COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

DRAGON FIRE MODELING AND SIMULATION IN 3D FOR COMPUTER ANIMATION

Diploma thesis

2015 Bc. Jozef Hladký

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

DRAGON FIRE MODELING AND SIMULATION IN 3D FOR COMPUTER ANIMATION

Diploma thesis

Registration number:	ce4faed8-df61-40da-aed9-15ae20583610
Study programme:	Applied informatics
Branch of study:	2511 Applied informatics
Educational facility:	Department of Applied Informatics
Supervisor:	prof. RNDr. Roman Ďurikovič, PhD.

Bratislava, 2015 Bc. Jozef Hladký

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jozef Hladký
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Dragon fire modeling and simulation in 3D for computer animation
Modelovanie a 3D simulácia dračieho ohňa pre potreby počítačovej animácie

Cieľ: Naštudovať si model dynamiky ohňa a diferenciálne rovnice, ktoré ho popisujú. Naštudovať si model pre simuláciu vetra. Implementovať časticový systém a sily v C
. Implementovať vizualizáciu a nástroje meniace správanie modelu ohňa. Časticový systém by mal umožňovať interaktívnu manipuláciu pomocou vonkajších síl.

Literatúra: SIGGRAPH 2002 and 2012 zborníky

Vedúci: prof. RNDr. Roman Ďurikovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, PhD.

Dátum zadania: 03.12.2013

Dátum schválenia: 03.12.2013 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I hereby declare that all parts of this thesis have been written by myself using only the references explicitly referred to in the text and consultations with my supervisor.

. . . . . .

Bratislava, 2015 Jozef Hladký

Acknowledgements

I would like to thank my advisor prof. RNDr. Roman Ďurikovič, PhD.for his guidance and invaluable advice throughout my work on this thesis. I would also like to thank my colleagues from YACGS seminar for useful tips regarding design and implementation of the method. Finally, I want to thank my family and friends for their support and encouragement during my studies.

Abstrakt

V tejto práci predstavujeme systém pre modelovanie dynamiky ohňa s dôrazom na realistické správanie a šírenie ako aj na rozsiahly systém ovládacích prvkov. Náš systém ponúka širokú škálu parametrických a procedurálnych ovládacích prvkov. Šírenie ohňa a jeho pohyb je dosiahnutý pomocou diferenciálnych rovníc ktoré berú do úvahy vietor, vztlakovú silu, pohyb horiaceho povrchu ako aj difúzne aspekty. Realistické správanie je dosiahnuté použitím stochastických modelov nadnášania a blikania ohňa. V práci tiež implementujeme veterné polia umožňujúce prídavný kontrolný procedurálny pohyb. Správanie ohňa pokrýva pohybujúce sa zdroje, blikanie, trhanie a spájanie plameňa.

Kľúčové slová: oheň, plameň, animačné systémy, časticové systémy, fyzikálne modelovanie, veterné polia

Abstract

In this paper we present system for modeling fire dynamics with emphasis on realistic behavior and spread as well as extensive controls system. Our system provides a wide range of parametric and procedural controls. Flame spread and motion is achieved using differential equations which take account of wind, buoyancy, diffusion and velocity of the burning surface. Realistic behavior is achieved using stochastic models of flickering and buoyant diffusion. We also implement wind fields for additional controllable motion. Flame behavior covers moving sources, flickering, separation and merging.

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

Contents

Chapter 1

Introduction

Due to its dramatic nature fire is one of the most demanded elements in 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 real world. It is a difficult and complicated task, as no one has ever seen how a dragon fire looks like. People can imagine that dragon fire looks similar to the flamethrower, but this is rarely the case, as most artists want to make dragon fire easily distinguishable from real fire and make it somewhat special. Due to 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. Imagination has no limits. The flame often needs to react with its surroundings and with other elements like water or wind. But most of all, it needs to amuse the viewer. 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.

In this thesis 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.

First, we provide 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 as of year 2015.

Next, we cover in depth the Structural Modeling method presented by [LF02]. We cover the structure of the basic elements of the flame as well as the differential equation describing the dynamics of the whole flame. We also explain the stochastic elements of the flame separation and flickering as well as buoyant forces.

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 as well as our own of visualization methods.

In the implementation chapter we present the fundamental structures essential to implementing our method and also provide pseudocode of key algorithms.

In results we provide some examples of how to achieve different goals with our model. We conclude this thesis with proposing few ideas for future work on this topic.

Chapter 2

Overview

In this chapter we present and summarize some of the existing methods and tools used to model fire for the purposes of 3D animation. These methods are among the most frequently used nowadays, used in various feature films and TV series as well as in the video games industry.

2.1 Autodesk Maya

Autodesk Maya is a 3D computer graphics software used by professionals in various fields of animation, be it video games, animated films, TV series or some other 3D application. By default, it provides 2 different ways of modeling fire - the Fluids method and the Dynamics method. Both are particle-based and provide control over the fire's appearance and behavior. However these methods are not the only ones available in this software. The functionality of Maya can be extended by various plugins and some of them offer cutting-edge fire visuals (e.g. fumeFx).

Both Dynamics and Fluids are based on emitting particles into space. The user can specify various fields to simulate forces like gravity or wind, add turbulence detail or specify rendering details. The software provides plenty of controls over every aspect of the fire's appearance or behavior. Finding a balance between all these variables and make the flame suit our

needs is a very demanding task. [Aut14]

2.1.1 Dynamics

Maya dynamics can be used to create effects like steam, fire or rain. Fire-wise this method is best suitable for burning objects, objects leaving a trail or fireworks explosions. In the case of burning objects, the whole surface acts as an emitter. The motion of the flame itself isn't as realistic as in the Fluid method, however this method provides much faster computing of simulation steps and rendering.

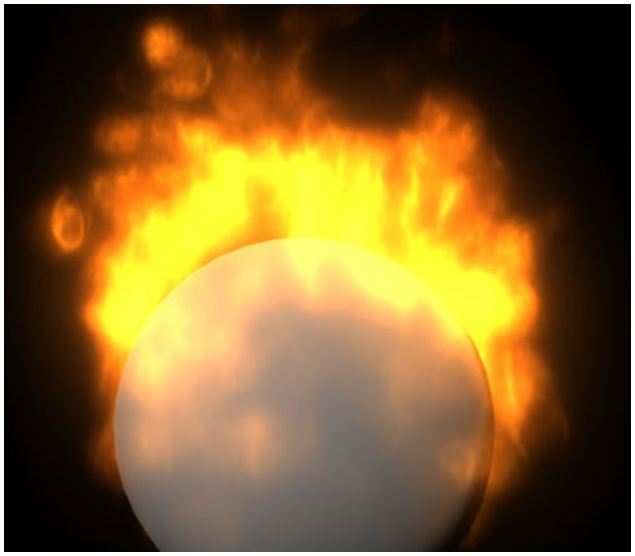

Figure 2.1: A burning sphere we created using Maya Dynamics method and rendered in mentalray.

2.1.2 Fluids method

This method can be used to create clouds, mist, fog, steam, smoke, fire, molten lava or ocean surface. In the scope of fire, this method is oriented mainly on creating realistic flame spread. It is best used to create fiery flamethrower-like projectiles, bonfires, explosions and nuclear blasts.

Contrast to Dynamics method, the fluid can spread only in a container, which must also contain the emitter. The motion of the fluid at each time step is simulated using solvers for the Navier-Stokes fluid dynamics equations.

The extra data needed to define such fluid effect may slow the simulation exponentially, because more calculations are needed at every step. For this reason Maya provides non-dynamic fluid effects, in which the flame appearance is created using textures and the flame motion is achieved by keyframing texture attributes.

For another school project we created animation of a dragon destroying a city. Our result can be seen on figure 2.2. Base grid resolution is 150x150, emitting 500 particles per second. To better illustrate the complexity of such task - the simulation time was 22 minutes, rendering of one frame in 1920x1080 resolution using mental ray on hexa-core AMD Phenom II X6 1090T 3.2GHz was 82 minutes.

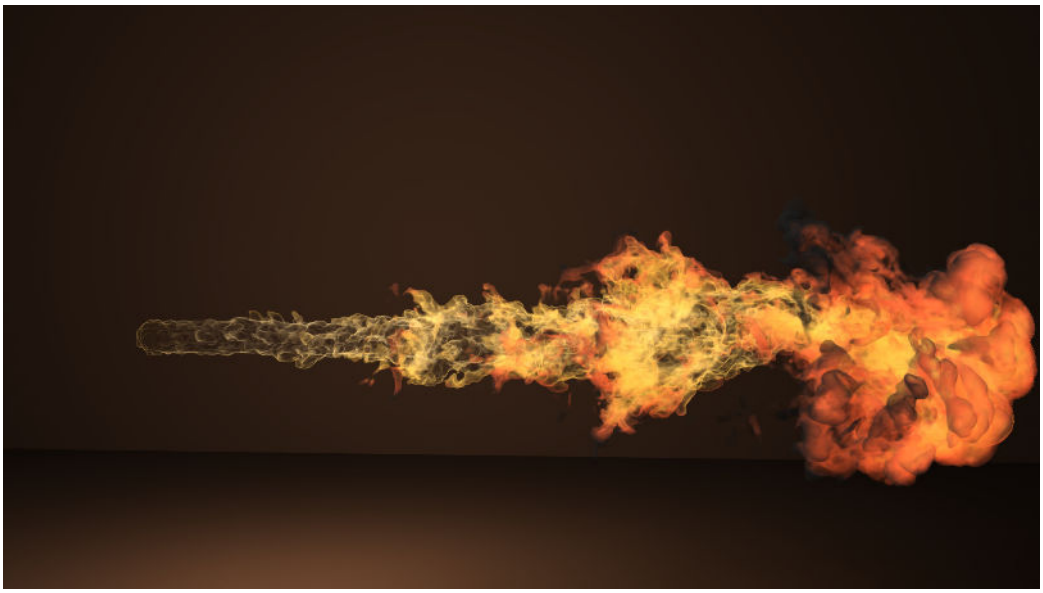

Figure 2.2: A flamethrower projectile we created in Maya using Fluid method and rendered in Mentalray renderer.

2.2 3D animation software plugins

The following list is a brief overview of some plug-ins aimed for fire modeling in 3D computer animation software. Most of them are compatible with at least one of the most used 3D animation software - Maya, 3Ds Max or Cinema4D.

2.2.1 AfterBurn

AfterBurn is an older plugin for creating volumetric effects. It's usage lies mainly in the movie and video games industry of the 2000's. It was used in motion pictures like Dracula 2000, Armageddon and Matrix Reloaded and video games like WarCraft 3 and StarCraft.

The method is based on building volumetric effects around the center of each particle. Each particle carries plethora of attributes (age, color, temperature, etc...), which can change over time of the animation using interpolation controllers. The usability and speed of the work flow are enhanced introducing the Animation Flow Curves (AFC). This tool allows a clear plot-like overview of how different attributes of the particle change over time. This method also introduces wind fields (which they called Daemons) that can change the combustion direction so that the flame appears as progressing in one direction, swirling around a vortex or explosion in all directions. [d.o06]

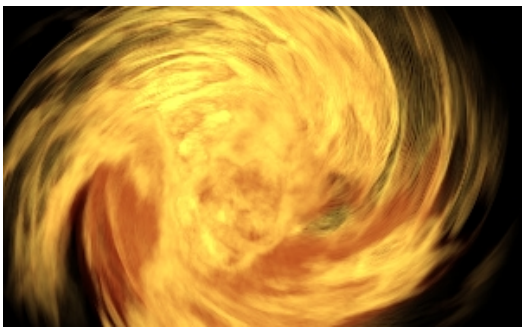

Figure 2.3: Example of vortex daemon wind field in AfterBurn.

2.2.2 FumeFx

FumeFx plugin can achieve one of the most realistic looking smoke and fire effects. On the top it also offers various improvements of workflow like GPU accelerated preview or multithreaded simulation. Developed by Sitnisati, this plugin is available for Maya and 3DsMax. This method was used for multiple award-winning feature films like The Avengers, Thor or Iron Man. [d.o13]

Figure 2.4: An example of flames created using FumeFx plugin. Pictures are from movies Ghost Rider and War Thunder, respectively.

2.2.3 TurbulenceFD

Developed by Jawset and available for Cinema4D and LightWave 3D (There is also a 2D version for Adobe After Effects). This system is based on solving the incompressible Navier Stokes equations. It utilizes a voxel grid to describe volumetric clouds of particles. The equations solutions describe the motion of fuid on the grid. Artist can *paint* sources of emission on any geometric object or particle system. In addition to fully tweakable fire shader the plugin provides a preset realistic fire shader based on the Black Body Radiation model. [Jaw09]

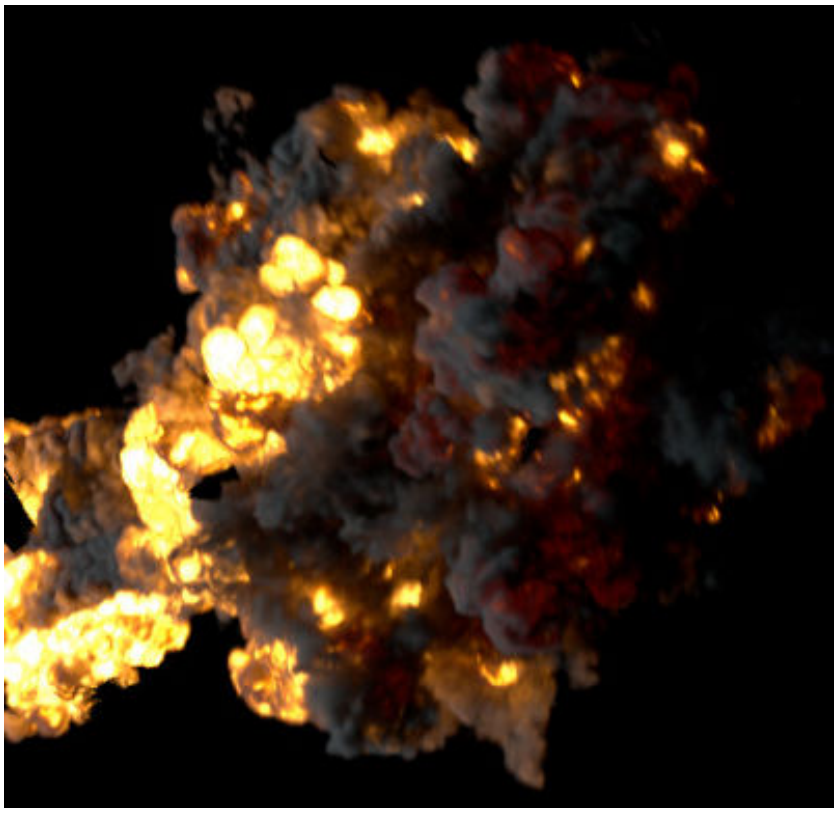

Figure 2.5: An example of a flame created using TurbulenceFD plugin. [Jaw09]

2.2.4 SOup upresNode

Animators often want to adjust the flame's appearance in lower resolutions because it provides faster turnaround and after achieving desired behavior they want to increase the resolution to improve detail for the final rendering. SOup is a set of plugins that extend the procedural capabilities of Maya software. The upresNode plugin eliminates one major drawback of Maya's fluids - that is, if you change the resolution of the grid the fluid behaves differently. Because of computational complexity, the tweaking of the flame's appearance in high resolution can be very tedious. 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 [KTJG08]. [ea11]

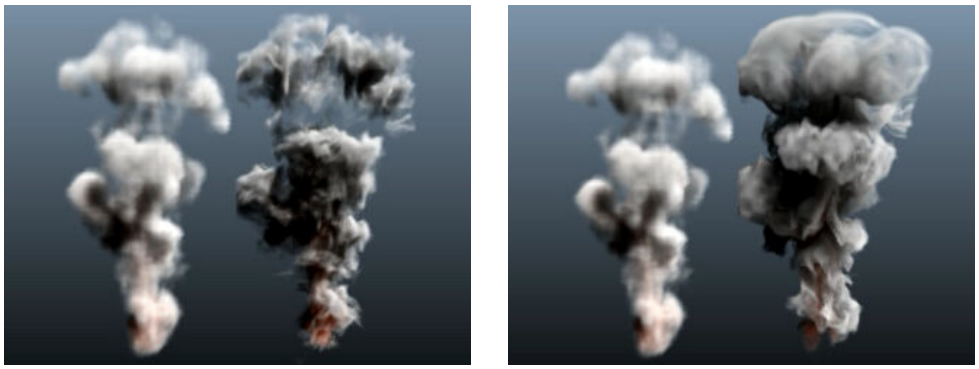

Figure 2.6: An example of increasing the resolution of the grid using the SOup upresNode method utilizing Wavelet turbulence algorithm. [KTJG08]

2.2.5 PhoenixFD

PhoenixFD 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 and is optimized to offer flexibility and speed even at very high particle

count. Fully utilizable with V-Ray renderer it offers proper refraction on liquids. [gro12]

Figure 2.7: Nuclear explosion created with PhoenixFD.

2.3 Shrek (2001)

This method was developed by DreamWorks and first appeared in the 3D animated movie Shrek (2001). The system's main focus is on efficiency and complete control over visual appearance and behavior. The flame is based on parametric space curves that evolve over time according to multiple procedural, hand-defined and physics-based wind fields. Physical properties are based on statistical measurements of natural diffusion flames. Around these splines is built an implicit surface with cylindrical profile. In this region the particles are point-sampled using volumetric falloff function. [LF02]

Figure 2.8: Screenshot from Shrek (2001). The flame was modeled using [LF02]. In this screenshot the flame evolves towards camera.

2.4 The Hobbit (2014)

The most complex animation simulation up to date was done by Weta Digital for the feature film Hobbit: The battle of the five armies. In this completely digital scene, a dragon destroys a wooden city situated on a lake. Combined simulations are used to achieve the resulting appearance, consisting of air, fire, water and rigid bodies simulations. Air flows and wind fields are modeled as volumes in which every piece of falling debris triggers wind movement, turbulence and pressure change. The dragon's fiery breath consists of emitted particles which act in a fluid stream moving in a water-like simulation. This fluid then serves as a fuel for the fire itself, which creates viscous appearance of the fire and napalm-like spread in the environment. [MS13]

Figure 2.9: Screenshot from the destruction scene in The Hobbit: Battle of the Five Armies (2014)

.

Chapter 3

Structural modeling of flames

3.1 Method overview

Here we provide an in-depth review of the state-of-the-art method for modeling fire presented by [LF02] and used in the motion picture Shrek in the year 2001. This method constructs a flame animating system with emphasis not only on realism, but also on artistic appearance. It is based on particle system with appropriate and effective particle control. This system provides a range of behavioral controls suitable for artistic animation. Realistic appearance is achieved using stochastic models of flickering and buoyant diffusion. Implementation of wind fields provides additional procedural control.Flame behavior includes moving sources, flickering, separation and merging, combustion spread and interaction with stationary objects.

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 $O(n^3)$ [FM96] [SF95]. 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 desired visual effect with numerical simulation, as even a small change in starting conditions can provide drastically different results.

The Structural modeling method provides different approach. It utilizes

the fact, that most of the visual aspects of the flame can be statistically modeled. By separating the statistic-based visual aspects of the flame, we are left with a set of structural elements which the animator can directly control by specified parameters. The statistical properties of the flame were measured on real flames. The model contains also numerous large-scale procedural controls such as wind, diffusion, fuel combustion or convection. These controls affect the local behavior of the flame particles. We will briefly cover the essential components of the model and then focus on the structure, particle creation and dynamics.

3.2 Essential components

This system utilizes different statistically-controlled or directly controlled elements. The whole model which can be divided into 8 components:

1. 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 B-Spline curve. This curve defines the spine of the flame and is the main actor in the shape and behavior of the flame. As exact structure was not mentioned in [LF02], we in our proposed method decided to further divide the curve into smaller structure elements, which we will cover later in chapter 4. Although one curve is sufficient for defining one flame, multiple curves can act in a scene. When these curves collide with each other, the curve is not affected at all. However, when the curves are close to each other, the flames these curves define appear as if they were merging into one.

2. The splines evolve through space in time. Each point of the curve carries data of the flame's 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. The model also covers realistic adaptation of the flame to the movement of the source.

3. 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 which can be also user-controlled. Along with the heuristic defining the breakaway height, the limitation of the separated spline lifespan and the rate of aging are based on engineering observations of real flames.

4. The region where actual particles of the flame are created is build using a cylindrical profile rotating around the central spline of the flame. The created particles gain the properties of the respective segments of the spline that spawned them. This region represents the visible part of the flame and provides a boundary with the oxidation region.

5. The first level of procedural noise is applied to particles created within the flame region. This 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 structural elements. This noise is not based on any empirical observations, so Flow Noise is recommended, as it provided good visuals.

6. Transformation into parametric space of the structural curve is applied to the particles. Then 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.

8. We define procedural controls to account for position, intensity, lifespan, shape, color, size, evolution and behavior of the flames.

These 8 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 levels when desired, compared to numerical simulation approaches.

We do not cover the last 4 stages in detail because the main focus of this thesis is on the dynamics and structure of the flame.

3.3 Base Curve

Each central spline defining one flame is essentially a B-Spline curve with control points being the particles in the center of the flame. It is the fundamental structure for the whole flame, as it affects the appearance and overall shape of the whole flame.

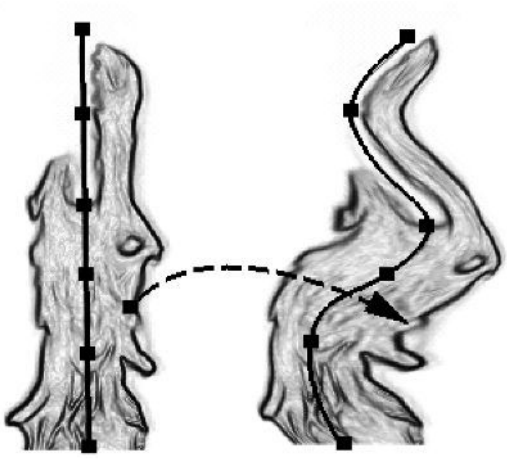

Figure 3.1: Spline movement impact on the shape of the whole flame. [LF02]

3.4 Dynamics

The particles in this model advance according to differential dynamics model describing a combination of physical terms as well as hand-defined procedural

wind fields. The differential equation is as follows:

$$\frac{\partial x_p}{\partial t} = w(x_p, t) + d(T_p) + V_p + c(T_p, t) \tag{3.1}$$

where 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 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

$$c(T_p, t) = -\beta_t g_y (T_0 - T_p) t_p^2 \tag{3.2}$$

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 cannot 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 sufficient to create realistic reaction of the flame to a moving source. You can see examples in figure 3.2.

3.5 Evolution of the flame

The flame moves according to combination of forces, some of which can be user-specified (like the procedural physics-based wind-fields) and some of them are statistical in nature and based on observation of natural diffusion

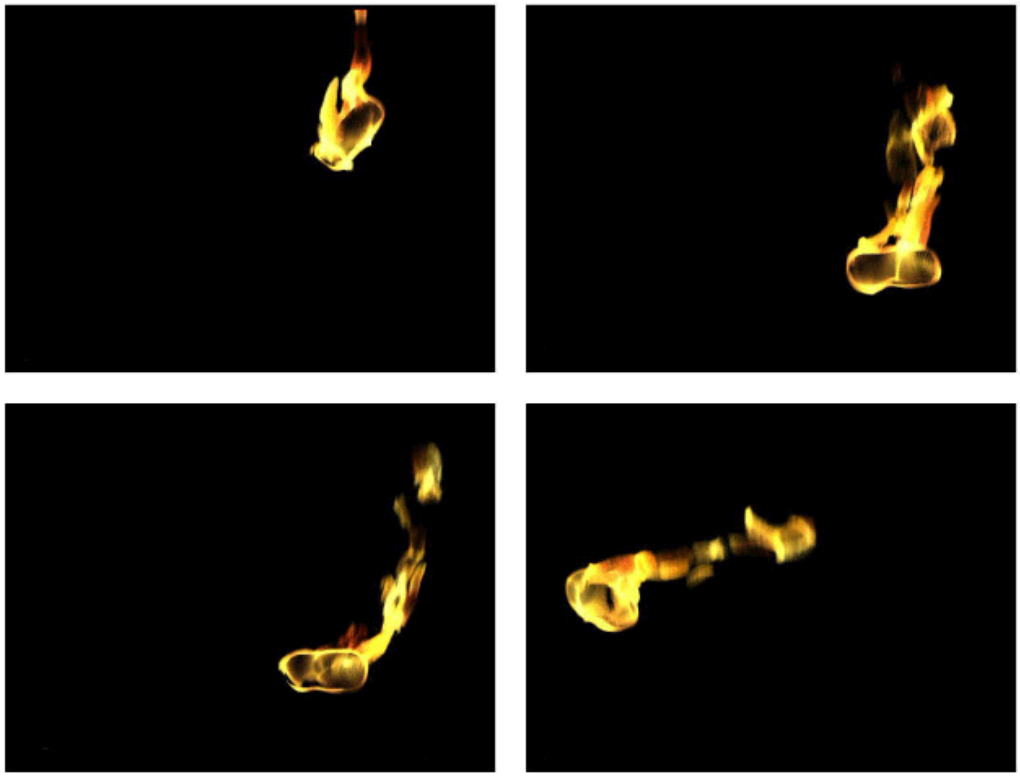

Figure 3.2: Torch waved through the air. A demonstration of flame adaptation to a moving source. [LF02]

flames. The motion is also influenced by hand-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 at the burning surface.

2. In the second frame of the animation, particle p_0 is released into the environment, where it moves according to equation 3.1 using explicit Euler integration method. [LF02] also mention that Runge-Kutta method for solving differential equations is sufficient and 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 at 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 move according to equation 3.1. The interpolating B-Spline is then fitted to pass through all of 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 predefined height, the spline can separate.

3.6 Separation

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. The first region will be the persistent flame region, it's 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 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 func-

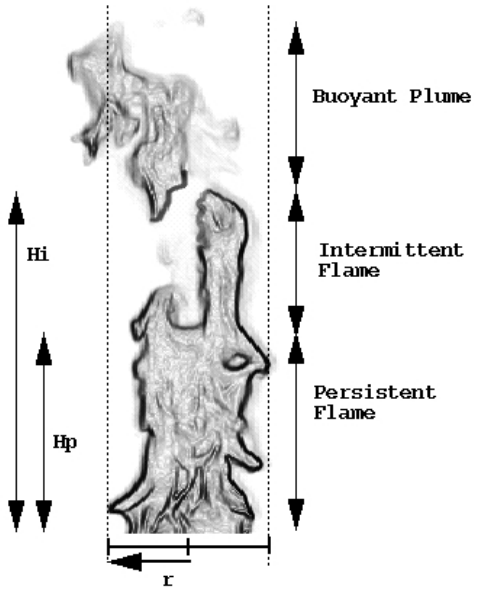

Figure 3.3: Illustration of persistent, intermittent and buoyant regions of the flame. [LF02]

tion

$$D(h) = \frac{1}{\sqrt{2\pi}(H_i - H_p)/2} \int_{-\infty}^{h} e^{-\left(h-\frac{|V_c|}{f}\right)/\left(2((H_i-H_p)/2)^2\right)} dh \qquad (3.3)$$

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 persistent flame region, $|V_c|$ is the average velocity of the structural control points. f is the aproximate breakaway rate in Hz. According to the observations by [Dry01] $f = (0.50 \pm 0.04)(g_y/2r)^{1/2}$ 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.

When we decide to separate the flame a portion of the spline is cutoff from the top of the flame. This portion ranges from the top to a randomly chosen point below. The distribution for this selection is not based on any observations. Normal distribution $\mathcal{N}(\mu, \sigma^2)$ proved sufficient with mean μ

and standard deviation σ chosen as:

$$\mu = H_p + (H_i - H_p)/2 \tag{3.4}$$

$$\sigma = (H_i - H_p)/4 \tag{3.5}$$

The separated segment of the spline is not re-sampled back to n control points as this prevents additional local detail appearing in the separated segment. To account for lack of accurate way of fuel content determination, the particles in the spline are given a limited life-span 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 i^3 most of the breakaway flames have very short life-span.

3.7 Profiles

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 photograph and creates a rotationally symmetric surface. In this 3D space fire particles are point-sampled and transformed into the spline structural curve. Two levels of procedural noise are added and then the particles are at their correct positions and are ready to be passed into the rendering stage.

Figure 3.4: Illustration of movement of the base splines and their impact on the overall look of the flame. [LF02]

Chapter 4

Proposed method

We propose a method that creates the basic structure and dynamics for a controllable 3D fire effects. 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 Structural modeling of flames conceived by Arnauld Lamorlette and Nick Foster [LF02]. As the paper described this method in outlines and in general was short on details, we have created our own structure, chosen the wind fields implementation and heuristics. Parameters allowing greater control were added whenever reasonable.

4.1 Weighted dynamics model

We extend the Structural modeling dynamics differential equation to following form

$$\frac{\partial x_p}{\partial t} = \alpha w(x_p, t) + \beta d(T_p) + \gamma V_p + \delta c(T_p, t) \tag{4.1}$$

where α, β, γ and δ are weights 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 additional

controls that help us shape the behavior of the flame to our needs.

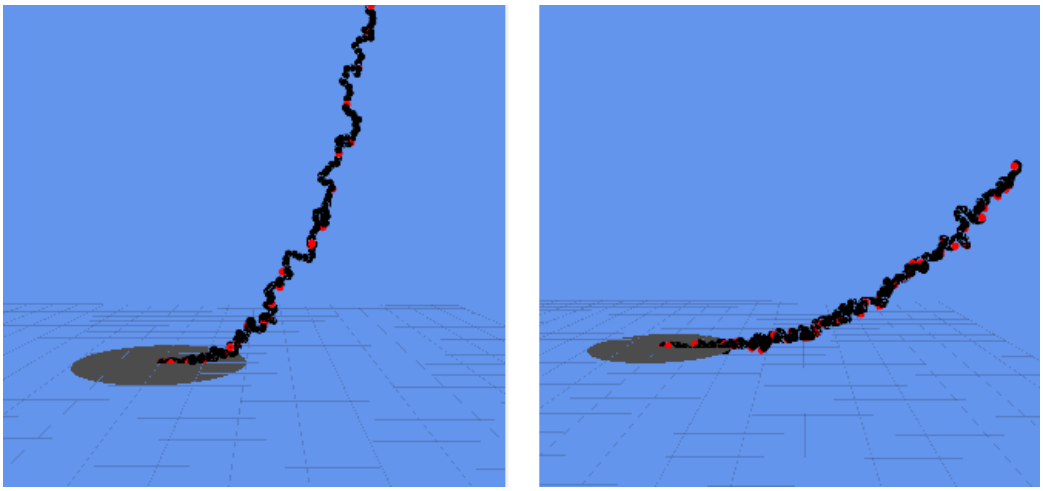

Figure 4.1: Example of modifying the parameters of equation 4.1. In the left $\delta = 1$, in the right $\delta = 0.1$. This scene has uniform wind field which blows to the right.

4.2 Amortization

The [LF02] paper was not describing any heuristic for the amortization of particle attributes. We have decided to use linear techniques of temperature decay

$$T_{t+1} = max(T_t - c\Delta t, T_0) \tag{4.2}$$

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

Similarly we defined the age amortization as

$$t_{t+1} = t_t + a\Delta t \tag{4.3}$$

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

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.

4.3 Structural elements

As the method described the structure very loosely, we created our own structures, their hierarchy and attributes. The center spline forming the spine of the flame still plays major role in the flame appearance and has the greatest impact on visuals and dynamics.

4.3.1 Emitter

The flame creation is handled by an emitter. It is the most fundamental element as it carries important parameters required for creation of the Base splines of the flames. The scene can contain 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.

Table 4.1: Table of emitter attributes

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

4.3.2 Base spline

The Base spline is emitted from emitter and consists of spline segments. At every frame, new segment is created at the bottom of the spline and according to random distribution presented in section 3.6. One or more segments can partly separate and form a new spline. The length L of the base spline s is

$$L = \sum_{i=0}^{n-1} l_{s_i} \tag{4.4}$$

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

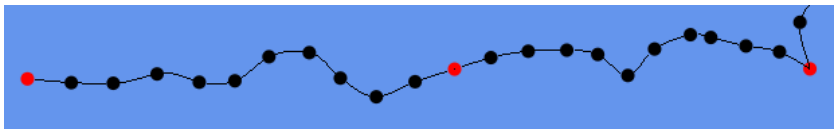

Figure 4.2: Our visualization of Base spline element. Red points are end points separating the segments, black points are control points and the spline is interpolating Catmull-Rom spline.

4.3.3 Base Spline segments

Each spline forms a spine for one individual flame. The individual splines consist of s segments. Each segment consists of a start point p_s, end point

p_e and n control points $cp_0 \dots cp_{n-1}$. 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 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 point, 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 equation 4.7

4.3.4 NURB as a Spline segment

It is not exactly stated in [LF02] what properties should the B-Spline curve have. The first version of our implementation contained a Non-uniform Rational B-Spline as it offered controls superior to general B-Spline. We used the general form of NURBs curve

$$C(u) = \frac{\sum_{i=1}^{k} N_{i,n} w_i p_i}{\sum_{i=1}^{k} N_{i,n} w_i} \tag{4.5}$$

where k is the number of control points p, w_i are their corresponding weights and $N_{i,n}$ are the recursively-defined basis functions of a NURB spline.

The intention was to control the visuals through the knot vectors, weighted control points and degree of the NURBs. However in the later phases where re-sampling of the control point in each frame is required (see 4.4), the NURB under-performed due to its lack of ability to pass through all the control points and the whole spline tended to flatten more and more during the simulation.

In the end we've decided to utilize Catmull-Rom spline, as it proved sufficient results while allowing us to pass through each control point.

4.3.5 Catmull-Rom spline as Spline segment

We define the Catmull-Rom spline as

$$p(t) = \left(1, t, t^2, t^3\right) \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{pmatrix} \begin{pmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{pmatrix} \tag{4.6}$$

where t is the interpolation parameter, τ is the tension and $p_{i-2} \dots p_{i+1}$ are points defining the spline. The spline itself is drawn only between points p_{i-1} and p_i, points p_{i-2} 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. We define the length of the spline as

$$L = \sum_{t=0}^{n-2} \left\| p\left(\frac{t}{n-1}\right) - p\left(\frac{t+1}{n-1}\right) \right\| \tag{4.7}$$

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

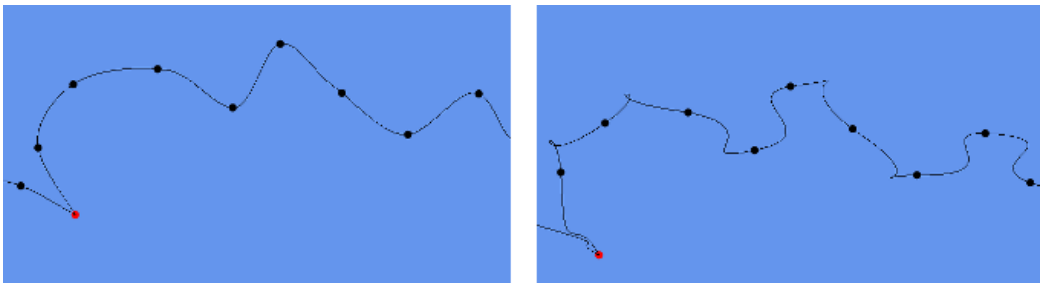

Figure 4.3: Example of different tension values for the Catmull-Rom spline. On the left $\tau = 0.5$, on the right $\tau = 2$.

4.4 Evolution of the flame

In this section we present detailed description of the modified evolution of flame in detail:

1. We can have one or multiple flame emitters. Each emitter has it's position p, radius r, velocity v and b, which is the number of Base splines it will produce concurrently. Following steps are the same for each emitter in the scene.

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 containing 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 B-Spline.

3. In the second time step frame of the animation, particle p_0 is released into the environment, where it moves according to 4.1 and is amortized with the methods proposed in 4.2. 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. The separation and flickering occurs as in described in section 3.6.

4. 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 from the structural elements on the corresponding level of the spline.

4.5 Wind Fields

We model our wind fields based on the method proposed by [WH91]. We define 4 types of wind-fields - Uniform, Sink, Source and Vortex. 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.

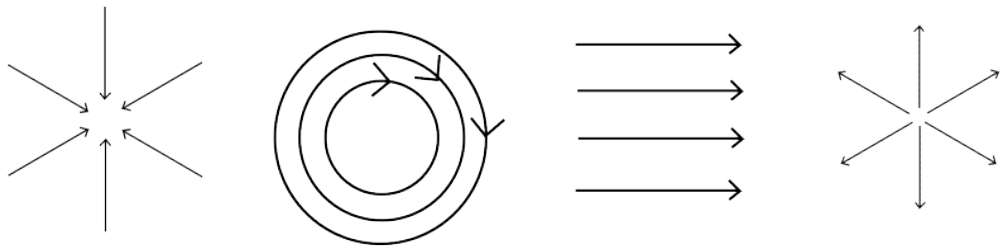

Figure 4.4: Schematic describing the sink, vortex, uniform and source types of the wind fields.

For the Uniform wind-field it is sufficient to specify only the direction and strength. However for the circular types this 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

$$v_r = \frac{s}{2\pi r}\Delta t; \qquad v_\theta = 0; \qquad v_z = 0; \tag{4.8}$$

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 size of our chosen time step. The Sink wind-field produces the same displacement vector as Source wind-field, except the stength 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 Vortex wind-field placed at origin with strength s is then given by

$$v_r = 0; \qquad v_\theta = \frac{s}{2\pi r}\Delta t; \qquad v_z = 0; \tag{4.9}$$

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 size of our chosen time step.

The resulting displacement vector $w(x_p, t)$ of equation 4.1 is the sum over each wind-field in the system.

$$w(x_p, t) = \sum_{i=0}^{n} v_i = v_{vort}(x_p, t) + v_{sink}(x_p, t) + v_{source}(x_p, t) + \cdots \tag{4.10}$$

4.6 Separation and Flickering

The separation stage of the flame is modeled as presented in section 3.6. We approximate the integral in the probability function in equation 3.3 using following method

$$\int_{-\infty}^{h} e^{-\left(h-\frac{|V_c|}{f}\right)/\left(2((H_i-H_p)/2)^2\right)} dh = \frac{h-g}{N}\sum_{i=g}^{h} e^{-\left(i-\frac{|V_c|}{f}\right)/\left(2((H_i-H_p)/2)^2\right)} \tag{4.11}$$

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 select the height of the transformation, we model the Gaussian distribution as Box-Müller Transformation. Given variables x_1 and x_2 which are

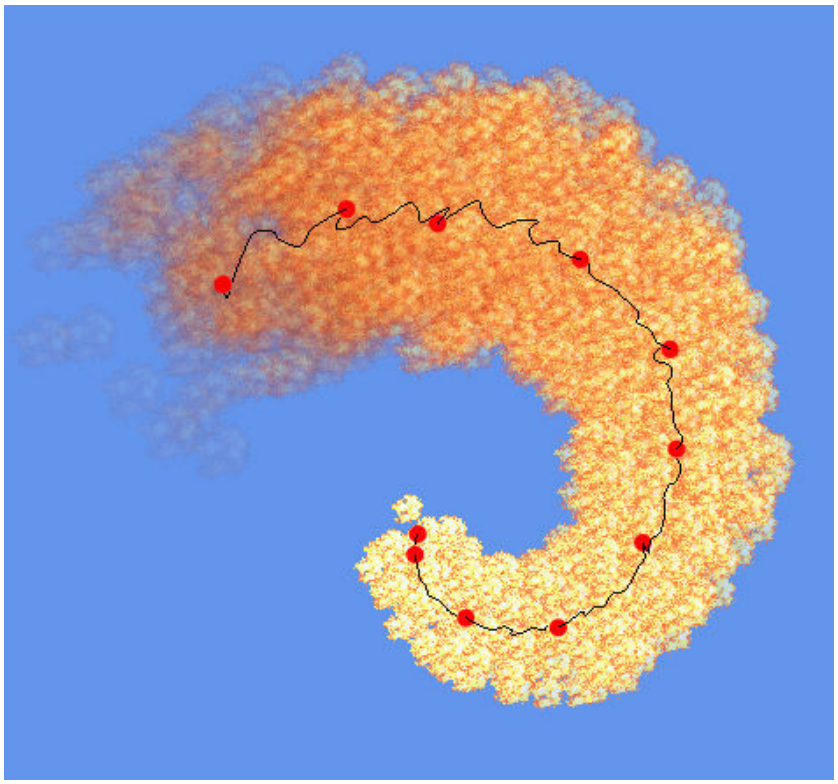

Figure 4.5: Example of flame behavior in combination of Vortex and Source wind-fields.

uniformly and independently distributed between 0 and 1, we define z_1 and z_2 as

$$z_1 = \sqrt{-2 \ln x_1} \cos(2\pi x_2) \tag{4.12}$$

$$z_2 = \sqrt{-2 \ln x_1} \sin(2\pi x_2) \tag{4.13}$$

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

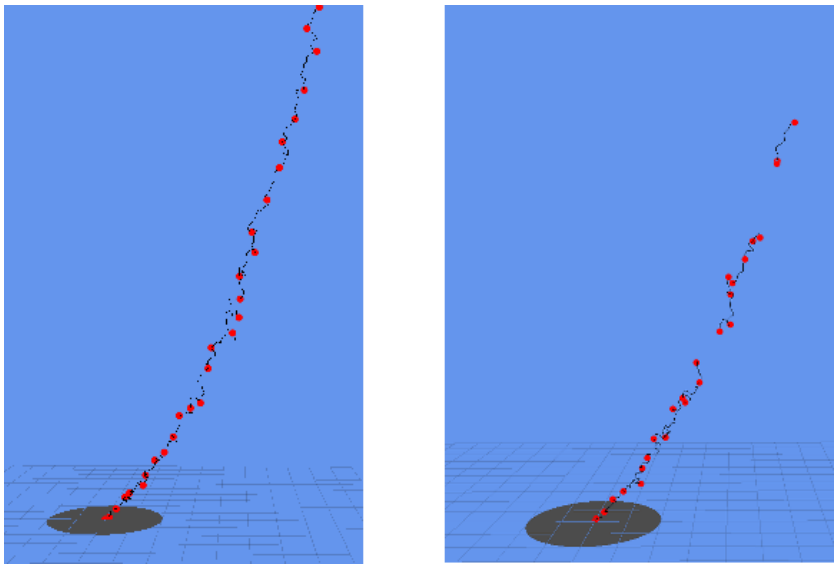

Figure 4.6: Example of spline separation. Evolution of the same spline with the same settings and wind fields. In the left, the separation is disabled, in the right it is enabled.

4.7 Flame profile generation

After specifying the 2D normalized flame profile, we create a normalized 3D cube where we rotate the 2D profile around the Y axis and thus achieve a symmetric space in which we can create particles. The particles' positions are randomly selected using uniform distribution and then tested against the symmetric 3D profile whether they are accepted or rejected. The number of accepted samples depends on the shape of the specified 2D profile. For most of the used profiles, around one half of the samples were rejected.

4.8 Visualization of the particles

As the rendering part of the [LF02] technique was quite complex and out of the scope of this thesis, we have decided to implement simplified particle rendering using billboards. We are rendering each particle as a rectangle

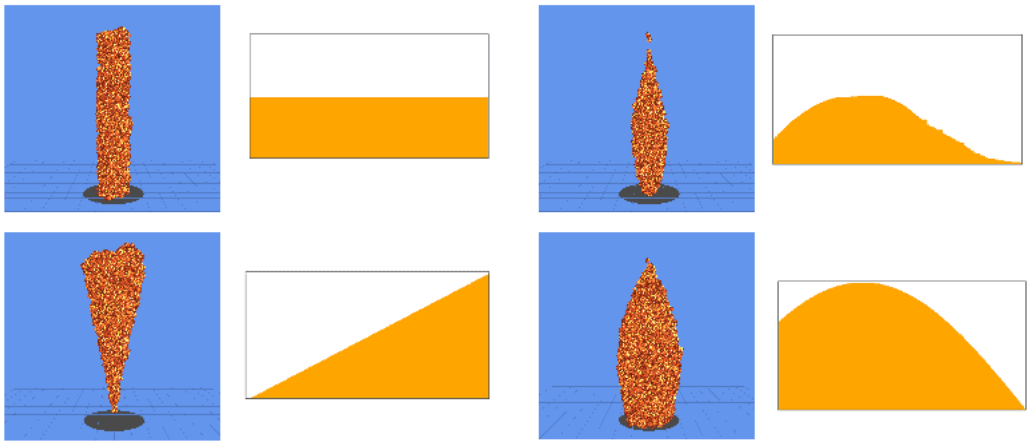

Figure 4.7: Examples of different profiles and their effect on the shape of the flame. The amortization shader is turned off.

which always faces the camera and is painted with miniature flame particle texture.

Since we have all the attributes of the painted particle at our disposal, we modify the color of each particle according to its age and temperature. The temperature of the particle is affecting red, green and blue channels of the particle color, while age determines the value of alpha channel responsible for particle transparency. The amount of darkening of the particle is based on the color of the texture, but for most textures that we tried we found suitable this method:

$$c_p = c_p \frac{T_p}{\frac{T_i}{4} - T_0}$$

where c_p is the color of particle p, T_p is the temperature of the particle p, T_i is the initial temperature which is each particle assigned at the moment of creation at the base of each emitter and T_0 is the ambient temperature.

Next we change the transparency of the particle. We use

$$a_p = 1 - \frac{t_p}{t_{max}}$$

where a_p is the transparency of particle p, t_p is age of particle p and t_{max} is

the maximal age at which particles die.

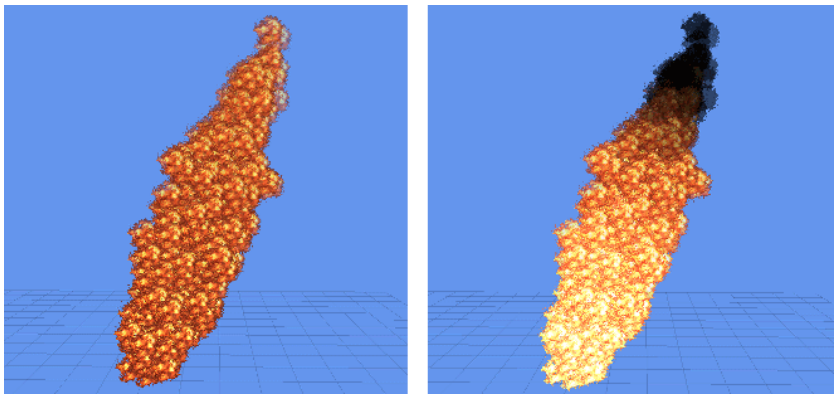

Figure 4.8: The effect of amortization shader.

Chapter 5

Implementation

Our method is implemented in C# and developed on operation system Windows 10 Technical Preview. The compiler is .NET 4.5. IDE used is Microsoft Visual Studio 2013 Ultimate. 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:

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

5.1 Flame representation

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.

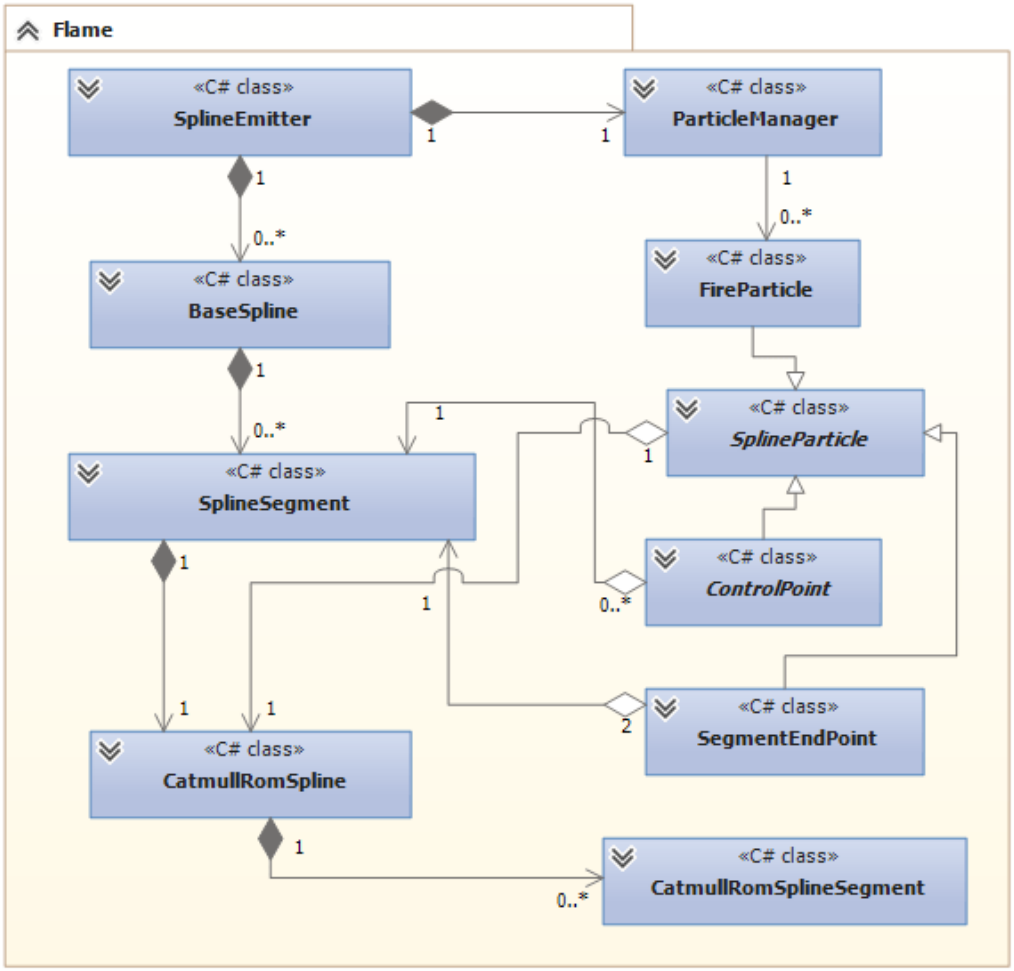

Figure 5.1: Class diagram depicting representation of the flame in our structure.

5.1.1 Spline structure

Because the 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.

Smallest 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 inehritance. 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 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 equation 4.6 For details, see section 4.3.5.

Spline Segment

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 *CatmullRomSplineSegment* class.

Spline segment contains 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 is responsible for re-sampling of the control points.

Base spline

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 required to draw points and polylines in OpenGL are implemented.

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

Spline Emitter

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.

Particle manager

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 *ParticleManager*. By utilizing 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.

5.2 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.

5.2.1 Settings and parameters

In our application the setting of the constants and parameters are implemented as a singleton classes having private static fields representing each parameter. Each field is encapsulated with non-static C# getter and setter to allow binding to controls in the main form.

Following is a table of the parameters we can change through the user interface to change the behavior of the flame. This is only a digest of the flame-controlling parameters, as these classes contain more parameters which are not related to the flame.

Table 5.1: Fire-controlling attributes in class *MainMovementEquation*

Notation	**Type**	**Description**
α	$float$	Wind-Field term weight
β	$float$	Diffusion term weight
γ	$float$	Source velocity term weight
δ	$float$	Buoyancy term weight
β_t	$float$	Thermal expansion coeficient
T_0	$float$	Ambient temperature
Δt	$float$	Time step size

Table 5.2: Fire-controlling attributes in class *SplineSettings*

Notation	**Type**	**Description**
sh_{alpha}	*bool*	Activated transparency in amortization shader
sh_{dark}	*bool*	Activated darkening in amortization shader
speed	*int*	Simulation speed in ms
growth	*int*	Simulation steps needed to create a new segment
$S_{sampling}$	*int*	Catmull Rom sampling density
fire density	*int*	Number of Fire Particles per segment tested in rejection sampling
start age	*float*	Initial age of newly created particles
A	*float*	Life time of separated segments.
τ	*float*	Tension of the Catmull-Rom splines.
starttemp	*float*	Temperature of newly created particles
cooling	*float*	Temperature amortization per one time step
aging	*float*	Life amortization per one time step
age_{max}	*float*	Maximal life-time of particles
CP_{count}	*int*	Number of control points per one *SplineSegment*
H_p	*float*	Height of the persistent flame stage
H_i	*float*	Height of the intermittent flame stage
H_{max}	*float*	Height after which the flame always separates

5.2.2 Wind-fields

As in particle implementation, in wind-fields 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.

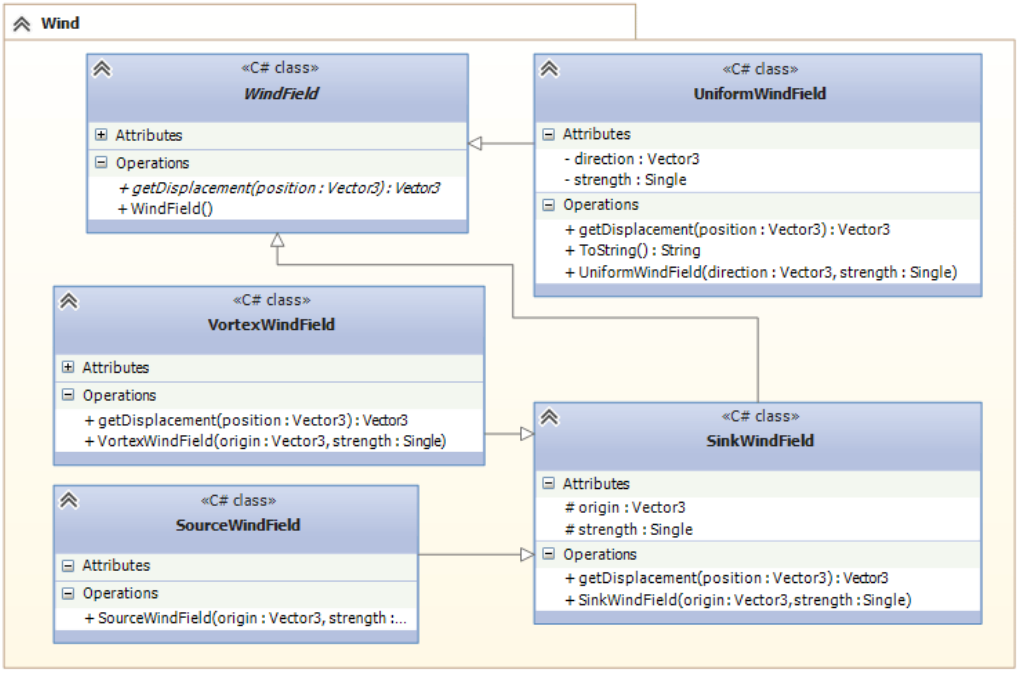

Figure 5.2: Class diagram depicting representation of the wind in our structure.

5.2.3 Simulation step calculation

SplineCalculator is a singleton class responsible for calculation of the main differential equation in each time step. It also contains a reference to a list containing 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 taking into account all four main equation terms - wind fields, diffusion, source movement and buoyancy, as well as their corresponding weights.

5.3 Key algorithms

Here we present some key algorithms we implemented that were essential to our proposed method. Because of limited size of this thesis we could not mention all the interesting methods, so we have selected the most important ones. You can review the rest in the source code, which is attached to this thesis.

5.3.1 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 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 5.1 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
```

5.3.2 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.

Algorithm 5.2 Box-Müller transformation

Input: 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

5.3.3 Wind-Fields transformation

Each wind-field affects the position of the particle in cylindrical coordinates. Here we present general algorithm depicting the necessary transformations for circular wind-fields.

Algorithm 5.3 Wind-Fields transformation

Input: position

positionInWFcoords = position - windfield.origin

cylindricalPos = CartesianToCylindrical(positionInWFcoords)

cylindricalDisplacement = Δt wf.getDisplacementVector

newCylindricalPos = cylindricalPos + cylindricalDisplacement

newCartesianInWFPos = CylindricalToCartesian(newCylindricalPos)

return (newCartesianInWFPos+origin) - position

5.3.4 Simulation step calculation

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

5.3.5 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 re-

Algorithm 5.4 Simulation step calculation

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

placed with double exponential transformation from MathNet library because it provided better calculation times.

The following algorithm takes *min* and *max* as input parameters specifying the range at which we want to calculate the integration. *stepsize* is are 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 5.5 Integration approximation

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

5.3.6 Spline space mapping

The 5.6 algorithm 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.

Algorithm 5.6 Spline space mapping

Input: 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 * CreateRotationMatrixFro-
mAxisAngle(newZVector, cylindrical.θ)
particle.position += radialDisplacement;
return particle
```

5.4 GUI

We provide user interface covering a wide range of controls. The controls are logically divided based into 5 panels arranged as tabs in a *TabControl* instance. Thanks to implementing the settings classes as singletons with public encapsulated fields we can directly bind the specified fields to their GUI elements. The logical values are represented via checkboxes, numerical parameters are utilizing numericUpDown controls.

The GUI also contains controls allowing adding and removing wind fields and emitters.

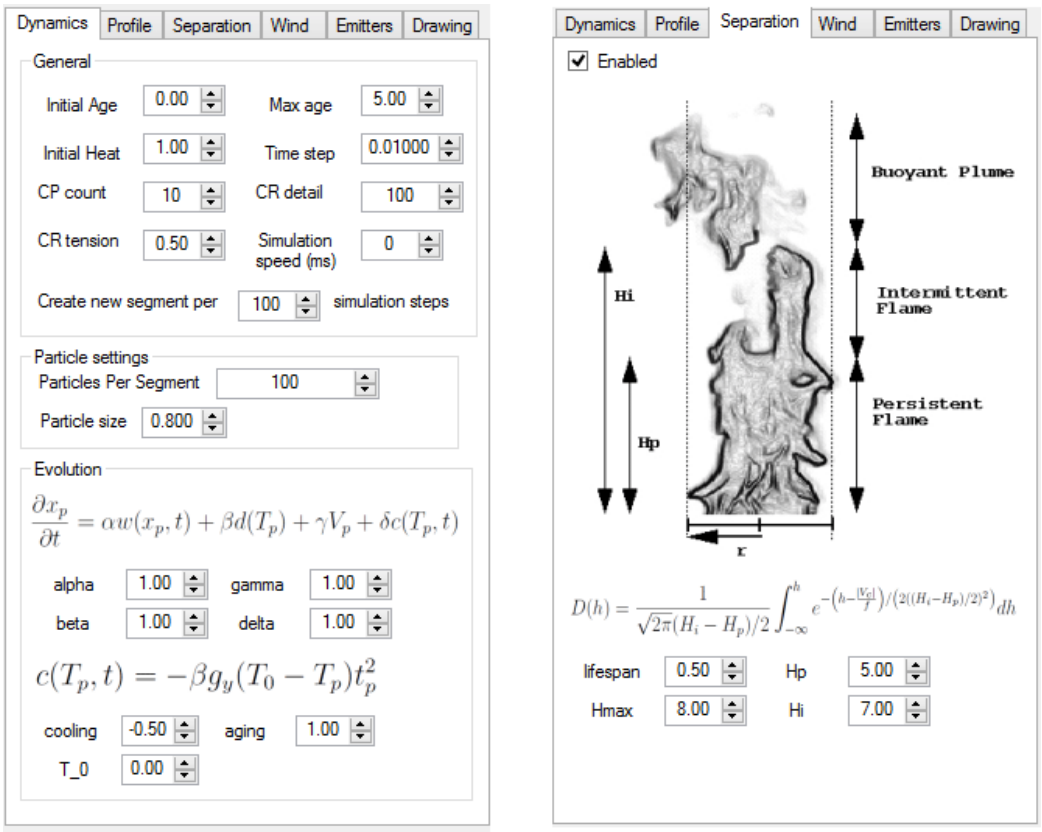

Figure 5.3: Example of GUI elements for controlling the dynamics and separation of the flame.

Chapter 6

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 simulation ran above 30 frames per second with one emitter containing one spline. The spline contained 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 100 000 particles per segment, although when populating the scene with this many particles, the application FPS count started to incline towards fps of applications using complex numerical solutions and offline rendering methods.

6.1 Direct control methods

One of the main goals of this work was to provide wide range of controls over the flame behavior and appearance. In professional applications the fire simulations are controlled via multitude of parameters and even slight change can produce drastically different results. Our application shares this feature. In this section we provide a few examples of achieving different goals with our simulation. Some of these results can be achieved by different settings, each of which produces different side-effects. This feature is also present in

professional methods.

6.1.1 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 the main differential equation. If we would like to boost only one specific wind-field, we increase its strength parameter. In our application we found that in order to see this effect in a bigger scope, often prolonging the spline is needed. For examples on this topic, see section 6.1.3. 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. Values in the range [0, 0.1] for buoyancy term δ proved usable.

6.1.2 Smoke simulation

Due to amortization shader, the colder the particles are, the darker texture they render in our billboarding visualization. If we want to boost the darkening the tip of the flame, we need to ensure that these particles are colder. Since the amortization shader takes account of the temperature span of the particles, reducing the initial temperature is not the right way to do this. We can achieve this effect by increasing the cooling rate. 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.

6.1.3 Prolonging the spline

Most common requirement for the flame behavior would be to prolong it's length. In order to achieve this, the user must first understand when are the end segments of the flame destroyed. Basically, the main precursor for destroying the end segments is their current age affected by life-span parameters

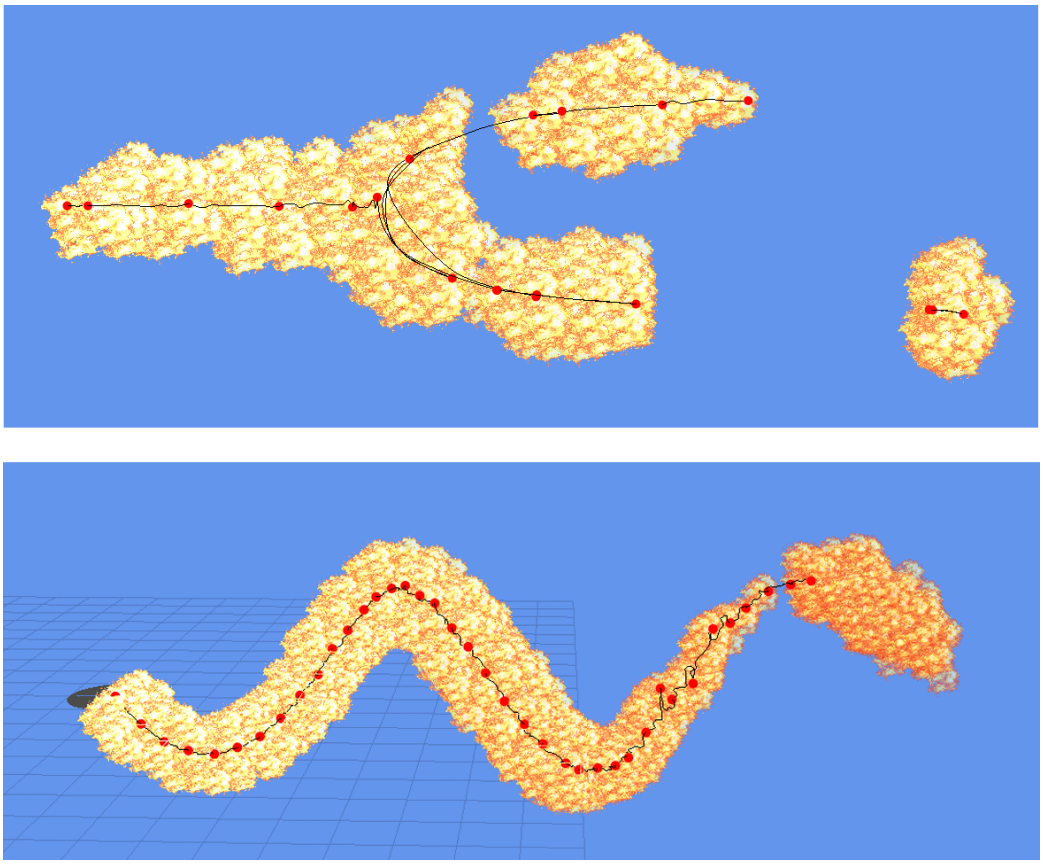

Figure 6.1: Direct control using wind-fields. In the first image we see uniform wind field colliding with source wind-field. The second image is combination of uniform and vortex wind-fields. Both examples have diffusion and buoyant terms suppressed to ≈ 0.1

- initial age and aging rate. 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 sufficient life-span is required, the length of the spline comes into play. As the separated segments maximum life-span gets modified and scaled by 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

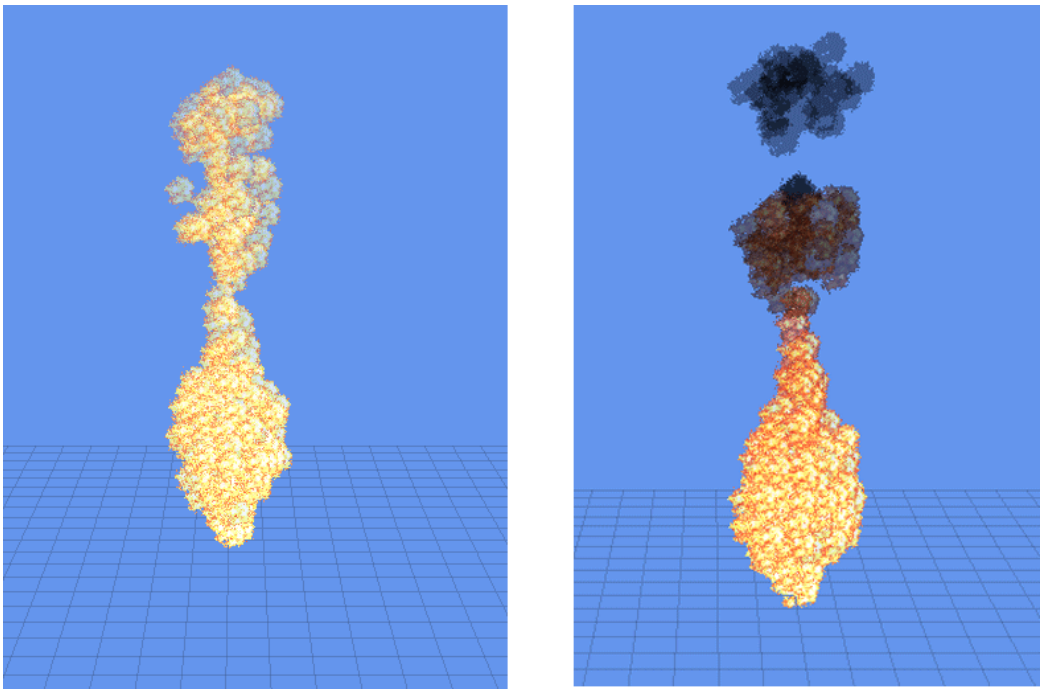

Figure 6.2: Example of darkening by increased cooling. The first image has cooling set to -0.5, the second has cooling -2.0

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 amount of segments which fall under the persistent and intermittent flame regions.

6.1.4 Artifacts

With many parameters and many ways to control our model, many ways to break the model emerge. While we tried to restrict the inputs for the parameters in a safe manner, the number of different parameter combinations allow for unpredicted behavior. One particularly interesting effect comes with

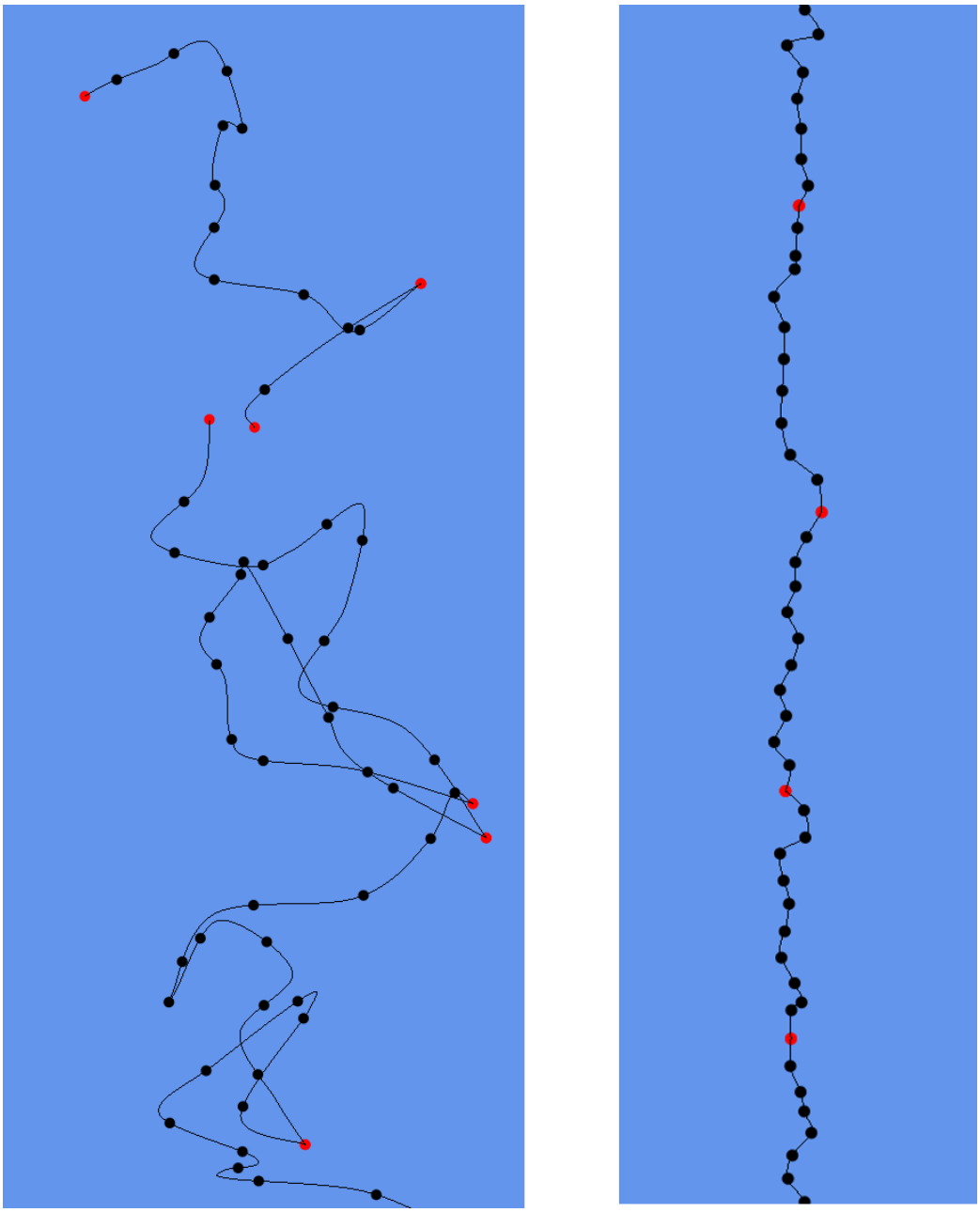

Figure 6.3: Example of flattening the spline by reducing the diffusion term β. The left image has $\beta = 1$, in the second $\beta = 0.1$

the Catmull-Rom tension $\tau > 1$. In this case the control points are quickly

displaced around the whole scene, making the Catmull-Rom spline stretch to great lengths and create random shapes. While by restricting the spline length we can produce a chaotic fireball effect, prolonging the spline with uniform profile provides a tentacles-from-hell like appearance.

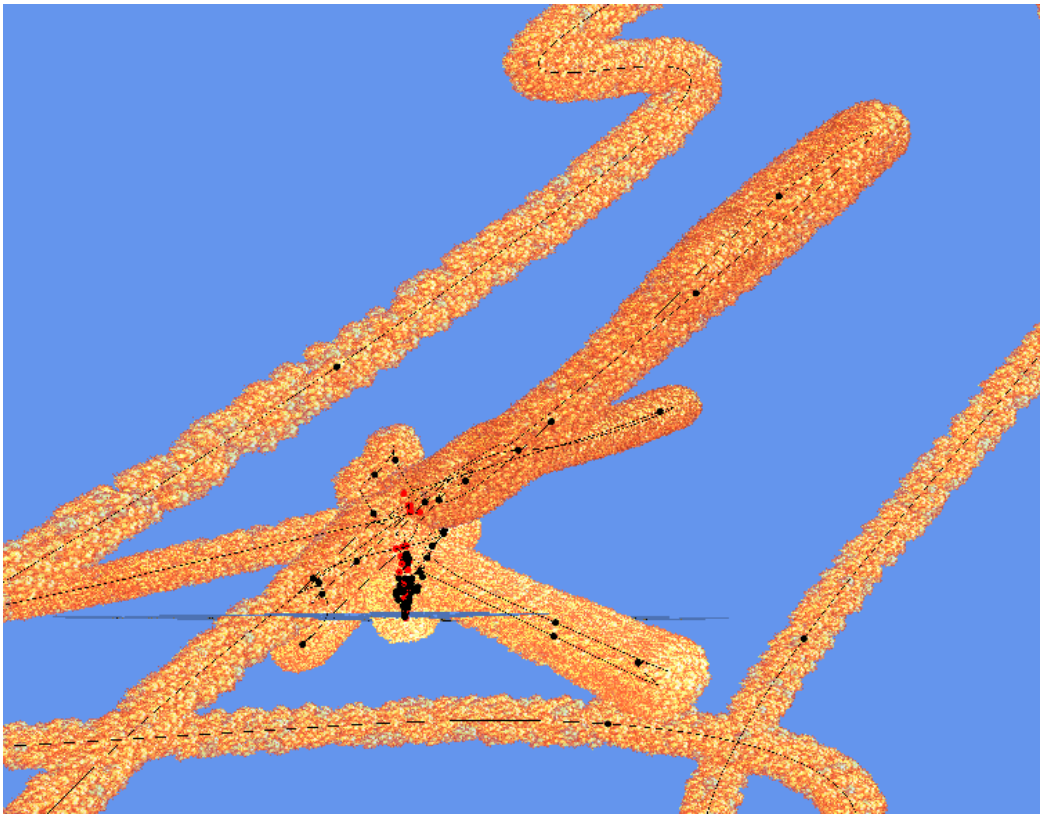

Figure 6.4: The tentacles of hell achieved by setting Catmull-Rom tension $\tau = 2$ and setting the particles tested against each segment to 100 000. During this simulation the FPS was essentially zero

Chapter 7

Conclusion

In this thesis we have presented a method for modeling dynamics of fire. We have presented an improvement of the structural modeling method and implemented usable application with a wide range of user controls for altering the behavior and visualization of the flame. The desired effect can be achieved by multiple paths and different configurations, which can produce different side-effects. This principle is present across all state-of-the-art flame modeling software. Because of this complexity, complete control over the flame requires at least a shallow understanding of dynamics principles presented in this thesis.

In the future work we need to enhance the generation of fire particles with noise and displacement to produce more realistic flame profiles. Wind fields could be enhanced to allow for more complex wind flows changing over time. Collision with objects in the scene needs to be added. Export to external rendering software such as Maya could prove beneficial. More complex visualization and high-quality offline rendering would also be useful. There is always space for optimization and parallelization.

Bibliography

[Aut14] Autodesk. Maya 2015 user guide. `http://help.autodesk.com/view/MAYAUL/2015/ENU/`, 2014. [Online; accessed 21-April-2015].

[d.o06] Sitni Sati d.o.o. AfterBurn plugin. `https://www.afterworks.com/AfterBurn.asp?ID=2`, 2006. [Online; accessed 21-April-2015].

[d.o13] Sitni Sati d.o.o. FumeFx plugin. `https://www.afterworks.com/FumeFX_Maya/Overview.asp?ID=1`, 2013. [Online; accessed 21-April-2015].

[Dry01] Dougal Drysdale. *Intro to Fire Dynamics*. Wiley, 2 2001.

[ea11] Peter Shipkov et al. SOup plugin. `http://www.soup-dev.com/examples2_1.htm`, 2011. [Online; accessed 21-April-2015].

[FM96] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483, September 1996.

[gro12] Chaos group. PhoenixFD plugin. `http://www.chaosgroup.com/en/2/phoenix_maya.html`, 2012. [Online; accessed 21-April-2015].

[Jaw09] Jawset. TurbulenceFD plugin. `https://www.jawset.com/`, 2009. [Online; accessed 21-April-2015].

[KTJG08] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. In *ACM SIGGRAPH*

2008 Papers, SIGGRAPH '08, pages 50:1–50:6, New York, NY, USA, 2008. ACM.

[LF02] Arnauld Lamorlette and Nick Foster. Structural modeling of flames for a production environment. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 729–735, New York, NY, USA, 2002. ACM.

[MS13] Jason Diamond Mike Seymour, Matt Walin. The VFX show. `http://www.fxguide.com/thevfxshow/the-vfx-show-177-the-hobbit-the-desolation-of-smaug/`, 2013. [Online; accessed 21-April-2015].

[SF95] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 129–136, New York, NY, USA, 1995. ACM.

[WH91] Jakub Wejchert and David Haumann. Animation aerodynamics. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 19–22, New York, NY, USA, 1991. ACM.

List of Figures