Realistic 2D Fire in Real-Time

Odd Erik Gundersen odderik@idi.ntnu.no

Samuel Rødal knuterro@idi.ntnu.no

Geir Storli geirstorli@yahoo.no

Department of Computer and Information Science
Norwegian University of Science and Technology

Abstract

We have reached our goal of implementing a computationally cheap algorithm for real-time rendering of realistic two-dimensional fire. The fire is simulated using computational fluid dynamics, vorticity confinement, and combustion modelling and visualized using black-body radiation. Small rotationally homogenous fires like torches and candles viewed in front are the best candidates when using this approach.

1 Introduction

Real-time graphics is an area of great interest mostly driven by the gaming industry. As the gaming industry produces new games like The Elder Scrolls IV: Oblivion™ and Half Life 2™ where a lot of attention is given to the physics engine, the users get more and more used to realistically-looking environments. Thus, it gets harder to impress the users for every new game title that is released. One area where lots of work needs to be done in order to convince the user visually is fire. Even in the highly acclaimed games mentioned above, the fire is not convincing.

It is the very turbulent nature of fire that makes it hard to produce a realistic imitation, especially in real-time. Both procedural and physically-based real-time methods have been proposed to generate non-repeating fires. Still, none have managed to produce a visually satisfying real-time fire. The introduction of general purpose programming on the graphics processing unit (GPU) has made it possible to do both a physically-based simulation and rendering of a simulated fire on the GPU. This movement of the simulation from the CPU to the GPU does not only lessen the CPU workload, it also speeds up the rendering process as the GPU is faster than the CPU at certain tasks.

Even though most of today's games and virtual environments are three-dimensional, we present an algorithm for visualizing two-dimensional fires. The reason for this is the computational demand of physically based simulations. In computer games, the computational resources need to be spent well in order to achieve both realistically-looking graphics as well as believable behaviour of non-player characters. Regardless of how much faster hardware gets, the users' demand for realistic environments in games keep growing. Thus, certain optimizations have to be made. In some cases, one can get away with only rendering a two-dimensional fire.

Our fire simulation is largely based on the approach in [1], which combines the stable fluid solver from [2] with a three-gas system. We use the vorticity confinement method from [3] to create a more turbulent flame. To visualize the fire, black-body radiation is utilized.

The rest of this paper is structured as follows. After briefly discussing some relevant related work, our approach for simulating and visualizing two-dimensional fire is presented. Then, the results are described and discussed. Finally, we conclude and give a prospective of possible future work.

2 Related work

Several real-time methods for simulating and visualizing fire have been proposed. The most prominent non-physically based method is presented in [4]. They use video-textured sprites for creating believable raging fires with smoke in real-time. In order to add variety to the flames, two flame animations are combined in various ways. Another non-physical approach is presented in [5]. They use volume rendering in combination with a set of textures to visualize animated amorphous materials such as fire, smoke, and dust. Dynamics and illusion of motion are created through cycling the textures in each voxel. In [6], a method that uses a photometric solid defining luminous intensities for a set of zentihal and azimuthal directions is presented. The intensities are stored in a 2D texture and by rotating this texture the fire is animated.

The Navier-Stokes equations are important in the physically-based approaches. [1] use the Navier-Stokes equations to control the motion of a three-gas system consisting of oxidizing air, fuel gases, and exhaust gases. The fuel gases, exhaust gases, and heat are simulated explicitly in separate fields and advected by the velocity field of the air. The fire is visualized using volume rendering. [7] also solves the Navier-Stokes equations using the stable fluid approach presented in [2], only implemented on the GPU. Fuel and exhaust gases are not part of the fluid simulation, and the fire is visualized using a particle system. [8] presents a method fully implemented on the GPU simulating fuel gas, exhaust gas, and heat using the Navier-Stokes equations in combination with vorticity confinement, an explicity modelled combustion process, a particle system, and black-body radiation.

3 Our approach

The core of the simulation is the interaction between the different fields. In this section, we first introduce the main components of our approach and then go into more detail about how the simulation is realized. We then present the method used for visualization and finally the complete algorithm is presented. The simulation is governed by a set of equations, and these equations are solved using Stam's stable fluid solver as described in [2].

3.1 Computational domain and boundary conditions

We use a grid data structure to represent the computational domain. The computational domain limits the area where a fire is simulated. There are two different kinds of cells; interior cells and boundary cells. Each cell contains a corresponding field value. The main fields we use in the simulation are a fuel gas field, an exhaust gas field, a temperature field, and a velocity field. We will refer to the fuel gas field, exhaust gas field, and temperature field collectively as the density fields. The density fields are scalar fields where a given field value specifies the amount of fuel gas, the amount of exhaust gas, or the temperature at that position. The velocity field is a vector field specifying the direction and speed air and gas moves in. When discretizing the fields into grids, the field values are defined in the centre of the grid cells and assumed to be uniform inside each cell. As for boundary conditions, the boundary cells are set to 0 in the density fields and closed boundaries with slip condition are used for the velocity field.

3.2 Velocity field

Equation 1 and equation 2 are the Navier-Stokes equations for incompressible flow with zero viscosity, also known as the Euler equations.

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

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

Equation 1 is a differential equation governing the velocity field **u**. The first term on the right-hand side of the equation is the self-advection of the velocity causing velocity to move along itself. The second term, $-\nabla p$, is the pressure gradient 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. Equation 2 is the non-divergence condition, which states that the velocity field should be mass conserving. The last term on the right hand side of equation 1 is the external force acting on the velocity field. The external force actually consists of several separate forces as shown in equation 3:

$$F = f_{vorticity} + f_{buoyancy} + f_{gravity} + f_{\exp ansion} \tag{3}$$

where $f_{vorticity}$ is the vorticity confinement force, $f_{buoyancy}$ is the buoyancy force due to heat, $f_{gravity}$ is the gravity force due to fuel and exhaust gases, and $f_{expansion}$ is the expansion force resulting from combustion. These forces are later explained in more detail in the following sections.

3.3 Vorticity confinement

To achieve real-time results, we have to use a rather coarse grid in the computational domain. Besides, 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 counter-balance this, we use the vorticity confinement method, also used by [3], 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, and the vorticity ω of the velocity field **u** is calculated using the following equation:

$$\omega = \nabla \times \mathbf{u} \tag{4}$$

Next, the normalized gradient N of ω, pointing from lower to higher concentrations of vorticity, is calculated using the following equation:

$$\mathbf{N} = \frac{\nabla|\omega|}{|\nabla|\omega||} \tag{5}$$

Finally, we calculate the vorticity force given by equation 6. The parameter ε controls the strength of the vorticity confinement and h is the distance between two grid cells.

$$f_{vorticity} = \varepsilon h(\mathbf{N} \times \omega) f_{weight} \tag{6}$$

The last term, f_{weight}, is used to control the vorticity separately in different areas. This allows us to have more vorticity outside the flame than inside, in order to approximate a turbulent wind field surrounding the flame causing the flame to flicker and behave more chaotically. More turbulence also helps to simulate parts of the flame detaching from the flame body before fading.

3.4 Gravity and buoyancy

Fuel gas and exhaust gas are pulled down due to gravity. The gravity force is given with the following equation:

$$f_{gravity} = f_g(g + a)\begin{pmatrix} 0 \\ -1 \end{pmatrix} \tag{7}$$

The constant f_g determines the strength of the gravity force while g and a are the amount of respectively fuel gas and exhaust gas. The buoyancy force causes hot air to rise and is given with the following equation:

$$f_{buoyancy} = f_b(T - T_{ambient})\begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{8}$$

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

3.5 Expansion

The blue core of a flame is the result of radicals in the chemical reaction zone where fuel is converted into exhaust products. In [3], the boundary of the blue reaction zone is tracked and then used to locate the positions of the fuel gas particles moving through the reaction front. Particles curve outward due to gas expansion as they move through the reaction front. This effect creates much of the turbulence that can be observed in flames. It also gives the flames their characteristic visual fullness.

Tracking the position of the blue core is computationally very demanding and therefore impractical for real-time simulation. We wish to approximate the effect of the blue core without keeping track of the position of the blue core. By adding a horizontal force to the velocity field, the gas expansion is approximated. We assume that the fire has one blue core and that the centre of the core is at position x_{center} in the simulation grid. The horizontal forces should be larger at the boundary of the blue core where the reaction actually takes place. Because we do not have any knowledge about the position of the blue core, we make the horizontal force proportional to the distance between x_{center} and x_{grid}, where x_{grid} is the x-position of the grid cell to be computed. The following equation describes the expansion force:

$$f_{expansion} = \begin{cases} f_e(x_{grid} - x_{center})\begin{pmatrix}1\\0\end{pmatrix} & if\, T > T_{threshold} \\ 0 & if\, T \leq T_{threshold} \end{cases} \tag{9}$$

where f_e is a scale factor. Positions on the left side of x_{center} will add a force to the left and positions on the right side will add a force to the right. The expansion force will only affect the velocity field if combustion occurs in a particular grid cell. Combustion only occurs if the temperature in a grid cell is higher than the threshold $T_{threshold}$.

3.6 Density fields

The density fields are three separate scalar fields specifying the amount of fuel gas, exhaust gas, and heat distributed throughout the computational domain. The evolutions of these three scalar fields are governed by the same equation:

$$\frac{\partial d}{\partial t} = -u \cdot \nabla d + \kappa_d \nabla^2 d - \alpha_d d + S_d + C_d \tag{10}$$

The parameter d is a scalar quantity that represents either the amount of fuel gas, exhaust gas, or temperature in a grid cell in the computational domain; denoted by g, a or T respectively. Equation 10 describes the evolution of a scalar field over time in the computational domain as the velocity field **u** affects the scalar field. We use a slightly modified form of the equations described in [1] and in [2].

The first term on the right-hand side in equation 10 governs the advection of the scalar quantity d by the velocity field **u**, while the second term governs the diffusion of the scalar quantity d. κ_d is the diffusion constant controlling the amount of diffusion associated with each of the density fields. Furthermore, the third term governs the dissipation of the scalar quantity d where α_d denotes the dissipation rate. The dissipation rate ensures fuel gas, exhaust gas, and temperature will decrease over time. S_d denotes a source term used for increasing the scalar quantity d. Only the fuel gas field and temperature field have sources used for injecting fuel and temperature while exhaust gas is produced solely in the combustion process. C_d is the combustion term that controls the effect of the combustion process on a specific density field.

3.7 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 that occurs inside a grid cell, we choose a similar approach to that described in [1].

The combustion inside a cell will only occur if the temperature in the cell is above a given threshold temperature $T_{threshold}$. We assume that there always will be enough oxygen to react with the fuel gas, which simplifies the combustion computation. Assuming enough oxygen should be a realistic assumption for the kind of free-burning fire we wish to simulate, and we end up with the following equation for the combustion parameter:

$$C = rbg \tag{11}$$

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. By using the combustion parameter *C,* we end up with the following equations describing the rate of change due to combustion of the fuel gas, exhaust gas, and temperature respectively:

$$C_g = \begin{cases} -\frac{C}{b} & \textit{if } \mathrm{T} > \mathrm{T}_{\mathrm{threshold}} \\ 0 & \textit{if } \mathrm{T} \leq \mathrm{T}_{\mathrm{threshold}} \end{cases} \tag{12}$$

$$C_a = \begin{cases} C\left(1 + \frac{1}{b}\right) & \textit{if } \mathrm{T} > \mathrm{T}_{\mathrm{threshold}} \\ 0 & \textit{if } \mathrm{T} \leq \mathrm{T}_{\mathrm{threshold}} \end{cases} \tag{13}$$

$$C_T = \begin{cases} T_0 C & \textit{if } \mathrm{T} > \mathrm{T}_{\mathrm{threshold}} \\ 0 & \textit{if } \mathrm{T} \leq \mathrm{T}_{\mathrm{threshold}} \end{cases} \tag{14}$$

The three terms C_g, C_a, and C_T are used when using equation 10 from section 3.6 to calculate the density fields. Because we assume there always will be enough oxygen to react with the fuel gas, the parameter b will only control how much oxygen is involved in the reaction. C_g will be used to decrease the amount of fuel gas, C_a to increase the amount of exhaust gas, and C_T to increase the amount of temperature, all due to combustion. Moreover, the parameter T_0 controls the amount of heat that is produced by the reaction.

3.8 Burning rate parameter

In [1], they state that the burning rate parameter r is the maximum percentage of the fuel gas that can be burned during one second and therefore should be set between 0 and 1. Our calculations show, nevertheless, that there is no upper limit to the parameter r. If we insert the term C_g from equation 12 into equation 10 ignoring all other terms, we get the following partial differential equation describing the rate of change of fuel gas due to combustion:

$$\frac{\partial g}{\partial t} = C_g = -\frac{C}{b} = -rg \tag{15}$$

The solution to the partial differential equation 15 is:

$$g(t) = ke^{-rt} \tag{16}$$

Equation 16 gives the amount of fuel gas g at time t, where k is a constant which can be calculated by specifying the initial condition g(0) = g0. We introduce a new parameter p as the maximum percentage of the fuel gas that can be burned in a second, and we can calculate p in the following way:

$$p = 1 - \frac{g(t+1)}{g(t)} = 1 - e^{-r} \tag{17}$$

We then get the following relationship between r and p:

$$r = -\ln(1-p) \tag{18}$$

This shows that there is no upper limit to the parameter r, and we use this approach rather than setting $0 \leq r \leq 1$, which would result in an artificial limit to the fuel consumption rate.

3.9 Visualizing the simulated result

The fields representing the different aspects of the fire can be combined and visualized in different ways. We adopt some aspects of the technique used in [3] because of their superior visual results. They treat fire as a participating medium with black-body radiation and render it using a stochastic ray marching algorithm. We use the black-body radiation emitted by the exhaust gases solely because this is what gives fire the characteristic yellow-orange color. Black-body radiation is only dependent on the amount of exhaust gas and the temperature of the exhaust gas making the values of the fuel gas field unnecessary in the rendering process. Because we do not explicitly simulate the chemical reactions, we cannot track the surface of the blue core in the simulation. Thus, we cannot visualize the blue core of a flame.

We use Planck's formula as shown in equation 19 in order to calculate the intensity radiated by the hot exhaust gas.

$$B_\lambda(T) = \frac{2\pi hc^2}{\lambda^5 \left(e^{\frac{hc}{\lambda kT}} - 1\right)} \tag{19}$$

By using the wavelengths of red, green, and blue light and the temperature of the gas, we get 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 [9]:

$$n = n_{\max}(1 - e^{\frac{-L}{L_{average}}}) \qquad (20)$$

where n is the resulting intensity, n_{max} is the maximum intensity of the limited range, L is the original intensity, and $L_{average}$ is a constant controlling the brightness of the resulting intensities. Higher values of $L_{average}$ give darker resulting intensities.

We use equation 20 with L set to B_{red}, B_{green}, and B_{blue} to get the resulting intensities n_{red}, n_{green} and n_{blue}. The resulting intensities are finally combined to create an RGB color value. By defining the minimum and maximum temperature T_{min} and T_{max} we expect to observe in the fire, the color spectrum of the fire can be precomputed with a specified number of different colors. When the color of a grid cell is to be computed, the amount of exhaust gas in the grid cell is multiplied with the temperature in the grid cell and a scale factor. This value is used to perform a lookup in the precomputed color spectrum giving the color value of the grid cell. This computation is performed for all grid cells in the simulation.

3.10 Complete Algorithm

The complete procedure for simulating and visualizing the fire is presented in algorithm listing 1. The velocity field and density fields are discretized into a simulation grid – the computational domain. Each step in the algorithm performs the same computation on each grid cell. The algorithm starts at time t and is finished at time *t* + *dt*. Thus, *dt* is the time step used for executing the complete algorithm on all cells in the computational domain.

A fire is started by injecting fuel and temperature into their respective fields, and these scalar fields will affect the velocity field, which in turn will spread fuel gas, exhaust gas, and temperature throughout the computational domain. After starting the fire, the vorticity forces are calculated in step 1 using equations 4 through 6, and in step 2 the gravity, buoyancy, and expansion forces are calculated using equations 7 through 9.

The forces describing the change of fuel gas, exhaust gas, and temperature due to combustion are calculated in step 3 using equations 11 through 14 respectively. The combustion forces are added to the density force field in addition to the dissipation terms from equation 10. In step 4, the fuel gas and temperature sources are added to the density force field.

The four subsequent steps correspond to the velocity fluid solver and modify the velocity field. These steps involve finding solutions to equations 1 and 2. First, in step 5 the velocity forces are added to the velocity field. These forces correspond to term F, introduced in equation 1 and further described in equation 3. The velocity field is projected in step 6 by ensuring that equation 2 holds making sure the field is mass conserving. In step 7, the velocity field is self-advected, corresponding to the first term in equation 1, and in step 8 it is again projected ensuring mass conservation.

```
Algorithm 1 A single simulation and visualization step
Fire specific velocity forces
1: Calculate vorticity forces
2: Calculate gravity, buoyancy, and expansion forces

Fire specific density forces
3: Calculate combustion forces and dissipation terms
4: Add fuel gas and temperature sources to density force field

Velocity fluid solver
5: Add velocity forces to velocity field
6: Project velocity field
7: Self-advect velocity field
8: Project velocity field

Density fluid solver
9: Add density forces to fuel gas, exhaust gas, and temperature fields
10: Diffuse fuel gas, exhaust gas, and temperature fields
11: Advect fuel gas, exhaust gas, and temperature fields

Visualization
12: Visualize result using the black-body radiation model.
```

The next three steps correspond to the density fluid solver. These steps involve finding solutions to the equation 10 for each of the density fields. In step 9, the density forces are added to the three fields causing the amount of fuel gas, exhaust, gas, and temperature in each grid cell to change. These forces correspond to the dissipation, source, and combustion terms in equation 10. In the next step, the three fields are diffused causing the amount of fuel gas, exhaust gas, and temperature to spread to neighboring cells. The diffusion corresponds to the second term in equation 10. In step 11, the three fields are advected by the velocity field. Advection corresponds to the first term in equation 10.

By using a black-body radiation model, the exhaust gas and temperature fields are used to create a visually realistic result. A fire color palette is pre-computed, and by combining the amount of exhaust gas and temperature in a grid cell, the resulting color for that cell can be fetched from the palette.

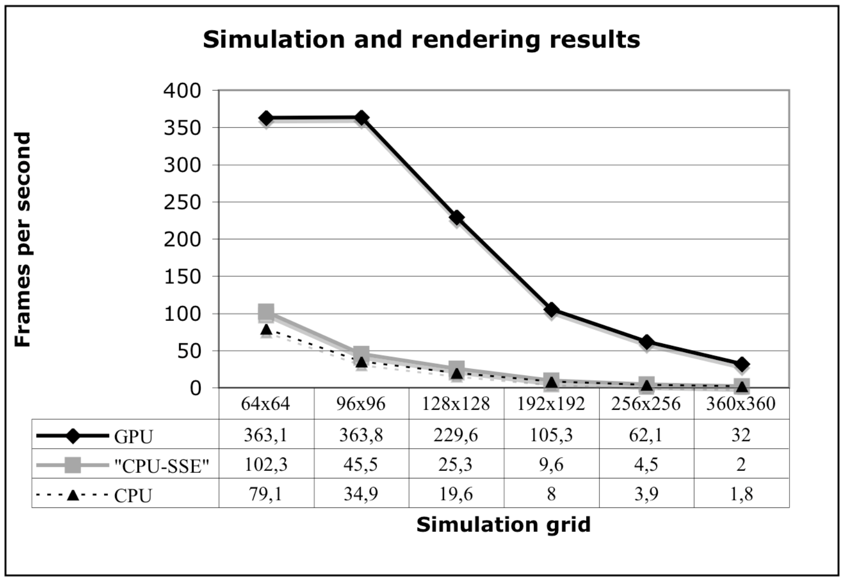
For the dissipation terms, source terms, and combustion terms explicit integration is used and the stable fluid solver from [2] is used as a fluid solver. Explicit integration may cause troubles when small time steps are used, but as our goal is real time rendering of the fire, this will not be a problem.

4.0 Results

In this section, we will present the achieved results. First, the performance of the algorithm is presented, and then the visual result is discussed.

4.1 Performance

We have implemented three versions of the proposed algorithm; one where all computations are fully done on the CPU, another one where all computations are executed on the CPU using streaming SIMD operations, and finally we have implemented a version where the complete algorithm is executed on the GPU. The tests were run on a Pentium 4 with a 3 GHz processor and 512 MB of system memory running Windows XP. The GPU used was a Gainward GeForce 7800GTX with 256 MB RAM. All tests were run twice for 20 seconds and the frame rates are the arithmetic mean of the two runs.



	64x64	96x96	128x128	192x192	256x256	360x360
GPU	363,1	363,8	229,6	105,3	62,1	32
"CPU-SSE"	102,3	45,5	25,3	9,6	4,5	2
CPU	79,1	34,9	19,6	8	3,9	1,8

Figure 1: Frame rates for the different implementations.

Figure 1 shows the frame rates achieved with the CPU, CPU with SSE enabled, and GPU implementations using various grid sizes for the computational domain. As can be seen from the results, the frame rate rapidly deteriorates when the simulation grid size increases. Still, the drop in frame rate is larger on the CPU than on the GPU. From simulation grid size of 64x64 to 96x96 the frame rate on the GPU is not changing. This is because the communication between the CPU and the GPU becomes the bottleneck, and the GPU is not fully utilized. With increased simulation grid size, however, the GPU is better utilized. The CPU implementation with SSE enabled achieves almost real-time frame rates with a simulation grid size of 128x128. However, the CPU is fully utilized then and cannot be used to perform other tasks without hurting the frame rate. The GPU implementation on the other hand achieves real-time frame rates with a simulation grid size of 360x360. Although a simulation grid size of 128x128 is used creating the visual results in the next section, a larger simulation grid size would create a more detailed simulation. By using a simulation grid size of only 128x128, however, the additional GPU computational power can be utilized to perform other graphic tasks instead.

4.2 Visual results

Figure 2 shows the result from the 2D fire simulation using the GPU implementation. A simulation grid size of 128x128, display scale of 4, and time-step of 0.04 was used, with default values for all other simulation parameters. We introduced a sleep time after each rendered frame before rendering the next frame. This was done to reduce the frame rate to 30 FPS, thus avoiding the stuttering of the fire because of the original high frame rate (192 FPS).

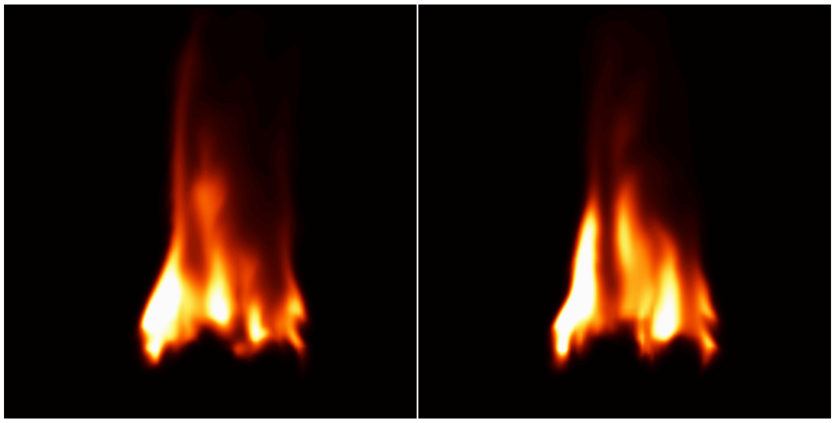


Figure 2: Two different frames from the same 2D fire simulation.

The black-body radiation-model approximates the colors realistically. The shape and movement of the fire is not easily seen in the figure, but these characteristics are clear in the animation. The flickering of real fire is present in that small parts detach from the flame body and move freely into the air. The unpredictable and chaotic movement of real flames is also present in that the flame body is constantly changing and swaying from side to side.

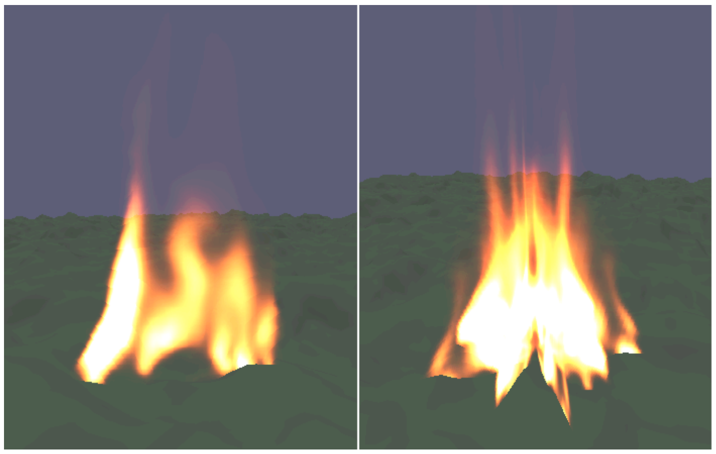


Figure 3: a) Single billboard b) Four rotated 2D slices.

We have experimented with two approaches for using the resulting 2D fire texture in a 3D environment as illustrated in figure 3. The first approach displays only a single billboard always facing the camera as illustrated in figure 3a. Even though the billboard is rotating according to the camera’s view direction, this rotation is not very noticeable unless the camera is positioned above or below the fire. The second approach, as shown in figure 3b, uses several rotated 2D billboards from the same simulation to create a flame. The symmetry due to using the same simulation for all the billboards can be removed by using different simulations for each billboard. Also this approach deteriorates when viewed from above, but not to the same degree as when just one billboard is used. Small rotationally homogenous fires like torches and candles viewed in front are the best candidates when using this approach.

5.0 Conclusions and Future Work

We have presented an algorithm for visualizing realistically-looking two-dimensional fires. The algorithm is physically-based and performs well as the simulation and rendering is fully performed on the GPU. However, there are limitations that could be addressed in future work. A fire is both affected by and affects the environment it is set in. As this is not the case with the presented algorithm, this would be a natural way of extending the work. Wind and obstacles should affect the fire and the fire should consume combustible materials and enlighten the environment. Another method for camouflaging the fire's flat nature is to map the fire onto a half-cylinder. For instance, this method would work better when used in a fireplace or for a torch hanging on a wall.

References

[1] Melek Z., Keyzer J.: Interactive simulation of fire. Tech. rep., Texas A&M University, 2002.
[2] Stam J.: Stable fluids. In *Siggraph '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. (New York, USA, 1999), ACM Press/Addison Wesley Publishing Co., pp. 121-128.
[3] Nguyen D. Q., Fedkiw R., Jensen H. W.: Physically based modelling and animation of fire. In *Siggraph '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, (New York, USA, 2002), ACM Press, pp. 721-728,
[4] 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.
[5] King S. A., Crawfis R. A. Reid W.: Fast animation of amorphous and gaseous phenomena. In *Volume Graphics '99*, pp. 333-346, 1999.
[6] Bridault-Louchez F., Leblond M., Rouselle F.: Enhanced illumination of reconstructed dynamic environments using a real-time flame model. In *Afigraph '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, and interaction in Africa* (New York, NY, USA, 2006), ACM Press.
[7] Krüger J., Westerman R.: GPU simulation and rendering of volumetric effects for computer games and virtual environments. In *Computer Graphics Forum 24*, 3, 2005.
[8] Rødal S., Storli G., Gundersen O. E.: Physically based simulation and visualization of fire in real-time using the GPU. In *Theory and practice of computer graphics 2006: Eurographics UK Chapter proceedings*. Lever L., McDerby M. (Eds.), (Middelsbrough, UK, 2006), Eurographics Association, pp. 13-20.
[9] Kresimir Matkovic: 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.