TN1008
Project Report

Viktor Nilsson, vikni067@student.liu.se
Axel Kinner, axeki412@student.liu.se
Johannes Deligiannis, johde901@student.liu.se
Johan Beck-Norén, johbe559@student.liu.se
Andreas Valter, andva287@student.liu.se

September 2, 2013

Abstract

This is a project report for the course TN1008 - Advanced Simulation and Visualization of Fluids in Computer Graphics held at Linköping University, 2013. In this report we present an implementation of a physically based simulation of fire. The implementation is largely based on the paper Physically Based Modeling and Animation of Fire by Nguyen et al. We simulate fire by using two sets of incompressible flow equations representing the unignited fuel and the hot gaseous products. These equations are coupled together using a ghost fluid method and the interface seperating the fluids are tracked using a level set implicit surface. We also track the temperature across the simulation to simulate external forces such as buoyancy, and to be able to render the simulation with a visually accurate result. The simulation was rendered using ray casting which was implemented in C++, as was the rest of the project.

Chapter 1

Introduction

1.1 What is fire?

By fully understanding the process behind fires one can easier simulate and render the phenomena. Fires are usually created by a chemical reaction between oxygen and fuel. As the amount of oxygen is reduced the flame becomes less clean resulting in appearing smoke.

The chemical reaction is most commonly producing carbon dioxide, heat and light. The actual flame that we can see is then the combined outcome of all three products. Depending on the temperature of the flame; it appears differently when observed by the human eye. This is due to the Black-body radiation emitted from the fuel, gas and soot particles.

1.2 Different ways to simulate fire

The most physically correct state of the art approach today for simulating fire is to track the burnt fuel and the unburnt fuel as two separate fluids. By tracking the temperature of the two; the simulation ignites the unburnt fuel when reaching a specific temperature and reduces the burnt fuel over the time. To create a lasting flame one has to add unburnt fuel over the simulation time. This assumes that the system is surrounded by an oxidizer (e.g air).

Other approaches includes creating the flames procedurally, without using any fluid system. This is of course much easier and faster to implement. There is also some fluid approaches that track one fluid; the flame surface (density) instead of the temperature.

1.3 General fluid simulation background

Fluid simulation has been around before we even started to use computers. The first mathematical approaches were presented around 1950. Incompressible and

free-surface fluids were first seen around 1995 [6]. Before this the fluids were computed as non physically-based and in 2D.

Since the Navier-Stokes equations are still yet to be solved many different approaches and methods for fluid simulation exists.

1.4 Previous implementations of fire

The most notable previous report, and the one this report follows closely is Physically Based Modeling and Animation of Fire [8]. It approaches the problem as described earlier with two seperated fluids and combines them with the help of a ghost fluid in between them. There exists many new and improved methods based on [8] such as Wrinkled Flames and Cellular Patterns [7], which introduces the characteristic surface a flame often have.

Chapter 2

Background

2.1 Physically Based Model

Visually, one can define two distinct components of fire or flames; the inner blue core and the hot gaseous products. In this report we follow the technique used in [8] for tracking the border between the blue core and the hot gaseous products with an implicit surface (level set). Points located inside of the surface are defined as gaseous fuel which has yet to be ignited, and points outside of the surface is defined as ignited hot gaseous products. The black-body radiation emitted by the ignited fuel (hot gaseous products) are what we typically see as the orange or yellow coloured component of fire. We also track the temperature in order to simulate external forces such as buoyancy, and to be able to render the simulation with visual accuracy. The gaseous fuel is injected at its ignition temperature, and as fuel crosses over our implicit surface (igniting) its temperature cools until the black-body radiation is indistinguishable from the surrounding air.

2.1.1 Blue Core

We separate the gaseous fuel from the ignited fuel by an implicit surface. This implicit surface moves at a velocity of the unreacted fuel velocity plus a flame speed S in the normal direction of the implicit surface. S dictates at what rate the fuel is burning (moving over the implicit surface), and will produce different kinds of flames for different values for S. A small value for S will result in a blue core with greater surface area, and vice versa for a larger value for S.

2.1.2 Hot Gaseous Products

The blackbody radiation emitted from the hot gaseous products is the part of the flame we often consider being yellow or orange. To be able to represent these colours when rendering the simulation we track the temperature across our grids. Another important aspect of the simulation is the expansion that takes place as the fuel passes over our implicit surface and ignites. A simplified explanation of what happens is an almost instantaneous expansion of the fuel as it ignites, causing a change in the fuel trajectory as it does so. We model this in the same fashion as in [8], by using a density ratio between the density for the fuel and the hot gaseous product respectively. Since we assume that mass and momentum are preserved, we use the following equations from [8] to couple the flow equations across the implicit surface.

$$\rho_h(V_h - D) = \rho_f(V_f - D), \tag{2.1}$$

$$\rho_h(V_h - D)^2 + p_h = \rho_f(V_f - D)^2 + p_f \tag{2.2}$$

In these equations V_h and V_f are the velocities in the normal direction for the hot gaseous products and the fuel respectively, $D = V_f + S$ is the implicit surface's speed in the normal direction, and p_h and p_f are the pressures for the hot gaseous products and the fuel. Note that this rapid expansion causes discontinuities in both the density and the velocities in the area of the blue core border (the implicit surface). We must therefore be careful when taking derivatives in that region, which brings us to the next section.

2.1.3 Two-phase Flow and Ghost Fluid Method

We model the fuel and the hot gaseous products by two separate sets of incompressible flow equations, namely Navier-Stokes equations. The problem with discontinuities along the blue core surface described in the previous section can be solved by extrapolating values for each section (fuel and hot gaseous products) by the Ghost Fluid Method [5]. If we for example are taking the derivative for the unignited fuel in the vicinity of the blue core, we will extrapolate fuel velocity values for cells adjacent to, but outside of, the blue core. These extrapolated values can then be used when taking derivatives and we do not have to worry about the discontinuities described earlier. Vice versa for ignited fuel in the vicinity of the blue core.

2.2 Level Set Method

To be able to track the interface between the blue core and the hot gaseous products a Level set method is used. The Level set method represents the interface by an implicit function ϕ defined as equation 2.3 in a 3D-space, where $h = 0$.

$$L = \left\{ \vec{x} \in \Re^3 : \phi(\vec{x}) = h \right\} \tag{2.3a}$$

$$L_{inside} = \left\{ \vec{x} \in \Re^3 : \phi(\vec{x}) \geq h \right\} \tag{2.3b}$$

$$L_{outside} = \left\{ \vec{x} \in \Re^3 : \phi(\vec{x}) < h \right\} \tag{2.3c}$$

This representation is useful for tracking and calculating topology properties of a interface. The representation makes it easy to find out where in the fluid a sample point is taken since it is only a matter of testing the sign. Since ϕ is a implicit function, the normal can be calculated as in equation 2.4.

$$\vec{n} = \frac{\nabla \phi}{|\nabla \phi|} \tag{2.4}$$

Moving an interface along a vector field $\vec{V}$, the velocity field for instance, is a matter of solving the hyperbolic PDE in equation 2.5.

$$\frac{\partial \phi}{\partial t} = -\vec{V} \cdot \nabla \phi \tag{2.5}$$

ϕ is also a signed distance function which means that the value of ϕ gives the distance to the closest point $\vec{p}$ of the interface, eq. 2.6. The direction of this point is also parallel with the surface normal, and since the length of the gradient is 1 [4], this point $\vec{p}$ can be calculated for a given point $\vec{x}$ by equation 2.8.

$$distance_L(\vec{x}) = \min_{\vec{p} \in L} \|\vec{x} - \vec{p}\| \tag{2.6a}$$

$$\phi(\vec{x}) = distance_L(\vec{x}) : \vec{x} \text{ is inside} \tag{2.6b}$$

$$\phi(\vec{x}) = -distance_L(\vec{x}) : \vec{x} \text{ is outside} \tag{2.6c}$$

$$\vec{n} = \nabla \phi \tag{2.7}$$

$$\vec{p} = \vec{x} - \vec{n} \cdot \nabla \phi(\vec{x}), \vec{p} \in L \tag{2.8}$$

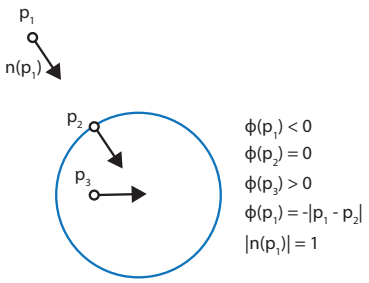


Figure 2.1: Illustration of the level set in 2D, and its properties.

2.2.1 Reinitialize signed distance function

Since the level set only is a sampled signed distance function in practice, a reinitialize operation is used to make sure that the level set is always close to a signed distance function. This is done by making sure that the length of the gradient is kept close to 1. By always having the grid described in this way, it is certain that whenever moving in the direction of the gradient towards the surface with the distance d, the closest point on the surface will not change and the distance to the surface is now d less then it was before.

2.3 Navier-Stokes Equation

The incompressible Navier-Stokes equations in equation 2.9 offer a way to describe the motion of fluids. This equation is the base of most simulations of fluids eg. water, smoke, and fire. The equation consists of a set of partial differential equations that describe how the fluid should behave through out the simulation.

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u}, \\ \nabla \cdot \vec{u} = 0 \end{aligned} \tag{2.9}$$

In this equation, $\vec{u}$ represents the velocity field of the fluid. ρ is the density of the fluid. p stands for the pressure and is the force per unit area that the fluid is affecting its surroundings with. The letter g represents external or body forces that acts on the whole fluid. This term handles things like gravity and buoyancy. The term $\nu \nabla \cdot \nabla \vec{u}$ represents the fluild viscosity. This term describes how much the fluid resists deforming while it is moving. Things like honey has a high viscosity, while fluids like water has low viscosity. For fluids with low viscosity, this term is usually not modeled because it has such small impact on the simulation.

2.3.1 Self-Advection

In order to completely solve the Navier-Stokes equations, an equation on the following from must first be worked out:

$$\frac{\partial q}{\partial t} + \nabla q \cdot \vec{u} = 0 \tag{2.10}$$

Where $q = q(t, \vec{x})$ being some quantity, moving with the velocity field of the fluid, $\vec{u}$, at a certain time t and point $\vec{x}$ in space. L.h.s in eq. 2.10 is also what defines the material derivative, denoted using capital D:

$$\frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \nabla q \cdot \vec{u} \tag{2.11}$$

The material derivative describes the change of a quantity in a velocity field. An equation using the material derivative is called an advection equation and since the material being advected is the velocity field itself, $q = \vec{u}(t, \vec{x})$ eq. 2.12 is often referred to as Self Advection.

$$\frac{\partial u}{\partial t} + \nabla u \cdot \vec{u} = 0 \tag{2.12}$$

From the Eulerian perspective, which is mainly used in this report, we are observing fixed points in space and tracing the change of velocity at these points, while the Lagrangian point of view is focusing on a fixed set of particles and tracing their trajectory (position). Equation 2.12 states that the velocity is only changing at a location due to movement/replacement of quantity, or more easily explained in Lagrangian terms, the particles are not changing their velocity to any external force, just moving with the flow. The Lagrangian reasoning is fundamental to a commonly used method to find a numerical solution to equation 2.12 as described in more detail in section 3.2.1.

2.3.2 External Forces

When fire is simulated as it would appear on earth, it is affected by the environment that it is in. The characteristics of a lit candle is caused by gravity, combined with boyancy that makes the flame rise and flicker. As the air around the flame increases in temperature the hot air rises, causing an upward motion to the flame. If a fire is lit in space with no gravity acting on it, it would take the shape of a sphere.

2.3.3 Projection

The last step in solving the incompressible Navier-Stokes equation is the pressure step, which is also sometimes called the projection step, by its linear algebraic properties of a projection. This step enforces that the velocity field is divergence free, and therefore incompressible, by calculating a pressure field p which satisfies this property, eq. 2.9.

This can be done by using the Helmholtz-Hodge decomposition theorem which states that a vector field $\vec{v}$ can be expressed as a sum of two vector fields, one curl free $\vec{v_{cf}}$ and one divergence free $\vec{v_{df}}$, eq. 2.13a. If $\vec{v}$ is then set as $\vec{v_{ext}}$ (the velocity field given by the external forces step) and $\vec{v_{df}}$ as the requested velocity field $\vec{v_{req}}$ and then set $\vec{v_{cf}}$ as $\Delta t \frac{\nabla p}{\rho}$, it is possible to reorder eq. 2.13a to eq. 2.13b which looks like a euler integration.

$$\vec{v} = \vec{v_{cf}} + \vec{v_{df}} \tag{2.13a}$$

$$\vec{v_{ext}} = \Delta t \frac{\nabla p}{\rho} + \vec{v_{req}} \Leftrightarrow \vec{v_{req}} = \vec{v_{ext}} - \Delta t \frac{\nabla p}{\rho} \tag{2.13b}$$

It is still two unkowns in eq. 2.13b which is one to many to be able to solve this equation. But since $\vec{v_{req}}$ is divergence free it is possible to rule it out by apply the divergence operator ∇ on both sides in eq. 2.13b. This results in a possion equation 2.14, which is possible to solve.

$$\nabla \vec{v_{ext}} = \Delta t \frac{\Delta p}{\rho} \tag{2.14}$$

2.3.4 Boundary Conditions

When solving the Navier-Stokes equation, there are two different types of boundary conditions that need to be enforced in order for the fluid to interact with walls or other solid objects. These conditions can be seen as constraints that rule out unwanted solutions for the PDE of the projection step. The first type of boundary condition is the *Dirichlet boundary condition*. It simply states that any velocity vector component that points into a grid cell marked as solid is set to have a velocity of zero for that component.

$$V \cdot n = 0 \tag{2.15}$$

In equation 2.15 V is the velocity vector, and n is the boundary surface normal. This boundary condition is enforced for the velocity field before and after the projection step. This results in that no velocity vectors will point into grid cells marked as solids.

The second boundary condition is the *Neumann boundary condition*, eq. 2.16, and it ensures that there is no change of flow between fluid cells and cells marked as solid. The condition is enforced when building the linear equation system during the projection step. The connection between a fluid cell and a neighbouring solid cell is removed by entering a zero-value in the correct location in the poisson matrix, thus ensuring that no exchange of fluid or change of flow will occur between the cells.

$$\frac{\partial V}{\partial n} = 0 \tag{2.16}$$

2.4 MAC Grid

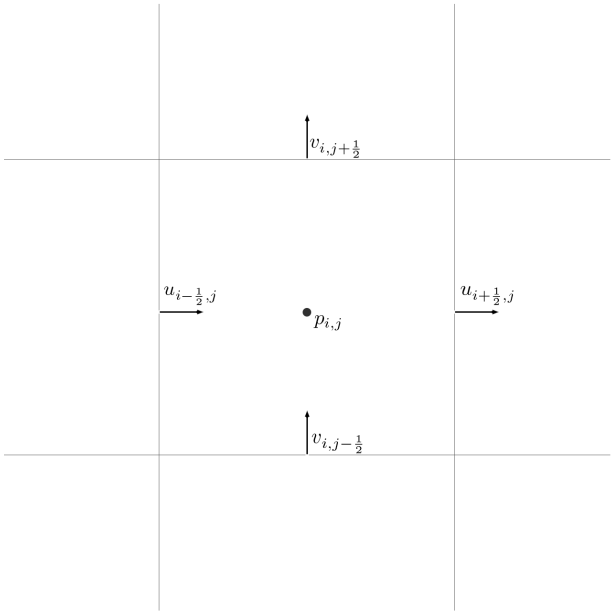


Figure 2.2: 2D MAC grid, p is pressure, u and v are velocity components. Half indices are used for illustrative purposes, an implementation should use whole indices.

In order to discretize the navier-stokes equations spatially, a technique called MAC (Marker And Cell) is often used. Space is quantified in equally sized cells, sampled quantities such as velocity components and pressure are spread out at different locations in the grid, see fig. 2.4. The pressure is stored in the center of each cell and the velocity is split componentwise and placed at cell faces orthogonal to its direction. The placement of velocity vectors and pressure might seem a bit odd but makes the central difference accurate for the pressure gradient and velocity divergence [4], it's easy to find out how much is flowing in or out of a cell which is essential when doing the pressure update.

A less appealing consequence on the other hand is that a velocity vector can not be accessed directly, it requires interpolation each time since the components are separated. Additional properties might be stored in the grid depending on the type of fluid being simulated, in the case of a fire simulation where buoyancy is a distinguishing feature that can be mimicked by evaluating the local temperature difference, temperature can be stored in the cell center. To track the interface between the blue-core and hot gaseous medium the level-set method is used. The ghost fluid method described in more detail in 3.1.1 requires a duplicate of each velocity component to account for discontinuous

density at the fuel interface.

2.5 Rendering

We are using a ray casting method for rendering our fire. The ray caster collects temperatures over the scene as samples and transform them into a color for each pixel using black-body radiation. The rendering part of the project is a separate work and is included in Appendix A for further reading.

Chapter 3

Method

3.1 MAC Grid

The grid is stored as a 1D array of size N where $N_x * N_y * N_z$ is the number of cells in dimension x, y and z. To access a grid index (i, j, k) a transformation from 3D to a 1D array index is made using the following formula:

$$f_{1D}(i, j, k) = i + jN_x + kN_xN_y \tag{3.1}$$

To avoid the half indices introduced in fig. 2.4, the following convention is used:

$$\begin{aligned}
u_{i-\frac{1}{2},j,k} &= u[f_{1D}(i, j, k)] \\
u_{i+\frac{1}{2},j,k} &= u[f_{1D}(i+1, j, k)] \\
v_{i,j-\frac{1}{2},k} &= v[f_{1D}(i, j, k)] \\
v_{i,j+\frac{1}{2},k} &= v[f_{1D}(i, j+1, k)] \\
w_{i,j,k-\frac{1}{2}} &= w[f_{1D}(i, j, k)] \\
w_{i,j,k+\frac{1}{2}} &= w[f_{1D}(i, j, k+1)]
\end{aligned}$$

Note that u,v and w represent different dimensions and could have different lengths N, more specifically:

$$\begin{aligned}
N_u &= (N_x + 1) * N_y * N_z \\
N_v &= N_x * (N_y + 1) * N_z \\
N_w &= N_x * N_y * (N_z + 1)
\end{aligned}$$

3.1.1 Two Phase Flow

The ghost fluid method requires extra caution when velocities at arbitrary points are evaluated. If the point is close to the interface chances are at least one velocity value is on the other side of the interface when interpolating. The balance equations in section (2.1.2), are used to enforce mass conservation. In practice, a second velocity field $\vec{u}^G$ ("ghost values") is used in parallel to the original velocity field $\vec{u}$. The ghost values are found using the adjusted velocity values described by following equation

$$\Delta V = (\frac{\rho_{fuel}}{\rho_{burnt}} - 1)S \tag{3.2}$$

$$\tag{3.3}$$

where ρ is the density of the fluid, the difference in density between burnt and fuel is what causes the reaction at the flame front. S is the speed in which the fuel is burning, a typical value is about 0.5 m/s. In general $\hat{n}$ is the normal pointing from the fluid region to the burnt region, in this case it's the normal of the level-set ϕ.

$$\vec{u}^G(\vec{x}) = \begin{cases} \vec{u}(\vec{x}) - \Delta V \hat{n}, & \text{if } \phi(\vec{x}) \geq 0 \\ \vec{u}(\vec{x}) + \Delta V \hat{n}, & \text{otherwise} \end{cases} \tag{3.4}$$

In practice this can be done once at the beginning of the simulation by storing the ghost values in a second grid or on the fly during simulation to save space. No matter what implementation is used it boils down to a simple check of the level set sign whether the point being used for interpolation is in the same kind of medium (burnt or fuel) and picking the value from same grid if the medium is the same, or choosing the ghost value otherwise.

3.2 Navier-Stokes

3.2.1 Self-Advection

Where more renowned methods of solving differential equations are failing to find a numerical solution to eq. (2.12) (such as forward Euler for the time derivative and central difference for the spatial derivative[1]), a method called semi-Lagrangian method is often used to ensure unconditional stability. As the name implies, it is motivated by taking a Lagrangian viewpoint (which treats the fluid as a set of finite particles) in order to aid the Eulerian grid based approach. Consider a particle at $\vec{x}$, with velocity $\vec{u}(\vec{x}) = \frac{d\vec{x}}{dt}$. If the particle started from grid location $\vec{x}_G$ and ended up at $\vec{x}_P$ after time step Δt, assuming constant velocity we have that:

$$\vec{u}(t, \vec{x}_G) = \vec{u}(t + \Delta t, \vec{x}_P) \tag{3.5}$$

[1] Which is unconditionally unstable for any Δt [4, p. 28]

The same argument can be made backwards: there exists a particle at $\vec{x}_{P_1}$ that will end up at grid location $\vec{x}_{G_1}$, finding $\vec{x}_{P_1}$ is done simply by tracing the current velocity field backwards in time and evaluating the velocity field at that point. Using simple Euler integration gives:

$$\vec{x_{P_1}} = \vec{x_{G_1}} - \Delta t \vec{u}(\vec{x_{G_1}}) \tag{3.6}$$

$$\vec{u}(t + \Delta t, \vec{x}_{G_1}) = \vec{u}(t, \vec{x}_{P_1}) \tag{3.7}$$

$\vec{x_{P_1}}$ is not very likely to be directly at a known grid point, so in order to find $\vec{u}(\vec{x_{P_1}})$ some kind of interpolation has to be used. Trilinear interpolation is prone to smooth the velocity field and better alternatives exists, such as Catmull-Rom interpolation, which reduces the dissipation (smoothing) and boosts the accuracy to second order (trilinear has first order accuracy). Despite the shortcomings of trilinear interpolation it showed to be adequate for convincing visual results.

Practical example in 2D

Below is a simplified 2D illustration to further illustrate the advection update of a velocity vector (e.q (3.5)-(3.7)) as well as a quick look at the bilinear-interpolation of the face-vectors. To extrapolate this to the third dimension trilinear interpolation has to be used in step 1 (fig. 3.1) instead of bilinear interpolation.

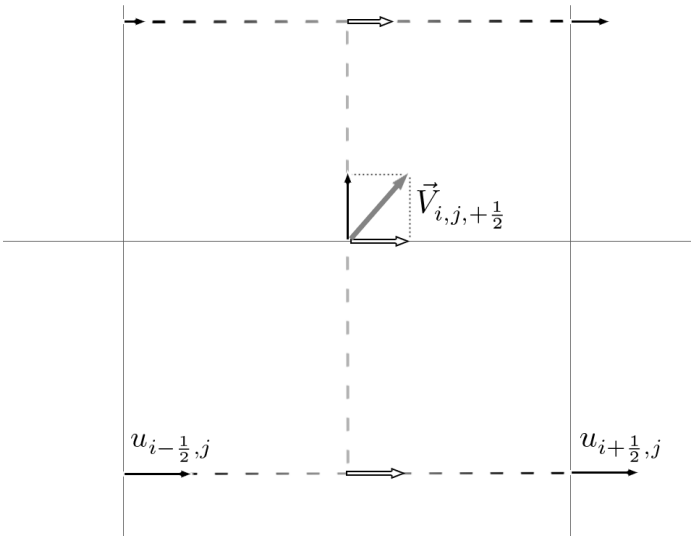


Figure 3.1: To find $\vec{V}_{i,j+\frac{1}{2}}$ linear interpolation is used on the v components. Black arrows represent known scalar values and white arrows represent interpolated values. Keep in mind that they are implicitly known to be vectors, not stored as such. The gray vector in the center $\vec{V}_{ij+\frac{1}{2}}$ is the vector found by combining the interpolated u-component and the explicitly known v component.

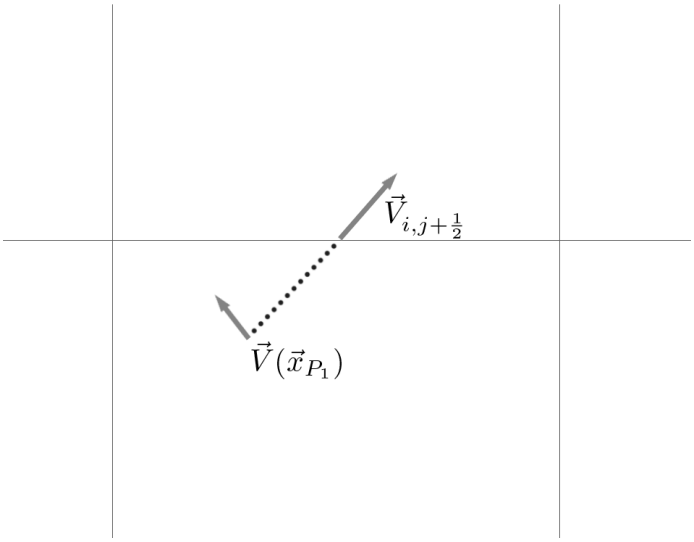


Figure 3.2: Once the location $\vec{x}_{P_1}$ is found by tracing backwards from $\vec{x}_{i,j+\frac{1}{2}}$ ($\vec{x}_{P_1} = \vec{x}_{i,j+\frac{1}{2}} - \Delta x \vec{V}_{i,j+\frac{1}{2}}$) as described in section 3.2.1. The vector $\vec{V}(\vec{x}_{P_1})$ is acquired by interpolating the nearby u and v components as in fig 3.1

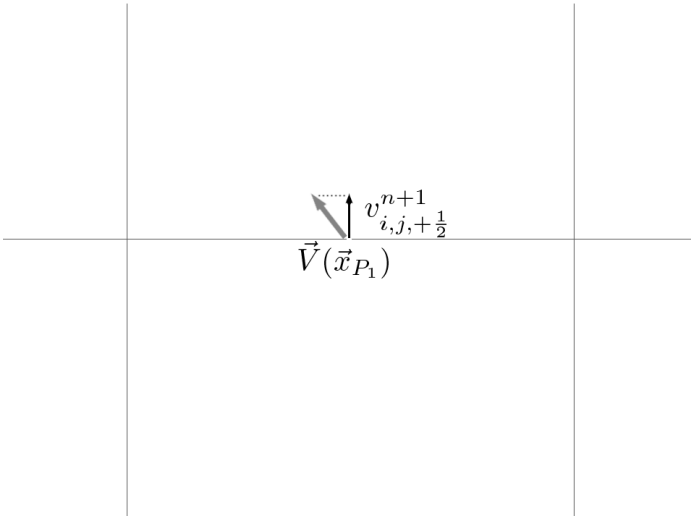


Figure 3.3: When $\vec{V}(\vec{x}_{P_1})$ is found the next time step can be updated as: $v^{n+1}_{i,j+\frac{1}{2}} = \vec{V}(\vec{x}_{P_1})_v$

3.2.2 External Forces

External forces are handled in a quite straight forward way. For each point in the grid all forces are calculated and added together. After that we calculate the new velocity by integrating the force. The gravitational force is applied uniformly over all grid points that has fluid in them. Buoyancy is described by a simple model that is directly connected to the temperature in each grid point. We define it as equation 2.9. The vector z is a unit vector pointing upward.

$$f_{buoy} = \alpha(T - T_{air})z \tag{3.8}$$

α is a positive constant, T_{air} is the ambient temperature of the room and T is the temperature.

3.2.3 Vorticity Confinement

Since we are using grids with finite spacing for our simulation, numerical dissipation will cause non-physical damping of some of the fine detail energy in the simulation such as vortices and small scale turbulence. We can add this missing energy back into our simulation by increasing the speed of existing vortices in the fluid. This method is called vorticity confinement and is implemented as in [8]. First we calculate the curl ω of the original vector field.

$$\omega = \nabla \times \vec{V} \tag{3.9}$$

We define normalized vorticity vectors $\vec{N}$ that will point from areas with low vorticity towards areas with high vorticity.

$$\vec{N} = \frac{\nabla|\omega|}{|\nabla|\omega||} \tag{3.10}$$

The resulting vorticity confinement force $\vec{f}_{vort}$ to be added to the velocity fields can then be calculated using these vorticity location vectors.

$$\vec{f}_{vort} = \epsilon\Delta x(\vec{N} \times \omega)) \tag{3.11}$$

In equation 3.11 ϵ is some arbitrary scalar to control the magnitude of the vorticity force being added. The dependency on Δx ensures that as the grid resolution is refined and Δx grows smaller the vorticity confinement force added decreases as well, resulting in the physically correct result being obtained. The resulting force $\vec{f}_{vort}$ is added to the velocity fields as an external force.

Since the velocity vector in eq. 3.10 is stored component-wise in a MAC grid, we need to interpolate a velocity value for the center of the cell for which we are calculating the vorticity confinement force to be able to calculate the cross product. This is done by a central difference to obtain the derivatives needed. For example, $\frac{\partial V_z}{\partial y}$ is discretized as $\frac{V_z^{j+1} - V_z^{j-1}}{2\Delta x}$ in the implementation.

3.2.4 Projection

The projection step is a linear operation to update $\vec{u}$ according to the pressure, p. This is done to maintain incompressibility as seen in the Navier-stokes equations, equation 3.12, we have to for each timestep compensate for the pressure to make $\vec{u}$ divergence-free.

$$\nabla \cdot \vec{u} = 0 \tag{3.12}$$

To retrieve the pressure p we have to solve the equation (3.13) [4].

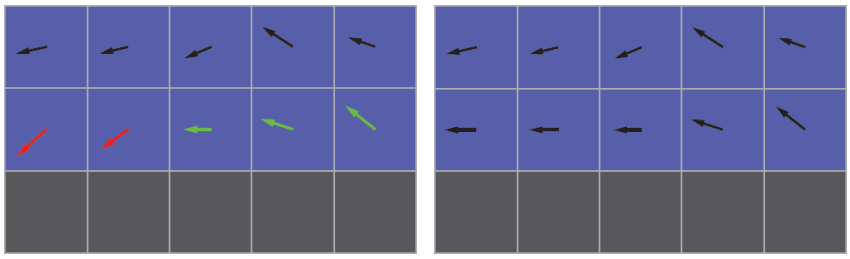
$$Ap = b \tag{3.13}$$

Where A is the coefficient matrix, each row corresponds to one cell in the fluid. So for large grids A will be extremly storage dependent. b is a vector with all the negative divergences for each cell of the fluid.

After A and b are calculated and set they are inserted into a PCG-solver (Preconditioned Conjugate Gradient), to retrieve the pressure vector p. As mentioned earlier, when the new pressure-gradient is retrieved we can for incompressibility calculate the new $\vec{u}$ which will be needed for the next time step.

3.2.5 Boundary Conditions

The *Dirichlet* boundary condition is enforced before and directly after the projection step. Our implementation loops over the grid containing information on solid cells. If for example a solid is present at grid index *(i,j,k)* defined at the center of the grid cell, all velocities in the staggered MAC grids for the fuel and hot gaseous products sharing a cell wall with *(i,j,k)* are set to zero, projecting the velocity vector onto the tangent plane of the cell surface. This results in that no velocity vectors point into cells that are solid, making sure that no fluid will flow into solid objects.



(a) before (b) after

Figure 3.4: Velocity vectors before and after enforcing the Dirichlet boundary condition. Image courtesy of *TNM079, LiU.*

The *Neumann* boundary condition is enforced when building the Poisson matrix during the projection step. If we for a fluid cell find a neighbouring solid cell, that solid cell's entry in the diagonal matrix is set to zero. This ensures that

no exchange of fluid will take place between the fluid cell and the cell marked as solid.

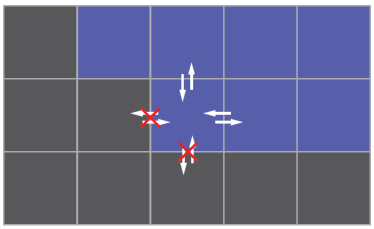


Figure 3.5: The Neumann boundary condition ensures no fluid exchange between solid- and fuel cells. Image courtesy of *TNM079, LiU*.

3.3 Advecting Level Set

The blue core does not only move with the velocity field $\vec{u_f}$, it is also burning with the speed S in the normal direction. This means that the level set which defines the blue core is advected by the velocity field $\vec{w} = \vec{u_f} + S\vec{n}$, where $\vec{n}$ is calculated using a central difference, eq. 3.14.

$$\frac{\partial \phi}{\partial x} \approx \phi_x^{\pm} = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{\Delta x} \tag{3.14}$$

To ensure stability in equation 2.5 [9], the finite difference is calculated using a upwind scheme, eq. 3.15. eq. 2.5 must also satisfy the time step constraint in eq. 3.16.

$$\frac{\partial \phi}{\partial x} \approx \begin{cases} \phi_x^+ = \frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{\Delta x} & , \vec{w_x} < 0 \\ \phi_x^- = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{\Delta x} & , \vec{w_x} > 0 \end{cases} \tag{3.15}$$

$$\Delta t < min \left\{ \frac{\Delta x}{\vec{w_x}}, \frac{\Delta y}{\vec{w_y}}, \frac{\Delta z}{\vec{w_z}} \right\} \tag{3.16}$$

$\vec{n}$ is not assured to be a unit vector since it is a numerical approximation of a signed distance function, $\vec{n}$ is therefore normalized before usage. $\vec{n}$ could even be evaluated as a null vector, in these cases a constant unit vector $(0, 1, 0)$ which is pointing upwards, is used.

3.3.1 Extrapolation

When accessing values outside the grid, extrapolation is needed to find out the value of the signed distance function, e.g. when calculating the finite difference. Some easy ways to do this is by using a constant value or by finding the closest real value to it. More sophisticated methods is to make the value fulfil the

properties of a signed distance function as described in [4]. The method below is a more simple way to mimic a signed distance function.

Find the closest real position p_r to the position p_e. The extrapolated value ϕ_e is then calculated using $\phi(\vec{p_r})$ subtracted with the distance between p_r and p_e, eq. 3.17. This means that the value is further away from the interface.

$$\phi_e(\vec{p_e}) = \phi(\vec{p_r}) - |p_r - p_e| \tag{3.17}$$

This solution would be true if the extrapolated value were in the negative normal direction from the real value.

Chapter 4

Results

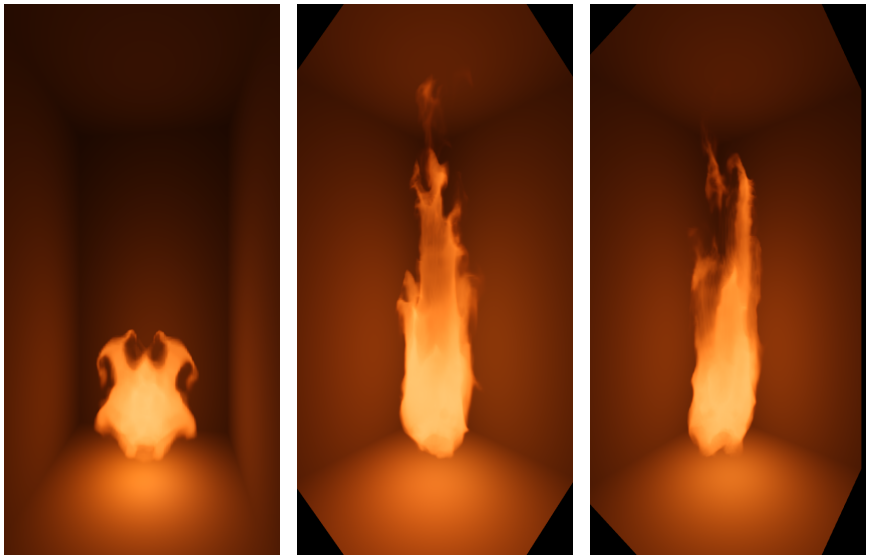
4.1 Results

Figure	$\vec{u}_{fuel}/\vec{u}_{burnt}$ grid	T grid	T_{ign}/T_{max}	S	ϵ_{burnt}	Simulation/Rendering time (s)
4.1	30x60x30	90x180x90	2200/3000	0.1	60	5/64
4.2	15x30x15	90x180x90	2200/3000	0.1	60	3/64
4.3	60x120x60	90x180x90	2200/3000	0.1	60	30/64
4.4	30x60x30	90x180x90	1500/1500	0.1	60	5/64
4.5	30x60x30	90x180x90	2200/3000	0.25	60	5/64
4.6	30x60x30	90x180x90	2200/3000	0.025	60	5/64
4.7	30x60x30	90x180x90	2200/3000	0.1	100	5/64

Table 4.1: Settings and performance for the figures

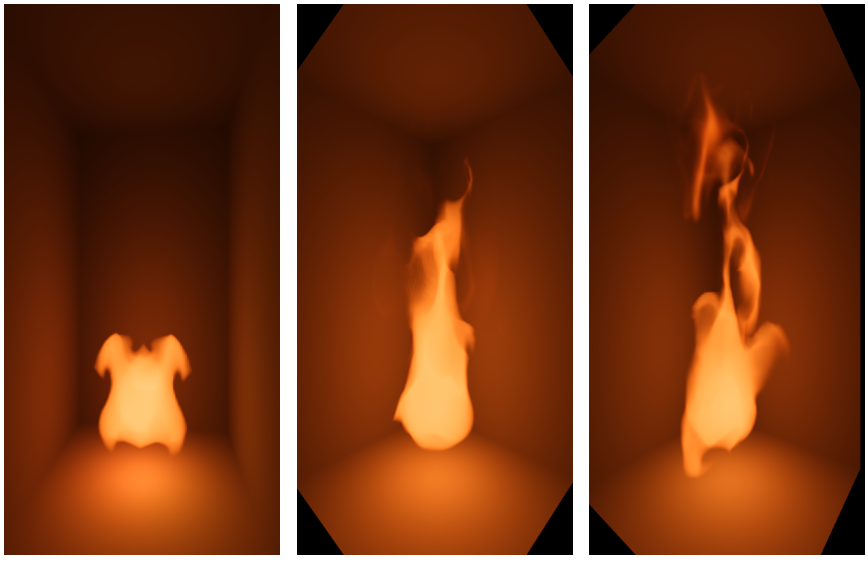
Common settings for all figures are $\alpha = 0.15$, $\epsilon_{fuel} = 16$, $\rho_{fuel} = 1.0$, $\rho_{burnt} = 0.01$ and $T_{loss} = 3000$. All figures have been rendered on a computer using a Intel 3770K CPU.

The fire in the figures shown does not consider the walls as boundaries in this simulation and therefore moves through them. The fuel is injected as a sphere in each frame for all but one of the figures.



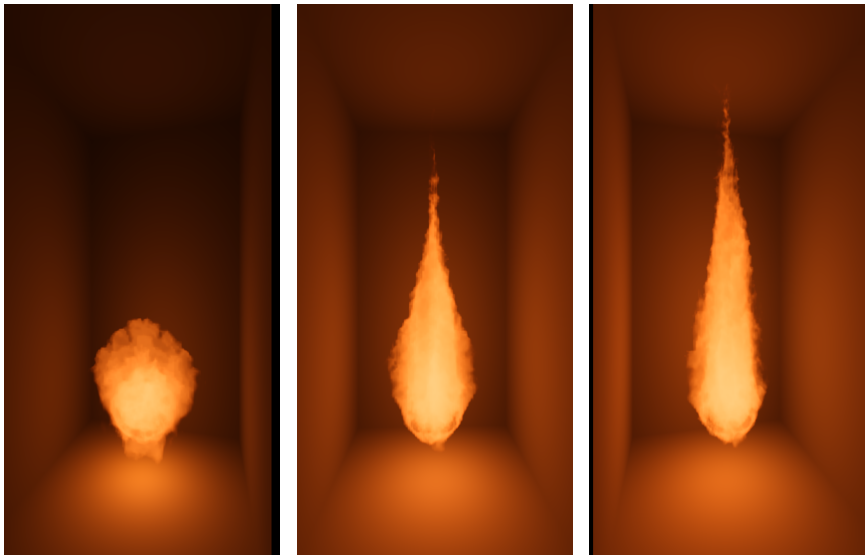
(a) frame 9 (b) frame 50 (c) frame 144

Figure 4.1: The figure shows the result of the settings which are used as reference settings for the rest of the figures.



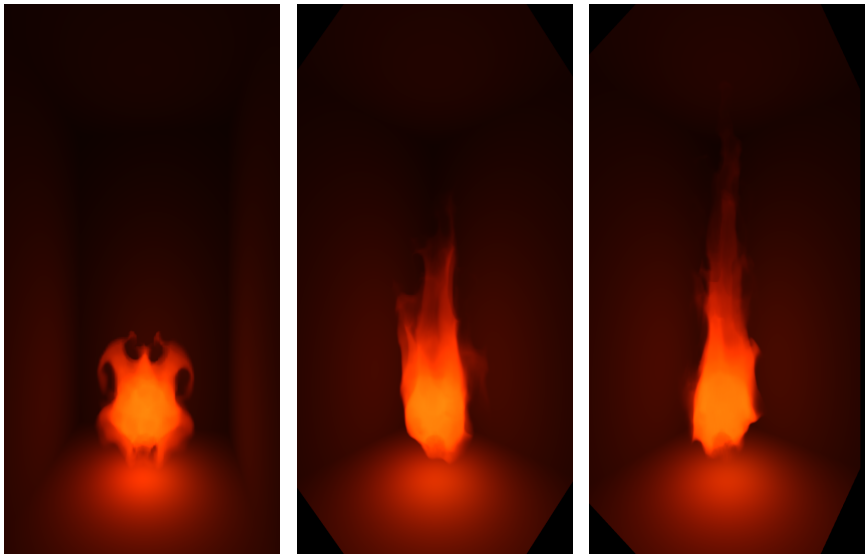
(a) frame 9 (b) frame 50 (c) frame 144

Figure 4.2: The velocity grid has half the reference resolution, i.e the simulation has half the resolution.



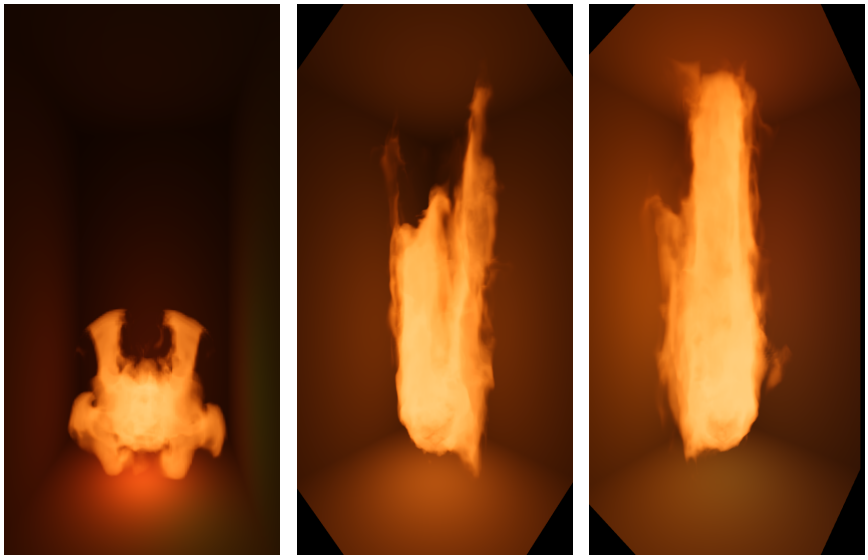
(a) frame 9 (b) frame 50 (c) frame 144

Figure 4.3: The velocity grid has twice the reference resolution.



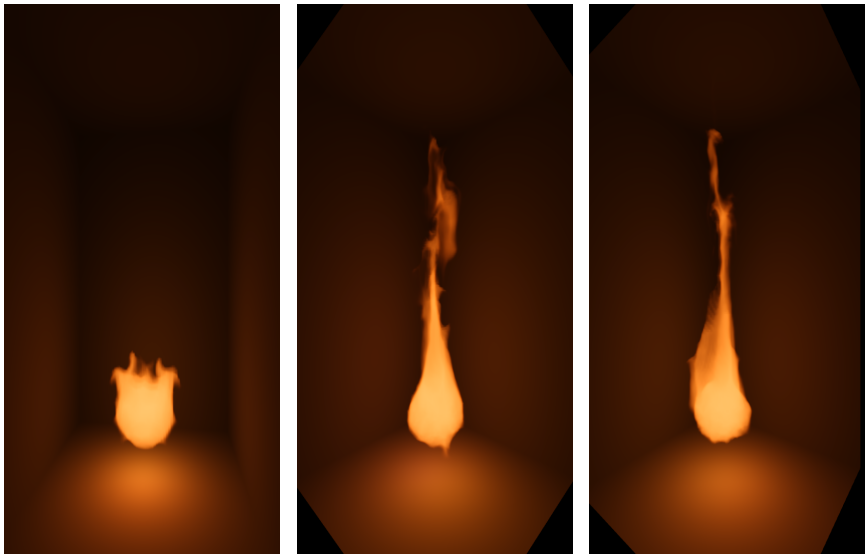
(a) frame 9 (b) frame 50 (c) frame 144

Figure 4.4: Using a lower ignition- and max temperature than the reference. The chromatic adaptation constant is set to 1 instead of 100.



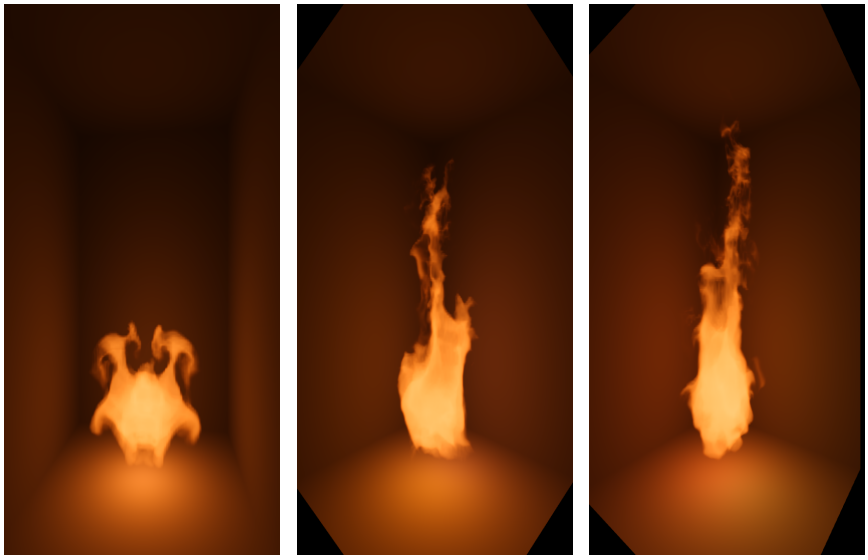
(a) frame 9 (b) frame 50 (c) frame 144

Figure 4.5: Using a higher flame speed S than the reference.



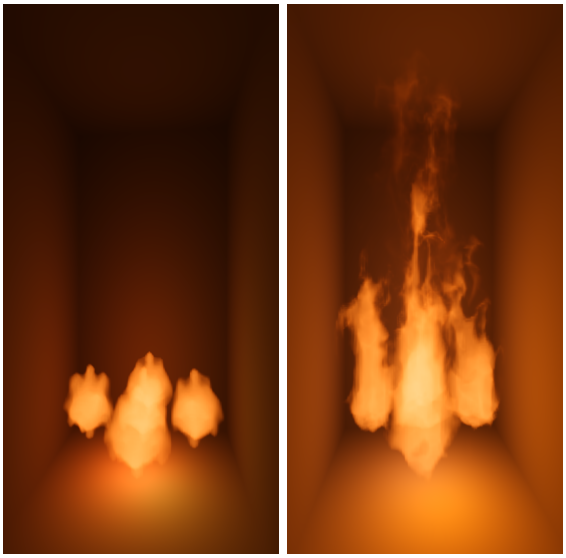
(a) frame 9 (b) frame 50 (c) frame 144

Figure 4.6: Using a lower flame speed S than the reference.



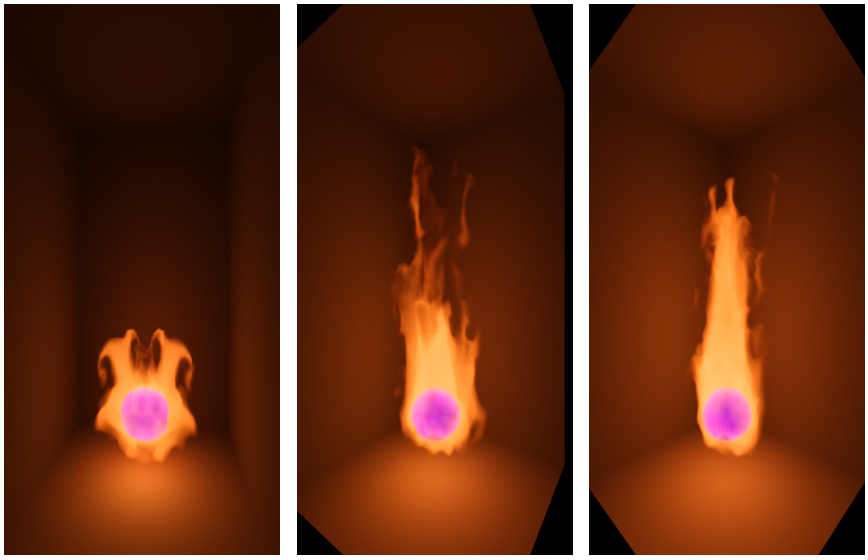
(a) frame 9 (b) frame 50 (c) frame 144

Figure 4.7: Using a higher ϵ_h than the reference.



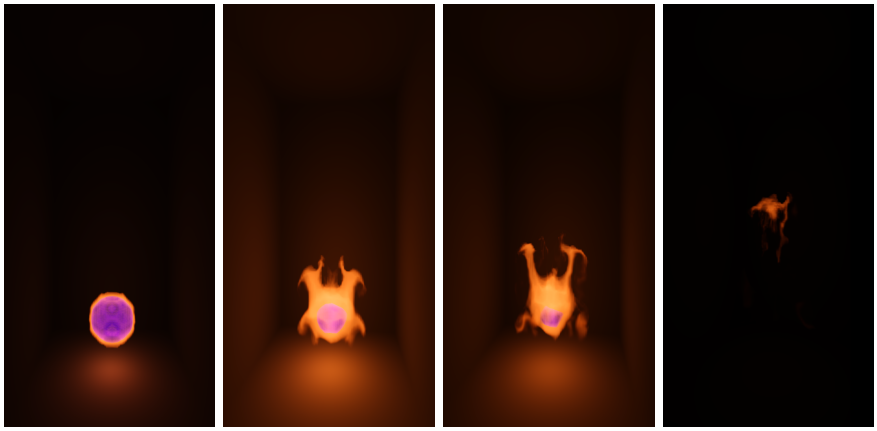
(a) frame 5 (b) frame 99

Figure 4.8: Injecting 5 smaller spheres with fuel instead of one big.



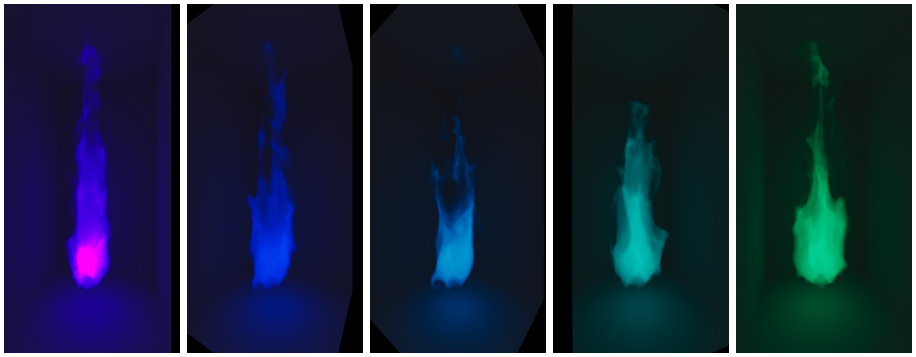
(a) frame 9 (b) frame 41 (c) frame 50

Figure 4.9: The blue core converts the radiance from the black body radiation to purple radiance only.

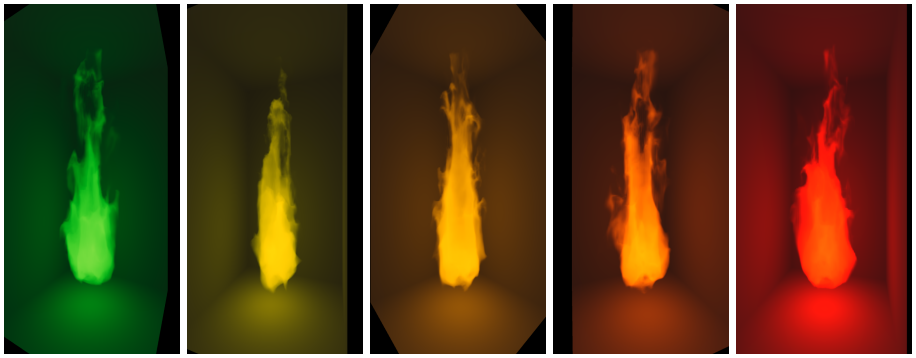


(a) frame 2 (b) frame 9 (c) frame 13 (d) frame 25

Figure 4.10: Same as figure 4.12, but without fuel injection.

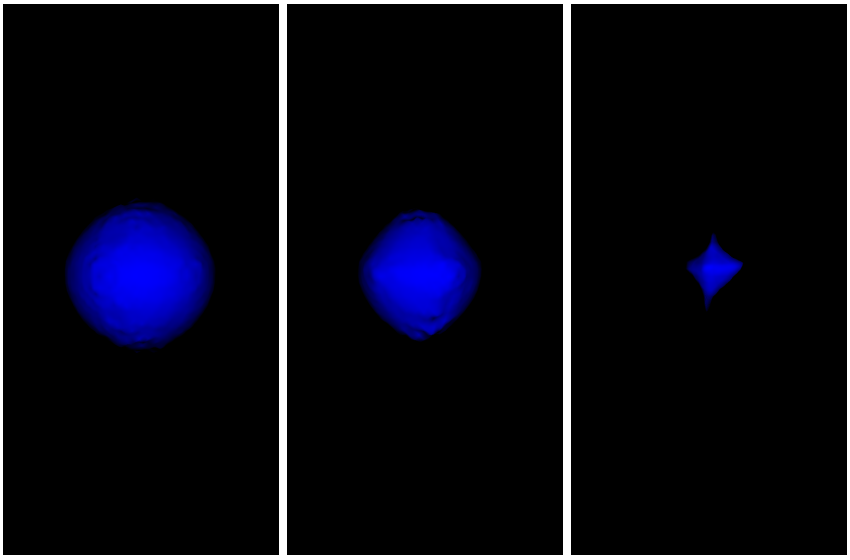


(a) frame 60 (b) frame 68 (c) frame 73 (d) frame 89 (e) frame 96



(f) frame 117 (g) frame 162 (h) frame 178 (i) frame 189 (j) frame 209

Figure 4.11: The black body only emits radiance for one wavelength at the time.



(a) frame 4 (b) frame 8 (c) frame 20

Figure 4.12: A burning blue core visualised as a surface.

Link to a video which demonstrates all the results in the figures:
http://youtu.be/F40_rowaLQg

4.2 Discussion

Our implementation of the simulation and rendering achieves a detailed and visually plausable fire. Physically based variables can be fine tuned to achieve a wide variety of flame types by adjusting fuel injection speed, fuel ignition temperature, flame speed, fuel density, cooling constants etc. Using world coordinates for our simulation grids enabled us to have different resolutions for different grids used in the simulation. For instance the temperature grid used for rendering can have considerably higher resolution than the grids used for the velocity fields in the same simulation run.

The design patterns used in our implementation is based on class inheritance, which turns out to be considerable performace hit, and created a sometimes unnecessary long simulation time. The current render method only samples from the simulation grids and does not perform any intersection tests against meshes or any triangles for that matter. We can therefore currently not render triangles, meshes, and so on or anything not present as data in the grids. The implemented calculation of the radiance values for the surrounding area is a quick method, but costly in memory since the radiance is calculated for every wavelength in every voxel, which is needed in order to calculate the mean radiance value. A more correct method would probable implement a Monte Carlo method instead.

Appendix A

Rendering report

Volume rendering of Physically Based Fire

TNCG14 Advanced Computer Graphics Programming - Project report

Viktor Nilsson
Linköping University
vikni067@student.liu.se

Axel Kinner
Linköping University
axeki412@student.liu.se

September 2, 2013

Abstract

We present a method for rendering physically based fire with help of volume rendering. The simulation produces a 3D dataset of temperatures which is then rendered with help of a basic ray casting technique and combined with a Black-body radiation conversion.

1 Introduction

Volume rendering is a widely used technique in scientific visualizations and computer graphics. The most common cases are sampled datasets out of CT scannings where the rendering is focused on the scanned density.

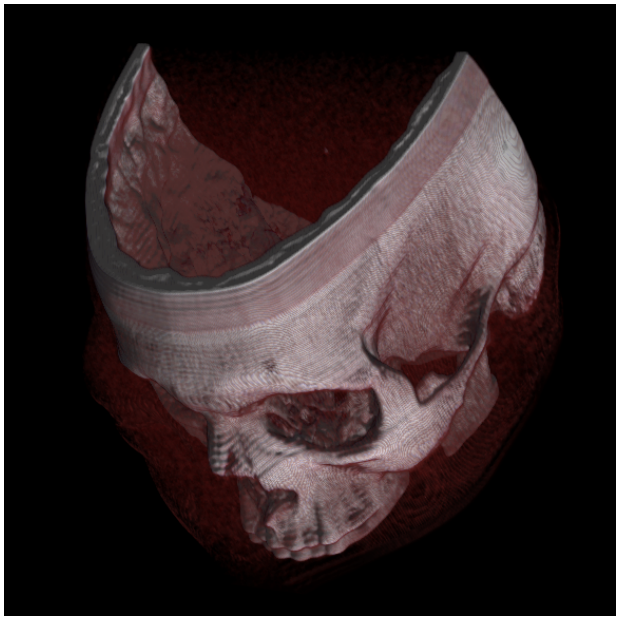


Figure 1: *Image of a cadavar head CT scan rendered using view aligned slicing with alpha blending. Lower density flesh and brain matter are assigned lower alpha values.* [3]

There are a wide range of techniques when rendering state of the art fluid simulations where the most common ones renders the actual surface of the fluid. There are also a big consideration when choosing between realtime and offline rendering.

Instead of focusing on rendering the density or the surface of a level set one could render the velocity or even the temperature. Since the human eye reacts to the light produced by the heat of a combustion i.e. flame, the temperature is the key element to the colors we see in a fire.

Our method is focusing on quality and the best looking results rather than speed and realtime effects. This resulted in that we render the simulation offline on the CPU as images, notice that we could reduce the quality and constants for the simulation and render so that we would achieve a realtime simulation and volume rendering but with very low detail level.

2 Previous work

Our project is an expansion of an already implemented fire simulation. The simulation is based on Nguyen02[7] and consists of two fluids, one for the ignited fuel and one for the surrounding fuel e.g. air. The simulation is described

by two separate Navier-Stokes equations, which can be solved numerically. The simulations are then coupled together using Ghost fluid method at the interface between the fluids. The Ghost fluid method enforces the conservation of mass and momentum when one fluid particle transforms from one fluid to another. This gives the fire the important expansion of the fluid, when the fuel is transformed into hot gas.

3 Black-body radiation

To be able to convert a temperature sample in the grid to a color, we use the black-body radiation model.

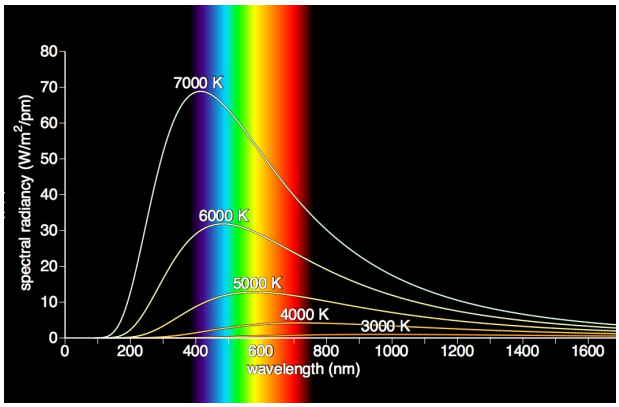


Figure 2: *Five different temperatures and their respective spectral radiancy depending on the wavelength. Notice the color spectrum that the human eye can observe, from 400 to 700 nm.* [2]

The black-body radiation model calculates an emitted radiance for a given wavelength and temperature, Planck's formula[7] Equation 1. Where $C_1 \approx 3.7418 \cdot 10^{-16} Wm^2$ and $C_2 \approx 1.4388 \cdot 10^{-2} m^\circ K$

$$L_{e,\lambda}(x) = \frac{2C_1}{\lambda^5(e^{C_2/(\lambda T)} - 1)} \tag{1}$$

This allows us to calculate the spectral radiance L for a given wavelength λ and position using the radiative transport, Equation 2, using the blackbody radiation as the emitted radiance.

$$\begin{aligned}(\vec{\omega} \cdot \nabla)L_\lambda(x,\vec{\omega}) = -\sigma_t(x)L_\lambda(x,\vec{\omega}) + \\ \sigma_s(x)\int_{4\pi} p(\vec{\omega},\vec{\omega}')L_\lambda(x,\vec{\omega}')d\vec{\omega}' + \\ \sigma_a(x)L_{e,\lambda}(x,)\end{aligned} \tag{2}$$

The scattering part, $p(\vec{\omega},\vec{\omega}')$, is neglected due to the high computation time and the fact that there are few scattering effects in flames.

The achieved radiance is then mapped onto the XYZ-color space by using the CIE standard observer 1931[1]. We then convert the XYZ colors to the LMS-color space with the CAT02 transformation method [4], Equation 3.

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.7328 & 0.4296 & -0.1624 \\ -0.7036 & 1.6975 & 0.0061 \\ 0.0030 & 0.0136 & 0.9834 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3}$$

We also add an chromatic adaption at this step to get a result as if the observers eyes and thereby vision have adapted to the intensity in the scene. Without this step a too bright flame was produced which is the case when looking at a flame with a normal to large sized pupil.

The LMS values are then converted back by inverting Equation 3. And finally the XYZ values are converted into the RGB-color space with an sRGB conversion, Equation 4[6].

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4}$$

4 Ray casting method

The radiative transport equation can be solved discreetly using ray-casting, by traversing a ray that goes from the eye and then through the volume. The temperature is then sampled several times along the ray. That gives us the discrete formulate Equation 5[7] and because we neglect the scattering effect as mentioned earlier we can also neglect $p(\vec{\omega},\vec{\omega}')$.

$$\begin{aligned}L_{n,\lambda}(x,\vec{\omega}) = e^{-\sigma_t \Delta x}L_{(n-1),\lambda}(x+\Delta x,\vec{\omega}) + \\ L_\lambda(x,\vec{\omega})p(\vec{\omega},\vec{\omega}')\sigma_s\Delta x + \\ \sigma_a L_{e,\lambda}(x)\Delta x\end{aligned} \tag{5}$$

The equation tells us that we have to traverse the ray backwards from its endpoint back to the eye, to simulate the energy loss from absorption, scattering and distance. We use a constant Δx during the whole rendering.

The ray-caster is using a perspective to give a more realistic view, which is calculated by first defining a forward, up and right vector (in unit lengths) from the eye. These vectors is then used to calculate the near plane position that belongs to a pixel (which has a u, v coordinate), Equation 6.

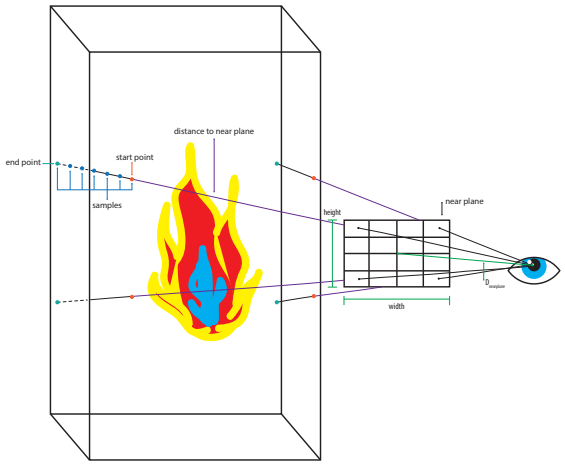


Figure 3: *Illustration of the raycasting algorithm. It illustrates how a ray starting from the viewer $\vec{x}_{eye}$, goes through the near plane $\vec{x}_{nearplane}$ and then through the first intersection point (start point) and then ends at the last intersection point (end point) of the box. The distance to near plane is the distance where the ray only lose energy. It also illustrates how the samples only exists between the start point and the end point (in this case no sample is in the fire, and would probably result in a pixel only lit by the wall). To simplify the illustraton only 4 rays are shown, in the real application one ray would go through each pixel in the nearplane, which would be 16 rays in this case.*

$$\vec{x}_{nearplane} = \vec{x}_{eye} + \vec{F} \cdot D_{nearplane} + \vec{R} \cdot u \cdot \frac{w}{2} + \vec{U} \cdot v \cdot \frac{h}{2} \quad (6)$$

Where u and v is screenspace coordinates, defined from -1 to 1 and w and h is the near plane width and height (which we calculate using a field of view calculation). This near plane position can then be used to calculate the direction of the ray.

To speed up the ray-caster we find the intersection points between the volume and the ray using a ray-box intersection algorithm. This allows us to have a start (closest intersection to the eye) and end sample point, which avoid redundant calculations outside the volume. If the eye is in the fluid, the near plane position is the start point. If the start position differs from the near plane position we have to add the energy loss between those positions.

To get a sense of depth in the rendering we render the sides of the volume as walls. We calculate the radiance that is reflected from a given point at the walls by integrate the radiance over all voxels incomming to that point, and using that value as the start value for λ in Equation 5, because the point is at the end position.

5 Discussion

We use OpenMP[5] for improving the speed when volume rendering our simulation. An average speedup on a Intel Core i7-3610QM CPU (2.30GHz, 4 cores, 8 threads) was calculated to 3.83 times faster. Notice that in some cases for specific time steps the speedup could be above 4, which is an superlinear speedup. Since the CPU has 4 cores the speedup would theoretically be impossible but due to the cache effect, where already calculated values and memories are reused; the speedup can sometimes be achieved.

The calculation of the reflected radiance from the walls that is described in x (blackbody chapter) is very naive, and therefor very slow. To improve the performance greatly we calculate the mean radiance and its mean position, for every wavelength. We then calculate the reflected radiance for each wall position from these values instead. This gives us almost the same results, but the rendering miss some small details like when a flame burst close to the roof or wall. This could also be solved more elegantly by using a Monte Carlo method.

Future work would be to add another render for the blue core. Since the blue core is due to a chemical reaction and cannot be rendered as with the flame. Also another future work is to add the very common smoke which occurs when the fire is getting a small amount of oxygen.

6 Results

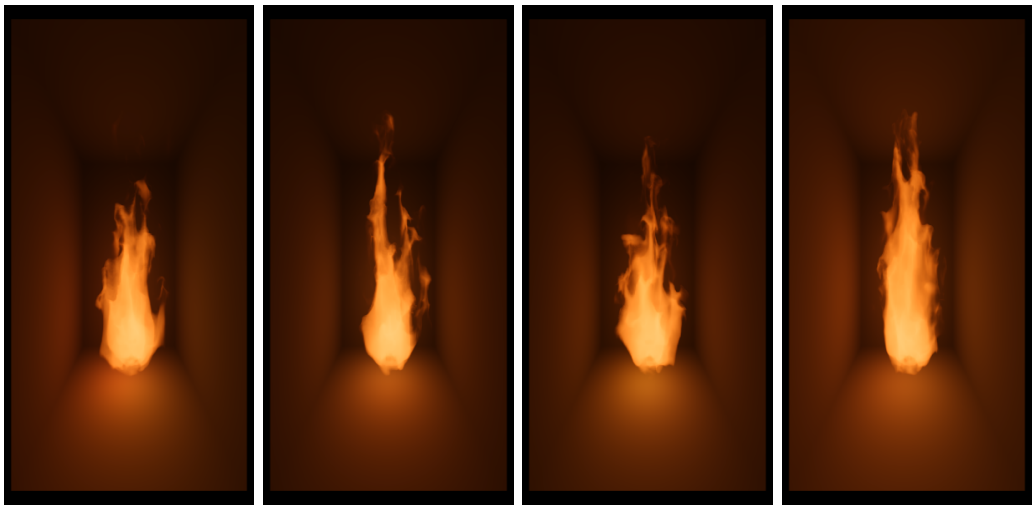


Figure 4: *Four images from our flame simulation and rendering (frame 90, 109, 224 and 454). Notice the intensity and position change of the reflected light at the walls and floor. Also notice that since we only use one average point as light source the radiance is hardly visible at the roof. Simulation grid size (30x60x30), temperature grid size (120x240x120), image size (300x600 pixels), flame speed* $S = 0.1$*, all wavelengths.*

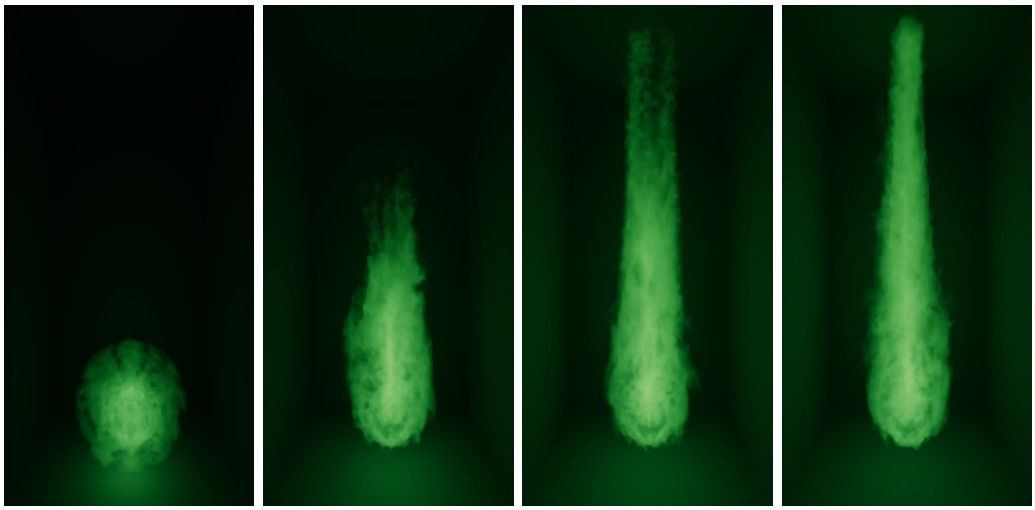


Figure 5: *Four images from our flame simulation and rendering (frame 7, 43, 89 and 214). Notice the intensity and position change of the reflected light at the walls and floor. Also notice that since we only use one average point as light source the radiance is hardly visible at the roof Simulation grid size (60x120x60), temperature grid size (120x240x120), image size (500x100 pixels), flame speed* $S = 0.2$*, only the green wavelenghts.*

References

[1] Cie starndard observer. 1931. `http://www.cis.rit.edu/mcsl/online/cie.php`.

[2] Blackbody. 2013. `http://www.exo.net/~pauld/workshops/Stars/Stars.htm`.

[3] Ct scan. 2013. `http://en.wikipedia.org/wiki/Volume_rendering`.

[4] Lms color space, cat02. 2013. `http://en.wikipedia.org/wiki/LMS_color_space`.

[5] Openmp. 2013. `http://openmp.org/wp/`.

[6] srgb. 2013. http://en.wikipedia.org/wiki/SRGB.

[7] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire, 2002.

Bibliography

[1] Cie starndard observer. 1931. http://www.cis.rit.edu/mcsl/online/cie.php.

[2] Lms color space, cat02. 2013. http://en.wikipedia.org/wiki/LMS_color_space.

[3] srgb. 2013. http://en.wikipedia.org/wiki/SRGB.

[4] R. Bridson. *Fluid Simulation for Computer Graphics*. 2008.

[5] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method), 1999.

[6] N. Foster and D. Metaxas. Realistic animations of liquids, 1995.

[7] J.-M. Hong, T. Shinar, and R. Fedkiw. Wrinkled flames and cellular patterns, 2007.

[8] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire, 2002.

[9] K. F. R. Courant and H. Lewy. Uber die partiellen differenzengleichungen der mathematischen physik, 1928.