, vorticity confinement [CIR52] is integrated and time-varying external body forces that modulate the resulting vector field are considered during the simulation. Figure 9 shows a number of different examples that visualize the effects of the described features.

4.2. Effects Modelling

To generate effects like explosions, fire and smoke, the GPU engine simulates flow dynamics using different flow and template parameters. Figure 10 shows the generated phenomena after 2D simulation results have been extruded to 3D. In contrast to low speed events such as fire, flames and smoke, for the realistic simulation of explosions compressible events like expansion and contraction are important. Nevertheless, in all scenarios the incompressible NSE in combination with pressure templates are employed.

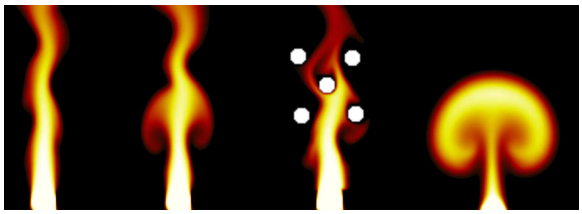

Figure 9: *From left to right we show a simple inflow, the same flow with enabled vorticity confinement, a few obstacles placed in the flow and a temperature driven inflow*

Explosions: For modelling explosions, we simulate a vast amount of energy that is instantaneously liberated by initially setting the temperature in near-by regions of the center of the explosion to be very high. In subsequent simulation steps the temperature of injected heat is decreased. The buoyancy force is computed and added as an external force. By using appropriate pressure templates, bigger structures are created in the very heart of the explosion while they slowly fade out towards the outer regions and finally disappear outside the explosion. As we continually inject dye at the source of the explosion, the dye density is used as scaling parameter for the pressure templates.

Fire Ball: To realistically simulate a corona, templates are positioned at the surface. In addition, they are oriented such as to generate flow structures moving away from the center of the corona. Buoyancy is not considered in this case. To simulate high frequency structures giving rise to a pumping like behavior, only for a short period rather small templates are inserted. These templates, however, are scaled by a large factor to amplify their effect.

Smoke: Smoke rising from the cigarette exhibits all frequency structures in the upper part, where cooling has already taken place. The rising hot air in the lower part is not modified. In order to prevent regular vortex-like structures as in the explosion, much smaller templates with a short lifetime are specified. In this example, heat is continuously injected and templates are scaled by temperature. Additionally, an outflow condition is specified at the upper boundary of the domain, and the Reynolds number is decreased to simulate less viscous material. In this way we achieve a much straighter and faster upward flow.

Log Fire: To simulate a log fire, we use many small templates at the base of the fire to disturb the otherwise too homogenous flow. Larger templates are used in the upper part of the simulation grid where the fire turns into smoke. Like in the cigarette smoke example, heat is constantly injected and templates are scaled by temperature.

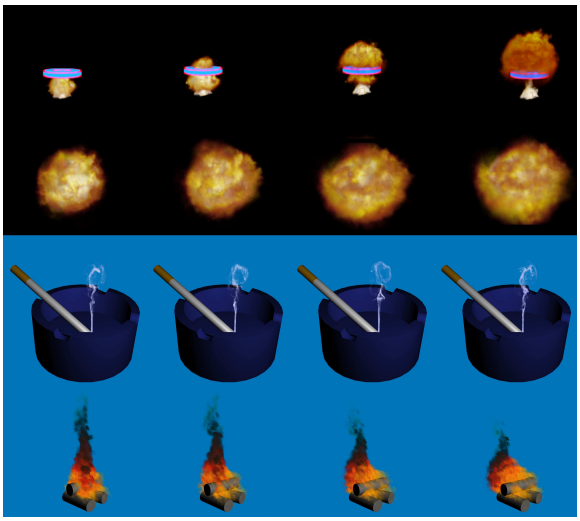

Figure 10: *From left to right we see the time evolution of different phenomena as they are generated by the GPU simulation engine.*

5. Volumetric Extrusion

While conceptually neither the pressure templates method nor the GPU NSE framework is restricted to two dimensions, a full direct simulation in 3D is not practical in real-time environments due to computational and memory requirements. Thus we restrict ourselves to the simulation of effects that exhibit a rotationally symmetric appearance. Then, simulation can be restricted to a few 2D slices from which the volume is extruded [RNGF03].

The simulation loop starts by computing multiple independent 2D fluid simulations to produce similar structures in each slice, only slightly different initial conditions are specified. Volumetric effects can now be generated by spherical linear interpolation between these 2D simulations (see Figure 11).

For every data point in the 3D domain the projection along the circular arc onto the closest two slices is computed. At these projection points the 2D simulation slices are sampled and reconstructed values are interpolated with regard to their circular arc distance from the original point in 3D space. Since this interpolation does not introduce new features along the circular arc, it results in a perfectly symmetric result. This is avoided by perturbing the initial point coordinate by a stochastic fractal distribution. This leads to the following compute kernel for every point in the domain:

- The initial point position is disturbed by a small turbulence offset which is fetched from a 3D hypertexture
- The point is radially projected onto the closest simulation slices
- Respective values are fetched from both slices
- Spherical linear interpolation is performed

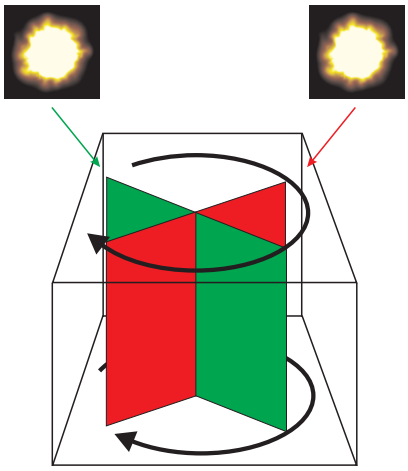

Figure 11: *The extrusion of 2D simulation results to 3D is illustrated.*

5.1. Texture based Volume Rendering

Because simulation results at arbitrary points in the 3D domain can be decoded procedurally, it is not necessary to reconstruct the whole volume for rendering. Instead, at runtime the data can be reconstructed on proxy geometries, i.e. view plane aligned slices that are rendered in a particular order [CCF94, WE98]. Fragments covered by these slices reconstruct the data by executing the described kernel in the fragment shader stage, and they blend the results into the frame buffer. Proxy geometries are always rendered into a texture render target with the same size as the 2D simulation textures, and they are then scaled to the current view port.

5.2. Particle based Volume Rendering

Using OpenGL Superbuffer [KSW04, KLRS04] or Vertex Shader 3.0 functionality, we can compute intermediate results on the GPU, save these results in graphics memory, and use them again in the geometry units to displace point primitives.

Initial particle positions are stored in the RGB color components of a floating point texture. For every particle, a fragment shader fetches the particle position and reconstructs the vector field at that position from 2D slices. This vector is then used in turn to perform an Euler step into the direction of the flow. Updated positions are written to a 2D texture render target, which becomes the particle container in the next pass.

To draw displaced particles, a static vertex array stored in GPU memory is rendered. In a vertex shader program the particle position is fetched from the current particle container, and this position then replaces the position initially

stored in the vertex array. Particles are rendered as point sprites, which gives them a similar visual appearance as geometric icons, but requires far less geometry processing. For instance, in Figure 1 the campfire was rendered with a much simpler sprite texture than in Figure 10. In addition to texture and size, the color of each particle can be modulated according to a specific color table or derived flow quantities like density of temperature.

6. Discussion and Performance Evaluation

To verify the effectiveness of the GPU engine for interactive simulation of real-world phenomena let us first present timings for solving the NSE on differently sized 2D grids (see table 1). All our experiments were run under Windows XP on an Intel Pentium 4 2.0 GHz processor equipped with an ATI X800 XT graphics card. Because particle advection requires a graphics card with Vertex Shader 3.0 support, a Nvidia GeForce 6800 GT is used in the respective examples.

Grid Size	Cells	Performance
128×128	16384	443 fps
256×64	16384	443 fps
256×256	65536	221 fps
512×512	262144	54 fps

Table 1: *Timings for solving the NSE on 2D grids.*

The table shows that the implementation scales well for grid sizes of 128×128 and larger. Even for a grid of 512×512 the simulation still runs at a frame rate of about 54 fps. Overall, we note that the simulation can almost be performed at a rate equal to what can be achieved by transferring precomputed simulation results from the CPU to the GPU.

The following table 2 shows the impact of pressure templates on the overall performance.

Templates	128^2	256^2
0	443 fps	211 fps
5	411 fps	209 fps
10	391 fps	208 fps
20	369 fps	206 fps
40	326 fps	204 fps
80	256 fps	202 fps
200	166 fps	153 fps

Table 2: *Performance impact of pressure templates for different grids. To simulate the corona, 40 templates are used.*

One can see that even for a huge number of pressure templates only about one fourth of the overall run-time performance is lost due to template insertion.

Next, timings for the complete simulation system including two 2D-simulations, pressure templates and the extrusion to 3D via 2D textures are presented (see table 3).

Volume Size	Performance
64^3	190 fps
128^3	152 fps
256^3	62 fps

Table 3: *Timings for the corona simulation using 40 pressure templates. Intermediate results are rendered to a* 256^2 *texture map.*

The results show that for a volume of 128^3 the best trade-off between visual quality and performance can be achieved. Running the simulation on larger grids introduces additional small-scale features while the overall appearance remains unchanged (see Figure 12).

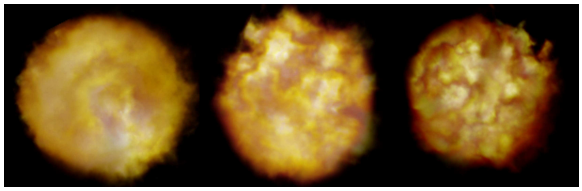

Figure 12: *The corona demo at resolutions of* 64^3, 128^3 *and* 256^3.

Finally we give timings for the particle based rendering, again the timings describe the overall performance including the simulation in two 2D slices, the particle advection and the final rendering of the point sprites and the surrounding scene (see table 4).

Particle Count	Performance
16384	190 fps
65536	128 fps
262144	60 fps
1048576	21 fps

Table 4: *Timings for the smoke simulation using 50 pressure templates on a* 128^2 *simulation grid.*

Figure 13 shows the differences in visual quality for 16384, 65536, 262144 and 1048576 particles. As can be seen, for a large amount of particles the size of the individual point sprite can be reduced to almost one pixel without losing the impression of a connected smoke cloud. Whereas with these very small particles very detailed structures can

be rendered at still interactive framerates. Yet even with relatively few particles the physically plausible advection conveys an impression of rising smoke while the framerates allow for an integration into a bigger system, like a game engine.

Figure 13: *The images show the smoke simulation rendered with* 16384, 65536, 262144 *and* 1048576 *particles. For the images with fewer particles we use larger points to maintain a constant smoke density.*

7. Conclusion and Future Work

In this work, we have described techniques for modelling *and* rendering volumetric real-world phenomena. By using a GPU simulation engine in combination with a novel approach to control the shape and appearance of simulated structures, interactive visual simulation of fluid effects is possible. As the timings show, and because neither the simulation engine nor the render engine impose considerable bandwidth or memory requirements, the proposed techniques have the potential to be integrated into real-time scenarios like computer games or virtual environments.

We have introduced pressure templates as a new modelling paradigm for fluid flow. Such templates provide an effective means for modelling small-scale features that can be added at almost arbitrary resolution. To give the animator control over the shape, the size and the dynamic behavior of evolving structures, the proposed flow templates can be instantiated with a variety of different parameters. In this way, custom design of fluid flow is made possible.

Because the 3D flow domain can be sampled procedurally using spherical linear interpolation, relevant flow quantities can be reconstructed on arbitrary proxy geometries. As a matter of fact, texture based volume rendering can be utilized without any significant loss of performance. In addition, by combining fragment processing and Vertex Shader 3.0 functionality, advection and rendering of particles can be performed on the GPU without any data transfer to CPU memory. This allows for the rendering of massive particle sets, at the same time enabling the use of different, e.g. oriented, point sprites to enhance the visual impression. The described particle-based technique has the advantage that external forces like wind fields and collisions with other parts of the scene can be considered during rendering.

In the future, we will integrate the proposed effects into a game engine and a virtual reality application. Besides performance issues, one of the most interesting questions is how to let these effects automatically be controlled by external forces in a given environment. Furthermore, we will extend the concept of flow templates to 3D by integrating volumetric templates into 3D simulations. Although we do not expect this method to be suitable in interactive scenarios, visual simulation of real-world phenomena on rather small grids should be possible in this way. Additionally, the described method will no longer be constraint to the simulation of radially symmetric effects.

References

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *ACM Computer Graphics (Proc. SIGGRAPH '03)* (2003), pp. 917–924. 2, 5

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings ACM Symposium on Volume Visualization 94* (1994), pp. 91–98. 6

[CIR52] COURANT R., ISAACSON E., REES M.: On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications on Pure and Applied Mathematics 5* (1952), 243–255. 5

[CMTM94] CHIBA N., MURAOKA K., TAKAHASHI H., MIURA M.: Two-dimensional visual simulation of flames, smoke and spread of fire. *The Journal of Visualization and Computer Animation 5* (1994), 37:53. 1

[EMP*94] EBERT D., MUSGRAVE K., PEACHEY D., PERLIN K., WORLEY M.: *Texturing and Modeling: A Procedural Approach*. Academic Press, 1994. 1

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *ACM Computer Graphics (Proc. SIGGRAPH '01)* (2001), pp. 22–32. 2

[FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. *Graphical Models and Image Processing 58*, 5 (1996), 471–483. 2

[FM97] FOSTER N., METAXAS D.: Modeling the motion of a hot turbulent gas. In *ACM Computer Graphics (Proc. SIGGRAPH '97)* (1997), pp. 181–188. 2

[FOA03] FELDMAN B. E., O'BRIEN J. F., ARIKAN O.: Animating suspended particle explosions. In *Proceedings of ACM SIGGRAPH 2003* (Aug. 2003), pp. 708–715. 2

[FvJS01] FEDKIW R., VAN JENSEN H., STAM J.: Visual simulation of smoke. In *ACM Computer Graphics (Proc. SIGGRAPH '01)* (2001), pp. 15–21. 2

[Gar85] GARDNER G.: Visual simulation of clouds. In *ACM Computer Graphics (Proc. SIGGRAPH '85)* (1985), pp. 297–384. 1

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for

boundary value problems using programmable graphics hardware. In *Graphics Hardware 2003* (July 2003), pp. 102–111. 5

[HBSL03] HARRIS M., BAXTER W., SCHEUERMANN T., LASTRA A.: Simulation of cloud dynamics on graphics hardware. In *Proceedings ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2003), pp. 12–20. 1, 2

[Ina94] INAKAGE M.: A simple model of flames. In *Proceedings of Computer Graphics International 89* (1994), pp. 71–81. 1

[KCR00] KING S., CRAWFIS R., REID W.: Fast volume rendering and animation of amorphous phenomena. In *Proceedings of Volume Graphics 2000* (2000), pp. 229–242. 1

[KLRS04] KOLB A., LATTA L., REZK-SALAMA C.: Hardware-based simulation and collision detection for large particle systems. In *Eurographics Symposium Proceedings Graphics Hardware 2004* (2004). 6

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), ACM Press, pp. 49–57. 2

[KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: A GPU-based particle engine. In *Eurographics Symposium Proceedings Graphics Hardware 2004* (2004), pp. 115–122. 6

[KW03] KRUEGER J., WESTERMANN R.: Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 908–916. 2, 5

[Mic04] MICROSOFT: Shader Model 3 Specification. http://msdn.microsoft.com, 2004. 2

[NFvJ02] NGUYEN D., FEDKIW R., VAN JENSEN H.: Physically based modeling and animation of fire. In *Proceedings of ACM SIGGRAPH 2002* (2002), pp. 721–728. 2

[OBH02] O'BRIEN J. F., BARGTEIL A. W., HODGINS J. K.: Graphical modeling and animation of ductile fracture. *ACM Trans. Graph. 21*, 3 (2002), 291–294. 2

[OH95] O'BRIEN J., HODGINS J.: Dynamic simulation of splashing fluids. *Proceedings of Computer Animation '95* (1995). 2

[Per85] PERLIN K.: An image synthesizer. *ACM Computer Graphics (Proc. SIGGRAPH '85)* (1985), 287–296. 1

[PP94] PERRY G., PICARD R.: Synthesizing flames and their spread. SIGGRAPH '94 Technical Sketch Notes, 1994. 1

[Ree83] REEVES T.: Particle systems - a technique for modelling a class of fuzzy objects. *ACM Computer Graphics (Proc. SIGGRAPH '83)* (1983). 1

[RNGF03] RASMUSSEN N., NGUYEN D. Q., GEIGER W., FEDKIW R.: Smoke simulation for large scale phenomena. *ACM Trans. Graph. 22*, 3 (2003), 703–707. 2, 3, 6

[SF93] STAM J., FIUME E.: Turbulent wind fields for gaseous phenomena. In *ACM Computer Graphics (Proc. SIGGRAPH '93)* (1993), pp. 369–376. 1, 3

[SF95] STAM J., FIUME E.: Depiction of fire and other gaseous phenomena using diffusion processes. In *ACM Computer Graphics (Proc. SIGGRAPH '95)* (1995), pp. 129–136. 1

[Sta99] STAM J.: Stable fluids. In *ACM Computer Graphics (Proc. SIGGRAPH '99)* (1999), pp. 121–128. 2

[SW92] SAKAS G., WESTERMANN R.: A Functional Approach to the Visual Simulation of Gaseous Turbulence. *Computer Graphics Forum 11*, 7 (1992), 107–116. 1

[WE98] WESTERMANN R., ERTL T.: Efficiently using graphics hardware in volume rendering applications. In *Computer Graphics (SIGGRAPH 98 Proceedings)* (1998), pp. 291–294. 6

[WLMK02] WEI X., LI W., MUELLER K., KAUFMAN A.: Simulating fire with texture splats. In *Proceedings IEEE Visualization '02* (2002), pp. 134–143. 1

[YOH00] YNGVE G., O'BRIEN J., HODGINS J.: Animating explosions. In *ACM Computer Graphics (Proc. SIGGRAPH '00)* (2000), pp. 29–36. 2