# Smoke simulation with obstacles outside the simulation grid

Adam Csendesi
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest / Hungary
E-mail: csendesi.adam@gmail.com

## Abstract

The integration of natural phenomena in a virtual scene is a difficult task. Most of them involve small scale dynamics that looks chaotic on large scale. Such phenomena are fluid motion, fire, and smoke. Although their simulation is difficult and usually has high resource requirements real-time simulation is possible by making compromises. In this paper we present a method to simulate smoke in real-time by taking advantage of the GPU. We also present a way to include solid objects considered as obstacles in the simulation. A disadvantage of the simulation is that the space for simulation is limited. The obstacles outside the simulation grid have no effect. In this paper we introduce a way to overcome this.

**Keywords:** Smoke simulation, Navier-Stokes equations, Obstacles, Real-time, GPGPU

## 1 Introduction

Natural phenomena are still a researched field in computer graphics. In engineering their simulation is essential to accurately predict their behaviour. In the game industry accurate visualizations make the virtual worlds more realistic. Even though we know their inner dynamics their simulation remains difficult due to the tremendous amount of the resources required. In most cases it is necessary to run these simulations fast. In computer games the phenomena are only part of the virtual scene that needs to be updated and rendered within a fraction of a second.

Smoke is one of the complex natural phenomena. It is basically a flow of matter, similar to fluids and fire. Its motion and dynamics are governed by the laws of physics on the scale of particles, but the tremendous amount of particles form a complex system that is difficult to simulate. The calculation capacity limits the number of particles we can use to create the effect, and a low number of particles make the effect visually unrealistic. The approach to simulate the individual particles is referred to as the Lagrangian view.

There is another approach, called the Eulerian view, where the space is divided into discrete cells forming a grid. Every cell contains macroscopic quantities that give information about the space and matter inside that cell (like velocity, pressure, density, etc.). A simulation in Euler space does not calculate the motion of each particle, but the changes of the quantities in the cells. An advantage of this view is that it is easy to approximate gradients in a grid. Using this view, the accuracy of the simulation highly depends on the resolution of the grid instead of the amount of particles used. The higher this resolution the better it approximates the real life behaviour.

The value of a simulation method is also measured by how well it can be integrated into a virtual world. An absolutely essential part of these scenes are solid objects. A world, even a virtual one would be boring with only fluid, or gas in it. Solid objects are considered obstacles and they drastically affect the motion of other objects, fluids, and gases. That is why their affect on smoke should be included in its simulation.

## 2 Related work

The particle based approach was the dominant solution for fluid simulations mostly because the necessary computing time can be controlled by limiting the maximum number of particles. In real-time applications there was no capacity for solving equations. With the particles the computations are very simple as long as there is no need to simulate particle-particle interactions. An accurate simulation however requires those calculations as well.

The Eulerian approaches had very limited potential until Stam introduced the “stable fluid” simulation in [8]. This presented the first unconditionally stable algorithm for solving the Navier-Stokes equations. The method was still far from applicable in real-time. Harris [9] presented an implementation that uses the GPU on a video card based on Stam’s work. He also tested algorithms and recommended the Jacobi iteration to be used in the solving process.

There are also articles and papers [1][3] on implementing a real-time fluid simulator based on the Navier-Stokes equations using the modern GPUs. The method is based on storing the simulation grid in textures and implementing the algorithms in pixel/fragment shaders. Rendering the texture that contains the current state onto another texture using the special shaders executes an algorithm. The target texture then contains the new state of the grid, containing the updated values in the cells. Even if some technical details are different in the implementations, the simulation in [1] and [3] are based on this method. We also used this technique to run the simulation on GPU.

# 3 Simulation

## 3.1 Dynamics

The basics of the simulation are to numerically calculate the flow in the simulation grid. Every cell has a velocity which describes the average velocity of the matter inside the cell. Based on this, it is possible to approximate the distribution of quantities in the next time step. To simulate the motion we need to update the velocities according to the physical quantities. The Navier-Stokes equations offer a possible solution to that problem. The momentum equation (1) describes the change of the velocity.

$$\frac{\partial\vec{u}}{\partial t}+\vec{u}\cdot\nabla\vec{u}+\frac{1}{\rho}\nabla p=\vec{g}+\nu\cdot\nabla^2\vec{u} \qquad (1)$$

In the equations $u$ stands for velocity, $p$ for pressure, $g$ for acceleration caused by gravity, $\nu$ for dynamic viscosity, and $\rho$ for density.

The second is the incompressibility equation (2), which guarantees the conservation of mass if the matter is incompressible.

$$\nabla\cdot\vec{u}=0 \qquad (2)$$

Fluids and gases are compressible but here we can make a general simplification and regard them incompressible. In most cases the compressibility is irrelevant because extreme circumstances are required to produce visible compression. We don’t lose much with this constraint but we gain a lot by enforcing the conservation of material.

The momentum equation is a special form of Newton’s second law. The numerical solving of this formula alone is easy since the variables are all contained in the cells as quantities or can be calculated from the stored quantities. It can be further simplified by dropping viscosity. It is a very important force in the equation and must be counted in for realistic results but we can still leave it out. The reason is that the numeric nature of the simulation has an effect which is similar to viscosity and is very strong and visible. This is called numeric dissipation because the accumulating numeric errors in the calculations eliminate the small details in the motion. An example is the blur of turbulent flow when the vortex is created by the simulation, but with time the numeric dissipation blurs it into a simple moving mass. We replace the gravity with a variable force that creates acceleration $a$.

$$\frac{\partial\vec{u}}{\partial t}+\vec{u}\cdot\nabla\vec{u}+\frac{1}{\rho}\nabla p=\vec{a} \qquad (3)$$

The incompressibility equation derives from the constraint that the amount of quantities flowing out from a certain place must equal the quantities flowing in. This is a constraint for the momentum equation which makes the solution difficult to calculate. We have to ensure that the velocity field is divergence free. A projection operator can be defined which takes $w$ as input and gives $u$ as output. It is called projection because executing the same operator again on a field leaves it unchanged. It is necessary in this case since we want a divergence free velocity field unchanged after a projection on it.

The projection operator should solve a linear equation. If $w$ is a velocity field containing divergence and $u$ is divergence free than (4) is true.

$$u=w-\nabla p \qquad (4)$$

Combining this with the incompressibility constraint gives equation (5). This can be interpreted as a linear equation and can be solved for pressure.

$$\nabla^2 p=\nabla\cdot w \qquad (5)$$

Using equation (5) we can update the pressure values in the cells so when changing the velocities using the pressure gradient we eliminate the divergence. Solving equation (4) for pressure is possible with a number of algorithms. We used the Jacobi iteration to do this. This is considered an any-time algorithm since it converges to the solution. We need to run several iterations to get an accurate update of the pressures. Based on experience 20-30 cycles produce acceptable results. The disadvantage of this method is that the results are not perfect so this cannot ensure a perfectly divergence free velocity field nor the perfect incompressibility of matter. The first creates an error in the simulation but the second can also be considered an advantage since it makes the simulated material slightly compressible.

Now that the velocity field is updated and divergence free, the simulator only needs to calculate the advection of quantities in the grid. It would seem almost trivial to just use a forward integration to get the next position of the quantities in a cell. It would be a solution with Lagrangian viewpoint, but in Eulerian space it is unconditionally unstable. Not to mention that this could not be done on a GPU. There is also an algorithm that traces back the flow by approximating the source position based on the velocity. The disadvantage of this solution is that it is only an approximation and only first order accurate. The main reason for the error is that the calculated position of the source is based on the velocity in the destination cell at the previous moment. Its advantage is that the advection is unconditionally stable so it can even be used with large time steps. Because the theoretical view of this algorithm is Lagrangian but it is an advection in Eulerian space, this advection scheme is called semi-Lagrangian advection. Many simulators use this because it is fairly easy to implement and it always remains stable. The only condition is that it requires a divergence free velocity field to work. In practice the relatively small divergence that is left after the projection is acceptable.

For the implementation, the solving of the Navier-Stokes equations is broken down into the steps introduced before. The first step is the advection on a divergence free velocity field. It is followed by the application of the accelerations, forces and other manipulations on the quantities. Then the simulator runs the projection operator to apply the pressure gradient and make the velocity field divergence free again.

## 3.2 Smoke

The basic simulation is able to approximate a general flow of the quantities in the cells. These quantities must include the velocity and the pressure in order to run the basic simulation. New quantities can be added to serve the needs of special simulations. These new attributes are not necessarily physical. For the smoke we should add two more quantities which both describe physical attributes. The new quantities are advected the same way the previously introduced quantities do so the advection scheme used previously is also applicable with the new quantities

The density of the smoke is an essential value. It does not affect the dynamics but is required for the visualization of the smoke. The other one is the temperature which creates buoyancy. This force is calculated by (5), where $m$ is the molar mass, $R$ is the universal gas constant, $T$ is the temperature of the smoke, $T_0$ is the ambient temperature.

$$\vec{f}_{bouyancy} = \frac{p \cdot m \cdot g}{R}\left(\frac{1}{T_0} - \frac{1}{T}\right)\vec{y} \qquad (6)$$

Using the Boussinesq approximation the force to apply in the simulation is sown in equation (6). It includes the gravity as well as the buoyancy, and simplifies them in a linear equation. The constants α and β can be chosen based on experiences and test results.

$$\vec{f}_{bouyancy} = \left(-\alpha \cdot \rho + \beta \cdot (T - T_0)\right) \cdot \vec{y} \qquad (7)$$

Applying this force after advection and using the density values for rendering the general flow simulation is capable of simulating the motion of smoke.

Based on these algorithms we implemented a smoke simulator in 2D. It is mainly for a sample implementation. The results are shown on figure 1, which pictures are taken as screenshots from the simulator.

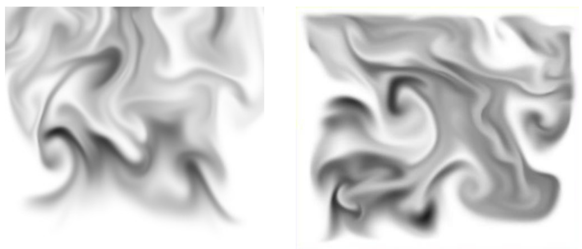
Figure 1 : Smoke simulation in 2D

## 3.3 MacCormack advection

The simulation is far from perfect. It has an inevitable error because of its discrete nature. There are several sources of this error. The trivial solution to decrease the error is to increase the resolution of the simulation grid. This approach works but has a high cost since the simulation algorithm is to be executed for every cell. The performance of the simulator is significantly lowered by a higher resolution of the grid as shown by Table (1).

Another source of the error is the advection scheme used in the implementation because it is only an approximation. The semi-Lagrangian algorithm is only first order accurate. Its accuracy highly depends on the divergence in the velocity field, and the distance between the source and the destination. A more accurate advection scheme could also greatly improve the results.

Another advection scheme that is called MacCormack advection gives second order accurate results and has low performance cost. We used the algorithm that [6] suggests and which is also used by [3]. It relies on the following equations if $A$ is a first order accurate advection scheme, $q^n$ is a quantity in the current moment, $q^{n+1}$ is a quantity in the next moment and $A^{inv}$ is the inverse of $A$ (like time was going backwards).

$$\hat{q}^{n+1} = A(q^n)$$
$$\hat{q}^{n} = A^{inv}(\hat{q}^{n+1})$$
$$e = \frac{\hat{q}^n - q^n}{2}$$
$$q^n = \hat{q}^{n+1} - e = \hat{q}^{n+1} + \frac{q^n - \hat{q}^n}{2}$$

The MacCormack advection uses a second advection step to estimate the error of the first one and then eliminates that error. This advection scheme is second order accurate and requires not much more than two semi-Lagrangian advection steps. There is a severe problem with this algorithm, its stability. It is only conditionally stable, which means that during the simulation it can destabilize and give completely useless results (as it is visible in Figure (2)). These incidents can be avoided by preventing the algorithm to create new extrema. To make it unconditionally stable limiters must be applied as presented in [6]. We decided that the advection should revert to the first order accurate result in case the limits would be passed. Since the semi-Lagrangian advection scheme and the results within the boundaries of the limiters are both unconditionally stable the resulting advection scheme remains stable under any circumstances.

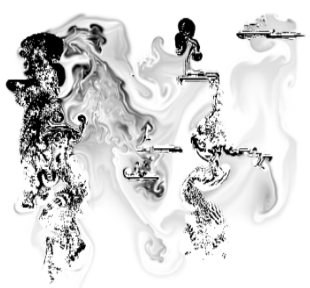
Figure 2 : Result of unstable advection

Figure 3 shows the quality gained by this more accurate advection because there are clearly more details, even though the grid resolution is the same. Table 1 presents the low performance cost of this method. The MacCormack advection can also be used to gain performance by lowering the resolution since (as Figure

4 demonstrates) the quality of the semi-Lagrangian advection in a high resolution grid can be achieved with the MacCormack method on a lower resolution.

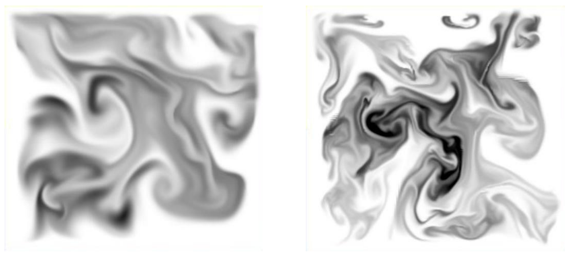
Figure 3 : left: semi-Lagrangian, right: MacCormack

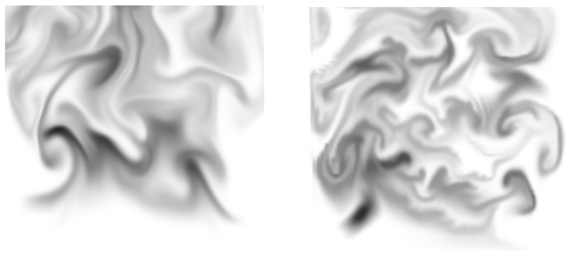
Figure 4 : left: 256x256 semi-Lagrangian, right: 128x128 MacCormack

| | semi-Lagrangian | MacCormack |
|---|---|---|
| **128x128** | 443 | 413 |
| **256x256** | 331 | 312 |
| **512x512** | 93 | 90 |

Table 1 : Performance[*] results of the 2D simulator [FPS]

## 3.4 Simulation in 3D

In a virtual world the smoke should flow in a three dimensional space. The equations do not limit the dimension of the simulation grid, it is only a matter of implementation. A way to use the GPU to run the simulation is to store the values of the grid cells in textures with floating point numeric representation. One channel of a texture cell stores the value of one quantity and since one texture has four channels we must use more than one texture for all the quantities. A special quantity is the velocity which being a vector requires one channel for each of its components.

A crucial question is what texture type to use. Today it is possible to use 3D textures because they can be used as render targets. This feature was introduced in Shader Model 4.0. An example for such implementation can be found in [3]. It makes the shader just as simple as in a 2D simulator leaving only the visualization as the main difference. We used a ray marching algorithm - similar to the one presented by [7] – to visualize the three dimensional grid.

However it is also possible to use 2D textures and divide them into tiles representing the layers of a 3D texture. This flat 3D technique was chosen in [1]. The coding is more difficult in this case because real 2D texture coordinates have to be converted to virtual 3D coordinates. Also the interpolation between the layers has to be done manually in the shaders. The price in difficulties is returned by the gain in performance.

We implemented both versions. The performance of the simulator that uses 3D textures is almost unreasonably low, as it is clearly visible from Table 2. These values do not contain the visualization of the data.

| | Flat3D | 3D |
|---|---|---|
| **32x32x32** | 336 | 18 |
| **64x64x64** | 76 | 9 |
| **128x128x128** | 10 | 4 |

Table 2 : Performance of the 3D simulator [FPS]

## 4 Obstacles

### 4.1 Voxelization

To simulate the effect of solid objects on the smoke's motion it is inevitable to provide data about them in Eulerian space. The vertex based description would be useless in the simulation. A useful form is a voxel based model of the objects. The operation which creates a voxel based model from a vertex based is called voxelization. The main goal is to sign in every grid cell whether or not an obstacle is present, and to store the velocity of a moving solid object. The velocity describes the average velocity of the obstacle's part that is located in the cell. This also means that the velocity is the same for a whole object if it is not animated and not rotating. The accuracy of the simulation with obstacles highly depends on the resolution of the simulation grid since it can be considered as the sampling resolution of the objects physical form.

The implementation depends on the specifics of the obstacles. If we assume all objects have closed surfaces we can use a special rendering technique to voxelize. The result will be the needed information, which voxels are inside and which outside the solid object. The problem is similar to the well known shadow volume, so the solution can be similar as well. There are variations of the implementation. Some use the stencil buffer, some use blending (like [1]). The advantage is that the objects can be any free form surfaces that are closed. The disadvantage is that the model used for the voxelization must be rendered for each layer of the grid. This is why a simplified object is recommended for the voxeliation and not the same as the one used in the scene.

If we limit the types of the obstacles to quadrics we can use a simple shader to calculate the inside-outside information for the whole grid. This is more performance-friendly since we do not need to render the object itself only specify its parameters. In case of a sphere the shader can simply determine if a voxel is inside by comparing its distance from the sphere's centre to the sphere's radius. The disadvantage is the obvious

[*] Test configuration: Athlon 64 X2 3800+, 4GB RAM, Radeon HD3870 512 MB, Vista (64 bit)

fact that the objects are limited. We used this method for voxelizing a sphere in our sample program.

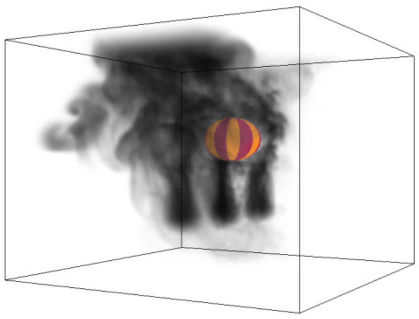

Figure 5 : 3D simulation with an obstacle

## 4.2 Boundary conditions

To create the effect of a solid object in the simulation grid we must modify the simulator. An important change is to ignore the values in those cells that are occupied by obstacles whenever calculating gradients, or divergences. This can be done by using the value in the centre cell for the calculation instead of the real value in that neighbouring cell. Without this modification the velocity divergence and the pressure gradient could be corrupt at the boundaries of the obstacles.

The other change in the shaders should be the enforcement of the boundary condition. This condition limits the velocity in the neighbour cells of the obstacles. It is essential to keep the matter from flowing inside the solid objects. If the velocities at the boundaries have to be corrected, then these values create a divergence in the velocity field. The created projection operator will then smooth the velocity field. This allows the boundary condition to affect the velocity field near the obstacles. During the final step of the projection the neighbour cell's velocity can be changed directly so it is the best place to enforce the boundary condition.

The condition itself may vary as the specific application requires. Usually the free-slip condition is used by fluid simulators and it is formally defined by equation (7) if $u$ is the velocity in a cell, $u_{obstacle}$ is the obstacle's velocity, and $n$ is the surface normal of the obstacle. It only allows the matter to flow parallel to the obstacle's surface which creates an effect similar to the real behaviour of fluids. They appear to stick to the surfaces and this can be reproduced using the free-slip condition.

$$\vec{u} \cdot \vec{n} = \vec{u}_{obstacle} \cdot \vec{n} \quad (8)$$

Although it is realistic for fluids this behaviour looks strange. Also it would require accurate surface normal vectors to produce the desired effect. Instead, we used a simple estimation by the relative position of the occupied neighbouring cell to the centre cell.

This estimation makes it possible to limit the velocity in a cell to zero which can produce an anomaly. When smoke flows into such cell it acts as a source because the neighbour cells are likely to pull matter from that cell while the smoke is still completely stuck there. For these reasons we chose to use a condition that lets the smoke flow away from obstacles. Its formal description, equation (8) is very similar to the free-slip condition only it allows more freedom.

$$\vec{u} \cdot \vec{n} \geq \vec{u}_{obstacle} \cdot \vec{n} \quad (9)$$

With this extra freedom it is possible that neighbouring cells pull out matter from cells occupied by obstacles. It has to be prevented by extending the advection to check for this condition and change the result accordingly.

Using the voxelization and the integration of the boundary condition the simulator is now capable of handling solid objects. It looks realistic and it does not require too much extra resources. Unfortunately there is a fundamental problem. The simulation grid is only a predefined part of the virtual space. It means that only those objects that at least intersect the box defined as the simulation space can affect the simulation. The objects outside the grid have no effect whatsoever and no matter how close they are.

# 5 Outer obstacles

## 5.1 Effect of inner obstacles

The simulation of the outer obstacles is not possible but the approximation of their effect can be applied indirectly to the simulation. We used an acceleration field to do this because it is not a direct manipulation of the velocities but it still offers some level of local control. The first step is to observe the simulated effect of the objects inside the grid.

For the observation it is necessary to visualize the velocity field. A ray marching algorithm like the one used for the density is suitable with a slight modification. The output colour should represent the vector of velocity, and the colours are linked to the three components. The problem is that the output is limited and the velocities are not. Because of this problem a visible range must be defined for the visualization.

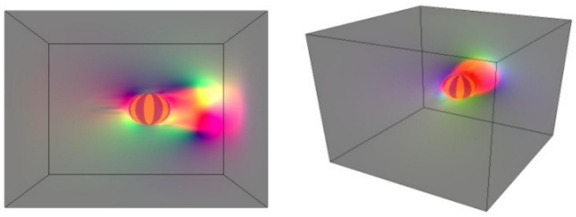

Figure 6 : Velocity field

In Figure 5 it is visible that the dominant effect is that the matter in front of the obstacle is pushed. It is like a shockwave before the moving object. This is the case when the pressure values are visualized on Figure 6. The negative and positive values are coded into separate colour channels. The pressure drops even in front of the object but before the shockwave.

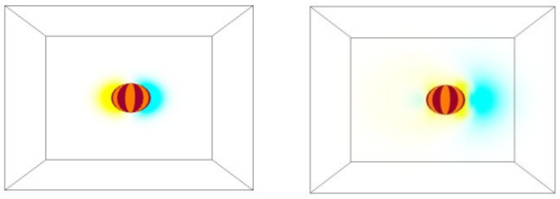
Figure 7 : Pressure field

The shockwave appears as the objects starts moving. It is getting stronger and moves away from the surface of the object. The amount of the changes in the velocity field depends on the velocity of the object. The effect remains similar but the strength differs as its speed changes.

## 5.2 Approximation

To approximately reproduce the effect of solid objects, the following algorithm attempts to create a shockwave similar to the simulated. Any vector in Cartesian space can be defined by the linear combination of the axes of the coordinate system's basis. Including the negative directions of the axes the vectors can be expressed using only non-negative coefficients. We express an objects velocity using the positive and negative directions of the three axes. This division into separate components allows storing data about the object's movement through time without any increase in the storage space. The program updates the parameters for each segment instead of always storing new values in every frame. Every segment has individual parameters used to calculate the accelerations. The parameters are the centre, the distance of the peak of the wave from the centre and the maximum of its strength. The strongest acceleration is located in the intersection of the sphere defined by the centre and distance parameters and the vector defined by the direction of the velocity. The accelerations fade away as they are farther from this point.

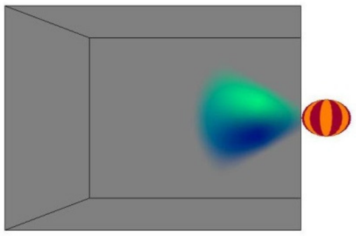
Figure 8 : Accelerations to approximate obstacle's effect

Also the direction is important so the accelerations are also fading as they are farther from the direction of the movement. Based on these assumptions the equations to get the acceleration in a position *p* are:

```
Dist[i] = length(p – Centre[i]);
DistStr[i] = max(PeakStr[i] – abs(PeakDist[i] – Dist[i]), 0);
DirStr[i] = max(dot(p – Centre[i], Dir[i]) – 0.8, 0)•5.0;
Acc [i] = normalize(p – Centre[i])•DistStr[i]•DirStr[i];
FinalAcc = $\sum_{i=0}^{5}$ Acc[i];
```

This calculation is implemented in a shader that renders onto the texture representing the acceleration field. This shader has to be executed for every solid object.

## 5.3 Parameters

The parameters used to calculate the accelerations should be updated every time step according to the object's movement. To get an effect that is getting stronger by time the maximum strength should be increased in case of a continued movement in the same direction. Also the distance of the wave should be slightly increased as well. When the object starts moving the distance is reset to a constant chosen based on the object's size. A possible choice could be the radius of the object's bounding sphere. The starting effect is almost always irrelevant since it will take some time for the object to get near the simulation grid. The centre parameter is updated as the object is moving. When an obstacle is no longer moving in the same direction, the centre should remain unchanged and the distance should be rapidly increased. The strength of the shockwave should be decreased so the effect will fade away after the object stopped. Based on these directives the algorithm to update the parameters is shown below.

```
SpeedStrength[i] = dot(ObstacleSpeed, Dir[i]);
if (SpeedStrength[i] > 0) {
    if (PeakStr[i] < DragCoeff[i]•SpeedStrength[i]) {
        PeakDist[i] = StartDist[i];
    } else {
        PeakDist[i] = PeakDist[i] + 0.1 • Δt;
    }
    PeakStr[i] = DragCoeff[i]•SpeedStrength[i];
    Centre[i] = ObstaclePosition;
} else {
    PeakStr[i] = PeakStr[i]•max(0.0, 1.0 – 5.0•Δt);
    PeakDist[i] = PeakDist[i] + Δt;
}
```

## 5.4 Results

The simulator program voxelizes the objects entering the simulation grid and uses the approximation of their effects otherwise. This made it possible to observe and compare the simulated and the approximated effect of obstacles. In the left side of Figure (8) the object is inside the simulation grid but in the sequence on the right side of Figure (8) it is outside. The distance between the objects starting position and the timing of its movement is the same in both cases. Therefore an ideal approximation should produce the same changes in the smoke's motion.

By looking at the two sequences (Figure 8) it can be stated that the result of the approximation is close to the simulated. The details are different but the main changes in the movement are very similar. The effect of the obstacle outside the grid can be considered as part of the simulation.

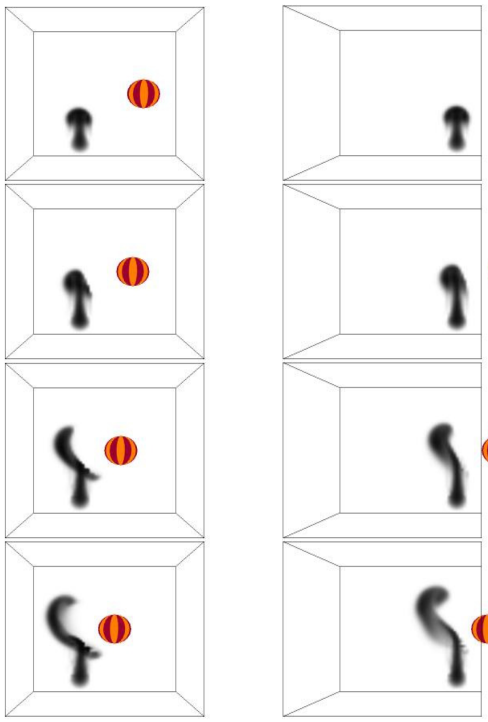
Figure 9 : left: simulated, right: approximated

## 6 Conclusion and future work

The theory and the mathematical basis of the smoke simulation is well established. Methods to implement them in a real-time simulator are also available and there are several sources that present them. Most of these rely on the computing power of the GPU because the algorithms can be executed in parallel on each cell. In this paper we presented a way to implement a real-time smoke simulator. The interaction with solid objects is a highly important part when integrating the simulator into a virtual world. The limits of the Eulerian view require some form of extension to make this a competitive option in more scenarios. The method present can approximate the effect of solid objects with a visually acceptable error. Depending on the requirements this could increase the applicability of simulators with Eulerian space. The advantage of the algorithm is that its requirements are independent from the movement of the objects. The disadvantage is that it has to be executed for every obstacle separately.

The algorithm can be extended with the monitoring of the obstacles and choosing only those that probably affect the simulation to run the approximation with them. This could save time in a general virtual world where many objects could be discarded easily. The approximation itself could be improved to give more accurate results. The drag coefficient used in the update of the parameters is manually specified. It would worth some research to automatically calculate this number.

The simulator part could also be improved. The voxelization could be modified to work with free form objects like in [1] or in [3] and implement the approximation for them as well. Also the rotation of the objects is not part of the voxelization and the approximation therefore it could be examined and determine if it changes the approximation's accuracy.

## Acknowledgements

This work has been supported by the National Scientific Research Fund (OTKA ref. No.: T042735).

## References

[1] Tamás Umenhoffer and László Szirmay-Kalos, *Interactive Distributed Fluid Simulation on the GPU*, In: Petar Biljanovic, Karolj Skala (editor) MIPRO 2008: Grid and Visualization Systems. Opatija, Croatia, 2008. pp. 236-242.

[2] Robert Bridson, Matthias Müller-Fischer, *Fluid Simulation*, International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2007 courses, San Diego, California, SESSION: Course 31: Fluid simulation, pp. 1 – 81, 2007

[3] Keenan Crane, Ignacio Llamas, Sarah Tariq, *Real-Time Simulation and Rendering of 3D Fluids*, GPU Gems 3, Chapter 30, pp 633 - 674, 2007

[4] Enhua Wu, Youquan Liu, Xuehui Liu, *An Improved Study of Real-Time Fluid Simulation on GPU*, Computer Animation and Virtual Worlds, Volume 15 , Issue 3-4 (July 2004), Special Issue: The Very Best Papers from CASA 2004, pp. 139 – 146, 2004

[5] Ronald Fedkiw, Jos Stam, Henrik Wann Jensen, *Visual Simulation of Smoke*, International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 15 - 22, 2001

[6] A. Selle, R. Fedkiw, B.-M. Kim, Y. Liu, J. Rossignac, *An Unconditionally Stable MacCormack Method*, J. Sci. Comput. 35, pp. 350-371, 2008

[7] Simon Green (NVIDIA Corporation), *Volume Rendering For Games*, Game Developers Conference 2005, 2005

[8] Jos Stam, *Stable Fluids*, Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference Series, pp. 121-128, 1999

[9] M. Harris, W. Baxter, T. Scheuermann, A. Lastra, *Simulation of cloud dynamics on graphics hardware*, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, San Diego, California, pp. 92 - 101, 2003