International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 8, Number 2, pages 186–204, July 2018

Fire Simulation in 3D Computer Animation with Turbulence Dynamics including Fire Separation and Profile Modeling

Jozef Hladký
Max-Planck-Institut fur Informatik
Saarland Informatics Campus
Department 4: Computer Graphics
66123 Saarbrcken, Germany
email: jhladky@mpi-inf.mpg.de

and

Roman Ďurikovič
Department of Applied Informatics, Faculty of Mathematics
Physics and Informatics
Comenius University Bratislava
842 48 Bratislava, Slovakia
email: durikovic@fmph.uniba.sk

Received: January 18, 2018
Revised: April 25, 2018
Accepted: June 4, 2018
Communicated by Susumu Matsumae

Abstract

In this paper, we propose a system for modeling fire dynamics with emphasis on realistic behavior as well as the extensive behavioral control system. We propose a wide range of parametric and procedural controls. Simulation of flame spread, and motion is governed using differential equations which takes into account wind forces, buoyancy forces, diffusion and velocity of the burning surface. We propose to enhance the realistic behavior by using stochastic simulation of flickering and buoyant diffusion. The proposed wind fields serve as added fire motion control by a user. Flame behavior covers moving sources, flickering, fire separation and fire merging.

Keywords: fire, flames, animation systems, particle systems, physically-based modeling, wind fields

1 Introduction

Due to its dramatic nature smoke, fire, and gas are essential in many applications such as in virtual environments, flight simulation, landscape design, animation, and movie industry. In the fantasy genre, fire-breathing creatures like dragons are nowadays very popular. This creates the demand for modeling the behavior of flame that doesn't exist in the real world. People can imagine that dragon fire looks like the flamethrower, but this is rarely the case, as most artists want to make dragon fire easily distinguishable from real fire and makes it somewhat special. Due to a wide variety of artistic requirements, sometimes the fire needs to act like a burning gas, sometimes like burning liquid or something purely out of fantasy. The flame often needs to react with its surroundings and with other elements like water or wind. Artists can provide a very detailed idea of the flame behavior and appearance, but it can be quite complex to transform these ideas into corresponding mathematical models for correct computation. The animation and visualization of such phenomena is a challenging task. Among the recent and impressive works to model fluid behaviors, the Navier-Stokes (NS) equations have been solved in different ways. Many articles are published in different areas to compute NS equations with numerical methods.

We present a method for modeling the dynamics of a flame with a set of behavioral controls. The model is based on particle system spreading in procedural environmental fields. This keeps many of the advantages of particle systems while also allowing animators to treat a flame as a high level structural element. The primary contribution of this paper is to show that it is possible to create fire animations, using a pipeline in which the style of automatic in-betweens is derived from hand-drawn key-frames as fire skeletons and the fire motion is derived from a simulation.

First, we give a brief overview of the methods currently used to model realistic fire for 3D animation purposes. We present some older methods as well as cutting-edge tools used in the production of the latest movies. Next, we cover the essential simulation steps of our method based on the central spline of the flame, see Sec. 3. Following is our proposed method, which presents modification of the structural modeling method using specific wind fields implementation, modification of the main differential equation responsible for the movement of flame particles, see Sec. 4. The motion of the central spline in the dynamic environment including the user intervention and re-sampling is proposed in Sec. 5. Vector wind field used in our model is reviewed in Sec. 6. We also explain the stochastic elements of the flame separation and flickering as well as buoyant forces in Sec. 7. Sec. 9 lists the fundamental structures and essential technologies followed by results with some examples and conclusions.

2 Overview of Previous Methods

Fluid simulation has been a highly active area of research in 3D computer animation [1]. Recent methods combine hand-drawn artwork with fluid simulations to produce animated fluids [2] that can be extended to hand drawn fire simulation. Numerical models from Computational Fluid Dynamics have not proven amenable to simplification without significant loss of detail [3]. Physically based models has also proved efficient at modeling the behavior of smoke [4, 5]. Authors [6] present a method for simplified modeling of fire applying the art coarse grid fluid dynamic equation solvers to model the motion of fuel and air in a unified system. For the shape editing and motion control of flames however, simulation has not been so useful. Fluid simulations are an expensive option for large scale fire animations without simplification.

Another category of modeling dynamic fire falls in visual modeling [7] focusing on efficiency and control. The high-detail textures were introduced to achieve a smooth boundary of the fire and gain the small-scale turbulence appearance [8]. Preliminary studies on implementation of physically-based models for fluids like fire and smoke effects on a mobile environment have been done [9]. Authors [10] introduced pure geometry based approach to fire modeling based on simulating spreading on polygonal meshes.

Particles systems used to model fire can interact with other primitives, are easy to render, and scale linearly (if there are no inter-particle interactions). Adding the inter-particle forces as the spring forces [11] models the kinematics of flames using a mechanical trick known as the "silk torch". The method [12] constructs a flame animating system based on particle system with appropriate and effective particle control. This system provides a range of behavioral controls suitable for artistic animation.

Here we list the well-known commercial applications available for fire modeling. Animators often want to adjust the flame's appearance in lower resolutions because it provides faster turnaround and after achieving the desired behavior they want to increase the resolution to improve detail for the final rendering. The upresNode plugin [13] eliminates one major drawback of Maya's fluids - that is, if you change the resolution of the grid the fluid behaves differently. The plugin offers a new node that increases the resolution and local detail without changing the fluid's behavior. It also allows additional detail at post processing by implementing wavelet turbulence algorithm proposed by [14].

Phoenix FD is a plugin developed by ChaosGroup and available for 3DsMax and Maya. The plugin handles fire, smoke, explosions, liquids, foam and splashes. It offers a hybrid simulation system including grids and particles. Fully utilized with V-Ray renderer, it offers proper refraction on liquids [15].

The most complex animation, simulation up to date was done by Weta Digital for the feature film Hobbit: The battle of the five armies. Combined simulations are used to achieve the resulting appearance, consisting of air, fire, water and rigid bodies simulations [16].

Modeling flame movement as direct numerical simulation is very expensive computation-wise. These models often act in a 3D grid. As the grid resolution increases, computational complexity raises by at least $O(n^3)$ [17] [18]. It is also difficult to implement intuitive control points for a physical based fire model. In addition, it is very hard for animators to achieve the desired visual effect with numerical simulation, as even a small change in starting conditions can provide drastically different results. Despite the problem complexity, we propose the real-time method $30fps$ for flame simulation with the export of simulated particles to external renderers for high quality off-line rendering.

Proposed method falls in direct numerical simulation methods where we prose the simplified NavierStokes equations [1] with the thermal therm buoyancy to be solved fully in 3D with Lagrangian method. This retains many of the advantages of particle systems while animators can treat a flame as a high level structural element. The primary contribution of this paper is to demonstrate that it is possible to create fire animations, using a pipeline in which the style of automatic in-betweens is derived from hand-drawn key-frames as fire skeletons and the fire motion is derived from a simulation.

Proposed simulation method for flames and smoke comparing to listed in this section satisfy the following requirements:

1. The motion of flames produced in an interaction with obstacles can be simulated while following the flame key-frames as fire skeletons.

2. The motion of flames and smoke can be easily controlled according to scenarios defined by flame skeletons, skeleton branching and wind forces.

3. The spread of fire can be simulated in full 3D particle based fluid dynamics in real time.

3 Essential Simulation Components

As is common in earlier methods the flame modeling, editing and simulation is based on particle system. Realistic appearance is achieved using proposed stochastic models of flickering and buoyant diffusion, introduction of wind fields gives additional procedural control. Our flame model behavior includes moving sources, flickering, separation and merging, combustion spread and interaction with stationary objects.

The whole proposed model can be divided into 7 essential components arranged into a pipeline:

1. A central spine is formed using the central particles of the flame. The positions of these particles form a set of points which act as control points to an interpolating spline curve. This curve defines the spine of the flame and is the main actor in the shape and behavior of the flame. The model also covers realistic adaptation of the flame to the movement of the source. We can have one or multiple flame emitters. Each emitter has its position p, radius r, velocity v and b, which is the number of Base splines it will produce concurrently. The following steps are the same for each emitter in the scene. When Base splines collide with each other, they are not affected at all.

2. In the first time step of the animation, the emitter creates a new segment. This segment is created at the present position of the emitter and consists only of one particle p_0 which will gain the user-specified values of initial temperature T_{p_0} and initial age t_{p_0} (typically zero). The particle also inherits the position p and the velocity $-v$ of the emitter. Gaining the negative velocity of the emitter covers the realistic flame behavior when the burning surface holding the

emitter moves. This particle acts as the start point and the end point of the segment, while the segment does not have yet any control points and thus no spline.

3. The splines evolve through space in time. Each point of the curve carries attributes of the flame height, age, temperature, etc. The evolution depends on hand-defined, procedural and physics-based wind fields. Some physical terms that affect the spline movement are based on statistical measurements of real-life flames. Therefore, in the second time step frame of the animation, particle p_0 is released into the environment, where it moves according to Eq. 1 and is amortized with the methods proposed in Eq. 4. New particle p_1 is generated at the origin of the emitter and N control points are uniformly sampled between the particles p_0 and p_1. The control points have their age and temperature linearly interpolated between the values of the points p_0 and p_1. We have tried also assigning the properties of p_0 to all of the control points, but this provided noticeable visual separation of the segments. All the particles of the segment, then move and amortize analogically. Every following segment is created and moved in the scene analogically.

4. The curves can break when reaching specified lengths. The separation is based on the statistic measurements of natural diffusion flames. The separated segments act as individual splines, but they do not generate new segments. They convect freely in the environment affected by the wind field. The separated segments are given limited lifespan, that can be also user-controlled. The separation and flickering occurs as described in Sec. 7.

5. The visible part of the flame is defined as a normalized 2D profile. We create a normalized 3D profile by symmetrical rotation of the 2D profile. We then randomly generate s particle positions and test them against the profile using rejection sampling. The resulting set of particles is then mapped from the parametric profile space into the space of the deformed Base spline using cylindrical coordinates. The particles inherit the age and temperature attributes of the structural elements at the corresponding level of the spline.

6. The first level of procedural noise is applied to particles created within the flame region. The noise is buoyant in nature, as it represents the combustion fluctuation of the flame base. It spreads up the flame profile based on the velocities of the Base spline segment. A second level of noise is applied using a vector field created using a Kolmogorov frequency spectrum. This second level of noise simulates turbulent distortion details.

7. The particles are rendered with a volumetric or a fast painterly method. Thanks to the color adjustments of each particle based on the color properties of the neighboring particles, the flame elements can merge realistically.

These 7 components form together a complex and general system for efficient and realistic flame animation along with some other similar natural fire effects. The computational complexity can be kept on a very low level when desired, compared to numerical simulation approaches. This due to the idea of driving the flame with skeleton spline curves reducing the necessarily amount of 3D particles in the flame simulation to only spline particles while number of generated particles around the spline can be reduced for simulation purposes. Later in the visualization process the number of visual particles that are out of simulation can be increased. We do not cover the last 4 stages in detail because the main focus of this paper is on the dynamics and structure of the flame.

4 Proposed Weighted Dynamics Model

We propose a method that creates the basic structure and dynamics for a controllable 3D fire effect. The flame's motion is achieved by solving differential equations that take account of procedural environmental factors as well as statistically measured factors, which are assessed on real-life observation and can be fine-tuned according to artistic taste. Our method is based on DreamWork's method conceived by Arnauld Lamorlette and Nick Foster [12], we have created our own structure, chosen the wind fields implementation, heuristics and parameters allowing greater control were added whenever reasonable.

We propose the dynamics by extend NS differential equations governing the flame dynamics simulating the dynamics of splines and particles in the following form

$$\frac{\partial x_p}{\partial t} = \alpha w(x_p, t) + \beta d(T_p) + \gamma V_p + \delta c(T_p, t), \tag{1}$$

where α, β, γ and δ are weight functions corresponding respectively to the wind field, diffusion, source motion and buoyancy terms. Setting the weights constant or interpolating their values over time of the animation creates added controls that help us shape the behavior of the flame to our needs. Interpolating the values can serve as key frame animation process, where key-frames define the exact flame shape at given simulation time. In addition, x_p is the position of particle p, $w(x_p, t)$ is the displacement vector due to wind fields, $d(T_p)$ represents Brownian motion scaled by the temperature of the particle, T_p, the temperature of the particle p. V_p is the displacement due to the motion of the source and $c(T_p, t)$ is the motion due to thermal buoyancy. The thermal buoyancy term is assumed to be constant over the lifespan of the particle, therefore

$$c(T_p, t) = -\beta_t g_y (T_0 - T_p) t_p^2, \tag{2}$$

where β_t is the coefficient of thermal expansion, g_y is the vertical component of gravity, T_0 is the ambient temperature and t_p is the age of the particle.

Because we are working with fire, the particle can get hotter than the environment, thus

$$\forall p : T_0 <= T_p.$$

It depends on our cooling heuristic, but in general the t_p^2 term affects the buoyancy term exponentially while $(T_0 - T_p)$ decreases linearly, which results in the rising of older particles despite their cold temperatures.

When a new particle is created at the source, it has initially the default user-specified parameters. If the source has velocity V, the particle is assigned velocity $-V$. This negative velocity is completely enough to create a realistic reaction of the flame to a moving source.

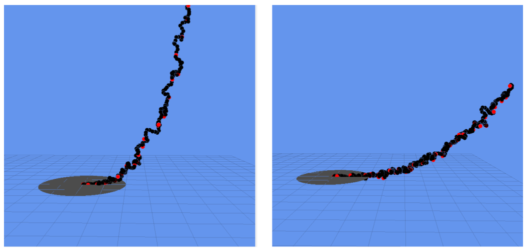

Figure 1: Example of modifying the parameters of Eq. 1. Left, $\delta = 1$. Right, $\delta = 0.1$. The uniform wind field which blows to the right.

4.1 Amortization

We propose to use linear techniques of temperature decay

$$T_{t+1} = max(T_t - r\Delta t, T_0), \tag{3}$$

where T_{t+1} is the temperature of the particle in the next time step, T_t is the temperature in the current time step, r is the cooling rate parameter and Δt is the time step. T_0 is the ambient temperature. Since, we are modeling fire, we have decided that the particle temperature cannot drop below ambient temperature.

Similarly, we define the age amortization as

$$t_{t+1} = t_t + a\Delta t, \tag{4}$$

where t_{t+1} is the age of the particle in the next time step, t_t is the temperature in the current time step, a is the parameter specifying the aging rate and Δt is the length of single time step. We also define maximal age t_{max}. If the age of the particle reaches t_{max}, the particle is dropped from the simulation pipeline.

The values for parameters c and a are chosen to suit our needs and are used as one of the controls for flame behavior and appearance. They mainly affect rendering, particle color and buoyancy in the upper stages of the flame.

5 Evolution of the Flame

The flame moves according to a combination of forces, some of which can be specified by user (like the procedural physics-based wind-fields) and some of them are statistical in nature and based on observation of natural diffusion flames. The motion is also influenced by user-defined parameters (like ambient temperature) and heuristics (cooling rate, aging rate). The evolution of the flame goes as follows:

1. In the first frame of the animation, a new particle p_0 with initial temperature T is created on the burning surface.

2. In the second frame of the animation, particle p_0 is released into the environment, where it moves according to Eq. 1 using explicit Runge-Kutta integration method that is sufficient and we can easily set the integration step such that method is completely stable in this case.

3. A new particle p_1 is generated at the surface and an interpolating B-spline is created between particles p_0 and p_1. According to preset density n, n control points are uniformly sampled between them.

4. At the third frame a new particle p_2 is created on the burning surface. A new set of n control points is created between p_2 and p_1. All the control points between p_0 and p_1 along with p_0 and p_1 themselves are moved according to Eq. 1. The interpolating B-Spline is then fitted to pass through all the control points. The control points are then re-sampled so even distribution along the length of the spline is achieved. The first and last point of each segment is left unchanged, only the control points are re-sampled. This enables us to maintain constant detail along the whole flame.

5. For each additional frame of the animation we continue analogically with creating new segments. After reaching a predefined height, the spline can separate in two pieces.

5.1 Emitter

The flame creation is handled by an emitter. It is the most fundamental element as it carries important parameters needed for creation of the Base splines of the flames. The scene can have multiple emitters and each emitter can emit multiple Base splines, which each form an individual flame. Due to the chosen rendering methods, all the flames emitted from one emitter appear as if they joined together to form one flame. So, while it may look like each emitter emits only one flame, in fact, they are emitting multiple flames which quickly visually merge. Each emitter has its position, velocity, density, and radius of the burning source surface, see Tab. 1.

5.2 Base spline

The Base spline is emitted from the emitter and consists of spline segments, see Fig. 2. At every frame, the new segment is created at the bottom of the spline according to random distribution

Table 1: Table of emitter attributes.

Origin	o	Specifies the position of the emitter
Velocity	$\vec{v}$	Velocity of the emitter
Density	n	Number of Base splines the emitter emits
Radius	r	Radius of the burning surface

presented in Sec. 7. The total length of the base spline s is

$$L = \sum_{i=0}^{n-1} l_{s_i}, \tag{5}$$

where n is the number of segments of the spline and l_{s_i} is the length of i-the segment in the spline.

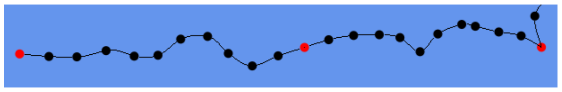

Figure 2: Base spline. Red points are end points separating the segments, black points are control points and the spline is interpolating Catmull-Rom spline.

5.3 Base Spline segments

The individual splines consist of s segments. Each segment consists of a start point p_s, end point p_e and n control points $cp_0, \ldots, cp_{n-1}$. The density of the control points specified by n doesn't need to be uniform in all segments of spline, as when we introduce flame separation, the spline can break in the middle of a segment, forming 2 splines with non-uniform density. The number of control points in the newly created segments can be user-specified to achieve the desired balance between computational complexity and detail level. The segments act as in a linked list, with the last point of $i-1$ -th segment being the start point of i-th segment, $p_{e_{i-1}} = p_{s_i}$. At i-th segment a B-Spline is interpolated from p_{s_i} through all control points of that segment arriving at p_{e_i}. Thus, a spline segment with density n consists of $n+2$ particles (start, end point and n control points) and contains $n+1$ segments.

The length of a spline segment is defined as the length of the spline starting in p_s, passing through every control point and ending in p_e, see Eq. 7.

5.4 Catmull-Rom spline as Spline segment

We use the Catmull-Rom spline defined as

$$p(t) = \left(1, t, t^2, t^3\right) \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 2-\tau & \tau-2 & \tau \end{pmatrix} \begin{pmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{pmatrix}, \tag{6}$$

where t is the interpolation parameter, τ is the tension and p_{i-2} , $\ldots$, p_{i+1} are points defining the spline. The spline itself is drawn only between points p_{i-1} and p_i, while points p_{i-2} and p_{i+1} serve only for computing the required tangents. Note that $p(0) = p_{i-1}$ and $p(1) = p_i$.

This spline proved sufficient for our implementation and provided satisfying results, see Fig. 3. We can approximate the length of the spline as

$$L = \sum_{t=0}^{n-2} \left\| p\left(\frac{t}{n-1}\right) - p\left(\frac{t+1}{n-1}\right) \right\|, \tag{7}$$

where n is our chosen sampling. In our work we found $n = 100$ sufficient.

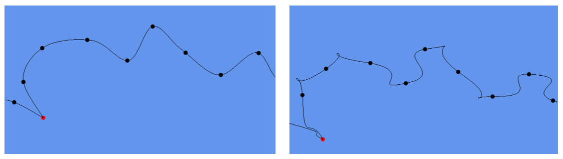

Figure 3: Example of different tension values for the Catmull-Rom spline. On the left $\tau = 0.5$, on the right $\tau = 2$.

6 Wind Fields

We model our wind fields based on the method proposed by [19]. We define 4 types of wind-fields - Uniform, Sink, Source and Vortex, see Fig. 4. Sink, Source and Vortex wind fields are circular and have always one fixed axis. Each acting wind field displaces the position of particle x_p. The displacement vector is affected by time step of our simulation, so it is scaled by Δt.

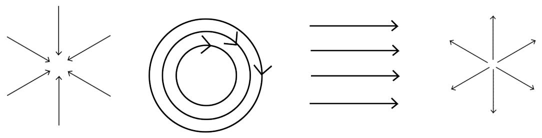

Figure 4: Schematic describing the sink, vortex, uniform and source types of the wind fields.

In the uniform wind-field, it is enough to specify only the direction and strength. However, for the circular types it is not sufficient, so we specify the displacement vectors in cylindrical coordinates (r, θ, z). For the source wind-field placed at the origin the displacement vector is defined as

$$v_r = \frac{s}{2\pi r}\Delta t; \qquad v_\theta = 0; \qquad v_z = 0, \tag{8}$$

where s is the strength of the wind field and r is the r coordinate of our particle's position in cylindrical coordinates and Δt is the size of our chosen time step. The Sink wind-field produces the same displacement vector as a Source wind - field, except the strength parameter s is set negative.

As the Vortex wind-field rotates the given point around the wind-field's origin, it changes only the angle θ in cylindrical representation. The displacement vector for a Vortex wind - field placed at the origin with strength s is then given by

$$v_r = 0; \qquad v_\theta = \frac{s}{2\pi r}\Delta t; \qquad v_z = 0, \tag{9}$$

where s is the strength of the wind field and r is the r coordinate of our particle's position in cylindrical coordinates and Δt is the size of our chosen time step.

The resulting displacement vector $w(x_p, t)$ of Eq. 1 is the sum over each wind-field in the system

$$w(x_p, t) = v_{vort}(x_p, t) + v_{sink}(x_p, t) + v_{source}(x_p, t) + \cdots .$$

7 Separation and Flickering

In order to model flame separation as a statistical process, we must first divide the flame into regions according to the height of the flame, see Fig. 6. The first region will be the persistent flame region, its height denoted H_p. In this region, the flame will never separate and so the spline will be always continuous. Then the intermittent region, where we will decide if the flame will or will not separate. The height of the intermittent region is denoted H_i. And the final region - the Buoyant plume - will

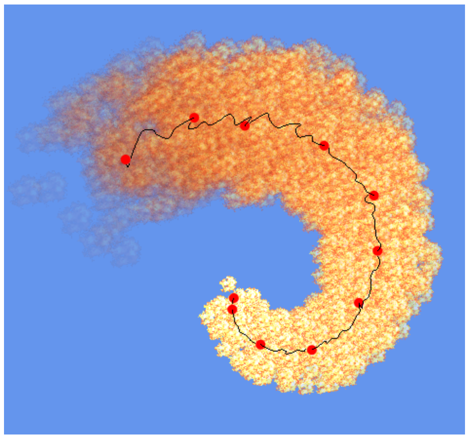

Figure 5: Example of flame behavior in combination of Vortex and Source wind-fields.

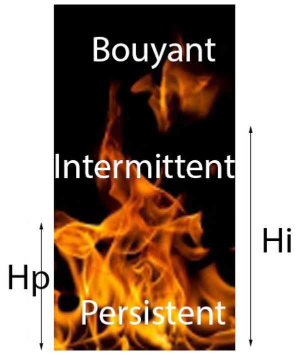

Figure 6: Illustration of persistent, intermittent and buoyant regions of the flame.

be the part of the flame which is separated from the base of the flame. The plume will be short-lived, and it will convect in the wind field freely.

The separation occurs in the intermittent section. When a particle exceeds H_i, we periodically test a random number against the probability function

$$D(h) = \frac{1}{\sqrt{2\pi}(H_i - H_p)/2} \int_{-\infty}^{h} e^{\frac{-(h-|V_c|/f)}{(2((H_i-H_p)/2)^2)}} dh, \tag{10}$$

where h is the height of the flame at which we are testing, H_i is the height of intermittent flame region, H_p is the height of the persistent flame region, $|V_c|$ is the average velocity of the structural control points. f is the approximate breakaway rate in Hz. According to the observations by [20] $f = (0.50 \pm 0.04)(g_y/2r)^{1/2}$ for circular sources with a radius r. We can also specify $D(h) \equiv 1$ for $h > H_{max}$, where H_{max} is our desired maximal limit for the flame height. We approximate the integral in Eq. 10 as follows

$$D(h) \approx \frac{h-g}{N} \sum_{i=g}^{h} e^{-\left(i-\frac{|V_c|}{f}\right)/\left(2((H_i-H_p)/2)^2\right)}, \tag{11}$$

where g is the chosen lower point from which we start approximating the integral and N is the number of steps that we wish to sample between g and h. Satisfying results were obtained using $h = -100$ and $N \approx 10^7$.

To separate the flame a part of the spline is cut off from the top of the flame. This portion ranges from the top to a randomly chosen point below, see Fig. 7. The distribution for cutoff point choice is not based on any observations. Normal distribution $\mathcal{N}(\mu,\, \sigma^2)$ proved sufficient with mean μ and standard deviation σ chosen as:

$$\mu = H_p + (H_i - H_p)/2, \qquad \sigma = (H_i - H_p)/4.$$

We model the normal distribution as Box-Muller Transformation. Given variables x_1 and x_2 which are uniformly and independently distributed between 0 and 1, we define z_1 and z_2 as

$$z_1 = \sqrt{-2 \ln x_1} \cos(2\pi x_2), \quad z_2 = \sqrt{-2 \ln x_1} \sin(2\pi x_2)$$

giving z_1 and z_2 as a normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. Due to our flame structure, after selecting the proper height of separation, we need to find the corresponding point on the spline.

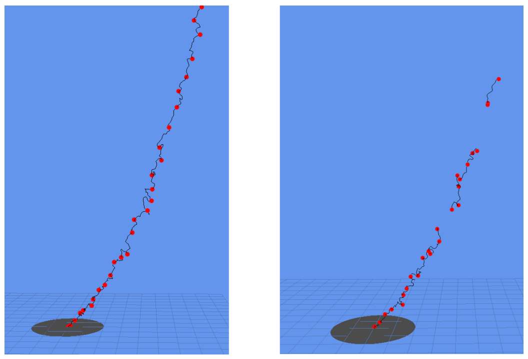

Figure 7: Example of spline separation. Evolution of the same spline with common settings and wind fields. Left, the separation is disabled. Right, the separation is enabled.

The separated segment of the spline is not re-sampled back with other control points as this prevents added local detail appearing in the separated segment. To account for lack of an accurate way of fuel content determination, the particles in the spline are given a limited lifespan of Ai^3, where i is a uniform random variable in the range $[0, 1]$ and A is a length scale ranging from $1/24^{th}$ of a second for small flames up to 2 seconds for a large pool fire. Because of term i^3 most of the breakaway flame has a very short lifespan.

8 Flame Profile Generation

At this stage when the spline creation and evolution were covered, we can now focus on the visible part of the flame. The flame is defined as the region between the burning surface and an oxidizing agent. We utilize a volumetric model created by rotation of a 2D normalized profile around the axis of the Base spline. The profile can be hand-drawn or derived from a photograph and creates a rotationally symmetric surface, see Fig. 8. In 3D space to fire particles are point-sampled and transformed into the spline structural curve. Two levels of procedural noise are added random positioning and turbulent distortion, respectively. Finally, the particles are in their correct positions and are ready to be passed into the rendering stage.

9 Implementation

Our method is implemented in C# and developed on operation system Windows 10. The compiler is.NET 4.5. IDE used is Microsoft Visual Studio 2013 Ultimate. The GUI is implemented using WinForms, the visualization using OpenGL API v3.3. To access OpenGL in C#, we use OpenTK library, v1.1. The data structure can be divided into 3 main parts:

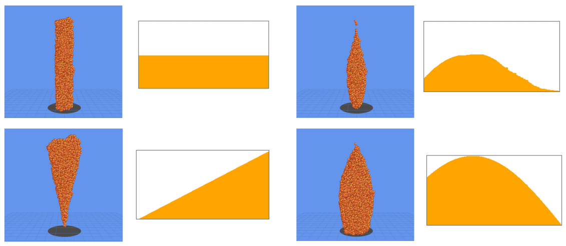
Figure 8: Examples of different profiles and their effect on the shape of the flame. The amortization shader is turned off. Second and fourth column show 2D flame profile drawn by user. First and third column shows the 3D flame model.

- Flame representation (Emitters, Splines, Particles)
- Parametric controls (Wind-Fields, Settings, Differential equation solver)
- Helper classes (Camera, Settings, GUI elements, Converters, Math classes)

The flame real-time preview is rendered with the splatting method in shader however the export to external rendering software is available for more complex scene. Understanding of the flame structure is essential to understand the algorithms and dynamics described later in this chapter. This section covers the classes that build up the whole flame structure including the particles, see Fig. 9. Because the spline structure is not trivial, we present a bottom-to-top explanation of the whole structure responsible for representation of one burning surface. We start with describing the basic particle types and move up the structure until we reach the topmost class.

9.1 Flame elements

The essential structures are Spline particles, which carry information about the age, temperature, position, and velocity of the particle. The *SplineParticle* is an abstract class, we define the different particle types using inheritance. We have three types of particles: *FireParticle* for representing the visible part of the flame, *ControlPoint* for representing the control points of each *SplineSegment* and *SplineEndPoint* for representing the first and last point of each *SplineSegment*. They all inherit the basic attributes from *SplineParticle* class.

The *CatmullRomSplineSegment* also belongs into the scope of the smallest elements of our structure. It represents one segment of a Catmull-Rom interpolating spline. Despite being defined by four spline particles $p_{i-2} \dots p_{i+1}$, it represents only the part between particles p_{i-1} and p_i. It contains basic methods of finding the corresponding point on the spline by implementing Eq. 6. For details, see Sec. 5.4.

The *SplineSegment* class contains two *SplineEndPoint*s representing the first and last particles, a linked list of n *ControlPoint* particles and a *CatmullRomSpline* class, which covers the representation of the spline stretching from the first *SplineEndPoint* through all the *ControlPoint*s in the linked list to the last *SplineEndPoint*. Each segment of this spline is represented by *CatmullRom-SplineSegment* class.

Spline segment has functionality responsible for evolution and amortization of its particles as well as methods for computing the length of the spline or finding a particle at specified height. *CatmullRomSpline* class handles re-sampling of the control points.

The *BaseSpline* class consists of linked list of *SplineSegments*. It is used for representing the persistent flame part as well as the separated buoyant plumes. While the persistent Base Splines have fixed life-time specified by parameter t_{max}, the separated segments are given limited life-span. This class is also responsible for drawing itself in the visualization window, so methods and data structures needed to draw points and polylines in OpenGL are implemented. Its functionality covers

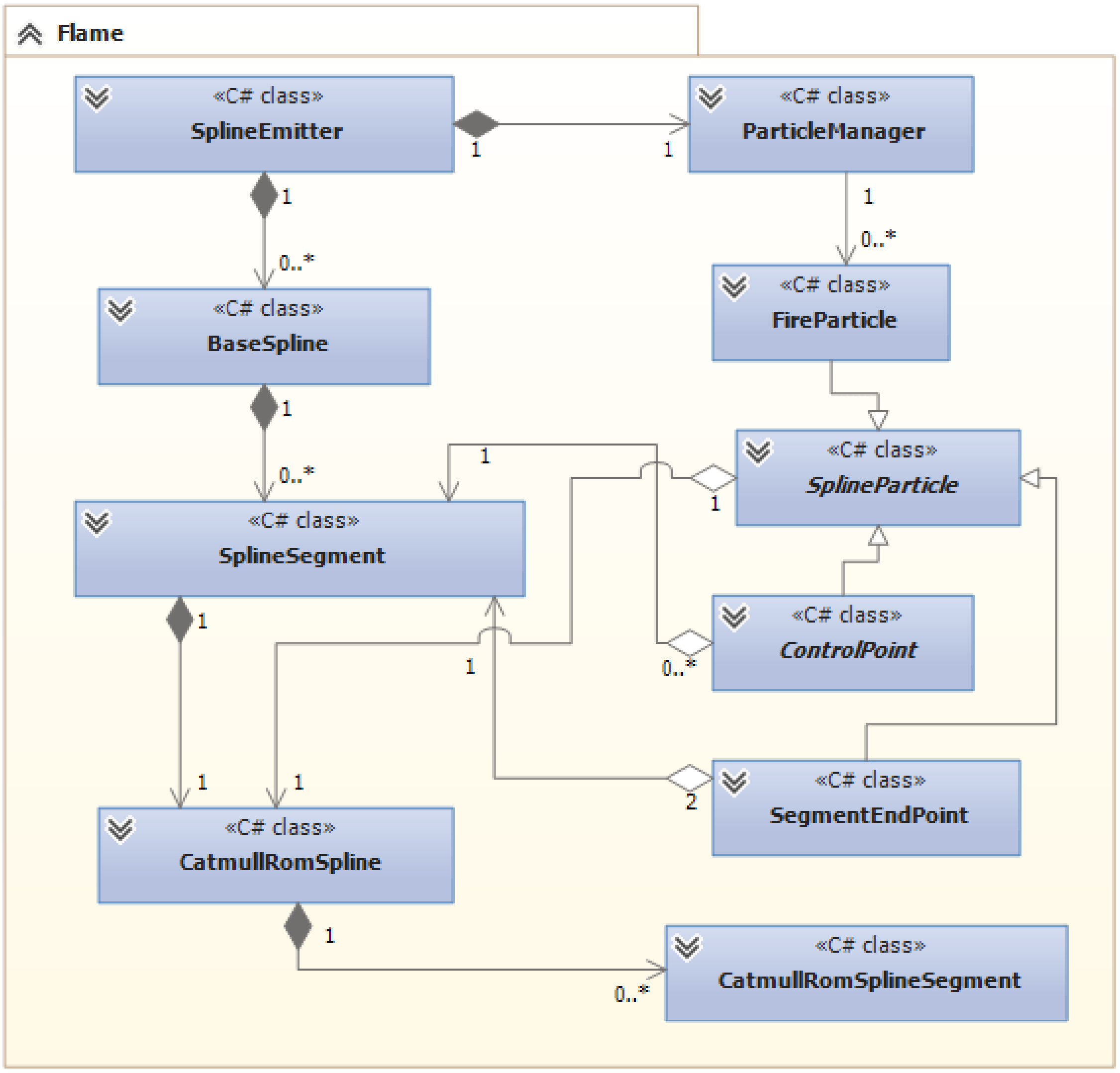

Figure 9: Class diagram depicting representation of the flame in our structure.

the calculation of the average velocity of all control points of the spline, the separation of the spline and a method for finding particle at specified height.

The topmost class in our structure is the $SplineEmitter$. It has specified radius r, position p and a spline count s. It also has two linked list structures, one representing persistent Base Splines and one representing the separated Base Splines. In the specified interval it emits s new Base Splines. It also governs functionality responsible for testing whether to separate the spline and it hosts the $ParticleManager$ class responsible for managing the visible part of the flame.

The $ParticleManager$ class is responsible for building up the visible part of the flame by creating the particles represented by *FlameParticle* class. Every spline emitter has one instance of *Particle-Manager*. By using the flame profile represented as an array of floating point numbers, it governs the creation of the particles for all the splines of the corresponding *SplineEmitter*. By using the helper classes that implement the needed transformations between normalized 3D profile space and structural spline space it outputs the flame particles with correct attributes and positions. Since it is responsible also for the rendering of the particles, this class also covers methods and data structures necessary to setup OpenGL shaders and buffer objects.

9.2 Parametric controls

In this section we will cover data structures responsible for the dynamics and control of the fire. We can control the fire by specifying various values for parameters describing the dynamics model as well as by specifying multiple wind-field types in the scene. In our application the setting of the constants and parameters are implemented as a singleton classes.

9.3 Wind-fields

Similar to particle implementation, in the wind field simulation and editing, see class diagram in Fig. 10, inheritance plays major role and is essential to the structure. We define one abstract class wind-field which has key method shared across all types of wind-fields - *getDisplacement*. This method returns the displacement vector affecting given position. The uniform wind field has only direction and strength attributes. The circular wind-fields also utilize inheritance, because they all share the same attributes - origin and strength. The difference is in the *getDisplacement* function, which each representation overrides.

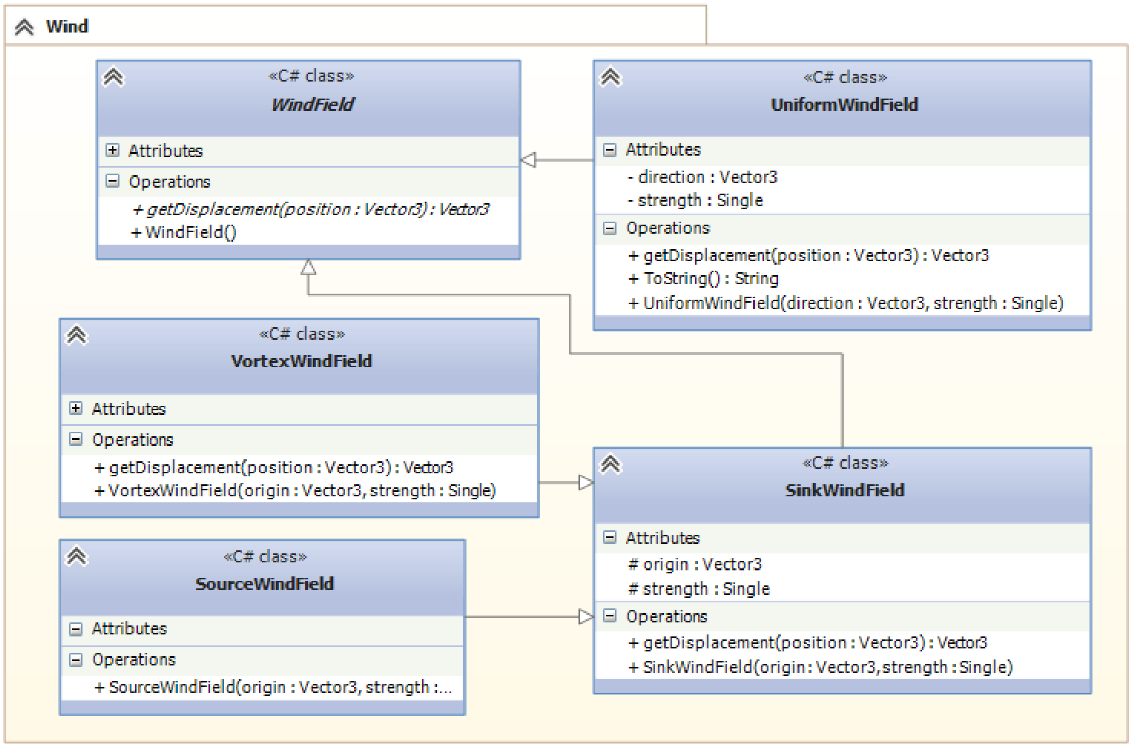
Wind
«C# class»
WindField
Attributes
Operations
+ getDisplacement(position : Vector3) : Vector3
+ WindField()
«C# class»
UniformWindField
Attributes
- direction : Vector3
- strength : Single
Operations
+ getDisplacement(position : Vector3) : Vector3
+ ToString() : String
+ UniformWindField(direction : Vector3, strength : Single)
«C# class»
VortexWindField
Attributes
Operations
+ getDisplacement(position : Vector3) : Vector3
+ VortexWindField(origin : Vector3, strength : Single)
«C# class»
SinkWindField
Attributes
# origin : Vector3
# strength : Single
Operations
+ getDisplacement(position : Vector3) : Vector3
+ SinkWindField(origin : Vector3, strength : Single)
«C# class»
SourceWindField
Attributes
Operations
+ SourceWindField(origin : Vector3, strength :...

Figure 10: Class diagram depicting representation of the wind in our structure.

Simulation step calculation - *SplineCalculator* is a singleton class responsible for calculation of the main differential equation in each time step. It also holds a reference to a list having all the wind-fields defined in a scene. After taking a *SplineParticle* descendant with required parameters such as temperature and age, it returns the displacement vector calculated by considering all four main equation terms - wind fields, diffusion, source movement and buoyancy, as well as their corresponding weights.

10 Key Algorithms

Here we present some key algorithms we implemented that were essential to our proposed method. Because of limited size of this paper we could not mention all the interesting methods, so we have selected the most important ones.

10.1 Control points re-sampling

In each time step we need to re-sample the control point positions according to a Catmull-Rom spline associated with the given segment. This algorithm, described in Algorithm 1, takes place in the *CatmullRomSpline* class, which represents the spline of one *SplineSegment*. If the *SplineSegment* has n control points, the corresponding *CatmullRomSpline* contains a linked list of $n+1$ *CatmullRomSplineSegment* instances.

Algorithm 1 Re-Sampling of the control points

```
  spline = new CatmullRomSpline(SplineEndPoints, ControlPoints)
  newSegmentLength = spline.Length/CatmullRomSplineSegments.Count
  height = 0
  for i=0 to CatmullRomSplineSegments.Count-2 do
    height += currentSegment.Length
    newPosition = spline.getPointAtHeight(height)
    newControlPointPositions.Add(newPosition)
  end for
  return  return newControlPointPoisitions
```

10.2 Box-Müller transformation

We use Box-Müller transformation principle in determining the separation height of the spline. Using two uniformly distributed random variables we can return a random variable with given mean and standard deviation, shown in Algorithm 2.

Algorithm 2 Box-Müller transformation

```
Require: mean, standardDeviation
  $r_1$ = uniform random in range [0,1]
  $r_2$ = uniform random in range [0,1]
  randNormal = $\sqrt{-2 \log r_1} \sin 2\pi r_2$
  return  mean+standardDeviation*randNormal
```

10.3 Wind-Fields transformation

Each wind-field affects the position of the particle in cylindrical coordinates. Algorithm 3 described here presents general algorithm depicting the necessary transformations for circular wind-fields.

Algorithm 3 Wind-Fields transformation

```
Require: position
  positionInWFcoords = position - windfield.origin
  cylindricalPos = CartesianToCylindrical(positionInWFcoords)
  cylindricalDisplacement = $\Delta t$ wf.getDisplacementVector
  newCylindricalPos = cylindricalPos + cylindricalDisplacement
  newCartesianInWFPos = CylindricalToCartesian(newCylindricalPos)
  return  (newCartesianInWFPos+origin) - position
```

10.4 Simulation step calculation

Simulation algorithm, Algorithm 4, takes place in *SplineCalculator* class. The input parameter is a *SplineParticle* instance. The method returns new velocity for the given particle.

Algorithm 4 Simulation step calculation

```
Require: particle
  displacement = (0,0,0)
  for all windfield in windFields do
    displacement += $\alpha$ windfield.getDisplacement(particle.position)
  end for
  displacement += $\beta$ normalizedRandomVector*particle.Temperature
  displacement += $\gamma$ particle.SourceVelocity
  displacement += $\delta\left(-\beta_t g_y\,(T_0 - \text{particle.temperature})\,\text{particle.age}^2\Delta t\right)$
  return displacement
```

10.5 Integration approximation

Here we cover our approximation of the integration needed in testing whether to separate spline at a given height. In the final release this method was replaced with double exponential transformation from MathNet library because it provided better calculation times.

The following algorithm depicted in Algorithm 5 takes *min* and *max* as input parameters specifying the range at which we want to calculate the integration. *stepsize* is also among input parameters. Step size is directly proportional to computation error and inversely proportional to time complexity. *func(x)* is the desired function we wish to integrate.

Algorithm 5 Integration approximation

```
Require: min; max; stepSize, funct(x)
  result = 0
  for step = min; min <max; step+= stepSize do
    result += funct(step) * timeStep;
  end for
  return result
```

10.6 Spline space mapping

The Algorithm 6 is used in each time step to map the *FireParticle* instances into the space of the structural spline curve. It also sets the correct values for the particles age and temperature based on their height on the spline. The input parameters are position of the particle in normalized 3D space profile and spline to which we wish to map the position.

11 Results

The implemented method was tested on a notebook with Intel Core i3 350m processor 2.26GHz (2 cores), 4Gb of RAM and an ATI Radeon 5145 graphics card with 512Mb of memory. The real-time simulation ran above 30 frames per second with one emitter holding one spline. The spline had up to 15 spline segments active and testing 100 particles per segment. Increased particle or segment count led to a rapid drop in frames per second. The method proved controllable and usable up to 100000 particles per segment, although when populating the scene with that many particles, the application FPS count started to incline towards fps of applications using complex numerical solutions and offline rendering methods. One of the main goals of this work was to give a wide range of controls over the

Algorithm 6 Spline space mapping

```
Require: position, spline
  cylindrical = CartesianToCylindrical(position)
  particle = spline.getParticleAtHeight(cylindrical.Z*spline.Length)
  nextStep = min(cylindrical.Z+0.001, 1)
  particle2 = spline.getParticleAtHeight(nextStep*spline.Length)
  newZVector = normalize(particle2.position - particle.position)
  unitX = (1,0,0)
  unitY = (0,1,0)
  helpingVector = (newZVector == UnitX) ? UnitY : UnitX
  newXVector = normalize(cross(newZVector, newZVector+helpingVector))
  radialDisplacement = (newXVector*spline.radius)*cylindrical.r
  radialDisplacement = radialDisplacement * CreateRotationMatrixFromAxisAngle(newZVector,
  cylindrical.$\theta$)
  particle.position += radialDisplacement;
  return particle
```

flame behavior and appearance. The fire simulations are controlled via a multitude of parameters and even a slight change can produce different physically based and artist looking results. In this section we provide a few examples of achieving different goals with our simulation.

Of course, to obtain a realistic interaction like most 3-dimensional techniques one can interact with the fire (in our case in real-time). For instance, it is possible to put an object in the flame, and let it catch fire by igniting new starting point of the spline skeleton. Since we are using a particle model, we do not need to perform collision detection, which is typically highly time consuming. Instead, we represent the interacting object as a local force field, which of course can change the shape and branch or split the flames by branching or splitting the skeleton. The user can set the wind field, the gravity, and the parameters for internal forces to control the strength of connectivity of the spline nodes, e.g. flame persistent, intermittent and buoyant regions, etc. All parameters can be modulated in real-time, and all simulations run in real-time.

11.1 Dominating the movement with wind-fields

Should we want to boost the effect of wind-fields, we simply increase the wind-field weight α in Eq. 1. In our application, we found that in order to see this effect in a bigger scope, often prolonging the spline is needed. As the buoyancy term is exponentially proportional to the flame age, it easily overpowers the wind-fields effect in the main equation. If we want to model loops of fire or bend the flame towards one point using wind-fields, we need to suppress the buoyancy term in Eq. 1. Values in the range $\delta \in [0, 0.1]$ for buoyancy term proved usable, see Fig. 11.

11.2 Smoke simulation

Due to amortization shader, the colder the particles are, the darker texture they render in our billboarding visualization. We can achieve this effect by increasing the cooling rate, see Fig. 12. The side effects of this cooling are that the flame temperature can more quickly drop to the levels of ambient temperature, at which the buoyancy term of the main equation entirely loses its effect.

11.3 Prolonging the spline

Most common requirements for the flame behavior would be to prolong its length. With flame separation disabled, increasing the initial age and decreasing the aging rate is sufficient to increase the length of the flame. However, the separation is often needed, and it complicates achieving what might at first seem like a simple task. While the long enough lifespan is needed, the length of the spline comes into play. As the separated segments maximum lifespan gets modified and scaled by

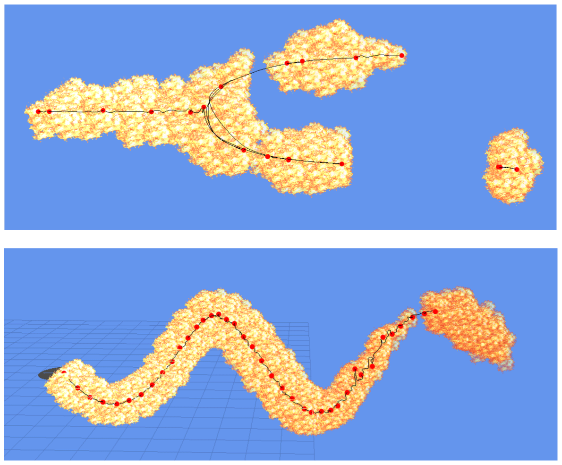

Figure 11: Direct control using wind-fields. Top image we see a uniformed wind field colliding with source wind-field. Bottom image is combination of uniform and vortex wind-fields.

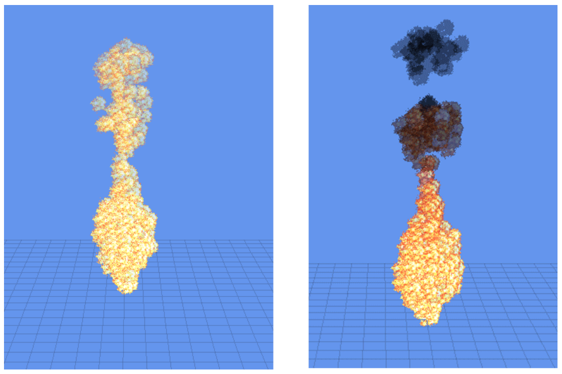

Figure 12: Example of darkening by increased cooling. Left image with cooling set to $c = -0.5$. Right, cooling $c = -2.0$, see Eq. 3.

a cubed uniform random variable in the range $[0, 1]$, once the separation occurs, the flames are short-lived and thus the flame appears shorter. One way of fixing this issue is changing the lengths describing the persistent and intermittent height, as well as specifying the maximum length at which the separation occurs with 100% probability. The other way is increasing the apparent length of the flame by suppressing the diffusion factor. This effectively flattens the Catmull-Rom spline in between the segment end points, thus providing a shorter segment and in turn increasing the number of segments which fall under the persistent and intermittent flame regions.

12 Conclusion

We have proposed a system that models flame dynamics for computer animation by improvement of the structural modeling method. We have showed that it is possible to create fire animations, using a pipeline in which the style of automatic in-betweens is derived from hand-drawn key-frames as fire skeletons and the fire motion is derived from a simulation. One can generate a complex animation of flames with the system assigning several parameters such as weights α, β, γ, and δ which corresponds to the wind field, diffusion, source motion, and buoyancy terms respectively. The desired effect can be achieved by multiple paths and different configurations, which can produce artistic side-effects. The motion of flames and smoke can be easily controlled according to scenarios defined by flame skeletons, skeleton branching, and wind forces.

The paper also shows how flame dynamics changes when one assigns different values to those parameters with several pictures the system generates. The spread of fire can be simulated in full 3D particle-based fluid dynamics in real time. Export to external rendering software such as Maya is beneficial for a more complex scene. In the future work we need to focus on optimization and parallelization.

References

[1] M. Chládek and R. Ďurikovič, "Particle-based shallow water simulation for irregular and sparse simulation domains," *Comput. Graph.*, vol. 53, no. PB, pp. 170–176, Dec. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2015.04.002

[2] M. Browning, C. Barnes, S. Ritter, and A. Finkelstein, "Stylized keyframe animation of fluid simulations," in *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*, ser. NPAR '14. New York, NY, USA: ACM, 2014, pp. 63–70. [Online]. Available: http://doi.acm.org/10.1145/2630397.2630406

[3] P. Witting, "Computational fluid dynamics in a traditional animation environment," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 129–136. [Online]. Available: http://dx.doi.org/10.1145/311535.311549

[4] T. Ishikawa, R. Miyazaki, Y. Dobashi, and T. Nishita, "Visual simulation of spreading fire," in *Proceedings of the NICOGRAPH International.* Tokyo: NICOGRAPH International Press, 2005, pp. 43–48.

[5] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen, "Physically based modeling and animation of fire," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 721–728, Jul. 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566643

[6] Z. Melek and J. Keyser, "Interactive simulation of burning objects," in *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.*, Oct 2003, pp. 462–466.

[7] N. Chiba, K. Muraoka, H. Takahashi, and M. Miura, "Twodimensional visual simulation of flames, smoke and the spread of fire," *The Journal of Visualization and Computer Animation*, vol. 5, no. 1, pp. 37–53. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/vis.4340050104

[8] X. Wei, W. Li, K. Mueller, and A. Kaufman, "Simulating fire with texture splats," in *IEEE Visualization, 2002. VIS 2002.*, Nov 2002, pp. 227–234.

[9] D. Park, S. Woo, M. Jo, and D. Lee, "An interactive fire animation on a mobile environment," in *2008 International Conference on Multimedia and Ubiquitous Engineering (mue 2008)*, April 2008, pp. 170–175.

[10] P. Beaudoin, S. Paquet, and P. Poulin, "Realistic and controllable fire simulation," in *Proceedings of Graphics Interface 2001*, ser. GI '01. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2001, pp. 159–166. [Online]. Available: http://dl.acm.org/citation.cfm?id=780986.781006

[11] M. Balci and H. Foroosh, "Real-time 3d fire simulation using a spring-mass model," in *2006 12th International Multi-Media Modelling Conference*, 2006, pp. 8 pp.–.

[12] A. Lamorlette and N. Foster, "Structural modeling of flames for a production environment," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 729–735. [Online]. Available: http://doi.acm.org/10.1145/566570.566644

[13] P. Shipkov, "SOup plugin," http://www.soup-dev.com/, 2011, [Online; accessed 21-April-2015].

[14] T. Kim, N. Thürey, D. James, and M. Gross, "Wavelet turbulence for fluid simulation," in *ACM SIGGRAPH 2008 Papers*, ser. SIGGRAPH '08. New York, NY, USA: ACM, 2008, pp. 50:1–50:6. [Online]. Available: http://doi.acm.org/10.1145/1399504.1360649

[15] ChaosGroup, "PhoenixFD plugin," http://www.chaosgroup.com/, 2012, [Online; accessed 21-April-2015].

[16] M. Seymour, M. Walin, and J. Diamond, "The VFX show," http://www.fxguide.com/thevfxshow/the-vfx-show-177-the-hobbit-the-desolation-of-smaug/, 2013, [Online; accessed 21-April-2015].

[17] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graph. Models Image Process.*, vol. 58, no. 5, pp. 471–483, Sep. 1996. [Online]. Available: http://dx.doi.org/10.1006/gmip.1996.0039

[18] J. Stam and E. Fiume, "Depicting fire and other gaseous phenomena using diffusion processes," in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 129–136. [Online]. Available: http://doi.acm.org/10.1145/218380.218430

[19] J. Wejchert and D. Haumann, "Animation aerodynamics," in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '91. New York, NY, USA: ACM, 1991, pp. 19–22. [Online]. Available: http://doi.acm.org/10.1145/122718.122719

[20] D. Drysdale, *Intro to Fire Dynamics*. Wiley, 2 2001. [Online]. Available: http://amazon.com/o/ASIN/0471972908/