REAL-TIME SIMULATION OF SMOKE USING GRAPHICS HARDWARE

Marinus Rørbech*
Department of Computer Science
University of Copenhagen
Denmark

Abstract

Based on earlier presented solvers for the incompressible Navier-Stokes equations, we implement a 3D fluid solver. The solver works in a few simple steps implemented in fragment shaders running entirely on the *Graphic Processing Unit* (GPU) of a modern graphics card. We present a simple visualization approach for rendering high-quality smoke animations in real-time. Finally, to handle interaction between fluid and stationary objects, we present a simple method for setting boundary conditions.

Keywords: Fluid dynamics, Navier-Stokes, Real-time, GPU.

Nomenclature

d	Density field
$\mathbf{f}$	Force field
h	Spatial discretization
$\mathbf{N}$	Vorticity location field
p	Pressure field
T	Temperature field
T_0	Mean temperature
t	Time
$\mathbf{u}$	Velocity field
β	Thermal expansion coefficient
ε	Vorticity confinement control factor
η	Gradient field of vorticity magnitude
ω	Vorticity
∇	Gradient operator $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$
ν	Kinematic viscosity
ρ	Density

Introduction

With the increase of computational power in standard home computers, more and more physical simulation is put into computer games. Recently, a quantum leap in computational power has been made with the introduction of the programmable GPU, available on modern graphics adapters. This has lead to higher quality in game graphics, and developers have started to use the GPU for non-graphics computations. This leap will believably open for more advanced effects in upcoming game titles.

Over the past years, a lot of the focus in game development has been dedicated to the simulation of rigid body physics. We believe that one of the new features which will show up in computer games in near future is the simulation of fluid dynamics.

To encourage this, we describe a simple method for achieving real-time animated 3D smoke.

Previous Work

The motion of a non-turbulent, viscous fluid is modeled by the *incompressible Navier-Stokes equations*

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

$$\nabla\cdot\mathbf{u} = 0 \quad (2)$$

where (1) describe the change in the fluid velocity, and (2) enforces conservation of mass.

Numerical methods for solving these equations have been researched extensively during the last decade, and today very realistic fluid animations can be generated off-line. Recent years, efforts have been made to take the methods for solving these equations to the next level: Real-time simulation.

One of the first methods for simulating the full 3D Navier-Stokes equations for the purpose of Computer Graphics, is presented in [4]. The terms of the equations are solved step by step, using a first order

central differencing approach. The velocity is made divergence-free, by solving the pressure term iteratively, using a *Successive Over-Relaxation* (SOR) method [2]. Because the solver in [4] is based on explicit methods the solver is only stable for small time-steps.

In [8], the first step towards real-time simulation is taken by making the solver unconditionally stable. The advection term, $(\mathbf{u}\cdot\nabla)\mathbf{u}$, is solved using a *semi-Lagrangian* integration scheme, where points are traced backwards to find advected velocities. Diffusion, described by the term $\nu\nabla^2\mathbf{u}$, is solved implicitly using a multigrid method [1]. To remove divergence from the velocity field, a pressure field is computed by solving the Poisson equation

$$\nabla^2 p = -\nabla\cdot\mathbf{u} \tag{3}$$

also using a multigrid method. The pressure gradient is then used to adjust the velocity field $\mathbf{u}$ by

$$\tilde{\mathbf{u}} = \mathbf{u} - \nabla p \tag{4}$$

producing the divergence-free velocity field $\tilde{\mathbf{u}}$.

In [3], the method from [8] is refined for the special case of simulating smoke. The diffusion term is neglected based on the assumption that the effect of diffusion is damped out, because of the low viscosity of air and the coarseness of the simulation grid. To make up for some of the small-scale detail thus missing, a vorticity confinement force [10]

$$\mathbf{f}_{vc} = h\,\varepsilon(\mathbf{N}\times\omega) \tag{5}$$

is introduced, where

$$\omega = \nabla\times\mathbf{u} \tag{6}$$

and

$$\mathbf{N} = \frac{\eta}{|\eta|}, \quad \eta = \nabla|\omega| \tag{7}$$

This adds a swirling behavior, which looks very natural for smoke. Derived from [5] a simple thermal buoyancy force, given by

$$\mathbf{f}_T = \beta\,(T - T_0)\cdot(0,1,0)^T \tag{8}$$

is also added, enabling various temperature effects, such as radiators causing the air to rise.

In [9], a simple, easy implementable solver is presented, based on the method from [8]. The multigrid method is replaced with the much simpler Gauss-Seidel method [1]. The entire solver consists of approximately 100 lines of C-code, and is thus presented directly in the paper.

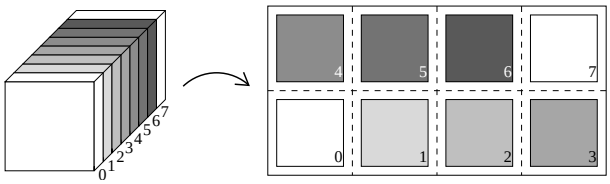


Figure 1: Flat 3D texture – the slices of a 3D field is laid out as tiles in a 2D texture. This figure is copied from [6].

Although this method is very efficient, software implementations are still not quite fast enough for real-time simulations in 3D. In [6], a method for simulating clouds is implemented using fragment shaders, running entirely on a GPU. To do this, 3D vector fields are represented as *flat 3D textures*, see Figure 1. Because of the inability of fragment shaders to read and write the same memory, the Gauss-Seidel approach of [9] is replaced with a variant of the Jacobi method, called *Red-Black Gauss-Seidel* [1]. Due to the power of the GPU, this simulation runs at interactive rates.

The Method

As in [8], we solve the Navier-Stokes equations in two steps. The first step updates the velocity field according to (1). This leads to a non-conserving velocity field. In the second step, we remove divergence, producing a mass-conserving velocity field upholding (2).

Updating Velocity

Advection of the velocity is solved using the semi-Lagrangian advection scheme presented in [8]. This method is stable for large time steps, which is crucial to our real-time simulation.

Following the method from [3] we neglect the diffusion term. Since the kinematic viscosity of air is very small, the effects of diffusion are assumably damped out by the numerical dissipation of our coarse grid.

Having left out the diffusion term, we adopt the vorticity confinement force, described in [3]. This

makes up for the small-scale detail in the velocity field that is dissipated by our coarse grid, and adds natural looking swirls to our smoke.
To easily interact with the smoke, we also include the addition of an external force field, which can respond to any user action.
Finally, we add a thermal buoyancy force, as described in [3], to be able to simulate various temperature-related effects.

Removing Divergence

As in [9], the divergence is removed in three simple steps. First the divergence is computed. Then we compute the pressure field by solving (3) iteratively. Following [6], we use the simple iterative Jacobi method [1], which, opposed to the Gauss-Seidel method used in [9], is implementable on a GPU. Finally, the velocity field is adjusted according to (4).

Implementation on GPU

The GPU is a programmable processor available on most recent graphics adapters. Even though it is designed for graphics processing, general purpose processing on a GPU is attractive due to its computational power.
The GPU works on streams of vertices and fragments (pixels with attributes, such as depth and color), and has only memory read access through texture maps. Thus, to implement a fluid solver to run on a GPU, the fluid state should be represented with texture maps, and the solver steps should be implemented as *fragment shaders*.

Vector Field Representation

We adopt the vector field representation described in [6]. A 3D vector field is represented by laying out each slice as a tile in a 2D texture, as can be seen in Figure 1. This allows us to update the entire field in a single render pass.

Fragment Shaders

The fragment shaders are executed by drawing regular primitives, such as triangles or quadrilaterals. When these primitives are rasterized, the fragment shaders are executed once on each resulting fragment, and the output is written to the frame buffer.

The input to a fragment shader, in the context of our fluid solver, is a texture coordinate referring to a specific cell in the simulation grid. The value of any field in this point can be looked up using a texture read instruction on the respective field texture map.

For looking up neighbor values, offsets can be added to the input texture coordinate to produce texture coordinates representing other cells in the grid. Neighbors in x and y directions can easily be found by adding the vectors $(1,0,0)^T$, $(1-,0,0)^T$, $(0,1,0)^T$, and $(0,-1,0)^T$, respectively, to the texture coordinate of the current cell. For neighbors in different tiles, however, offsets to the respective tiles must be taken into account. As opposed to [6], we pre-compute relative offsets between tiles, and pass them to our fragment shaders as texture coordinates. Compared to [6] this saves us a texture-read instruction per grid cell per simulation step (approximately 10^6 texture instructions per frame at resolution 30x30x30).

For each solver step we have a fragment shader which processes only a single cell in the grid with respect to that solver step. The entire grid is then processed by drawing primitives covering all the *interior cells* in the flat 3D texture, corresponding to the shaded areas in Figure 1. The *boundary cells* (white areas in Figure 1) are processed separately with special fragment shaders, to enforce the desired boundary conditions of the system.

Visualization

One of the hurdles of simulating fluid in real-time applications is the visual presentation of the fluid. A popular method for rendering smoke in games is to use a particle system, and render each particle using a bill-board texture, see Figure 2. Small scale effects, such as exhaust from a car, can be animated convincingly using a limited number of particles (< 500). However, the particle method is not very scalable since the simulation time increases linearly with the number of particles.

Instead, for animating the smoke we use a density field, as proposed in [9]. The density should simply be moved along with the velocity, so we model the

Figure 2: Bill-boarding – the particles to the left are rendered using the texture mapped quadrilateral in the center, yielding the smoke- or cloud-like appearance to the right.

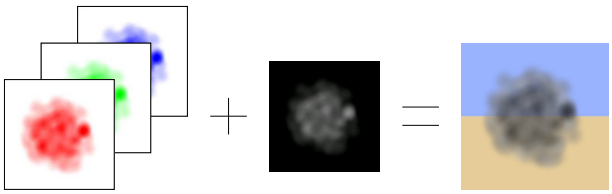


Figure 3: The density texture represents four density fields of red, green, blue (left), and alpha densities (center). When rendered, these four density fields represent colored, transparent smoke (right).

density movement by

$$\frac{\partial d}{\partial t} = (\mathbf{u} \cdot \nabla) d \tag{9}$$

which is similar to the advective term of (1). Thus, we update the position of the density by using the same semi-Lagrangian advection scheme, as described for updating fluid velocity.

Similar to all other vector fields, we represent our density field in a flat 3D texture. This allows us to easily represent colored and transparent smoke, by using the standard RGB color model and the alpha channel of the texture. This actually corresponds to having four density fields, (d_r, d_g, d_b, d_a), representing the red, green, blue, and alpha channels, respectively, see Figure 3. Since fragment operations are vector based, we can update all four density fields at the same speed as updating a single density layer.

The flat 3D texture representation of the colored density field allows us to easily render the density field by simply rendering each slice of the field as a bill-board. This involves drawing a texture mapped quadrilateral per slice in the texture. Slices are drawn in order of distance to the viewer, starting from the back. Each slice is blended with the background, according to density color and alpha value, giving a transparent look. This method is extremely fast, thus, allowing high-rate animations.

To avoid using expensive volume rendering techniques, for instance as presented in [7], we only use 16-bit fixed-point precision in the density texture. In this way, the density texture can be automatically filtered by the graphics adapter, improving image quality and speeding up the updating process.

By representing smoke with a density field, the animation speed is independent of the amount of smoke in the simulation. Unfortunately, this also means that the resolution of the smoke is limited in the same way as the resolution of the simulation grid.

Interaction With Objects

To extend our smoke simulation, we present a simple way to simulate interaction with stationary objects. We represent voxelized objects in a flat 3D texture. In this *obstacle map* we represent cells occupied by an object with the value 0, and cells occupied by the fluid with the value 1. Using a fragment shader, values in any field can easily be masked with regards to the objects in the fluid, allowing simple boundary conditions. Thus, simple stationary objects can be modeled on cell-level.

Results

In this section we show some examples of smoke simulated with our solver. All examples are run on an ATI Radeon 9600 PRO graphics adapter. The grid resolution is $30 \times 60 \times 30$ grid, and the obtainable frame rate is approximately 30 fps. The animations are available for downloading at `http://www.roerbech.dk/marinus/sims.html`.

First, we show a simple smoke stream, which rises due to thermal buoyancy and an external vent force. Frames from this animation can be see in Figure 4.

In the second example, we add a spherical obstacle into the scene from Example 1, to show interaction with stationary objects. Frames from this animation can be seen in Figure 5.

Finally, to demonstrate the capabilities of our colored density field, Figure 6 shows frames from an animation using two different colors of smoke.

Conclusion

We have presented a fast way of animating 3D smoke based on simulation of fluid dynamics.

We keep in mind, that our solver is not yet fully optimized, and we estimate a possible improvement in performance of about 10–15% on this account alone. Further more, the presented examples have been run on an ATI Radeon 9600 PRO graphics adapter, which is already considered obsolete by current graphics hardware standards.

Thus, even though our solver has some limits, it gives the impression that real-time fluid effects in games should be possible in the very near future. Especially when taking into account the extended features for general purpose processing presented on next-generation graphics adapters.

Since our solver is based on the Navier-Stokes it can be used for other purposes than animating smoke, for instance the simulation and animation of explosions, animation of miscellaneous wind effects, such as dry leafs swirling along the ground, etc.

References

[1] Demmel JW. *Applied Numerical Linear Algebra*. Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 1997.

[2] Faires JD, Burden RL. *Numerical Analysis*. 6th edition. Brooks/Cole Publishing Company, 1997.

[3] Fedkiw R, Stam J, Jensen HW. *Visual simulation of smoke*. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pages 15–22. ACM Press, 2001.

[4] Foster N, Metaxas D. *Realistic Animation of Liquids*. Graphical models and image processing: GMIP 1996, 58(5):471–483.

[5] Foster N, Metaxas D. *Modeling the motion of a hot, turbulent gas*. Computer Graphics 1997, 31(Annual Conference Series):181–188.

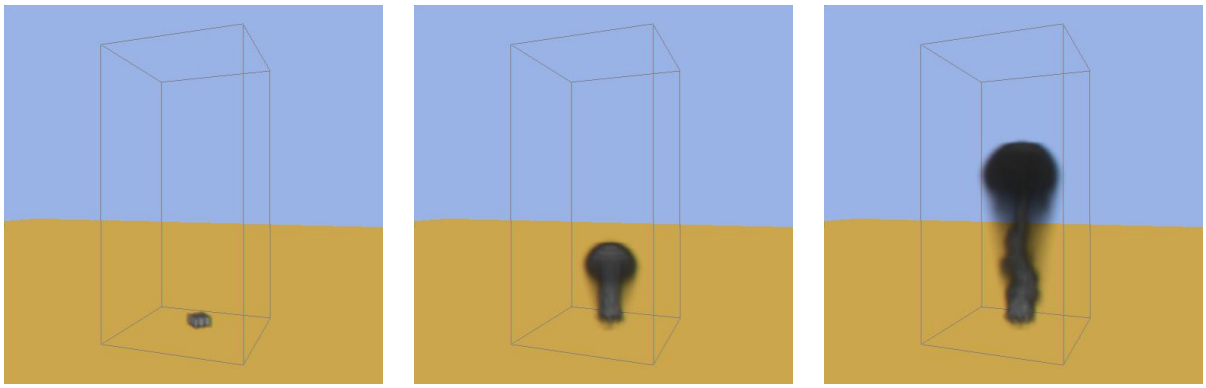
[6] Harris MJ, Baxter WV, Scheuermann T, Lastra A. *Simulation of Cloud Dynamics on Graphics Hardware*. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, Eurographics Association, 2003

[7] Meissner M, Guthe S. *Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators*. In: Graphics Interface Proceedings, pages 209–218. Canadian Information Processing Society, 2002.

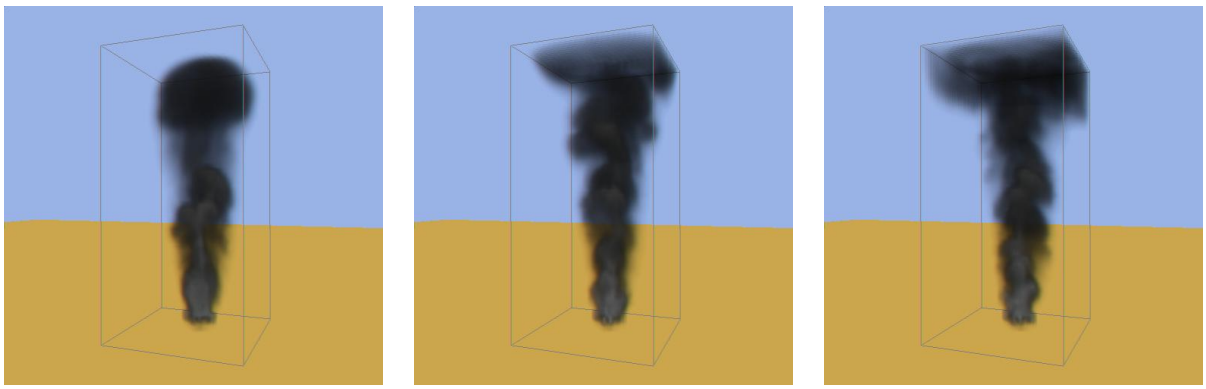
[8] Stam J. *Stable Fluids*. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[9] Stam J. *Real-Time Simulation of Fluid Dynamics for Games*, 2003, Available at `http://www.dgp.utoronto.ca/~stam/reality/Research/pub.html`.

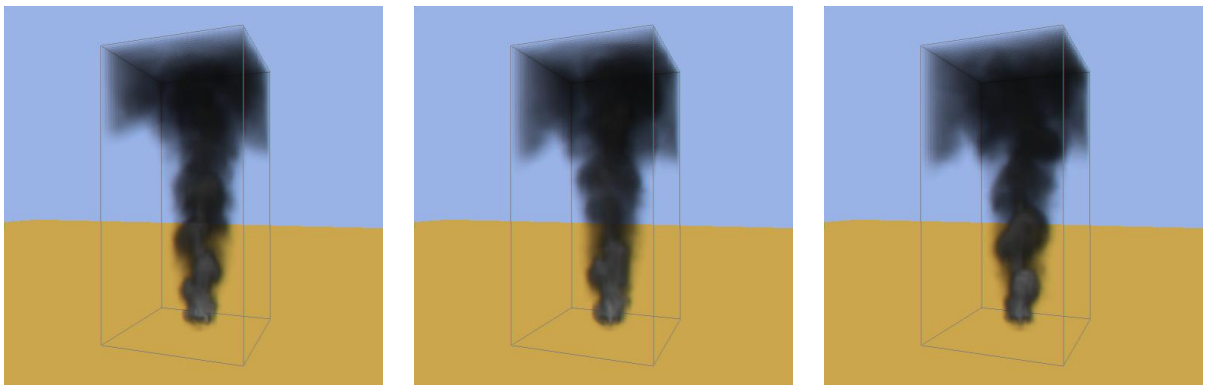
[10] Steinhoff J, Underhill D. *Modification of the euler equations for "vorticity confinement": Application to the computation of interacting vortex rings*. Physics of Fluid 1994, 6(8):2738–2744.



$t = 0.0$ sec. $t = 1.0$ sec. $t = 2.0$ sec.

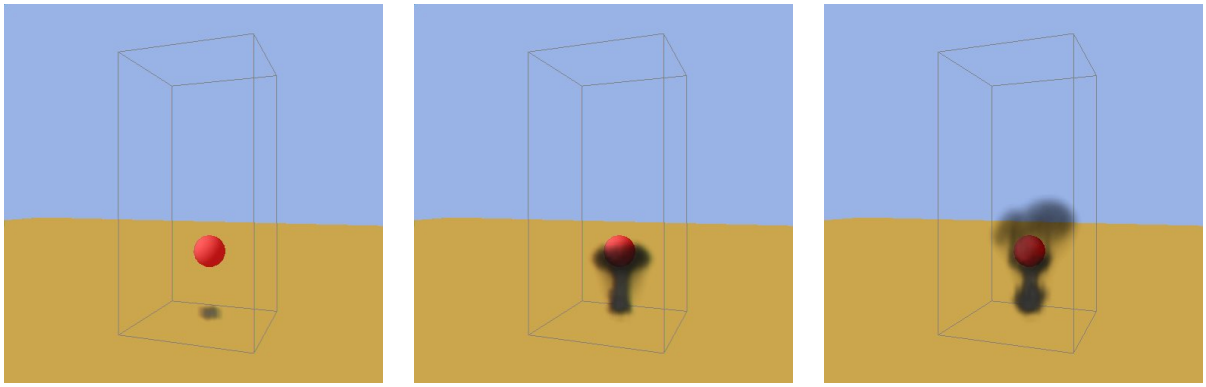


$t = 3.0$ sec. $t = 4.0$ sec. $t = 5.0$ sec.

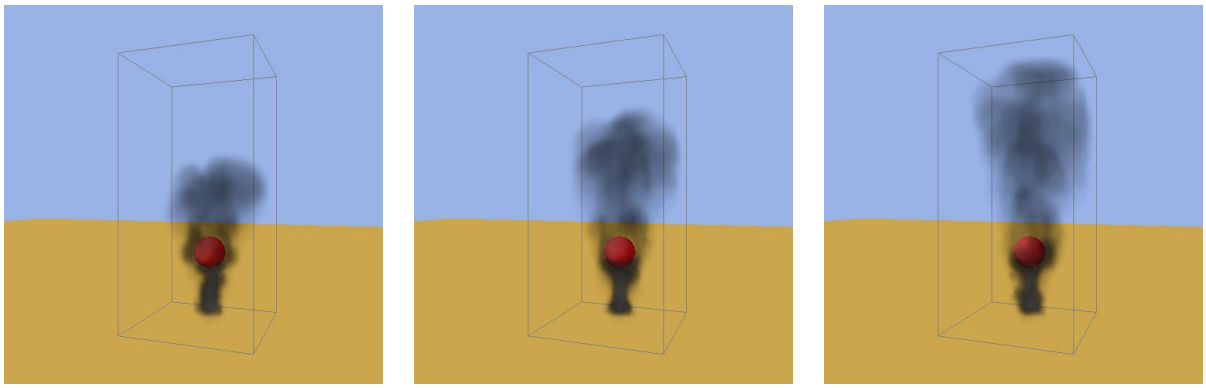


$t = 6.0$ sec. $t = 7.0$ sec. $t = 8.0$ sec.

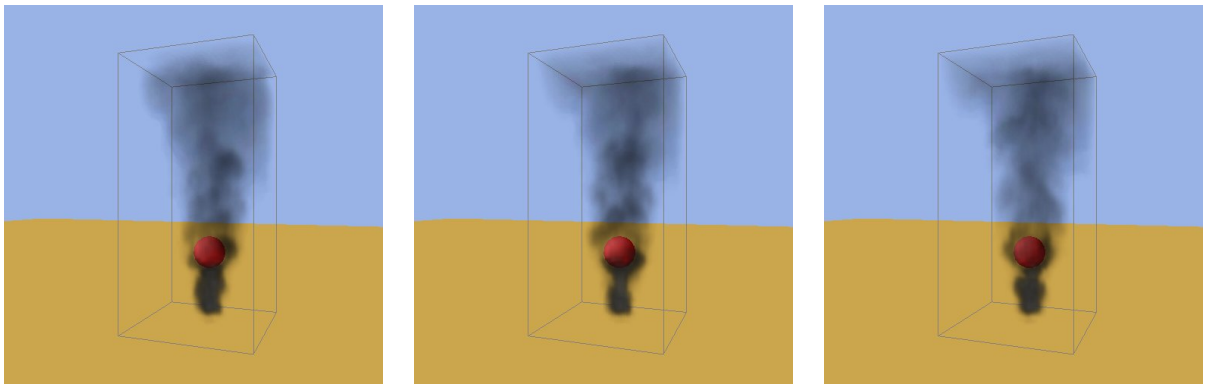
Figure 4: Example 1 – Rising smoke. The smoke has natural looking swirls due to the vorticity confinement, and the thermal buoyancy adds realism to the general smoke movement.



$t = 0.0$ sec. $t = 1.0$ sec. $t = 2.0$ sec.

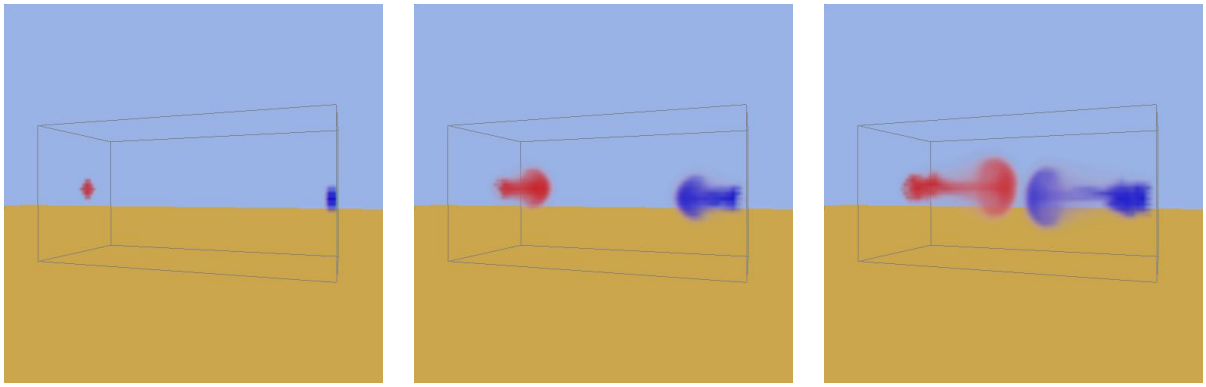


$t = 3.0$ sec. $t = 4.0$ sec. $t = 5.0$ sec.

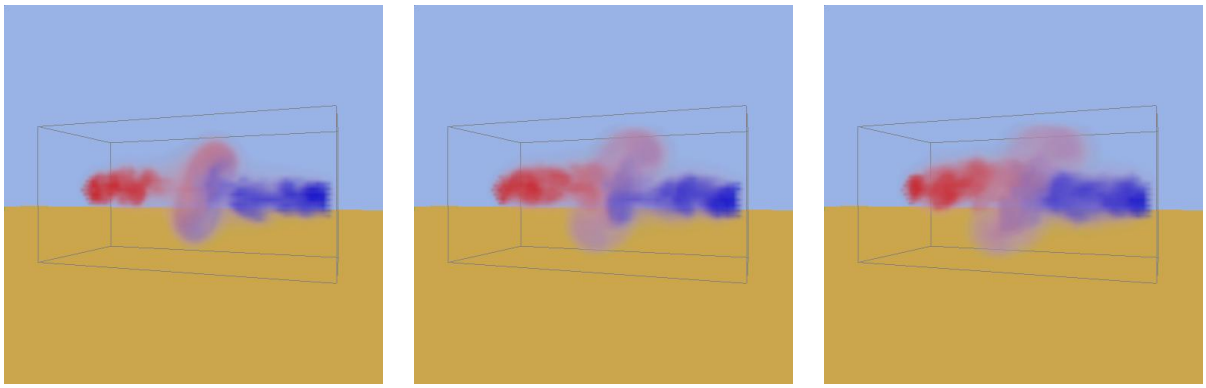


$t = 6.0$ sec. $t = 7.0$ sec. $t = 8.0$ sec.

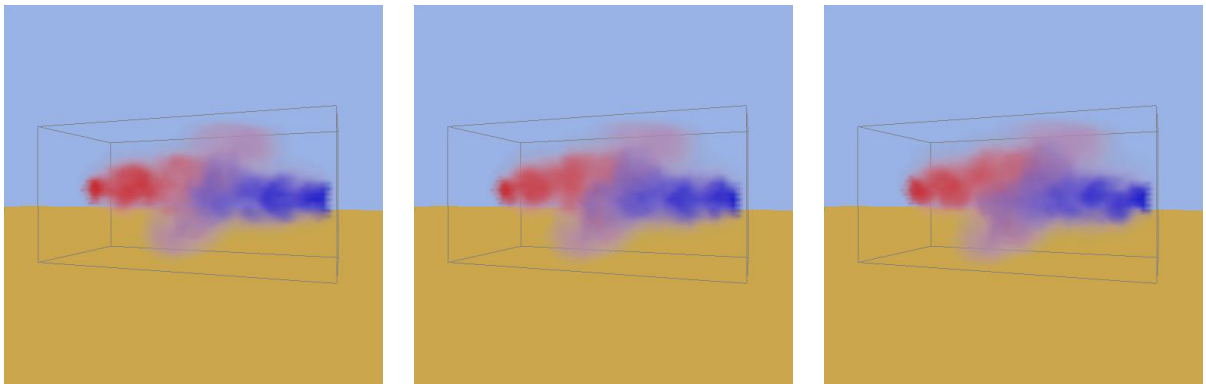
Figure 5: Example 2 – Interaction with a simple object. The smoke evolves a bit unnatural around the object due to the very simple handling of the boundary. However, an effect of interaction is achieved. The voxelized object is presented visually, using a spherical mesh, to improve visual quality.



$t = 0.0$ sec. $t = 1.0$ sec. $t = 2.0$ sec.



$t = 3.0$ sec. $t = 4.0$ sec. $t = 5.0$ sec.



$t = 6.0$ sec. $t = 7.0$ sec. $t = 8.0$ sec.

Figure 6: Example 3 – Multi-colored smoke. Two smoke streams with different colors meet and blend into a mixed color.