EG UK Theory and Practice of Computer Graphics (2006)
M. McDerby, L. Lever (Editors)

Physically Based Simulation and Visualization of Fire in Real-Time using the GPU

Samuel Rødal[†], Geir Storli[‡], and Odd Erik Gundersen[§]

Visualization Group, Department of Computer and Information Science, NTNU, Norway

Abstract

In this paper we present a physically based framework for real-time simulation and visualization of fire using the GPU. The physics of fire is modeled through a combination of a fluid solver and a combustion process causing the characteristic motion of fire. The simulation results are then rendered using a particle system combined with a black-body radiation model where the physically based simulation governs both the motion and appearance of the particles. By performing individual slice simulations in 2D and combining them using volumetric extrusion we achieve better performance than by performing the simulation in 3D without compromising the visual quality. Thus, achieving our goal of visualizing bonfire and torch-like fire effects with high visual quality in real-time.

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

1. Introduction

Fire is a powerful effect and can be used in virtual environments like games in order to increase immersion, suspending the disbelief of the user. However, it is hard to reproduce the chaotic and turbulent behavior of fire using traditional procedural methods. Fire is a very intense visual phenomenon, and thus needs to be realistically reproduced in order to convince the eye of a human observer.

Our view is that the laws of nature play an important part and must be taken into consideration when simulating visually convincing fire. Thus, we take a physically based approach to the simulation and visualization of fire. This also reduces the need to come up with various ad hoc approximations which can be hard to justify. With the increased power and programmability of modern GPUs, more processing power is available for performing physically based simulations while still achieving real-time frame rates.

Our main contributions are performing the combustion process and fluid simulation from [MK02] on the GPU and using a black-body radiation model in combination with a particle system also running on the GPU. The combustion simulation is combined with vorticity confinement as used in [NFJ02, WLL04, KW05]. The black-body radiation color table is precomputed and stored in a lookup texture on the GPU and used when calculating the color radiated by the hot exhaust gas. Another contribution is combining the combustion process with volumetric extrusion [RNGF03, KW05]. We have achieved our goal of visualizing bonfire and torch-like fire effects with high visual quality in real-time.

This paper is organized as follows. After a brief overview of related work in chapter 2, we describe both the theory and implementation of our framework for simulating and visualizing fire in chapter 3. In chapter 4 we present the results and analysis and chapter 5 concludes with a summary and future work.

2. Related work

We divide between methods used for simulating and methods used for visualizing fire and will treat them separately. Although other techniques have achieved visually pleasing

† knuterro@idi.ntnu.no
‡ geirst@idi.ntnu.no
§ odderik@idi.ntnu.no

results [Ngu04, LF02], our focus is on physically based simulation techniques.

2.1. Simulating fire

[NFJ02] present an offline physically based method for modeling fire. A thin flame model is developed, using an implicit surface to represent the reaction zone where vaporized fuel reacts with oxygen and creates hot gaseous products. The movement of the implicit surface is tracked using the level set method, and the flow of vaporized fuel and the flow of hot gaseous products are modeled independently using the incompressible Navier-Stokes equations.

[MK02] model fire in real-time by using a combustion process in addition to the incompressible Navier-Stokes equations. The equations are discretized into a 3D grid and solved using a fluid solver similar to the approach in [Sta99]. The fluid solver is used to control the motion of a three-gas system consisting of oxidizing air, fuel gases, and exhaust gases. The combustion process models the reaction that occurs between oxygen and fuel gases at a certain temperature threshold, resulting in the generation of exhaust gases and the spread of heat.

[WLMK02] use the Lattice Boltzmann model instead of the Navier-Stokes equations to simulate the fire process using a temperature field to model the generation of smoke. The fire behavior is mainly affected by the direction of wind and the location of fuel and non-burning objects.

[KW05] use volumetric extrusion of a 2D fluid simulation to simulate fire. However, instead of using a physically based combustion model to guide the fluid simulation they use pressure templates to disturb the flow in order to create a turbulent fire. Heat is modeled and used to scale the pressure templates to create more turbulence at the base of the fire.

The simulation is performed on the GPU in [WLMK02] and [KW05], and on the CPU in [NFJ02] and [MK02].

2.2. Visualizing fire

In [NFJ02] a Monte Carlo ray tracing approach is used to visualize the fire, treating the hot gaseous products as a participating medium. The radiance emitted by the medium is modeled using black-body radiation.

A hardware based volume rendering technique is used in [MK02] for fire visualization. The output of the fire simulation is voxelized data of the fuel gas, exhaust gas, and heat in the system, and each voxel is replaced by a semitransparent polygon where the level of transparency is controlled by the density of fuel gas and exhaust gas in the voxel. The fuel gas is shown in yellow and the reaction zone where the combustion occurs is shown in red.

Texture splats are used to visualize the fire in [WLMK02]. The velocity volume from the simulation is used to advect the display primitives, which are removed from the system when the fuel they are holding is consumed by combustion. The display primitives are rendered using texture splatting.

[KW05] use a GPU implemented particle system to visualize the fire. The velocity vector field from the fluid simulation is used to update the particle positions, and the particles are rendered using textured point sprites.

Recently, the ability of the GPU to perform both the simulation and visualization of particle systems has been explored. [Lat04] and [KSW04] introduce a full GPU implementation of the simulation and rendering of a particle system. The particle positions are stored in a 2D texture on the GPU and the current particle velocities in another 2D texture. The velocity texture and position texture are both updated in separate rendering passes. In [KSW04] the updated particle positions are rendered into a vertex array memory object. This array is processed when rendering the particles to the screen. [Lat04] propose the use of vertex texture fetch to read the particle positions from a vertex shader responsible for rendering the particles to the screen.

3. Simulating and visualizing fire on the GPU

Here we first give a general overview and then separately present our approach for simulating fire and our approach for visualizing fire. Then, we present the complete algorithm, and finally the implementation details are discussed.

3.1. Overview

We simulate the evolution of a fuel gas field, an exhaust gas field, and a temperature field, in co-evolution with a velocity field. The combustion process converts fuel gas to exhaust gas and heat when the temperature exceeds a certain threshold. Buoyancy due to heat then causes the hot exhaust gas to rise, which in combination with vorticity confinement causes the characteristic fire-like motion. All the simulation steps are efficiently performed on the GPU by packing the fuel gas, exhaust gas, and temperature field into a single texture. This packing is similarly performed for the components of the velocity field. We implement two variations of the simulation. One performs the simulation in full 3D and the other performs individual 2D slice simulations and combines them through volumetric extrusion.

We visualize the result of the fire simulation using a black-body radiation model and a particle system. A black-body radiation table is precomputed and stored as a lookup texture on the GPU. The black-body radiation approximates the radiation from the hot exhaust gas. Particle data is stored in textures on the GPU, which are updated by the particle system simulation and used by the vertex shader to specify the particle positions. The velocity field from the physically based simulation is used to advect the particles, while the temperature and exhaust gas fields are used in combination

with the black-body radiation table to compute the particle colors.

3.2. Simulating fire

Our fire simulation is largely based on the approach presented in [MK02], which combines the stable fluid solver from [Sta99] with a combustion process modeling fuel gas, exhaust gas, and temperature fields. The fields are limited to the finite computational domain, which represents the area where the fire is located, and are discretized into a voxel or grid structure. In addition, we use the vorticity confinement method used in [NFJ02], [WLL04] and [KW05] to create a more turbulent flame.

3.2.1. Velocity field

The velocity field $\mathbf{u}$ is a vector field specifying the direction and speed of air and gas. It is governed by the Navier-Stokes equations for incompressible flow with zero viscosity, shown in equation 1 and equation 2.

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\,\mathbf{u} - \nabla p + \mathbf{F} \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2}$$

The first term on the right-hand side of equation 1 is the self-advection of the velocity, causing velocity to be pushed along itself. The second term, $-\nabla p$, is the gradient of the pressure field, causing velocity to move from areas of high pressure to areas of low pressure. The pressure field is used as a correcting term, ensuring that equation 2 holds by projecting the velocity field onto its divergence free component. The velocity field needs to be divergence free in order to conserve mass according to equation 2.

The last term on the right hand side of equation 1 is the external force acting on the velocity field. The external force consists of several separate forces, shown in equation 3:

$$\mathbf{F} = f_{vorticity} + f_{gravity} + f_{buoyancy}, \tag{3}$$

where $f_{vorticity}$ is the vorticity confinement force, $f_{gravity}$ is the gravity force due to fuel and exhaust gases, and $f_{buoyancy}$ is the buoyancy force due to heat. These forces are described later in equations 9, 10, and 11.

3.2.2. Fire density fields

We use the fire density fields as a common term for the scalar fuel gas, exhaust gas, and temperature fields. To create a realistic and moving flame, we use the stable fluid solver and the velocity field to advect the fire density fields throughout the computational domain.

We use a slightly modified version of the density advection equation presented in [Sta99] to describe the evolution of the three fire density fields: fuel gas g, exhaust gas a, and temperature T. The modified equations governing the evolution of respectively the fuel gas field, exhaust gas field, and temperature field are given by 4, 5, and 6. These three equations correspond to the equations used in [MK02].

$$\frac{\partial g}{\partial t} = -\mathbf{u}\cdot\nabla g + \kappa_g\nabla^2 g - \alpha_g g + S_g + C_g \tag{4}$$

$$\frac{\partial a}{\partial t} = -\mathbf{u}\cdot\nabla a + \kappa_a\nabla^2 a - \alpha_a a + C_a \tag{5}$$

$$\frac{\partial T}{\partial t} = -\mathbf{u}\cdot\nabla T + \kappa_T\nabla^2 T - \alpha_T T + S_T + C_T \tag{6}$$

The first term on the right hand side of each equation is the advection term, causing the fire density fields to be carried along by the velocity field. The second term is the diffusion term, simulating the tendency of the densities to spread out. The third term governs the dissipation of the fire density fields, and is controlled by the dissipation rates α_g, α_a, and α_T. We also have source terms, S_g for fuel gas and S_T for temperature, which are used to inject fuel gas and temperature in order to get the fire started and to keep it going. The remaining terms C_g, C_a, and C_T are related to combustion, and are discussed in detail in section 3.2.5.

3.2.3. Vorticity confinement

To achieve real-time results, we have to use a rather coarse grid in our simulation. In addition to this, the stable fluid solver suffers from some numerical dissipation, meaning that much of the low-level turbulence that is important for achieving a realistic flame is lost. To counterbalance this, we use the vorticity confinement method, also used by [NFJ02], [WLL04], and [KW05], to find the vortices that are formed in the velocity field. A vortex is the center of a rotational movement. The vorticity at a given field point thus measures how much rotational movement is present there. The vorticity ω of the velocity field $\mathbf{u}$ is calculated with equation 7.

$$\omega = \nabla\times\mathbf{u} \tag{7}$$

Next, the normalized gradient $\mathbf{N}$ of $|\omega|$, pointing from lower to higher concentrations of vorticity, is calculated from equation 8, and finally we calculate the vorticity force $f_{vorticity}$, given in equation 9. The parameter ε controls the strength of the vorticity confinement, and h is the distance between two grid cells.

$$\mathbf{N} = \frac{\nabla|\omega|}{|\nabla|\omega||} \tag{8}$$

$$f_{vorticity} = \varepsilon h (\mathbf{N} \times \omega) \tag{9}$$

3.2.4. Gravity and buoyancy

Fuel gas and exhaust gas are pulled down due to a gravity force acting on the velocity field. The gravity force is given by the following equation:

$$f_{gravity} = f_g (g + a) \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} \tag{10}$$

The constant f_g determines the strength of the gravity force, while g and a are the amount of fuel gas and exhaust gas respectively.

Like the gravity force, the buoyancy force acts on the velocity field. The buoyancy force causes hot air to rise, and is given by the following equation:

$$f_{buoyancy} = f_b (T - T_{ambient}) \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{11}$$

The strength of the buoyancy force is determined by the buoyancy constant f_b, the temperature T, and the ambient temperature constant $T_{ambient}$, which is the temperature of the surrounding air. To create realistic fire the buoyancy force is crucial, as rising air is one of the main causes of the characteristic and turbulent appearance of the flame.

3.2.5. Fire combustion

An important part of the physically based simulation is to take into account what happens when fuel gas reacts with oxygen creating exhaust gas and heat. To simulate the combustion we choose an approach similar to the one in [MK02].

The combustion will only occur if the temperature is above a given threshold temperature $T_{threshold}$. In contrast to [MK02] we assume that there will always be enough oxygen to react with the fuel gas, which simplifies the equation for the combustion parameter:

$$C = rbg, \tag{12}$$

where r is the burning rate parameter describing how fast the fuel gas can be burned, b is the stoichiometric mixture describing the amount of oxygen required to burn one unit of fuel, and g is the amount of fuel gas. Equations 13, 14, and 15 use the combustion parameter C, and describe the rate of change of respectively the fuel gas, exhaust gas, and temperature due to combustion.

$$C_g = \begin{cases} -\frac{C}{b} & \text{if } T > T_{threshold} \\ 0 & \text{if } T \leq T_{threshold} \end{cases} \tag{13}$$

$$C_a = \begin{cases} C(1 + \frac{1}{b}) & \text{if } T > T_{threshold} \\ 0 & \text{if } T \leq T_{threshold} \end{cases} \tag{14}$$

$$C_T = \begin{cases} T_0 C & \text{if } T > T_{threshold} \\ 0 & \text{if } T \leq T_{threshold} \end{cases} \tag{15}$$

Because we assume there will always be enough oxygen to react with the fuel gas, the parameter b only controls how much oxygen is involved in the reaction, and will not directly affect how fast the fuel is consumed. In addition the parameter T_0 is used in equation 15 to control the amount of heat that is produced by the reaction.

3.3. Visualizing fire

Using a precomputed black-body radiation lookup table, a fire color field is computed based on the exhaust gas and temperature fields. Then, the fire is visualized using a particle system. The particle positions are updated based on the velocity field and the particle colors are read from the fire color field.

3.3.1. Computing the fire color field

We use Planck's formula for black-body radiation given by equation 16 in order to calculate the intensity radiated by the hot exhaust gas.

$$B_\lambda(T) = \frac{2\pi hc^2}{\lambda^5 \left(e^{\frac{hc}{\lambda kT}} - 1 \right)} \tag{16}$$

By using the wavelengths of red, green, and blue light and the temperature of the gas, we calculate the three intensities B_{red}, B_{green}, and B_{blue}. These intensities have a very high dynamic range whereas the resulting color should have a limited dynamic range suitable for display on traditional computer monitors. To map the given intensities onto a limited dynamic range, we use the exponential mapping function from [Mat97], shown in the following equation:

$$n = 1 - e^{\frac{-L}{L_{average}}}, \tag{17}$$

where L is the original intensity, and $L_{average}$ is a constant controlling the overall brightness. The resulting intensity n will be in the range $[0, 1\rangle$.

Equations 16 and 17 are used to precompute black-body radiation color values for a user specified range of temperatures, which are stored in a one dimensional lookup table.

At the beginning of each visualization step, the exhaust gas and temperature fields are used in combination with the black-body radiation lookup table in order to compute the fire color field. This is done for each cell in the computational domain. Equation 18 shows how the color $\mathbf{c}$ in the fire color field is computed, based on the temperature T, exhaust gas a, and a temperature scaling factor T_{scale}, which is used to control the resulting brightness of the fire. *lookup* is the black-body radiation lookup table.

$$\mathbf{c} = a \times lookup\,(T_{scale}T) \tag{18}$$

3.3.2. Particle system

Like [KW05], we visualize the fire using a particle system defined in the computational domain. Each particle represents a small element of the fire and has a set of associated variables: spawn position, current position, initial spawn delay, current velocity, and color. Spawn position and initial spawn time are given at the beginning of the simulation, whereas the other variables are dynamically updated. A particle's color is specified by an RGBA color value.

Initially, after the given spawn delay, a particle's position is set to the spawn position of the particle. A simple Euler step is later used to update a particle's position as described by the following equation:

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \delta t, \tag{19}$$

where $\mathbf{x}_i$ and $\mathbf{v}_i$ are the position and velocity of particle i respectively and δt is the timestep. Based on the particle position, the particle's velocity and color are found by interpolating samples from the discretized simulation velocity and fire color fields.

When a particle's intensity drops below a certain threshold, it is respawned by resetting its position to its spawn position. A minimum initial lifetime ensures that the particle is not respawned before it has had a chance to enter the fire. Particles are textured like in [WLMK02], creating more low-level detail. Figure 1 shows an example of a texture we used for this.

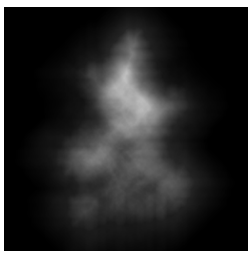

Figure 1: *Texture used for particles*

3.4. The complete algorithm

Both the fire simulation and the visualization are performed for each frame. The fire simulation evolves the velocity and fire density fields based on the combustion process and a stable fluid solver. The velocity and fire density fields are discretized in a grid structure throughout the computational domain and each step of the simulation is performed for all cells in the grid. The fire simulation steps are shown below:

1. Compute velocity forces and add them to the velocity field.
   - Vorticity confinement using equation 9.
   - Gravity using equation 10.
   - Buoyancy using equation 11.
2. Compute fire density forces and add them to the fuel gas, exhaust gas, and temperature fields.
   - Dissipation, part of equations 4, 5, and 6.
   - Source terms, part of equations 4 and 6.
   - Combustion forces using equations 13, 14, and 15.
3. Self-advect and project the velocity field based on equations 1 and 2 using the stable fluid solver.
4. Advect and diffuse the fuel gas, exhaust gas, and temperature fields based on equations 4, 5, and 6 using the stable fluid solver.

After the velocity and fire density fields have been calculated, they in turn are used by the particle system to visualize the fire. A discretized fire color field is now computed based on the temperature and exhaust gas fields, using the precomputed black-body radiation color table. The discretized velocity and fire color fields are then used by the particle system to locate corresponding velocity and color values based on the particle positions. This is shown below:

1. Compute the fire color field using the precomputed black-body radiation color table as described in equation 18.
2. Based on particle positions, sample particle velocities and colors from velocity and fire color fields respectively.
3. Advect particles based on the sampled velocities from step 2 using equation 19.
4. Render particles using a texture splat colored by the sampled colors from step 2.

3.5. Implementation details

To implement the fire simulation and particle simulation on the GPU, we use the FBO (framebuffer object) extension, which allows us to render directly to textures. Textures are used to store both the main velocity and fire density fields as well as particle data (including positions, velocities, and colors). Computations on textures are performed using Cg fragment shaders. GPU implementations of stable fluid solvers are presented in [LLW04] and [Har04].

We implement two versions of the algorithm. One performs a full 3D simulation and the other performs individual

2D slice simulations and combines them using volumetric extrusion [RNGF03, KW05]. In both cases we use flat 3D textures [HBSL03] to be able to simulate several slices at once (depth slices for the full 3D simulation and individual 2D slices for the slice simulation). Figure 2 shows an example of a flat 3D texture used to store two 2D slices of the fire color field.

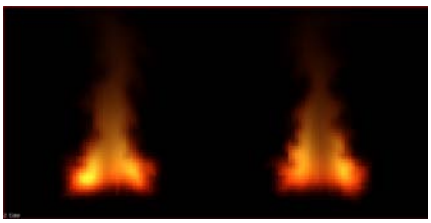

Figure 2: *A flat 3D texture representing two slices of the fire color field at grid size 64x64.*

Sampling from the computational domain using the particle positions requires two different approaches when the computational domain is represented as respectively a full 3D voxel volume or a set of individual 2D grid slices. When sampling from the full 3D voxel volume, simple trilinear interpolation can be used. Sampling from the 2D grid slices is more complicated though. Cylindrical interpolation [RNGF03] is used between the two closest slices and then bilinear interpolation is used within the two slices.

Particles are rendered as quads, which requires four vertices for each particle. As the particle positions are stored in textures on the GPU, we create a vertex buffer object. Each vertex contains the particle's index in the particle textures as well as an offset specifying the corner of the quad. The vertex shader then uses the index to read the position and color from the particle position and color textures respectively. The offset is transformed based on the modelview transformation matrix to make sure that the plane of the quad is parallel to the the viewing plane. Reading from textures in vertex shaders requires a GPU with VS3.0 support. Using a vertex buffer in combination with texture fetch in the vertex shader for particle system rendering was suggested by [Lat04].

4. Results and analysis

In this section, the performance of our algorithm is evaluated and it is compared to other approaches for visualizing fire. The limitations of the algorithm are also discussed.

4.1. Performance evaluation

In this section we first present and compare the performance results from full 3D and 2D slice simulations. We then use the 2D slice simulation coupled with the particle system and compare frame rates using different particle counts with and without rendering the particle system. We also present some visual results. All the tests were run on a 3 GHz Intel Pentium 4 CPU with 512 MB RAM and a NVIDIA GeForce 7800 GT with 256 MB VRAM.

Table 1 shows frame rates from the full 3D simulation with and without visualization. As expected, the frame rate rapidly decreases as we increase the grid dimensions. Also, the visualization is the most time-consuming part at lower grid sizes. As can be seen in figure 3, 32x32x32 is sufficient for achieving visually pleasing results in combination with the particle system.

Grid size	Fire simulation	Visualization
16x16x16	503.3 fps	49.4 fps
24x24x24	202.0 fps	42.5 fps
32x32x32	93.9 fps	34.0 fps
48x48x48	28.2 fps	19.0 fps

Table 1: *Frame rates for full 3D fire simulation, both with and without a particle system with 2048 particles.*

The frame rates achieved by running the 2D slice simulation with and without visualization using both 2 slices and 4 slices are presented in table 2. The results show that a large speed increase is gained from not performing the simulation in full 3D, although the difference is less explicit when visualizing. A simulation grid of 64x64 with two slices is sufficient for good visual results (figure 3). The frame rate is around 5 times as high as for the 3D simulation without visualization at 32x32x32.

	Fire simulation		Visualization	
Grid size	**2 slices**	**4 slices**	**2 slices**	**4 slices**
32x32	608.8 fps	597.1 fps	50.4 fps	49.5 fps
64x64	481.7 fps	297.8 fps	48.0 fps	44.9 fps
128x128	170.5 fps	90.9 fps	41.1 fps	33.9 fps
256x256	47.4 fps	24.0 fps	25.3 fps	16.1 fps

Table 2: *Frame rates for 2D slice fire simulations, both with and without a particle system with 2048 particles.*

Figure 3 shows a side by side comparison of the resulting fire for full 3D and 2D slice simulation. Performance-wise, the 2D slice simulation with two slices at 64x64 clearly outperforms the full 3D simulation at 32x32x32 without compromising the visual quality. In fact, the fire may even appear more detailed because of the higher grid resolution used in the individual slices. The fire has a flickering and turbulent behavior and a lot of low level details. This is representative of a raging bonfire, whereas small camp fires and candle flames usually are smoother.

The number of particles used when visualizing a fire simulation affects the performance and this is illustrated by the results presented in table 3. The number of particles has been

Figure 3: *Visual results from 64x64x2 slice (left) and 32x32x32 3D (right) simulations, with 2048 particles.*

Figure 4: *Particle system with 16384 particles (left) and 256 particles (right).*

varied when running 2D slice simulations with two slices of dimension 64x64. The particle size was the same for all tests resulting in very high fill rate requirements for high particle counts. The first column shows the results from performing the fire simulation and the particle system simulation (updating positions, velocities, and colors) but not rendering the particles. In the second column the particles are rendered as well. As can be seen, visualization is the main bottleneck of the algorithm due to the high fill rate requirements when rendering a large number of particles. The simulation on the other hand is pixel processing limited. Figure 4 shows the result of the particle system visualization at two particle counts. The size of the particle splats has also been varied to compensate for the different particle counts.

Particle count	No rendering	Rendering
256	471.5 fps	221.4 fps
1024	471.5 fps	87.8 fps
2048	457.8 fps	48.1 fps
4096	443.9 fps	25.6 fps
16384	374.3 fps	6.7 fps

Table 3: *Frame rates for two 2D slice fire simulations of size 64x64 and a particle system simulation with and without rendering.*

4.2. Comparison with other approaches

In this section, we briefly compare our fire visualization against the visual elements and performance of other approaches for visualizing fire. We have defined a set of criteria to guide the visual comparison. These criteria are flickering, turbulent behavior, color, texture, and smoke.

Flickering and turbulent behavior as well as a smooth appearance with realistic colors is best achieved by the offline method presented in [NFJ02]. The focus in [MK02] is on simulation and their fire is not visually convincing. The fire in [KW05] fails in capturing the flickering and turbulent elements as convincingly as ours, in addition to having quite saturated colors. Like our fire visualization, the one in [WLMK02] has some low level texture and realistic colors. As opposed to our work, both [NFJ02], [KW05], [MK02], and [WLMK02] include smoke.

[MK02] achieve 20 fps with a 20x20x20 simulation grid. Using the same grid size, we achieve a frame rate at around 360 fps, mainly because we have implemented the complete fire simulation on the GPU.

When using a grid size of 32x32x32 on a NVIDIA GeForce3 Ti 200, [WLMK02] achieve a frame rate of around 215 fps for the fire simulation alone, and 65 fps when rendering 100 texture splats. At the same grid size we achieve a frame rate of 94 for the fire simulation alone, and are able to render 512 particles at 65 fps. Obviously, the LBM implementation is a lot faster than the stable fluid solver, although the accuracy of the former is uncertain. Since they render the display primitives back-to-front, they use more computation power during the rendering step as sorting is needed.

Finally, [KW05] achieve a frame rate of 190 fps on a NVIDIA GeForce 6800 GT when using two 2D simulations at 64x64 each and extruding them to a full 3D volume. In comparison, we achieve a frame rate of 480 with the same grid size. However, we do not compute pressure templates, nor do we explicitly extrude the 3D volume.

4.3. Limitations

Our algorithm for simulating and visualizing fire has several limitations. First of all, the fire is non-interactive in that it does not react to its environment. The fire will thus not be affected by items or obstacles coming in contact with it, which might cause it to look unrealistic.

The characteristic blue core, which appears at the base of small flames and gas flames, is missing from our visualization. As the blue core is not explicitly simulated, we can not

visualize it without resorting to ad hoc approximations uncoupled from the physically based simulation. Smoke is basically exhaust gas which has cooled down and is thus supported by the simulation framework. However, smoke visualization has not been implemented yet.

The 2D slice simulation with volumetric extrusion limits the fire to rotationally homogeneous phenomena like torches and bonfires. For other effects, like for example a flame thrower, a full 3D simulation needs to be performed.

There are no high-level options for controlling the fire. Thus, an animator wishing to create a certain behavior and appearance needs to manually experiment with the quite large amount of different parameters for the algorithm. Particle spawn positions must also be synchronized with the fuel gas source field.

5. Summary and future work

We have presented an algorithm for simulating and visualizing fire completely on the GPU. The algorithm is based on an underlying fluid simulation coupled with a model of the combustion process. A black-body radiation model and a GPU-driven particle system are used in combination to visualize the result from the simulation.

Future work will include exploring whether volume rendering for visualizing the underlying simulation could produce better visual results or better performance than the current particle system. Another possibility is to explore whether it would be possible to model the interaction between the fire and other objects by incorporating obstacle boundaries in the simulation step. Some work has already been done on fluid and complex object interaction on the GPU [LLW04]. Other possible extensions of the algorithm include blue core simulation and smoke.

References

[Har04] HARRIS M. J.: Fast fluid dynamics simulation on the gpu. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Fernando R., (Ed.). Addison-Wesley Professional, 2004, pp. 637–665.

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

[KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (New York, NY, USA, 2004), ACM Press, pp. 115–122.

[KW05] KRÜGER J., WESTERMANN R.: Gpu simulation and rendering of volumetric effects for computer games and virtual environments. *Computer Graphics Forum 24*, 3 (2005).

[Lat04] LATTA L.: Massively parallel particle systems on the gpu. In *ShaderX3: Advanced Rendering with DirectX and OpenGL (Shaderx Series)*, Engel W., (Ed.). 2004, pp. 119–133.

[LF02] LAMORLETTE A., FOSTER N.: Structural modeling of flames for a production environment. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 729–735.

[LLW04] LIU Y., LIU X., WU E.: Real-time 3d fluid simulation on gpu with complex obstacles. In *Pacific Conference on Computer Graphics and Applications* (2004), pp. 247–256.

[Mat97] MATKOVIC K.: *Tone Mapping Techniques and Color Image Difference in Global Illumination*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1997.

[MK02] MELEK Z., KEYSER J.: *Interactive Simulation of Fire*. Tech. rep., Texas A&M University, 2002.

[NFJ02] NGUYEN D. Q., FEDKIW R., JENSEN H. W.: Physically based modeling and animation of fire. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 721–728.

[Ngu04] NGUYEN H.: Fire in the "vulcan" demo. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Fernando R., (Ed.). Addison-Wesley Professional, 2004, pp. 87–105.

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

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

[WLL04] WU E., LIU Y., LIU X.: An improved study of real-time fluid simulation on gpu. *Journal of Visualization and Computer Animation 15*, 3-4 (2004), 139–146.

[WLMK02] WEI X., LI W., MUELLER K., KAUFMAN A.: Simulating fire with texture splats. In *VIS '02: Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 227–235.