Video-based Fire Analysis and Animation using Eigenfires

Nima Nikfetrat

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Electrical Engineering and Computer Science

University of Ottawa

Ottawa, Ontario, Canada

Abstract

We introduce new approaches of modeling and synthesizing realistic-looking 2D fire animations using video-based techniques and statistical analysis. Our approaches are based on real footage of various small-scale fire samples with customized motions that we captured for this research, and the final results can be utilized as a sequence of images in video games, motion graphics and cinematic visual effects. Instead of conventional physically-based simulation, we utilize example-based principal component analysis (PCA) and take it to a new level by introducing "Eigenfires", as a new way to represent the main features of various real fire samples. The visualization of Eigenfires helps animators to design the fire interactively through a more meaningful and convenient way in comparison to known procedural approaches or other video-based synthesis models. Our system enables artists to control real-life fire videos through motion transitions and loops by selecting any desired ranges of any video clips and then the system takes care of the remaining part that best represent a smooth transition. Instead of tricking the eyes with a basic blending only between similar shapes, our flexible fire transitions are capable of connecting various fire styles. Our techniques are also effective for data compressions, they can deliver real-time interactive recognition for high resolution images, very easy to implement, and requires little parameter tuning.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Won-Sook Lee, for her continuous communication, excellent ideas, and providing financial support. Her vast knowledge and experience proved to be a vital part of my entire research.

Special thanks go to Dr. Daniel Amyot for his amazing software engineering guidance and inspiration, which directed me to acquire knowledge by exploring the latest tools and libraries for the development of my system in the beginning of the research. Thanks Mr. Andrew Zlotorzynskiin who trained me the safety regulations of working with fire and fuels, and provided a safe place on campus to record small-scale flames. I wish to extend my appreciation to all my friends and colleagues, who have always been supportive and inspiring.

Finally, I would like to thank my family for their infinite support and encouragement that empowered me to pursue my dreams. I would not have been able to accomplish anything without their devotion.

Table of Contents

List of Figures

List of Tables

Abbreviations

2D	Two Dimensional
3D	Three Dimensional
4D	Four Dimensional
CFD	Computational Fluid Dynamics
CUDA	Compute Unified Device Architecture, NVIDIA's parallel computing platform and programming model
DSD	Detonation Shock Dynamics
FPS	Frames per Second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HD	High-definition
LS-SVM	Least Squares Support Vector Machine
PCA	Principal Component Analysis
Qt	A cross-platform application and user interface framework
VFX	Visual effects

Chapter 1 Introduction

1.1 Overview

The modeling of natural phenomena, such as fire, explosion and smoke, is a challenging field in computer graphics that has been widely used in motion picture industry, visual effects productions, 3D animations and videos games for a long time. This field of research should continue to evolve and meet the growing expectation of animators, content creators and even viewers. Complexity of motions in fire and the turbulence in such phenomenon cause the simulation of fire to be an intricate process in computer graphics.

Physically-based approaches are one of the most popular simulations that take advantage of numerical methods in computational fluid dynamics, which is a field in fluid mechanics. In some physical models the turbulence is injected into the fluid with vorticity field which causes parts of the fluid to curl or spin, and therefore provides a situation that act as the motions of a realistic-looking fire. However, all these physical calculations along with rendering techniques require tremendous amount of computing power, which are still too much for current available hardware, and may not even provide a proper real-time previewing in most cases. Contrarily, procedural simulations tend more to emulate the turbulence of the fire using a few fast and simple parametric methods that most of them are not from physical models, or by means of noise functions in multiple dimensions [45][47][48]. Some researchers might combine their techniques with custom designed textures, or incorporate their simulations with particle systems. Therefore it tends not to have low-realism, even though it is fast in processing.

It is common that some users involved in VFX movie production and video games are not interested in utilizing artificial fire simulations, and instead, they take advantage of real captured video footage. Although this decision might be due to the preference of animators, we consider the following disadvantages of fluid systems as the possible reasons:

expensive computation of processing and rendering fluid simulations, complexity of producing realistic simulation by non-experts, dealing with a large number of technical parameters for fluid dynamics and many more.

Despite all the experience with various fluid systems and fire simulators, setting the correct parameters of the physically-based tools is still a very overwhelming procedure. In order to achieve a small, simple and realistic-looking fire, a huge list of necessary parameters and technical terms of fluids have to be changed randomly or through studying the accompanying documents, which are mostly followed by repetitions of a time-consuming rendering to preview a higher resolution of the simulated fire. For the problem of setting the parameters, even if the terms are known for experts, it is still sophisticated to set the correct values for various styles of fire, e.g. setting a velocity for a candle or a log on fire. FumeFX plugin for Autodesk 3D Studio Max and Autodesk Maya Fluids are two instances of such fire simulation systems that provide much flexibility with less usability. Our aim is to provide sufficient flexibility and usability simultaneously for a user to model fire intuitively to fill this gap. Our results are generated from various real videos of fire and motions recorded in 60 fps and 720p format, and the animators are still capable of controlling the fire, defining a scenario and generating another distinct fire animation, either longer or shorter than the original video. Our methods are more realistic-looking in comparison to existing procedural or particle-based approaches, faster and easier to control in comparison to physical-based approaches, and less complicated than current fire synthesis techniques.

In our approach, we are not concerned with measuring microscopic physical properties of fire, but with generating new fire frames by decomposing and recomposing existing fire frames which are performed through recognizing and/or synthesizing macroscopic patterns of fire using Principal Component Analysis (PCA). These techniques are based on “Eigenfires”, which are the eigenvectors of the covariance matrix for videos of various kinds of fire samples. The final quality can be as good as the quality of the recorded videos, and the users are still capable of compressing the videos. As a result, anyone is

capable of building a new database of Eigenfires with their own camera and video, or they can load one from our pre-processed library via a user-friendly interface to generate a new animation in a few seconds.

More precisely, our research seeks two fundamental objectives. The first objective is *fire analysis and visualization* in PCA subspace as a promising concept and useful for future research with similar topics. The second main objective is video-based *modeling and animation of fire*.

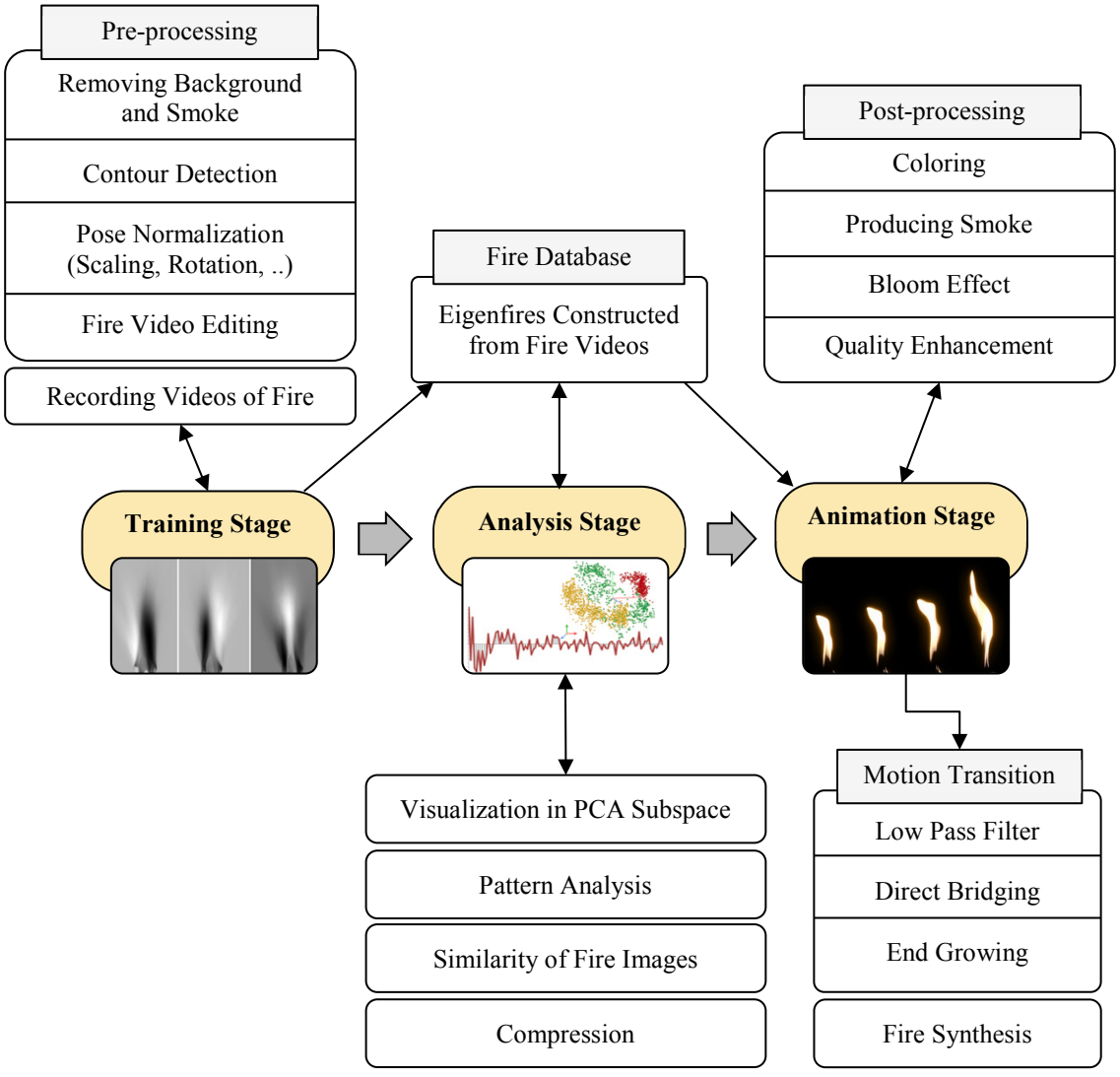


Figure 1: The main stages of our system and related tasks

Our system is divided into three major stages: (1) *Training stage*, (2) *Analysis stage*, (3) *Animation stage*. Figure 1 illustrates the overall structure of our system.

1.2 Objectives of This Research

One of the objectives of our research is to achieve some video-based techniques that enable animators with basic skills to produce new fire animations interactively, with the least amount of parameter adjustment. We are interested to collect information about the patterns of small-scale fire samples in such a way that can be used for fire synthesis or animation. An important goal here is to develop new techniques that can generate realistic-looking fire animations based on real footage of fire. For this purpose, we would like to develop a system that allows animators to easily pick the names of their desired motions, and then the system can automatically generate new animations without being concerned about physical properties of fire.

1.3 Proposed Solutions

In order to achieve our objectives, we propose the following solutions in this thesis. To understand the pattern of various fire samples, we perform analysis and visualization of small-scale turbulent fire videos in PCA subspace. We develop novel techniques of generating realistic-looking fire animations by recognition of similar flames in a database of fire or through synthesizing a new frame. It is important that the videos for our database should be recorded with at least 60 fps or higher to get practical results. In order to enhance the accuracy of recognition for generation of smoother animations, we propose a few image-based techniques to isolate the fire from the background. We develop our system and algorithms in a general way, so that they can be applicable on various types of fire videos under different lighting conditions. We also suggest algorithms of synthetic colorization that provide additional level of compression for fire videos.

The concept of our fire synthesis and analysis using Eigenfires is published in the Springer book "Intelligent Computer Graphics 2012", and it is mentioned at the end of this thesis.

1.4 Organization of the Thesis

In Chapter 2, the literature review of the most well-known fire modeling and simulation approaches is presented. Since we are taking advantage of Principal Component Analysis, we also introduce some related works based on PCA that we benefited from, or they inspired us to think about how we can evolve. Related research about video-based image synthesis is also covered in this chapter.

Before starting the training stage, we need to prepare our dataset of fire videos in such a way that the modified dataset can be used properly in the training stage. This phase is called *pre-processing* in our system and is presented in Chapter 3.

Chapter 4 discusses the thorough details of the first two major stages, training and analysis stage, with 2D and 3D visualizations of fire.

The 3rd major stage, animation stage, is discussed in Chapter 5, in which a few novel techniques are introduced for fire synthesis, modeling and animation with motion transitions.

Coloring and *post-processing* are also part of animation stage, and are presented in Chapter 6.

Experiments, performance statistics and discussion of possible methods to extend the 2D results into 3D are covered in Chapter 7, and the thesis ends by conclusions and a summary of contributions in Chapter 8.

Chapter 2 Literature Review

In this chapter, we review existing methods of fire simulation and video-based fire synthesis. We discuss the advantages and drawbacks of these methods, and compare them with our procedural methods. It must be mentioned that our aim is not replacing all the current techniques with our approaches, but to take fire analysis and animation to a new level by introducing simple unique ideas that is for sure interesting for many video game developers or people involved in visual effects industry. By utilizing our system, video game designers may automatically produce loops for fire animations to be used with 2D billboards, or generate new realistic-looking animations of fire in real-time. Similarly, film industry may take advantage of video recording equipment and high speed cameras to prepare their own high resolution database of fire. Therefore, a new animation can be generated in various productions based on scenarios, and still the database is capable of being extended by new recorded videos.

2.1 Fire Simulation in Computer Graphics

This section is categorized into three major subsections to introduce current techniques of fire simulation. Although most of the simulations are hybrid and a combination of two or all the three general approaches are involved in each work, they are classified in a group with the most major characteristics.

2.1.1 Particle-based Approaches

Particle-based methods have been very popular for decades and are still extensively used in current generation of video game consoles and mobile devices such as tablets. Particles systems use a large number of camera facing sprites or other classical graphic geometries (spheres, cubes, lines, etc.) to simulate fuzzy phenomena due to absence of sharp edges. Examples are fire, explosions, smoke, clouds, snow, and dust, which can be created

using specific textures, blending and blurring methods. Particles can be generated from a virtual emitter with different shapes (rectangular, circular, etc.) to imitate the source of the fuel (Figure 2). Each particle contains a set of attributes that can be changed during time steps and this is what makes the particles systems so flexible; attributes such as color, size, weight, life time, etc.

Particle systems carry much lower computational cost of rendering than volumetric rendering, and their assigned attributes can be modified procedurally or by the laws of physics. Unlike particle-based simulations, our approaches aim to deliver high quality results almost the same as the captured videos. As a result, we produce our final results in two different styles, using either the original fire video or from fire synthesis.

Although particle-based flames are usually placed in a category with lower quality of rendering, they are widely utilized because of limited hardware resources and ease of use in most cases in conjunction with other virtual environments or platforms

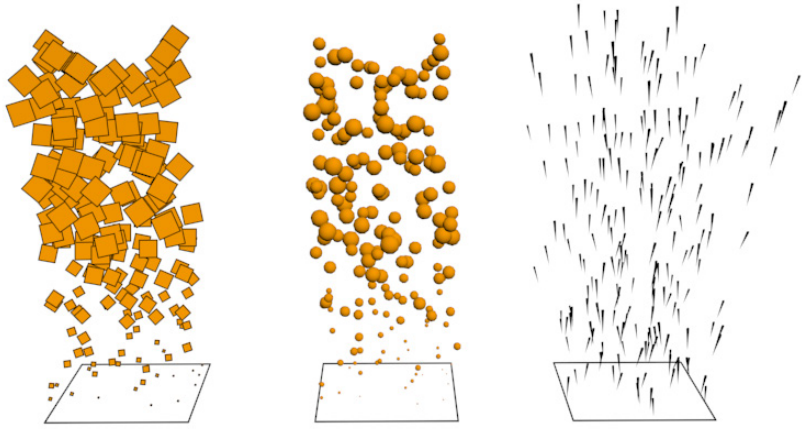


Figure 2: Particle systems with various particle geometries

2.1.1.1 Particle Systems and Particle Rendering

Reeves presented particle systems and their attributes in [52], and discuss how they were used to generate the fire element in *Star Trek II: The Wrath of Khan* by Lucasfilm Ltd., which was amazing at the time and a useful source for future works with particles.

Perry and Picard [49] employ a modified particle system to synthesize flames that can spread over polygonal objects, and it would give the user control over parameters such as external forces. A flame is modeled as a series of primitives along the trajectory of each particle that can blend and deform in each time step. The shape of the particles consists of regular 2D N-gon primitives with Gouraud shading, and the particles move using basic velocity equations, Newton's second law of motion as a force field ($\vec{F} = m\vec{a}$), and intentional perturbations. The drawback of this technique is the poor quality of both rendering and animation.

Chiba et al. [9] employ a basic 2D particle system using a vortex field (Figure 3) to simulate smoke and fire, which is not based on exact physical simulation and the motion of flames or propagation can be controlled based on defined scenarios. Their hypothesis is that the flames and smoke are formed by visualizing turbulence, consists of vortices of various sizes, and the particles behave as tracers in the field of turbulence. One of the problems of the particles that they could solve was rendering particles as sampled points of a fluid using a set of interpolated line segments instead of points. Collision detection is also performed between a vortex and obstacle at the center of a vortex. Their extended model supports appearance of smoke due to incomplete combustion over a fire, which is based on life span of particles. In our system, we also allow users whether to enable or disable the original smoke available in the real fire videos. We first remove smoke to gain a higher accuracy in PCA recognitions, and later restore them for final animations.

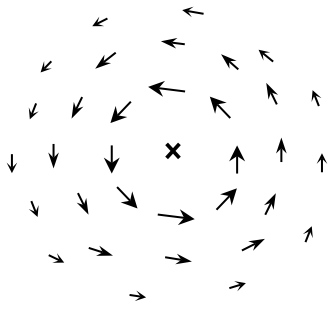


Figure 3: A 2D vortex field

Stam and Fiume introduce a solution to the advection-diffusion equation using warped blobs [58] for better convincing appearance of fire, instead of unwarped spherical blobs. In this work, the thorough introduction of diffusion equations for the evolution of the density, temperature and diffuse of fire makes it clear that the huge set of interdependent parameters causes the controlling of fire very complicated for non-experts.

An efficient fire model is proposed by Takahashi et al. [62] that acts as volume light sources and cast illumination into the static environments. The particles are used as some tracers, and the turbulent wind fields are added by three dimensional vortices. The image synthesis is also performed using pre-calculated intensity maps. Images of the scene are then rendered using a ray-tracing algorithm, by shooting rays from viewpoint to each pixel, together with volumetric rendering of the flames.

Since particle-based systems may not always result a visually appealing fire, therefore they are usually combined with other physical and combustion methods [19][51], which are mostly under heavy processing calculations for rendering and visualization.

2.1.1.2 Texture Splats

Xiaoming Wei et al. [69] proposed a fire modeling with textured splats to achieve small-scale turbulence. A texture splat is a small image of turbulence detail that is multiplied with a smoothing function to provide good blending in areas where splats partially overlap. By associating texture splats with particles as a set of display primitives and blending them, the turbulence in a fire can be visualized. Texture splats can be generated procedurally (Figure 4) using noise functions such as Perlin noise [47][48], or be taken from real fire images. There are many terrain generators available that may assist artists for this purpose, such as “Terragen Classic” and the open source “TexGen” tools. Some modifications however are required for the fire.

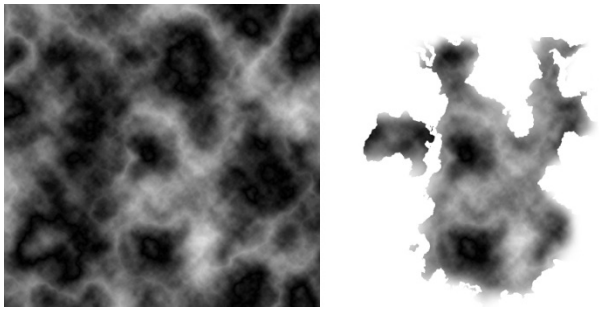


Figure 4: Our experiment for procedurally generated texture splats

The proposed model of Xiaoming Wei et al. is based on Lattice Boltzmann Model (LBM) for fire evolutions and simulation of physically-based equations. To achieve real-time interactions, two main grids are defined, one small grid for the LBM and another larger one for calculations of Navier-Stokes equations. Smoke as an incomplete combustion of a substance is also generated using a temperature field, same as Chiba et al. [9]. However, appearance of the smoke around the fuel, which shows up irregularly inside the fire, makes it unrealistic-looking for their examples in comparison to volumetric rendering or real footage of fire (Figure 5).

Figure 5: Fire modeling with texture splats. Reprint from Xiaoming Wei et al. [69]

A similar fire modeling is also proposed by King et al. [34] using coarse voxel grids and rendering with a splatting approach, in which the details of fire are incorporated into the splats and are updated at each time step. Their model is also able to randomly sample a

photograph of actual fire, and then a new series of textures can be generated based on that for rendering of a new fire.

Although rendering of regular particles with texture splats provide real-time simulations, their low quality cannot compete with recent volumetric renderings such as GPU simulation of fire by Krüger and Westermann [35], or with video-based techniques such as our fire animation in this thesis.

2.1.1.3 Billboards in Computer Graphics

Regular billboards in computer graphics are planar rectangles that always face the camera. They can be spread in 3D space or randomly on a surface using particles, or they can be positioned in specific distances from each other inside a cube. Then 2D textures of real fire can be applied to these billboards to make it look like a three dimensional fire or explosion (Figure 6). The issue of classical billboards are clipping artifact, in which the intersection of the 3D objects and billboards produces visible layers of planes that influence the transparency of the fire, caused by ignoring their distance from the objects.

To solve the clipping problems of fire, an improved type of billboard, spherical billboard, is proposed by Umenhoffer et al. [67] to render explosions for both fire and smoke in real-time on GPU. This is achieved by obtaining depth values of the front and the back surfaces, and then computing the opacity based on the distance inside the particle sphere in camera space until reaching the objects. While particles are quadrilateral primitives, the spherical shape is only assumed during fragment processing. The smoke and dust layers are calculated using Henyey-Greenstein phase function with zero value for emission term and then some more details are added by a few gray textures to perturb the billboard fragment opacities and temperatures. Although their work shows a better outcome than planar billboards, it still looks very artificial in comparison to what we produce.

The emergence of current generation of console games such as Microsoft Xbox 360 and Sony PlayStation 3, and huge advances in the graphical processing power of personal

computers in recent years, led to a wider use of billboard in video games. Now game developer are capable of creating thousands of extra quad surfaces to imitate grasses, plants, smoke, flames or even the flow of the drops of water from a waterfall. New features such as hardware tessellation, allow more details and meshes to be generated in real-time on the GPU, and let the central processing unit be free for other calculations.

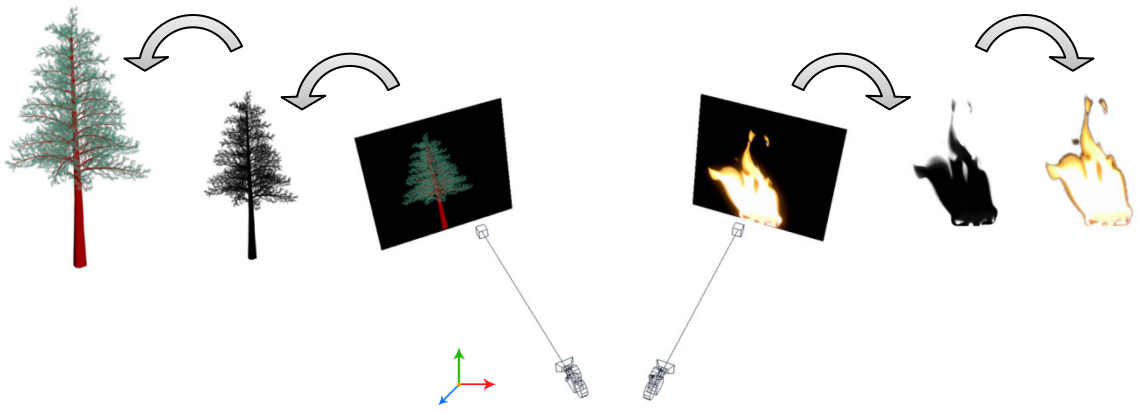


Figure 6: Camera facing billboards. Masks as alpha channels and textures with transparent background are depicted. On the right, one frame of our fire videos is shown with a transparent background

Many game designers that aim for high-quality games (AAA), commonly with high budgets, are eager to utilize recorded videos of fire directly as one of their primary choices. In many games and film's visual effects, in order to show that a house or object is set on fire, a few low poly 3D surfaces are created that surround the burning object, similar to curtains hanging from a wall in case of a burning house. The videos of real-life fire are their textures, and the corresponding alpha masks are used for transparency. Offsetting the coordinates of the textures sometimes acts as a procedural simulation of fire.

Billboards are one of the most popular approaches that animators and game designers take into account to display the output of our system in 3D environments. Thus, the mask images we produce in pre-processing phase are used for transparency of the background and blending in 3D scenes using billboard or similar techniques.

2.1.2 Physically-based Approaches

In this section, we first introduce the main equations of fluid mechanics for computer graphics, and then discuss such methods with their weaknesses in comparison to our video-based techniques.

2.1.2.1 Fluid Mechanics

In computer graphics, the most realistic models of fire are simulated with physics of fluid mechanics. Because of mathematical complexity of fluid mechanics, the problems can be solved using numerical methods. Computational fluid dynamics (CFD) is a discipline of fluid mechanics that are introduced to solve the problems of fluid flow. The fundamental basis of CFD problems is the Navier–Stokes equations which are the differential equations describing the dynamics of fluid flow, and have been used in majority of the fire simulations. By considering an incompressible flow of a Newtonian fluid, simplified flow equations can be obtained. Taking the incompressible flow assumption into account and considering a constant viscosity for a fluid, the Navier–Stokes equations are as follows:

$$\frac{\partial \vec{u}}{\partial t} = \frac{1}{\rho}\,\vec{f} - (\vec{u}\cdot\nabla)\vec{u} + v\,\nabla^2\,\vec{u} - \frac{1}{\rho}\,\nabla\vec{p} \tag{1}$$

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

where $\vec{u}$ is the velocity as a vector of 2D or 3D dimensions, t is the time, ρ is the density, $\vec{f}$ is the force field, v is the viscosity of the fluid, p is the pressure and ∇ is the gradient operator. Equation 1 ensures that momentum is conserved. As the incompressibility feature of a fluid under constant density, Equation 2 ensures that mass is conserved. This is expressed mathematically as the divergence of the velocity is zero in this equation.

Since the forms of Navier–Stokes equations could vary according to viewpoints of the fluids [63], we introduce the two well-known models that have been extensively used in

fire simulations to track the motion. The two approaches are: *Lagrangian* model and *Eulerian* model [7]. Figure 7 shows the complex procedure of setting simulation parameters for a fluid system.

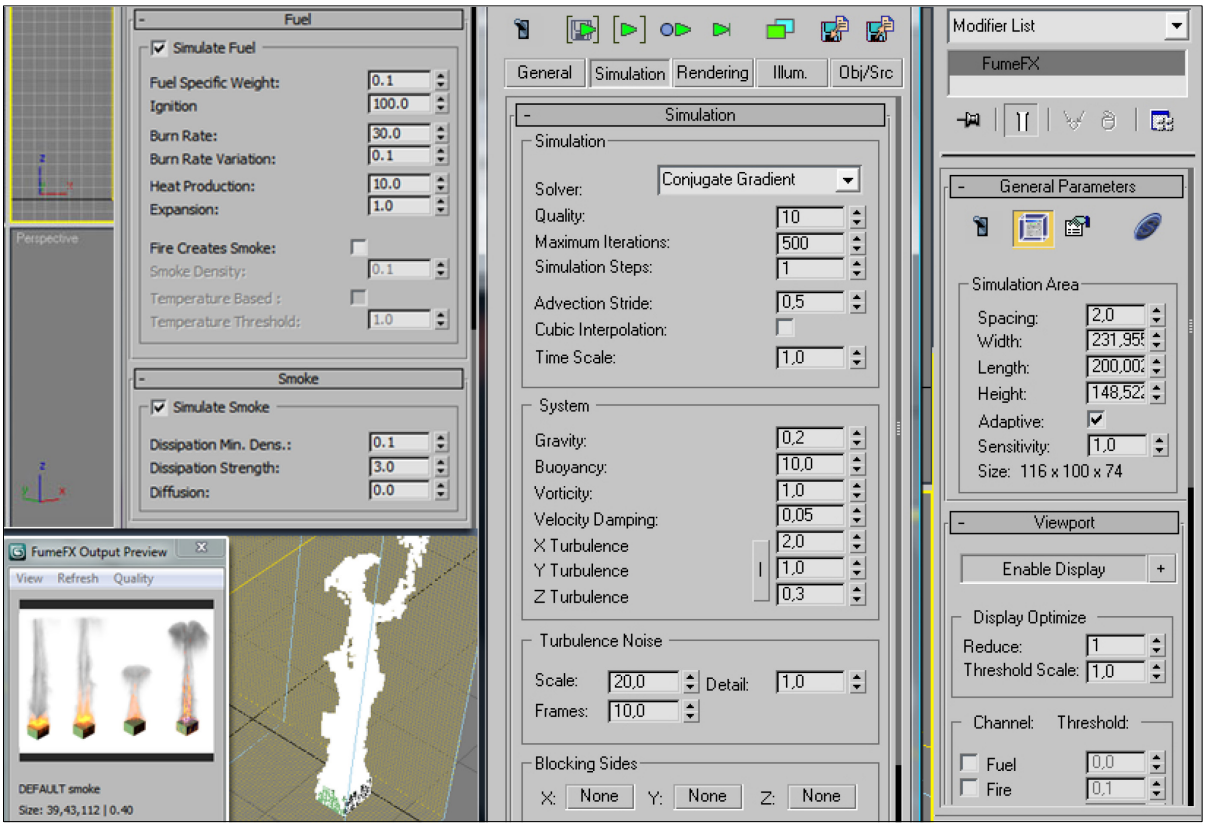


Figure 7: An screenshot from some complicated simulation parameters of FumeFX as a fluid dynamics engine from Sitni Sati

2.1.2.2 Lagrangian and Eulerian Appraoches

The Lagrangian approach treats the continuum moving as a particle system. In this model (Figure 8), the fluid is composed of a set of particles in specific distance from each other that are floating in the fluid and the features of the fluid such as temperature and pressure are moving along with the particles. A few challenging problems in this model are the required number of particles for fine details, surface generation from particles and rendering methods that can be solved in different ways. The particles provide a straight and

convenient way for collision detections but it is an expensive process. For simulation of solid objects, Lagrangian approach is usually taken into account as a discrete set of connected particles.

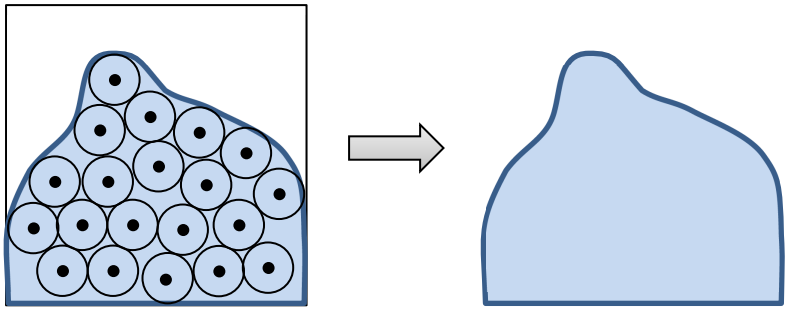


Figure 8: Lagrangian particle-based fluid structure in a 2D bounding box

The Eulerian approach is not based on particles anymore, and instead it follows grid based structures of different sizes (Figure 9), which is usually used in fluids. The particles are not tracked in this model, and instead we look at fixed points in space to compute fluid quantities and then update those data for the next time step. In many hardware accelerated implementations, fluid quantities such as density, velocity, temperature and color are stored in a series of 2D or 3D textures for easier or faster access. A few problems of this model are the fixed size of the grid, redundant calculations for empty regions, and expensive volumetric rendering.

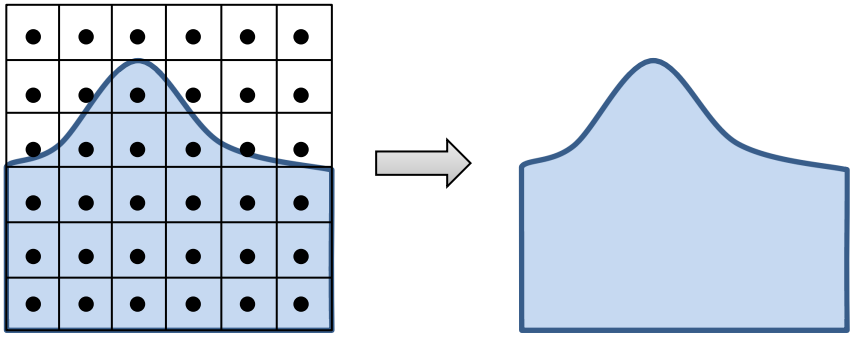


Figure 9: Eulerian grid-based fluid structure

2.1.2.3 Physical Simulation of Fire

Nguyen et al. [43] use a semi-Lagrangian stable fluids approach to model both 3D fire and smoke; one set of incompressible flow equations to model the fuel and another one for the hot gaseous products. This model is capable of animating the combustion of both solid and gaseous fuels or to interact with other objects. An implicit surface, which is called blue core region, is created to divide the regions between gaseous fuel and soot. Tracking the temperature is one of the interesting parts of their work, which influences fluid velocity. In this model, the temperature rises up until reaching a specific degree to ignite the fuel by considering the implicit surface. Temperature continues rising until reaching a maximum degree and begins to cool down to produce smoke and soot. For rendering, the fire is assumed to be a blackbody radiator and a medium with scattering, absorption and emission properties. A realistic color and rendering using a stochastic ray marching algorithm complete their model which recursively samples multiple scattering of the light (Figure 10). Despite the complexity of working with this physically-based technique, the obvious disadvantage is the huge performance issues, in which rendering a fire with a grid size of 120×120×120 takes around 5 minutes per frame. That is one of the advantages of our proposed methods that each frame can be computed in a few milliseconds, and still the results are from the real flames with sacrificing some flexibility. However, our 2D model cannot interact with other objects.

A grid-less stable numerical simulation of fire is introduced by Adabala and Hughes [1] using particle systems on GPU. To model the dynamics of the fire, a stochastic Lagrangian approach and a chemical composition evolution model are utilized. Although the parametric model allows real-time controlling of flicker in a fire, the results are not even as good as recent procedural approaches, or close to our realistic-looking results.

A combination of fluid and combustion models was discussed in a work by Min and Metaxas [42], using semi-Lagrangian method for advection of the fluid model. Their model

is capable of emitting the light and heat of fire into the environments by means of a photon casting procedure, where the voxels of the fire act as photon generators.

Figure 10: Physically-based modeling of fire. Reprint from Nguyen et al. [43]

Pegoraro and Parker [46] employ detailed simulation of the radiative emission and refractive transfer to achieve realistic renderings of fire. Based on fundamental molecular physics, they compute the emissive properties of the flame and the spectral properties of the different chemical compounds of the fire. The refractive properties of the fire are also taken into account to produce a wavy visual feedback that warps the objects behind the flames in the background. Their interesting fire rendering however takes about one to two minutes for an image of 512×512 resolution, almost similar to the processing time of other physically-based fluid simulations.

A modified version of Stam's sable fluids approach [57] is used by a work from Melek and Keyser [41] to produce fire in such a way that the motions of air, fuel, and exhaust gasses can also be simulated. However, the generated fire does not include sufficient fine detailed.

Simulation of gaseous phenomena in turbulent wind fields was depicted by Stam and Fiume [59] using a clustering algorithm, and the gas was modeled as a fuzzy blobby, in

which advection term was responsible for moving a blob, and diffusion term to deform it by an advection-diffusion equation.

Detonation shock dynamics (DSD) is used in a work by Hong et al. [28] to produce cellular patterns in flames. The fire is generated by coupling the third order DSD equations to the Navier-Stokes equations, combined with level set method. This model produces astonishing results and details for large-scale flames. However, the huge amount of required processing restricts the users of the system, as it takes around 4 to 8 minutes on average to render a single frame.

Producing high resolution flames in 3D is achieved by Horvath and Geiger [30] on GPU for visual effects of movies. The first stage is a coarse particle grid simulation that allows users to direct and control the motion of fire. Next, fine details will be added in the refinement stage, consisting of specific number of camera-facing image planes. Attributes of particles will be projected onto these planes. A GPU-based volume rendering and a farm of 10 GPUs make it feasible to obtain high quality simulations with layers of details, which might not be affordable for a small business, or freelancers. Another drawback of this technique is the massive processing time that should be distributed into farms of graphics cards or it will takes days to render a single frame in HD resolution. The issue here is that a quick previewing of a draft for the fire is not possible, unless the first processing stage of the animation is completed prior rendering.

Harris [23] describes mathematical background and implementation of 2D fluid simulations using Navier-Stokes equations for incompressible flow on the GPU, which significantly increased the performance in comparison to Stam's simulation on CPU [56][57]. Valuable GPU rendering tricks and implementations of dynamic fluids and fire are also discussed in [11] by Crane et al., which is a great reference for everyone interested in fluid simulations.

Despite the fact that physically-based approaches require significant amount of time to be processed, they mostly produce high quality visualization of fire and are more realistic

if the correct parameters are set by the users for the variables of dynamic fluid equations in the system. In recent years, a few GPU-based techniques have been introduced that enable users to calculate physical equations of fire and fluids in real-time with lower qualities, and they take advantage of storing densities, velocities and other large information of the fluid in 2D or 3D textures for faster access.

An advantage of our non-physical-based approach is that it supports beginners with no prior knowledge and experience of dynamic fluids or its technical terms (such as velocity, buoyancy, vorticity and kinematic viscosity) to produce an elementary fire. Another benefit of our approach is the capability of controlling available real fire animations through deforming their shapes, rather than starting from scratch which would be sophisticated or overwhelming. In this way, animators proceed faster than physically-based approaches.

2.1.3 Procedural Approaches

Fuller et al. [20] take advantage of the improved Perlin noise and M-Noise [45], and then combine them with an interesting curve-based volumetric free-form deformation to create a procedural model of fire (Figure 11). The 3D and 4D noises are employed based on the version of supported Pixel Shaders in various graphics cards. Their 3D hardware-accelerated volumetric rendering allows an artist to easily manipulate and deform the fire along a B-spline curve in real-time. By constructing a lattice around the volume, texture coordinates can be controlled and interpolated while the flame deforms along the curve. For volume rendering, the deformed lattice is sliced into evenly spaced view-aligned planes, and then the pixels of the associated textures are accumulated on GPU using additive blending. Although this system produces a very good looking fire with an amazing flexibility in real-time, the animations are not close to natural fire, because it is a system solely based on random noises. In our system, we provide a similar noise feature only as an additional synthetic option and keep the quality of results as close as the original video. This optional

noise is based on our noise generation which is the user's control on contribution Eigenfires via weight manipulation.

Vanzine and Vrajitoru [68] integrate the same system of Fuller et al. [20] into a 3D game engine, and discuss statistical results of performance obtained from different procedural noises with variety of volume sizes. Statistics of the comparisons are also provided for implementations with both DirectX and OpenGL.

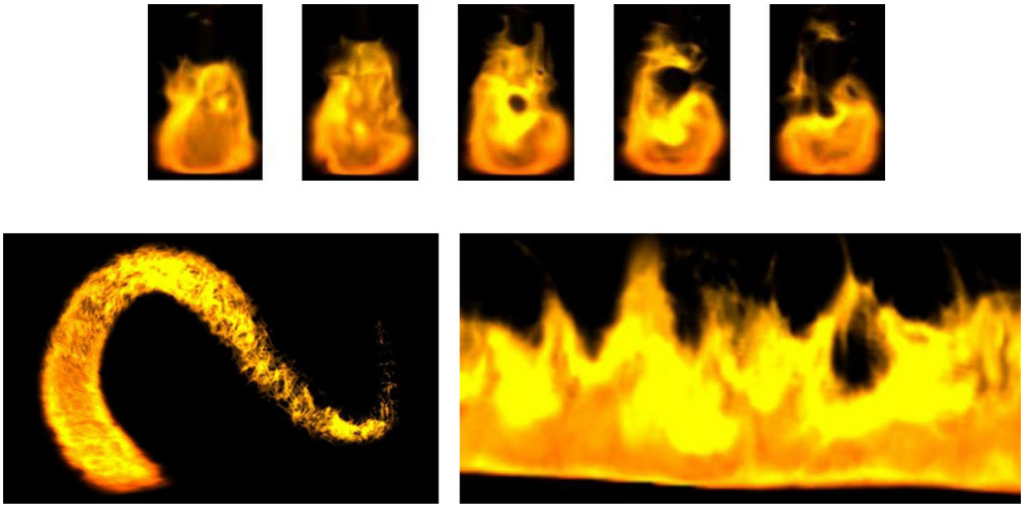


Figure 11: Real-time procedural volumetric fire. Reprint from Fuller et al. [20]

Based on a proposed technique by Lamorlette and Foster [37], a flame profile is created from a set of points as *spine* of the flame, using observed statistical properties of a real fire. This curve, or spine, can break and evolve with a combination of physically-based and procedural fields, and it deforms implicit surface of the flame. Particles are sampled on this surface, in addition to incorporating animated procedural noises into the process, and then the particles can be rendered volumetrically. This model is capable of interacting with stationary objects, and was used in the 3D animated movie *Shrek*. The complexity of working with such a system is the drawback here, in which an animator becomes productive within a week, while it takes less than 10 minutes to learn how to work with animation part

of our system, because the visual behaviors of the fire is not dependent on the parameter adjustments of a fluid anymore in our work.

Another representation of fire is introduced by Beaudoin et al. [4] that supports fire propagation. In this work, the fire is represented as a small set of flames instead of working with large numbers of particles, and it can spread progressively over the meshes. Skeleton technique, which is a small group of connected particles, forms the flame animation and is moved by turbulent as a time-varying vector field. Flames are modeled using implicit surfaces and are obtained from created skeletons. Finally, the model is rendered using a ray tracing algorithm. This approach is also utilizing noise functions or user defined parameters for air velocity field to animate the skeletons of the flames, in which they actually reduce the realism. In our approach, we use the natural patterns of fire instead of entirely deforming the fire with noise functions.

Amarasinghe and Parberry [2] discuss real-time rendering and deformation of burning objects on GPU while generating procedural fire using particles. In order to show that burning polygonal objects are consumed by fire, the objects are divided into uniform blocks that deform the vertices of the meshes on GPU in a Vertex Shader using CUDA. Although the consumption of burning objects looks great, the low quality of the generated fire makes it unbelievable as a natural phenomenon. Therefore, we considerably pay attention to the quality criteria in our research and later recommend proper numbers of Eigenfires and a few coloring techniques to prevent lowering the quality for professional industries.

Current procedural approaches tend more to imitate the characteristics of real flames by utilizing procedural noises or offsetting of texture coordinates to incorporate turbulence into the fluid. For fire and smoke generation in computer graphics, procedural methods can be used for various stages of production such as rendering, modeling, texturing and colorizing, and mostly intended as a substitute for existing time-consuming techniques.

Occasionally they are partially combined with physical equations to model a more visually convincing fire. Our animation technique belongs to a procedural approach.

2.2 Statistical Approaches

Principal Component Analysis (PCA) is a method of identifying patterns in data and representing data in such a way that highlights similarities and differences. PCA is a simple and non-parametric method of extracting information or "*features*" from complex datasets. Its capability is to identify relatively fewer features or components that still can represent the entire object, and that is why they are named as "*Principal Component*". It is important to consider the fact that the extracted components may not exactly express an actual feature that humans can understand. However, our attempts to use the PCA for fire analysis provided us with very interesting results, in which the essential features of our recorded fire videos could be extracted and visualized in such a way that animators can observe and perceive each of the main features one by one, and hence even use them to synthesize a new flame from scratch. These principal components are expressed in various ways and introduced as "*Eigenfires*" in this research.

In modeling fluid motion, selecting a basis of Laplacian eigen functions also provides novel and interesting properties leading to visually convincing simulations with far fewer degrees of freedom than existing approaches. De Witt et al. presented this idea in [14].

2.2.1 General Recognition Techniques

Applications of PCA are so vast and therefore we review a few interesting works that are sharing some basic ideas with some parts of our fire analysis, or directed us to achieve more accurate results. Face and gender recognition, handwriting analysis, hand gesture detection, cloud or object classifications, and geometry shape recognition are a few useful research areas we discuss here.

Face Recognition:

Eigenface [64][65] is a well-known face recognition technique developed by Turk and Pentland. By means of PCA, each set of face images can transform into a corresponding eigenface. Each eigenface represents only certain features of the face, and can be visualized as blurry combination of different faces, in which distinguishing most of the features from each other might be vague for viewers based on the size of the dataset [38]. Eigenface has been employed in many other fields outside of facial recognition and our fire analysis is also derived from their algorithms.

Face gender classification is a method proposed by Quanhua et al. [50] that combines eigenfaces with Least Squares Support Vector Machine (LS-SVM) classifier. The results of their research demonstrate that LS-SVM has a better classification performance in comparison to other known classifiers.

Handwriting Recognition:

Zuo et al. [74] present a simple algorithm for identification of writers' handwriting from a few characteristic words. The characteristic words are calculated by comparing the average distance between the feature vectors that are projected onto the PCA subspaces. Their dataset consists of 16,000 Chinese words, in which 40 Chinese words written by 40 different writes, and each word is repeated 10 times by the same writer. Instead of performing comparisons with the entire texts, a small number of characteristic words are used to identify writers, and therefore an average identification performance of 97.5% is achieved. However, this success rate is because of the customized dataset, produced by rewriting the words that makes the classification easier.

Another similar method is presented in [13] for handwritten character recognition, captured with a digitizer and pen position sensors.

In our approach, we are not concerned with classification like other recognition systems, unless the user restricts the outputs to specific types of fire accessible in the dataset.

The repetitions are also automatic with our techniques because of the nature of motions occurring in a real fire.

Hand Recognition:

Hand gesture recognition has been widely studied with PCA in the past two decades. Birk et al. [6] presented a real-time hand recognition for international sign language hand alphabet. In their system, they first map the hand pixels to very small gesture images of 32×32, and then normalize the images of the hands using segmentation and transformation. For classification a Bayes classifier is used, and finally their experiments show a Gaussian distribution for alphabet hand gestures.

In a recent work by Dardas and Petriu [12], the hand gesture recognition was enhanced in such a way that the bare hand could be detected in cluttered backgrounds. The idea is to remove the face of the humans from the input images, and then to track the hand based on the color of the skin with HSV color space. The database is prepared for fist, index, little and palm as the template of hand postures, and recognition is based on contour comparisons. No matter how large the output resolution of the webcam is, the images of the detected hands are converted to small images of 160×120 pixels to be able to run the system in real-time.

Hand shape recognition is a robust method introduced by Choi et al. [10]. Hand shape decomposition is the interesting part of their approach, in which the front view of the hand is divided into a few segments in order to isolate regions of the fingers. Then, the movement of the fingers can be analyzed and detected.

Object Classification:

PCA based classification of both single-layered and multi-layered clouds is presented in [3] by Bajwa et al. which is used for weather forecasting applications. In this system, the cloud images are scaled down to be 50×50. The classification in this technique is simple and

similar to [38], in which the threshold criterion is based on comparisons with half the largest distance between two images in PCA subspace.

Shape Recognition:

There are many shape recognition techniques based on PCA. Chi-Min Oh and Lee [44] proposed a shape recognition applicable to silhouette contour of segmented objects, capable of detecting basic geometries. This method is performed with Bayesian probability and Modified Chain Code (MCC), by means of normalization of chain code histogram. Another method is introduced by Tzimiropoulos et al. [66] that is capable of detecting 2D planar shapes such as aircraft silhouette identification. Their procedure is based on mapping the affine transformed boundary of the shape to its canonical form, and then analyzing the boundary locally. Hamadache et al. [21] recently proposed a method to control a 3-DOF manipulator robot using shape recognition of simple shapes that are given as input to the system.

In this thesis, although some parts of the fire analysis might look similar to some of the mentioned methods in this section, however the definitions of some regular terms are slightly different. For example, "*recognition*" for fire animation does not mean detection of exactly the same shape of the fire, but something proper for next frame of the fire animation. Therefore, later all these differences will be elaborated in its corresponding section.

2.2.2 Fire Recognition Techniques

Although recognition of fire and flames have been studied using PCA, they do not consider analysis, visualization and animation of fires to the same extent as we are trying for the use in computer graphics. Some of those PCA related works about fire recognition are presented in this section.

Weitao Li et al. [39] proposed an image-based technique for burning states recognition of captured flames from rotary kiln (a device to heat solids). The flames are a

series of poor quality images (Figure 12) and their method does not involve any image segmentation for the flames. The samples for the training set are obtained for 314 images of different burning states. The eigen-flame images are ranked using Fisher ratio class separability measures, and the burning state is classified with probabilistic neural network into three classes: over-burning, under-burning and normal-burning states. Therefore, this technique is only covering the recognition of these burning states for specific purposes other than computer graphics. In contrast, what we are trying to achieve is preparing Eigenfires in such a way that they are combinations of various fire samples, and suitable for both fire synthesis and fire recognition, and therefore it is performed through a series of pre-processing in conjunction with our further analysis and visualization for the purpose of producing new animations.

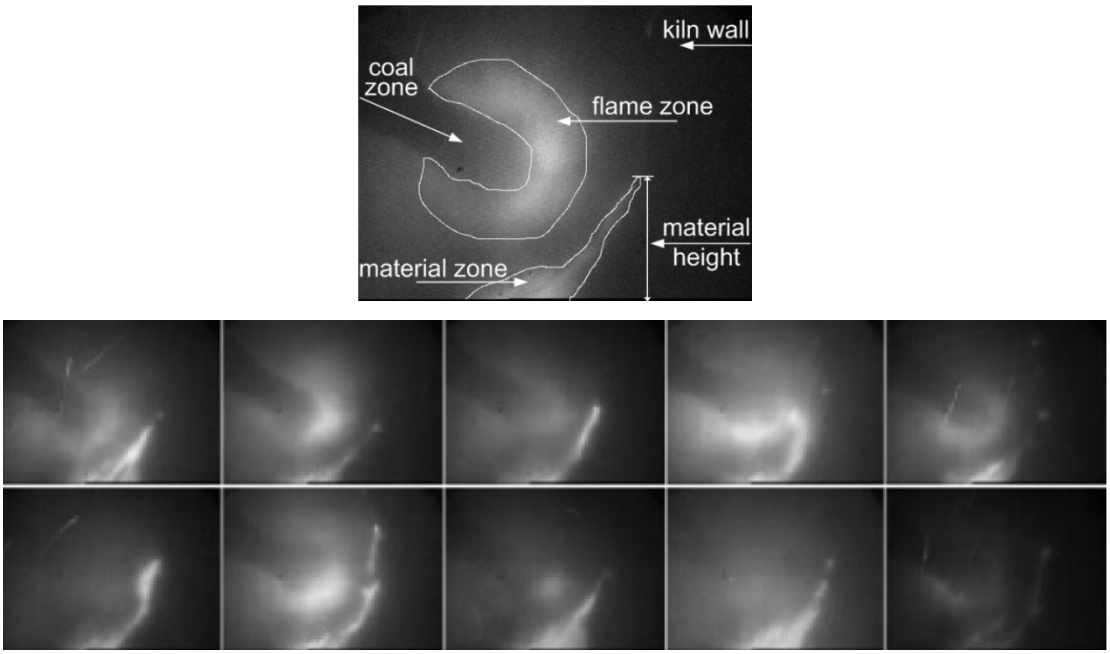


Figure 12: Poor quality flame images caused by smoke, dust and high temperature inside rotary kiln. Each image consists of several parts including kiln wall, coal zone, material zone and flame zone. Reprint from Weitao Li et al. [39]

Another fire recognition algorithm is proposed by Hongliang et al. [29]. They use a Multi-features Fusion algorithm (MFF), and analyze the twinkling frequency of flames and their contour. Fourier transform is also employed to obtain dynamic features of changes.

2.3 Sample-based Image and Video Synthesis of Fire

There are a few computer vision projects that are sharing some similarities with our algorithms. Although some of them apply their techniques to some basic fire samples, they do not concentrate on the problems of combining various fire animations or visualization in PCA subspace as we explore in computer graphics.

2.3.1 Video Texture

Video texture is a very interesting approach, introduced by Schödl et al. [53], as a continuous infinitely varying stream of images which is synthesized using random loops and transitions. In the first step the video is analyzed and the repetitions of the video are identified. Then they generate a new video by inserting the transitions stochastically or producing a few loops to play the video indefinitely. They use transition using various techniques such displaying original video frames, cross-fading and morphing.

Video textures work perfectly for different sequences of video as long as the camera's angles are fixed, background is not much cluttered and videos are not under varying lighting conditions. Therefore, transitions are limited for fire samples and it cannot be easily performed on our fire samples because of huge changes of lighting conditions, emitted from various flames. In contrast, we concentrate on combining various controlled fire samples and introduce similar concepts but in different ways suitable for fire. While they use L_2 distances and probabilities with an exponential function for analysis, we take advantage of PCA to perform our techniques in smaller dimensions. PCA also enables us to perform recognition steps faster, compress the final videos, visualize fire samples in various

ways, and modify or synthesize a frame as transition using Eigenfires. Although video sprites and background subtraction (using thresholding) are presented in video texture, they cannot be successfully applied on the multiple backgrounds of different fire samples we are trying to combine. Removing the background of turbulent fire samples is another problem that we solve in this thesis. In our system, we use frame rate of 60 fps with controlled motions of fire, and it is possible to generate new fire animations that are not already recorded.

Their large samples also include two fire examples created with video texture. They extended the regular motions of a candle flame because it was not turbulent at all and included no smoke, and similarly the loop was produced for a turbulent campfire using crossfading over only four frames. In contrast, our proposed motion transitions are capable of generating more frames for transitions of various flames (e.g. more than 20 frames for a transition) and still loops and transitions look visually convincing without tricking the eyes of viewers by simple blending.

In video texture, the system first store good transition points and then synthesize a new video stochastically based on them. In our work, the animator specifies preferred range of frames, and then the system identifies the best similar frames based on the request of the users in PCA subspace, suitable for fire animation by avoiding morphing or blending techniques, unless the animator chooses morphing. The reason is that following this approach the generation of fire animation resembles a procedural technique of fire simulation.

2.3.2 Graphcut Textures and Flow-based Synthesis

The graph cuts algorithm [36] merges volumes of pixels as patches along minimum error seams to create new images and videos (Figure 13). Interactive merging and blending allow users to synthesize images easier. Graph cuts can also produce a better frames for transitions in video texture [53] by using two 3D spatio-temporal texture patches instead of blending. In this thesis, one of our objectives is to introduce other motion transition

approaches that perform better for our chaotic motions in comparison to transitions in video texture [53].

Flow-based Video Synthesis by Bhat et al. [5] is an interesting approach for synthesizing and editing arbitrarily long videos of natural phenomena from shorter video clips such as waterfall, smoke and fire. The idea is to capture the variation of the particles along user-defined lines that are marked as the direction of the flow in an input video, and then to synthesize new videos along another user-specified flow lines, as a way to edit the input video. This is performed by tacking texture patches of specific sizes moving with particles, and rendering these patches with blending and certain warping along the new flow paths.

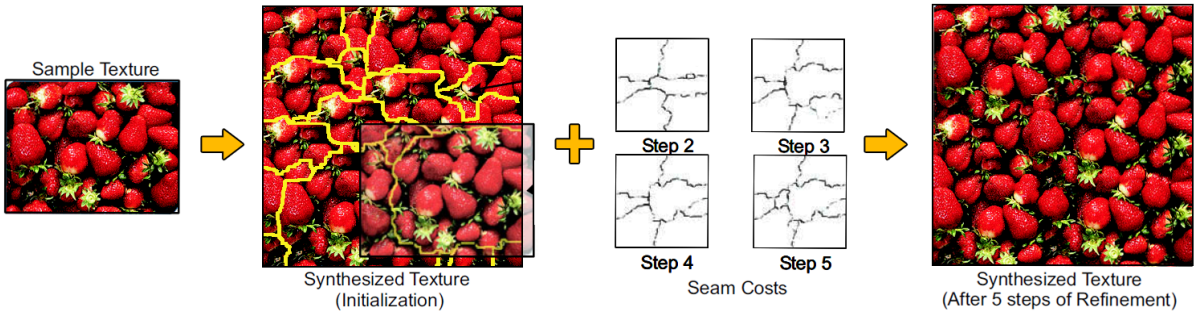


Figure 13: The process of synthesizing a larger texture from an example input texture using Graphcut algorithm. Reprint from Kwatra et al.[36]

2.3.3 Dynamic textures

Dynamic textures with linear models are presented by Doretto et al. [15] as a noise-driven linear dynamic system (LDS). Dynamic textures (Figure 14) are sequences of images of videos that exhibit certain stationary properties in time. (e.g. for sea-waves, smoke, turbulent fire, foliage, whirlwind). In their research, they analyze videos as visual signals. Doretto and Soatto extended the idea of dynamic texture in [16] by interactively modifying the temporal statistics of a video-based model of a dynamic texture to change the speed or intensity of the driving noise. In our research, we present another interesting way of synthesizing fire using Eigenfires.

Subsequently, Yuan et al. [73] propose a closed-loop LDS. For training stage of LDS, hidden state vectors are used to fit a linear dynamic system by considering state-space equation. Using the learned LDS and by sampling system noise the image can be synthesized. A work from Masiero and Chiuso [40] also deals with dynamic texture synthesis using non-linear dynamical models and compares it with the two previous linear dynamic texture techniques [15] and [73].

In our research, we also synthesize a few textures of flames, but we first solve the issues of mixing various fire samples, and then represent the outcomes of analysis stage (e.g. Eigenfires, features, etc.) in such a way that our own synthesizing process becomes interactively more convenient or predictable in comparison to some parameter changes methods in [15] and [16]. In this respect, in order to prevent colorful artifacts that usually appear in dynamic textures and also for a proper visualization of Eigenfires, we use grayscale and colorization procedures that consequently increase the performance.

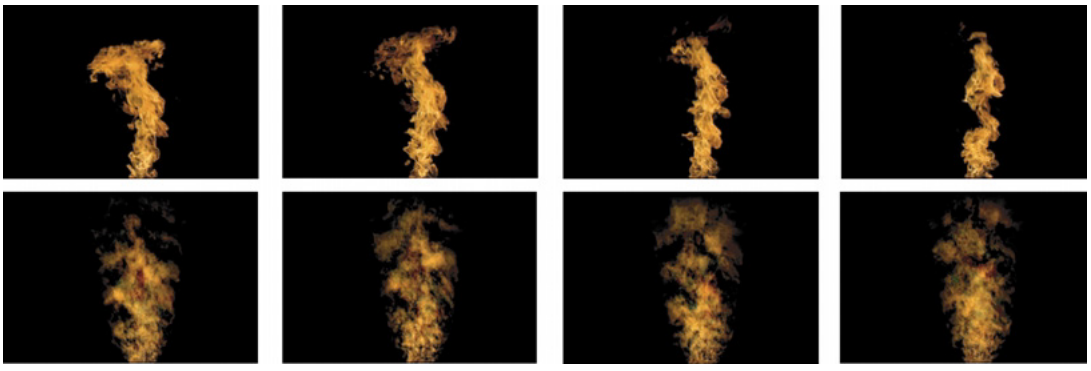


Figure 14: Examples of dynamic textures for fire. Top row: original sequence. Bottom row: the synthesized sequence. Reprint from Doretto et al. [15]

Chapter 3 Real-Life Fire and Pre-processing

Our large fire database includes variety of styles and motions we created and recorded to be able to produce new animation of fire, which does not necessarily exist in a recorded video but the animator may still generate it by combining the images of various videos of fire. In this chapter, we propose our new techniques to prepare masks for our fire samples that provide better results for fire synthesis and animation. These masks can also be utilized for transparency or blending with other non-solid backgrounds and environments. Our new methods are generalized to be applicable on videos with various lighting conditions, and therefore they can easily deal with possible undesirable noises in various recorded videos and still provide a consistent mask for the entire video. During the research, we have also tested our pre-processing algorithms on fire samples recorded by others (even large-scale fire) to make sure of their wider functionalities.

3.1 Recording Videos of Fire

For this research, we picked various materials as our fuel and ignited them in a fume hood cabinet, in absolute darkness in front of a black background. The videos are recorded from real small-scale flames with two professional HD video cameras at frame rates of 60 fps, in 720p format. Figure 15 shows this setup.

Figure 15: The setup for capturing small-scale flames in a fume hood cabinet

The three diverse fire samples are chosen to build the database for the results of this thesis. We name those samples respectively as fire *A (a torch with lighter fuel)*, *B (a pack of three solid fuel cubes)* and *C (a single burning cube)* throughout this thesis, and construct our results based on them. Figure 16 shows one frame of the selected three real-life flames without performing any pre-processing. Videos *A*, *B* and *C* contain 500, 1000 and 697 frames respectively in our experiments, which are comprised of 2197 images in total. The three fire samples have been selected to prepare a proper training set for our experiments that includes three sequences of entirely different in shape and size, and therefore we may better assess effectiveness of our algorithms. The lengths of the three fire samples do not necessarily need to be the same, as each frame is converted to a point in PCA space.

(*A*) (*B*) (*C*)

Figure 16: The recorded video samples without any pre-processing. (*A*) A small torch soaked in lighter fuel, 500 frames. (*B*) A pack of three solid fuel cubes, 1000 frames. (*C*) A single burning cube, 697 frames

The videos should be cropped aligned at the center of their fuel or burning object, so that our flames become centered and a smooth morphing can be performed later among them while reconstructing or synthesizing new images. Because of the health and safety regulations, the fuel could not be placed far from the background. Therefore, the brightness

of the flames illuminated the black background in some frames, and we have to eliminate it by our Hermite darkening process in next steps.

In this thesis, the results are presented as two-dimensional sequence of images, and we only work on one view of the videos even though Figure 15 shows a setting with two cameras. This setting is aimed for 3D reconstruction, which is not covered in this thesis. Nonetheless, we provide interested readers with the related references that can be used to convert the results of our current system from 2D to 3D model. We discuss this topic more in section 7.4.

3.1.1 Characteristics of Recorded Small-scale Flames

Features of our selected videos are as follows: Video *A* is a short flame with a small volume, repeatedly bends toward the left side of the view, and contains mixtures of colors such as blue and red around the fuel. This flame does not generate any smoke and the edges of the flame are clearly visible. Flame *B* consists of larger width and volume, in addition to a unique shape, in which constantly separates into smaller flames or branches every couple of frames. This feature is slightly visible in Figure 16(*B*) at the tip of flame. The third sample, fire *C*, includes a higher length, smaller width in comparison to the second sample, and more distortions due to external forces (*e.g. wind*) which are made intentionally during recording.

3.2 Contours

In the research presented on the topic of facial expression in [70], multiple control points were defined for both facial features and contour of the faces in order to produce new texture models. Subsequently, new facial expressions could be represented with parameter adjustments after performing PCA. Similarly, relationship learning in PCA space is discussed in [71], which is the association between 3D body meshes of a database and their corresponding 2D projected images. Thus, a new 3D human model can be reconstructed

from the 2D silhouettes of single photographs of individuals. Although we extract contours of flames similar to above mentioned papers, our methodology is different in concept, in which the segments of edges are not directly utilized for recognition step, but are used for pre-processing and post-processing purposes, such as background removal, image quality enhancement, pose normalization, custom masks and colorizing. The reason is that it is required to preserve smooth corners of flames in Eigenfires for proper reconstructions.

3.2.1 Contour Detection

The first step for preparation of images is detecting contour of the flames, which is based on detection of different color regions. There are many techniques that can be used to extract information about the boundaries of different objects from images. Standard Snake [32], Gradient Vector Flow Snake [72], and Contracting Curve Density (CCD) algorithm [22] are a few methods of shape matching to approximate the object's contour. However, based on our previous experiments and observations, we found them time-consuming, CPU intensive, and not proper for complex chaotic shapes, such as fluids. Utilizing these methods make the process of the detection non real-time or tedious for thousands of images of videos in HD format. They might produce incorrect contours with minor errors for consecutive frames, which are usually due to the amount of the noise in videos that are recorded in absolute darkness. The constant changes in burning state of the solid fuels also affect their final results. Because of the high intensity and brightness of the flames in comparison to our darker background, working on ranges of colors directs us to obtain proper contours. These contours are more consistent throughout the entire video while the shape of our flames deforms constantly.

HSV color space is employed by Chenebert et al. [8] to detect fire in a video or image by isolating the illumination component of a scene. Their methods can detect the pixels of fire regions in real-time and still distinguish the fire from other red alike objects using neural network classification, 2-stage color spectroscopy and decision tree classification.

For our contour detection, we take advantage of HSV color space and convert our RGB images into HSV color space using the following relations:

$$
\begin{aligned}
&V = max(R,G,B)\\
&S = \begin{cases} \dfrac{V - min(R,G,B)}{V} & if\ V \neq 0 \\ 0 & otherwise \end{cases}\\
&H = \begin{cases} 60(G-B)/_{S} & if\ V = R \\ 120 + 60(B-R)/_{S} & if\ V = G \\ 240 + 60(R-G)/_{S} & if\ V = B \end{cases}
\end{aligned}
\tag{3}
$$

where H, S and V are "*hue*", "*saturation*" and "*value*". The animator may define minimum and maximum thresholds for ranges of H, S and V to pick specific pixels, and create a mask image for them. Based on its cylindrical geometry, minimum and maximum pure colors may be set for Hue to find any combination of them (*e.g. finding orange color between red and yellow*). Similarly, Value and Saturation assist us to accept or reject that color based on the amount of colorfulness or brightness. This step can be repeated a few times if more regions for separate range of colors must be removed or kept untouched (*e.g. blue or green color of the fire in specific frames*). The mask for each repetition of the process is created using the following formula:

$$
\begin{aligned}
M(x,y) = \begin{cases} 1 & = \begin{cases} hue_{\min} \le p(x,y) \le hue_{\max} \\ saturation_{\min} \le p(x,y) \le saturation_{\max} \\ value_{\min} \le p(x,y) \le value_{\max} \end{cases} \\ 0 & = otherwise \end{cases}
\end{aligned}
\tag{4}
$$

$$
hue \in [0,360],\quad saturation \in [0,255],\quad value \in [0,255]
$$

where $p(x,y)$ is a pixel on the image in HSV space at pixel location of (x,y), and $M(x,y)$ is our binary map that can be used to extract our contours. Based on the cylindrical model of HSV color space, hue is usually a value between 0 and 360 degree. However, some libraries support other ranges for the value of hue (e.g. OpenCV 2.0 accepts it between 0 and 180. By defining a blending mode for each generated mask, such as "adding", "subtracting" or "difference", it is possible to find the union, intersection, or complement of regions throughout the entire video with the least amount of changes. This multi-color region is presented in Figure 17.

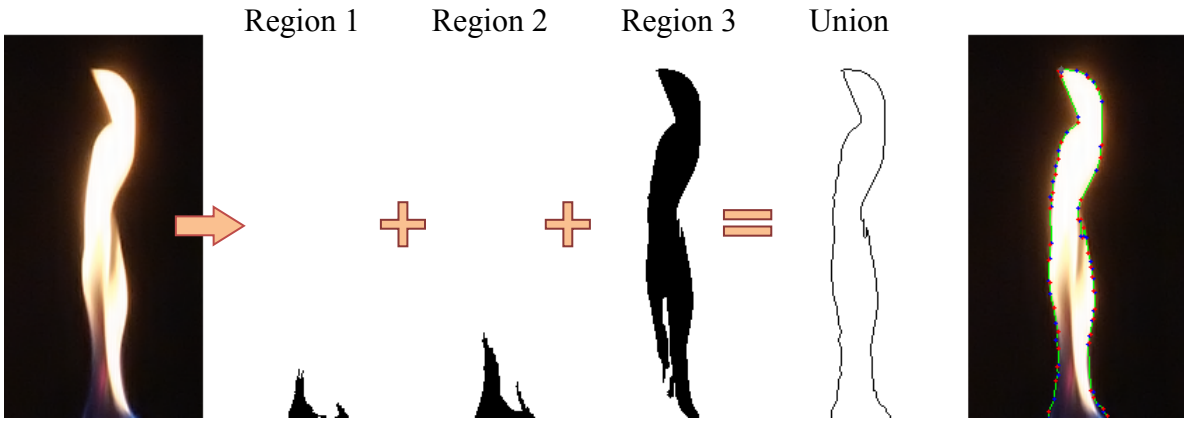


Figure 17: The mask equation is repeated three times with different HSV ranges to obtain mask regions for blue, red and yellow colors of the fire. The union of the three regions creates the consistent final mask and contour for the entire video

Morphological operations can be employed to filter out potential noises. Our application allows the users to apply real-time filters and operations (*e.g. Dilation, Erosion, Opening, Closing, Gaussian smoothing, contour optimization, hole removal, etc.)* based on a bottom-up layer style for minor improvements, to obtain a consistent contour for the entire video. It should be considered that all the employed techniques will be applied to the entire frames of the video to eliminate or reduce the need of manual parameter adjustments for specific frames. As a result, more inspection of techniques was required in comparison to regular image based approaches. Using Canny edge detection algorithm, binary thresholding

and border following algorithm [60], contours were retrieved from the binary mask we created.

3.2.2 Hole Removal Process

It should be noted that fire is not always a closed shape, and it can split or create a hole in a few frames. We therefore store our contours hierarchically in a tree, to be able to remove unwanted contours based on a depth level system assigned to each of them. The area inside each contour is also calculated and stored. We call this process as "*Hole Removal*", which can be *area-based* or *depth-based*, to eliminate specific small contours that appear unexpectedly during an entire video. Figure 18 shows the process of the contour detection and Hole Removal. A depth level of "0" means the contour is separate from other detected ones, and depth level of "1" indicates a contour which is one level inside another one, and so forth.

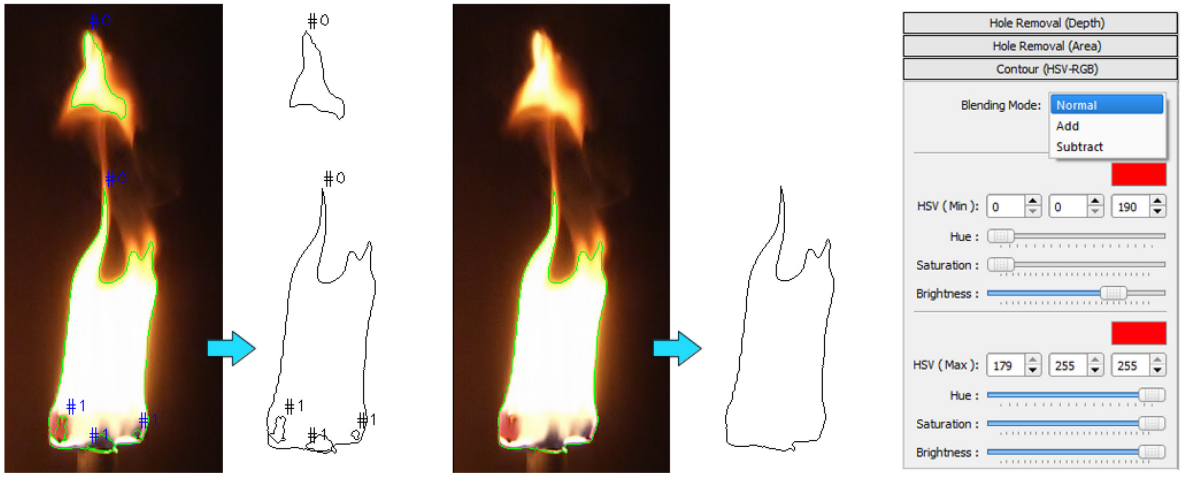


Figure 18: Contour detection and Hole Removal. The left two images: all detected contours with any possible depth levels. The next two images: contours with depth levels of zero, in addition to area-based restriction

Using this approach, different contours may be calculated and then used in certain scenarios. For instance, if the background of a small-scale fire video is dark enough, the

animator might be interested to separate smoke region from bright region of the fire for post-processing purposes or smoke visualization. It can be achieved by setting approximate values for HSV components and limiting the contour detection to a depth level of maximum "1" to obtain two contours in one step only, in which smaller one is the internal contour as the boundary of the bright fire, and the other one is the outer contour for smoke or soot.

3.2.3 Contour Optimization

Because we are combining variety of fire videos with diverse shapes, sizes and details, obtaining a coherent and consistent contour is the key part of the process. Hence, Douglas-Peucker algorithm [17] isemployed to optimize and approximate the contours whenever required, in which it concentrate on the selection of points rather than their deletion. Considering our contours as a few closed curves, there are two methods available for optimization of the contours using Douglas-Peucker algorithm.

In the first method, the two end points of the curve are selected which form a straight line. First point is an "*anchor*" and the last is a "*floating*" point. The algorithm looks for a point on the curve with the furthest perpendicular distance from the line. If the furthest distance is greater than a user-defined tolerance, the algorithm recursively divides this line by selecting the furthest point as the new floating point, and in each step the floating point moves toward the anchor. Otherwise, by meeting the condition of tolerance, the line represents an optimized segment of that part of curve. This way anchor is moved to the floating point and the last point on the line is assigned as the floating point, and this procedure is repeated. At the end, all anchor points comprise the final simplified contour. In other words, the points which had not been marked as an anchor point during the process should be removed from the curve. An example of this approach is illustrated in Figure 19 for by different values for tolerance. From left to right the maximum tolerance is increased.

The second method is exactly the same as the techniques mentioned above, except the fact that the floating points in each iteration should be stored in a stack or vector, so that

the new floating point can be chosen from this stack in next iterations to accelerate the process. It must be mentioned that the outcome of the second method might be slightly different from the previous method.

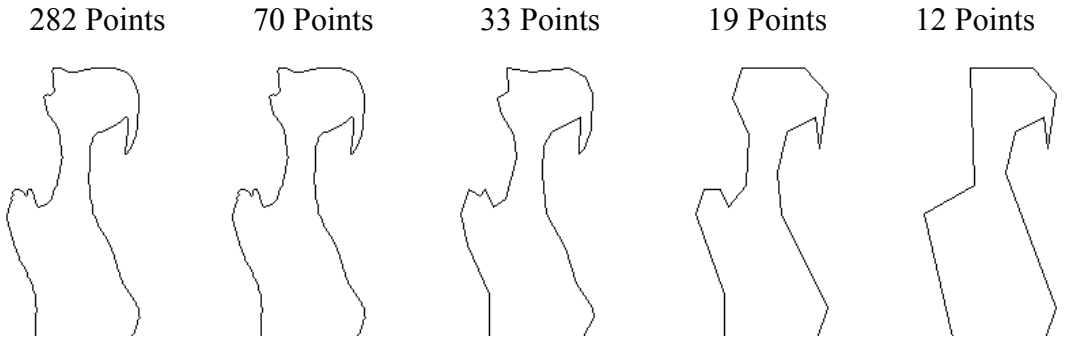


Figure 19: Optimizing contour of a flame using Douglas-Peucker algorithm by setting different values for tolerance

3.3 Background and Smoke Removal

As we already discussed, luminosity of flames brightened both background and smoke. We are planning to diminish that undesirable brightness by a darkening process outside the contours. It is important to preserve the smooth shape of the fire, specifically around contours. We are trying to take advantage of this smoothness to produce images with soft corners while calculating eigenfires, and consequently create realistic flames, in case compression or dimensional reduction is planned by animators. In majority of the previous works with PCA such as [64][65][70][71], the images are usually modified in such a way that the background is ignored outside the target and the shape is left with sharp boundaries. The reason is that they are mostly interested in detection and comparison of features inside the objects, while for fire both areas inside and around the contour are entailing important features that discarding them has a negative impact on how realistic the new reconstructed fire looks.

3.3.1 Hermite Background Removal

At this stage, the training images of the videos must be converted to grayscale to be prepared for PCA calculation. Next, the background and smoke should be darkened or completely removed. Our experiments show the fact that excluding and removing smoke from the rest of the fire at this level will result a more accurate recognition of similarities later by performing PCA. In other words, the results of automatic identification would be closer to what humans expect to see, and accordingly incorrect identification basically due to high length or volume decreases considerably. It should be mentioned that with choosing a large training set, as we did for the database, incorrect identification here does not necessarily mean two entirely different flames, but rather less visually similar to be selected as the next frame of an animation. This topic is discussed in more detail in next chapters.

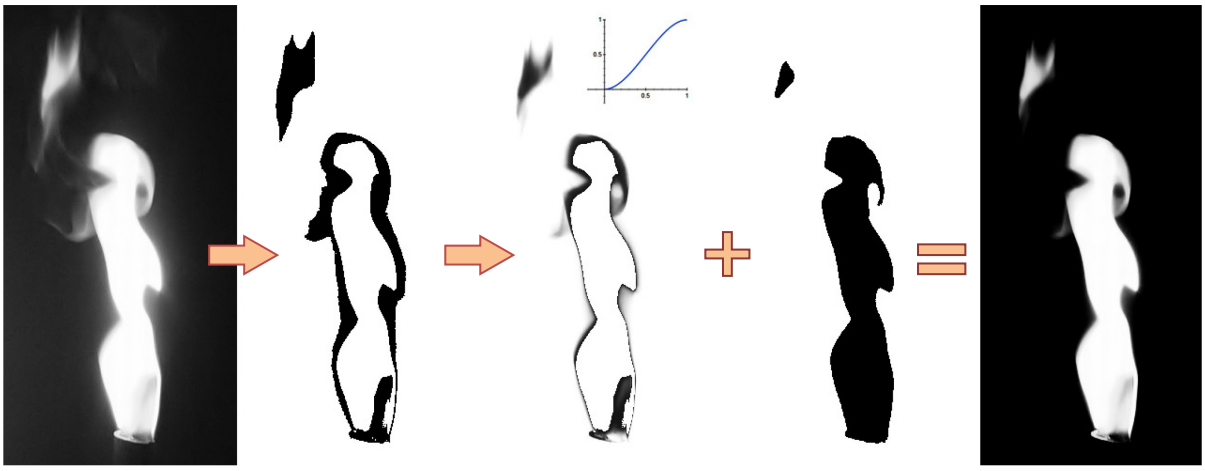
After experimenting with linear, cubic and Hermite curve approaches, the second Hermite basis function is found to yield smooth transitions from pixels with very high intensities to pixels with low intensities for fire samples. The equation of this basis function and our intensity criteria are as follows:

$$Intensity(S) = -2S^3 + 3S^2 \qquad S \in [0,1] \tag{5}$$

$$S = \frac{p(x,y) - I_{\min}}{(I_{\max} - I_{\min})} \qquad if\ I_{\min} \le p(x,y) \le I_{\max} \tag{6}$$

where S is based on two values of maximum and minimum intensities selected by animators. The value of any pixels with intensity of higher than $I_{\max}$ does not change, but any pixels lower than $I_{\min}$ become zero or black, which create "*intensity mask"* (Figure 20b). Any pixel between them is gradually darkened in respect to $Intensity(S)$ and produces a smooth transition (Figure 20c). To get better results, the calculated contour of a flame can be considered as a bitmap mask to ignore any changes of intensity for pixels that reside in the

interior part of the contour (Figure 20d). The result of this technique is shown in Figure 20. Note that the mask obtained from contour detection is inverted in this figure.



(a) (b) (c) (d) (e)

Figure 20: Hermite background removal process. (a) Original image converted to grayscale, (b) *intensity mask* with choosing $I_{\max} = 220$ and $I_{\min} = 110$, (c) modified pixels obtained from multiplying *Intensity*(S) with current pixels of the grayscale image in respect to the intensity mask, (d) the mask from contour detection, (e) the final image with smooth edges

3.4 Pose Normalization

While our cameras' positions are fixed, we have three videos of flames with different heights and widths (e.g. fire C is much taller than fire A), and created by different fuels. Therefore, they have to be repositioned and aligned at the center. Scaling and translation are performed once for each video which is easily done via our user-interface within a few clicks. However, rotation is not necessary unless the purpose is synthesizing fire images without respect to direction. For recognition of hand gesture by Birk et al. [6], all the images were subject to a simple normalization procedure. Hence, for the irregular shapes of fire, we require a more advanced procedure for direction normalization.

3.4.1 Removing Direction Factor using Weighted Least-squares

For the rotation, we eliminate direction factor from fire, which is automatically performed by fitting a line on the vertices of the calculated contours using weighted least-squares algorithm. Using this line, we can rotate our image around the start point that is located in lower part of the line. Unlike linear and nonlinear least squares regression, weighted least-squares regression includes an additional scale factor, as weight, to improve the fit. It works by incorporating extra nonnegative constants or weights, associated with each data point into the fitting process. Weighted least-squares regression minimizes the error estimate:

$$S = \sum_{i=1}^{n} \mathrm{w}_i \, {r_i}^2 \tag{7}$$

where n is the number of data points included in the fit and w_i are the weights. The residual for the *i*-th data point r_i is defined as the difference between the response data and the fit to the response data at each predictor value, and it is identified as the error associated with the data. If the variances (σ^2) of the measurement errors are known in the data, then the weights are given by the following equation:

$$\mathrm{w}_i = \frac{1}{{\sigma_i}^2} \tag{8}$$

If estimates of the error variable are only available for each data point, it is possible to use those estimates in place of the true variance. If the variances are not known, we may specify weights on a relative scale to the data points, or by following a particular form and measurement.

After finding this line, we should trim it to calculate its length and to get end points. Because it is not clear which point is the first or the last point along this fitted line, all the points should be projected into the line, and therefore farthest points are the end points for cutting the line. Let point (x, y) be projected to point (X, Y) onto the line. Intersecting the line joining them with our line ($Y = mX + b$) gives us the new coordinates. Let (y_2, x_2) and

$(y_{1,}x_1)$ be any two points on the line, and m as the slope of the line, the following equations perpendicularly project a 2D point (x, y) onto a 2D line:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{9}$$

$$X = \frac{my + x - mb}{m^2 + 1} \tag{10}$$

$$Y = \frac{m^2 y + mx + b}{m^2 + 1} \tag{11}$$

Next, the contour optimization approach with Douglas-Peucker algorithm [17] is employed to simplify the contour, and to eliminate unnecessary vertices. This process is illustrated in Figure 21. Sometimes the line might be constructed incorrectly in another direction if the width of a flame span horizontally or become shorter in height. We suggest three techniques as remedy to this problem:

1) Optimizing the contour and converting it to a low poly shape, in such a way that it only contains two points at the root. This is the best approach in our experiment.
2) Increasing the number of points on the contour around the fuel.
3) Special weights may be assigned to data points for weighted least-squares algorithm.

Although auto-rotation yields interesting results for shape recognition, we allow the user to disable it in our user interface as an optional choice. The reason is that the direction factor can be one of the main features of Eigenfires, which is explained in next chapters. By enabling direction normalization, we may remove the direction feature from Eigenfires, and instead always synthesize an image vertically, and then manually rotate the fire or use the angles that we stored with the direction line.

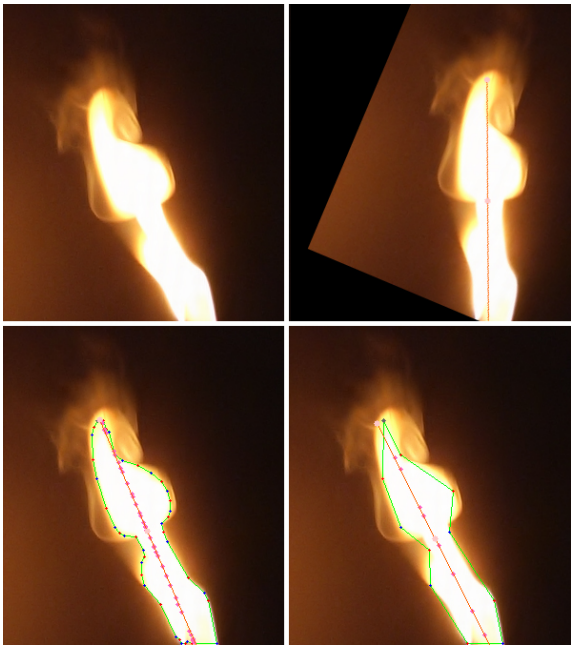


Figure 21: Pose normalization. Top row: (left) original image, (right) auto-rotated image after pose normalization. Bottom row: two contours with different levels of optimization that show almost the same direction. The fitted line is illustrated with a pink color and projected contour points on it

Chapter 4 Eigenfires and Analysis of Fire

In the beginning of this chapter, we first review the main stages of our system and outline a summary of what should be performed to acquire the features of fire using principal component analysis. By analyzing our fire video and computing the necessary parameters, the final fire database can be prepared. This database then will be utilized in next chapters to produce new animations of fire using our proposed recognition approaches.

4.1 Stages of the System

Our system is divided into three major stages: (1) *Training stage*, (2) *Analysis stage*, (3) *Animation stage*. This chapter covers the thorough details of the first two stages, and the last stage is discussed in Chapter 5. A short review of these three stages and their general concepts are as follows:

Training Stage:

This stage allows the animators to choose a name and path to build a custom database of fire, and includes the following main processes:

1) Performing pre-processing tasks on the video, if it is not already done.
2) Calculating Eigenfires using PCA (treating as 32-bit floating-point images), in addition to eigenvalues and average of fire images.
3) Exporting Eigenfires as series of 8-bit grayscale images for visualization.
4) Projecting all training images of the video into eigen space (fire space) to obtain weights corresponding to each Eigenfire

The training set can be either a combination of various recorded videos of fire, or only a single long fire video but with plenty of motions. This provides the artists with the maximum control over the way a new fire can be created.

This stage can be carried out either "*on-the-fly*" or in "*off-line*" mode in our system. The on-the-fly mode only calculates and stores compulsory steps directly in the system's memory that is faster when we are working with short length videos. In off-line mode, the entire steps will be stored on the disk as a database, in such a way that measured thresholds, details of video and other small parameters are written into an XML file and the rest of the large portion of data (e.g. EigenFires, average image, etc.) are stored in packages of binary formats for efficiency reasons.

Analysis Stage:

At this stage, what we deal with is series of vectors that contain decimal values. The data in these vectors are called "weights" with positive or negative float numbers. Calculated Eigenfires from training stage are also numbered and sorted based on their importance and contribution. Each image that is sent to the training stage is now represented as a vector of weights, and the index of each weight in these vectors shows how much of the corresponding Eigenfire should be used to create that image. For instance, by adding the result of [30% of Eigenfire #1 + 15% of Eigenfire #2 + …] to the average of all training set, we can reconstruct one frame of our videos. However, instead of these percentage values, we deal with positive and negative weight numbers that indicate how much the corresponding Eigenfires should be added or subtracted, in order to represent one final image.

As a result, training images can be visualized in a 3D view by showing their corresponding weights values for the first three Eigenfires (first three dimensions), which we call 3D representation of images in PCA subspace. Since each image can be displayed as a point in 3D space, consecutive frames of the fire video can be visually connected and tracked in real-time for pattern analysis of fire animation in PCA subspace, which are called "paths"

in our system. Similarly, charts can be generated for further analysis of the weights in higher dimensions that represent our images. The video can also be compressed by ignoring some high frequency components of the fire, which means ignoring Eigenfires with insignificant contributions.

Animation Stage:

This stage commences by loading the prepared database from the previous stages into the system. We need to come up with a plan to split the video into proper small fire animations and name them based on their style of motions such as “slight wind”, “strong wind”, “distortion”, “regular motion – fire A”, etc. Next, the animator may define a scenario and insert the desired motions into the motion list. Then, our system is responsible to create loops, motion transitions and finally a new animation by connecting selected motions.

The animation techniques are based on our proposed recognition algorithms that are presented in Chapter 5. “*Recognition”* in our system is a process in which the shape of a flame is identified to be slightly similar to another flame in a video. We therefore propose a few motion transition algorithms using both recognition and fire synthesis approaches that can connect two different animations by finding a suitable pattern from the last frame of one animation to the first frame of another one.

These recognition algorithms require us to define a few “thresholds”. Technically, a minimum threshold means calculating distances of weights corresponding to two images in PCA subspace and then deciding whether that distance is above our desired minimum threshold. In other words, it means how much two images are allowed to be similar. Therefore we need to define a threshold of slightly bigger than zero. A minimum threshold of zero, or more precisely an epsilon value, means recognizing exactly the same frame in the video that is not in our interest, because we are looking for other specific similarities and patterns that are useful to generate new animations. One of the benefits of PCA is that the detection of similar images and their patterns can be traced in our 3D view as well.

4.2 Calculating Eigenfires

Our primary idea of employing principal component analysis (PCA) for fire analysis is inspired by Pengcheng Xi et al. [70] on the topics of facial expression, and [71] about 3D human body cloning. In these work PCA was utilized to reconstruct sophisticated textures and shapes, other than the well-known use of PCA in the areas of recognition of objects. Unlike mentioned methods, we do not put our effort on interpolations among feature points or contours, but on video-based techniques (e.g. [39][53]) or image synthesis (e.g. [15][16]) using real fire samples.

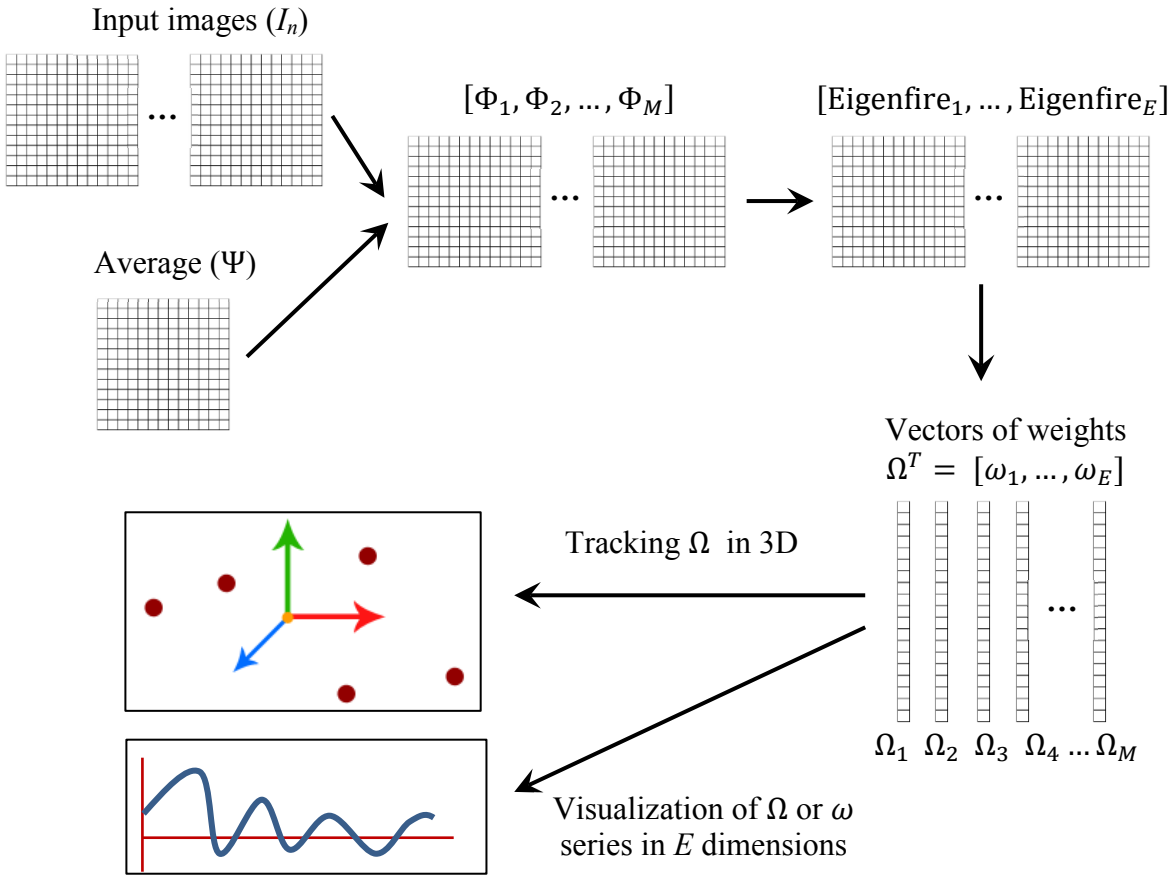


Figure 22: The process of computing Eigenfires, corresponding weights and visualization in PCA subspace

Eigenface is a popular face recognition algorithm proposed by Turk and Pentland [64][65], and our system is implemented based on their algorithms, but for analysis of fire. We later introduce our approaches suitable for generation of new fire animations using either recognition of similarities or combining our Eigenfires..

The idea of principal component analysis is to linearly find a series of vectors that describe the distributions of data (intensities of fire images) among all frames of the video. These vectors are the eigenvectors of the covariance matrix for videos of various kinds of fire, and we call these vectors as "Eigenfires", which appear as smooth combination of flames, expressing actual features of fire. This is because of the repetition of similar motions in a fire that occurs many times even during a few seconds of our recorded videos, in addition to occurrence of small amount of changes between each two consecutive frames using a frame rate of 60 fps. The concept of our procedure is illustrated in Figure 22 for better understanding

By performing all the required pre-processing tasks, the main part of training stage begins. The wide images of our three samples are cropped to be 410x670 resolutions to avoid unnecessary calculations for the black regions of the background (outside the region of fire), but the original size remained unchanged to get the highest possible quality of the fire from the videos. Let a frame of the pre-processed video as I(x,y) be a vector of length N = x × y, in which x and y are the width and length of our video resolution, and it is converted to grayscale, and its background and smoke are darkened. Considering N as dimension of eigenvectors, each image is dealt with as a single data point in N = 410 x 670 = 274,700 dimensional space based on our image resolution.

Let M be the number of frames in the video, and image I_n be n-th frame of the video, the average of fire Ψ is calculated as:

$$\Psi = \frac{1}{M}\sum_{n=1}^{M} I_n \tag{12}$$

This average image is relative to the length and shapes of selected video samples and preprocessing. It must look blurred around the contours of the fire and no sharp layers of flames appear on it, so that we receive the least amount of artifacts during reconstructions. Based on principal component analysis, we need to look for *M* orthogonal vectors of u_k as the best description for the distribution of the intensities of the fire in the video. Each u_k is computed using the following relation that is a maximum:

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}(u_k^T\,\Phi_n)^2 \tag{13}$$

subject to:

$$u_l^T\,u_k = \begin{cases}1, & if\ l = k\\ 0, & otherwise\end{cases} \tag{14}$$

where λ_k is the eigenvalue and u_k is the eigenvector of the covariance matrix C, and Φ_n is the difference of each image from the average image as: $\Phi_n = I_n - \Psi$.

We obtain the covariance matrix C in the following manner:

$$\mathrm{C} = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\,\Phi_n^T = AA^T \qquad A = [\Phi_1, \Phi_2, \Phi_3, \dots, \Phi_M] \tag{15}$$

The matrix C is *N* by *N*, which makes the calculations difficult to deal with. Because the number of images in space is less than the dimension of space (*M* < *N*), the number of meaningful Eigenfires (*E*) is equal to the number of our data points or fire images minus one as *E* = *M* – 1, rather than *N*, and the remaining Eigenfires will have a value of zero for their eigenvalues. For this purpose, *N* dimensional eigenvectors (Eigenfires) can be determined by first solving a smaller matrix of *M* by M, rather than *N* by *N*, to reduce the computations significantly. Therefore, we create an *M* by *M* matrix of $L = AA^T$, where:

$$L_{mn} = \Phi_m^T \Phi_n \tag{16}$$

Then we need to compute *M* eigenvectors of *L* as V_l , and use them in the following equation that describes linear combination of the *M* frames of the video to construct our final Eigenfires:

$$eigenfire_l = \sum_{n=1}^{M} V_{l_k} \Phi_k \qquad k = \{1, \dots, M\} \tag{17}$$

As it is obvious, the calculations are decreased from a huge order of *N* to a small order of *M* only. Finally, all Eigenfires will be sorted based on their eigenvalues, in such a way that the largest eigenvalue corresponds to our first Eigenfire, and then we store them into the database (*M - 1* meaningful Eigenfires).

Robin Hewitt gives a comprehensive explanation about implementation of PCA through a series of articles in SERVO Magazine [27]. It is also an impressive reference for better understanding of principal component analysis and related issues. Although the source code is based on traditional C-style programming interface of OpenCV library (Open Source Computer Vision), it can be easily ported to newer object-oriented versions. A document by Smith [55] is also an excellent source of information about mathematics and analysis of data using principal component analysis.

4.3 Contribution of Eigenfires Using Weights

By projecting our fire images into its Eigenfire components, a vector of weights is constructed. Each weight represents contribution of its Eigenfire to form a fire. By assigning a negative or positive value for weights, we may reconstruct a new flame which can be a combination of three flames. The experiments suggest that using about 50% of Eigenfires (e.g. 1100 Eigenfires, out of 2196, in our main database), we can properly identify similar

frames through recognition stage, because our reconstructed images only lose some noise appearance pixels inside the boundaries of flames, and it is negligible in detections. Likewise, employing a higher numbers of Eigenfires but still below 50% of the total number of Eigenfires would be sufficient for reconstructing high quality images. For instance, we may only consider the first 700 Eigenfires to create fire images which probably produce some unpleasant artifacts outside the flames, and then we can enhance the quality by applying Hermite background removal approach. The definition of a weight (ω) and projection is subtracting an image I with the average of images Ψ, and then multiplying it with its Eigenfire denoted as $eigenfire_k$:

$$\omega_k = eigenfire_k^T\,(I - \Psi) \qquad \Omega^T = [\omega_1, \omega_2, \dots, \omega_E] \tag{18}$$

where E is the number of Eigenfires, and the weights would be saved in a vector Ω. The weights are numbers in range of [-15000, +15000] for our samples, and this range changes based on the training set of images. Note that in many systems, weights are also referred as *coefficients*, because a fire image can be represented as a linear combination of the Eigenfires. As part of the training stage, all the frames of the video (training set) will be transformed into fire space once and then these vectors of weights are saved into the database to decrease the processing time for generation of animations later.

4.4 Visualization of Eigenfires in 2D and 3D

The first 9 Eigenfires and another 3 with lower eigenvalues are showed in Figure 23. For more detail, consider a blank image as I. The influence of a negative value for a specific weight is similar to filling image I with a mask which is dark pixels of corresponding Eigenfire visible in Figure 23. Similarly, a positive value fills image I using lighter pixels of that Eigenfire. The gray regions remain almost unchanged, such as the background. Although this is not a precise description, it is what you may visually perceive by changing a weight for one specific Eigenfire.

Since eigenvectors are stored in separate 32-bit floating point images, they should be converted to 8-bit gray images to be displayed as Eigenfires for visualization purposes. Since our eigenvectors may contain very small decimal numbers (positive or negative), we should perform the following scaling and shifting on the pixel values:

$$scaling = \frac{255}{pixel_{max} - pixel_{min}} \tag{19}$$

$$shifting = \frac{-255\, pixel_{min}}{pixel_{max} - pixel_{min}} \tag{20}$$

Where $pixel_{min}$ and $pixel_{max}$ are respectively the smallest and the largest pixel values among *N* dimensions in each eigenvector, which are small decimal values in our experiments.

It might be useful to know that there are some libraries and tools that can be used to read and display complex 32-bit floating-point images directly, such as OpenEXR that is a high dynamic-range (HDR) image file format from Industrial Light & Magic. However, Eigenfires may not be correctly displayed by them as we visualize in this thesis.

To elucidate the pattern among all the data points in PCA subspace, the first 3 principal components of all training images are visualized in 3D subspace in Figure 24. This figure shows that the data points related to each fire sample are distributed apart from each other. The center of the coordinates system is a vector of weights with zero values that produces an average image. In the right side of Figure 24, it shows that the fire can be tracked over time in 3D PCA subspace, like a "*path*" or a curve in three dimensions that is connecting the corresponding data points (consecutive frames of the fire video).

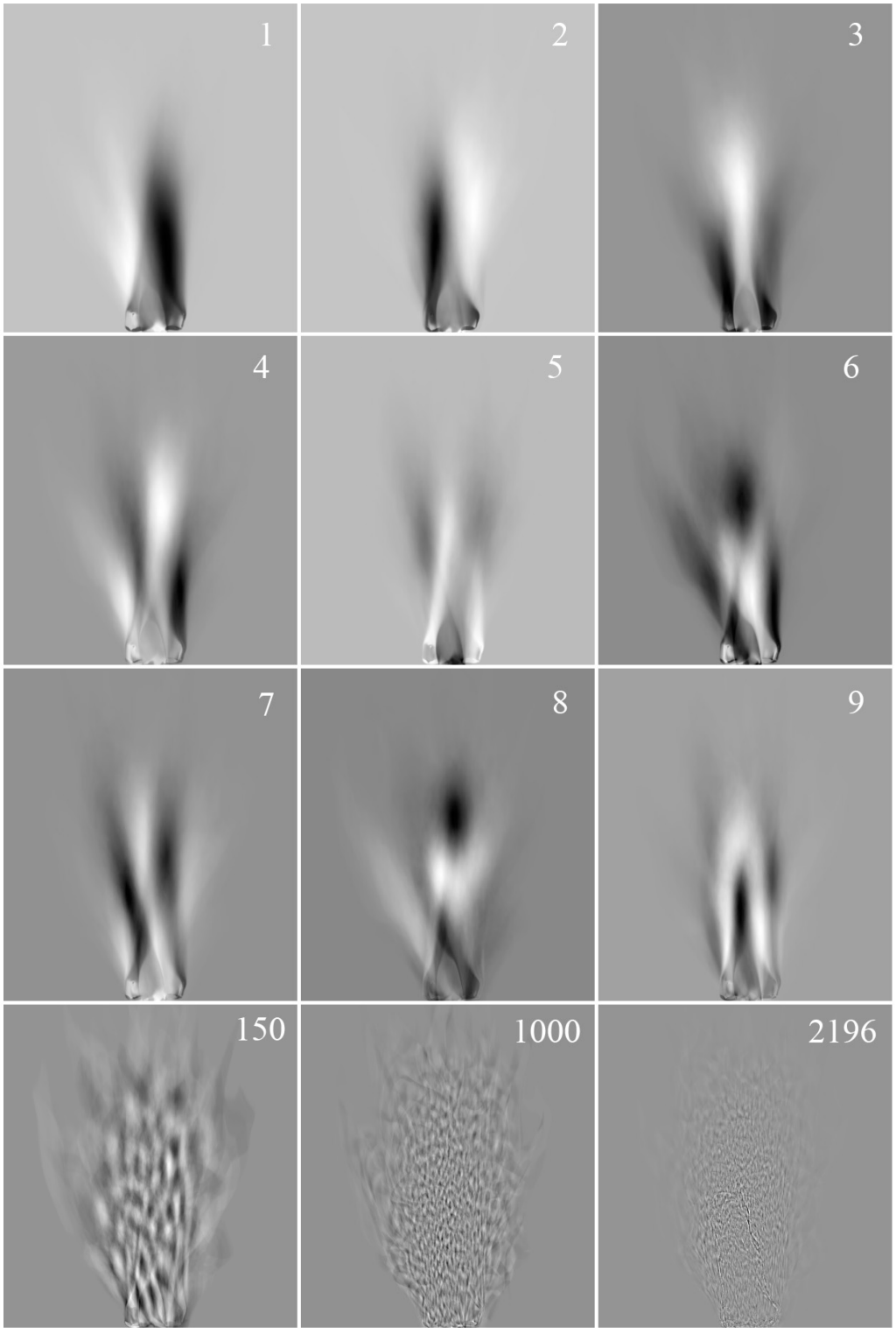


Figure 23: The first nine principal components, in addition to principal components of 150, 1000 and 2196, are viewed as Eigenfires. Eigenfires with smaller eigenvalues produce fine details and the edges of our flames, depicted as some sort of noise

This procedure can be followed frame by frame in both training stage and recognition stage for the first 3 principal components or any other principal components by inserting the corresponding weights to a vector we dedicated for 3D visualization. This helps us to analyze the motions of the fire based on their main features, and monitor the trend of changes for different types of flames simultaneously. Using this procedure, later animators may trace new generated images, as a few additional data points for validations.

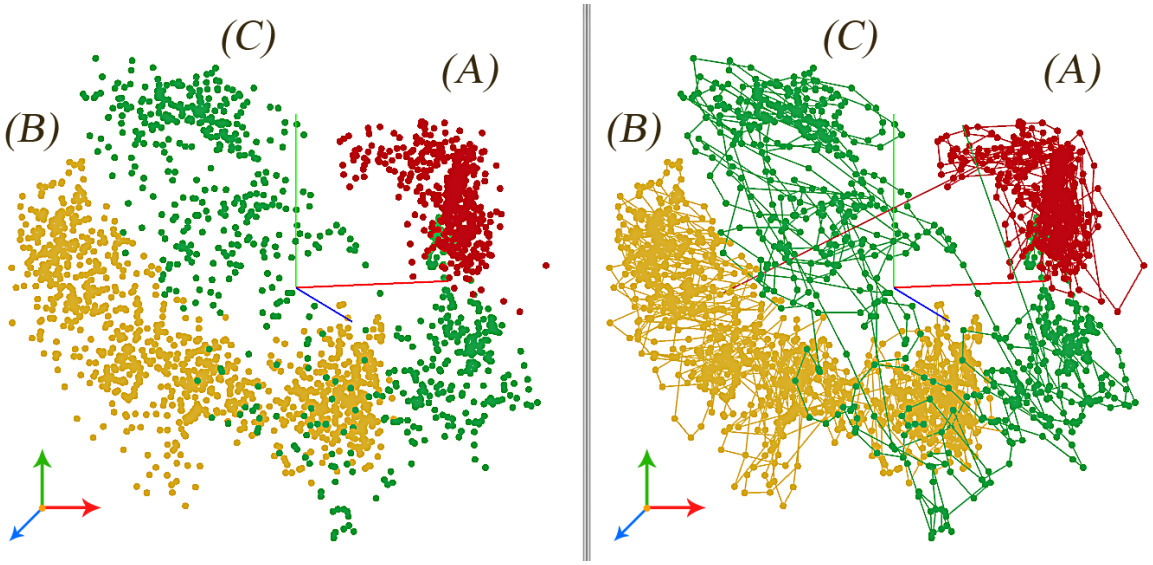


Figure 24: All training images projected in 3D PCA subspace, illustrated as a series of weights for the first three Eigenfires. Red, green and blue axes are the 1st, 2nd and 3rd principal components respectively. Red, yellow and green points are labeled with their relative fire's name (remember: Fire A, B and C). The image on the right shows "*paths*" which are the relations of consecutive frames by connecting each frame to the previous frames of the video using lines

4.5 Fire Recognition Procedure and Thresholds

The typical definition of recognition or identification in PCA related literature is detecting one abject in two images that are mostly the same. Recognition may also fail if the objects are very different, and it is called as an unknown recognition. However, "recognition" in our system is slightly a different process, in which the shape of the fire is identified to be "somewhat" similar to another flame, instead of being "exactly" the same. In

this way, recognition never fails for fire, because we are always looking for a frame that is not exactly the same but suitable to be chosen as our next frame of a fire animation. Unlike other systems such as face recognition, failure in our system means producing unrealistic animation, and it can be fixed by using longer fire videos and various types of flames. Our classification here is related to each style of fire and it can be even one class.

To determine which image (b) is providing the best description similar to another fire (a), a frame that minimizes the Euclidean distance D_{ab} is our choice:

$$D_{ab} = \|\Omega_a - \Omega_b\| \tag{21}$$

where Ω_a is corresponding weights of an image in the video, already stored in the database, and Ω_b is the vector of weights for a new projected image or one from the database if the video is large enough. Figure 25 illustrates the definition of this equation.

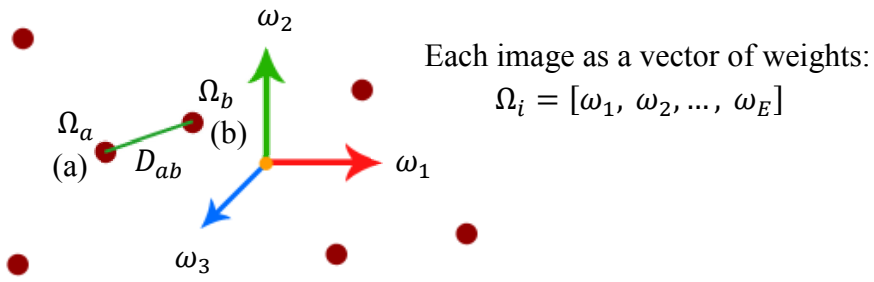


Figure 25: Each fire image becomes a point in high E-dimensional PCA space. A few images are shown in PCA 3D-subspace as red points for visualization. The smallest distance among all the images belongs to images (a) and (b), by computing the distance D_{ab} between their corresponding vector of weights, Ω_a and Ω_b, in three dimensions for this figure or in E dimensions by considering other remaining dimensions. This means images (a) and (b) are the most similar images

Because our system is designed for fire, the experiments shows that a large video (as training set) of at least 20 seconds is sufficient for obtaining successful recognition, and therefore we may only look in the same database for similarities. This enables us to generate our techniques in real-time for high resolutions. Remember that shapes of flames are

controlled by us in pre-processing stage (e.g. by ignoring smoke), and that is the key for a better recognition.

Three different frames from our fire videos are shown in Figure 26. These frames are recognized similar to each other by finding the shortest Euclidean distances among them in PCA subspace. By calculating and defining an important minimum threshold θ_{min} for each fire video, we avoid slow-motions or fast-forwards in animations, because we are not interested in repeating very similar flames over and over again or displaying parts of the videos. Unlike [53], our aim is random selections of similar frames. Note that we reject the results that fall below this θ_{min}, and instead accept the rest based on their shortest distance.

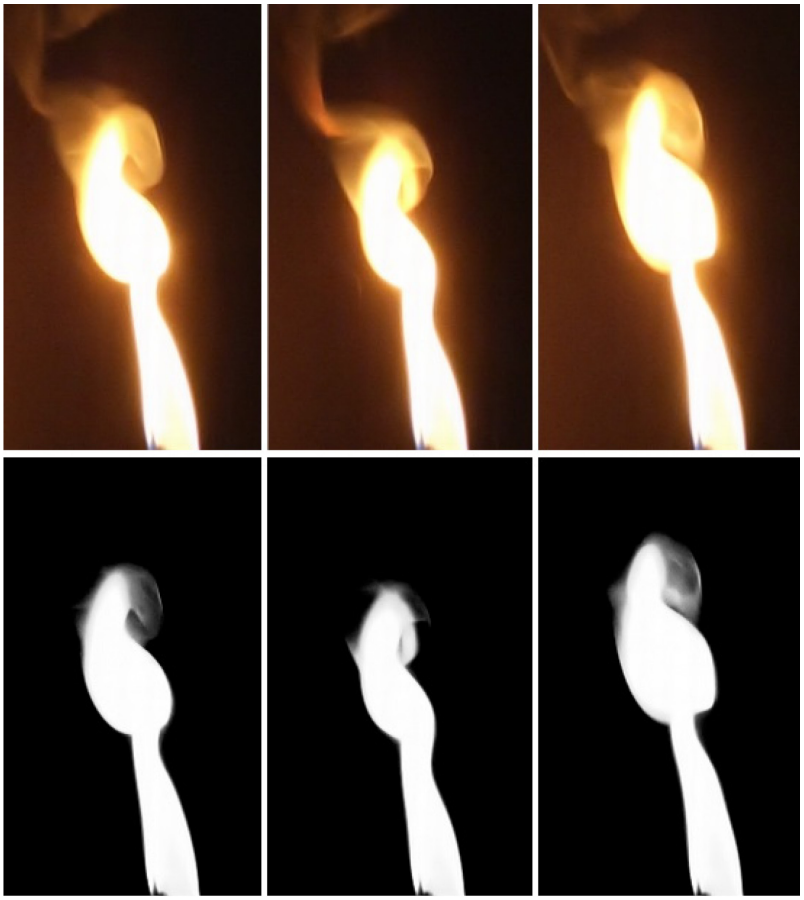


Figure 26: Three frames are found similar to each with the shortest Euclidean distances without choosing θ_{min}. Second row shows what we compare in PCA subspace

4.6 Analysis of Eigenfires and Weights

It is important to know that combining various videos of flames requires us to gather and store at least some subtle but important details in PCA subspace about each type of flame; Some details such as minimum or average of *thresholds* for each type of flame. Since fire *A* is much smaller than fire *C* in our database, the possible thresholds are up to 5 times different for their succeeding frames in each fire.

To clarify, the weights of the first 10 Eigenfires are illustrated in Figure 27 using clustered charts. For each video sample, 31 consecutive frames are depicted in clustered columns for comparison of the patterns and changes of the weighs. It is clear in the charts that the first 10 Eigenfires usually have the highest weights for each flame, but it does not necessarily mean the first Eigenfire, with highest eigenvalue, is always going to be the most important feature for all of our three flames. For example, although the 1st Eigenfire of fire *C* is almost neutral (zero value) for frame number 20, it is near to the maximum peak of the chart for the 3rd Eigenfire in the same frame. This is marked on the last chart of Figure 27.

To obtain more information about the changes of weights, the weights corresponding to one frame of our three fire animations are illustrated in Figure 28, using the first 100 Eigenfires. This figure shows that Eigenfires larger than number 50 have small values which are mostly some values less than 10% of the largest weight in the graph.

It is also a good practice to split the complete diagrams into proper sections at this level, and then to calculate the average of those weights as $\theta_{average}$, in addition to the values of the peaks for that section. This is usually sufficient to be done for the first 30 Eigenfires that keep essential features of fire, and then to be stored in the corresponding database. The reason is that users of the system may utilize this data later as a hint for initializations when producing a new fire animation or synthesizing an image based on approaches we discuss in Chapter 5. Otherwise, animators have to make a guess for the required parameters, which is still very easy and straightforward but requires the experiments to be repeated a few times to find out which values give them more accurate results.

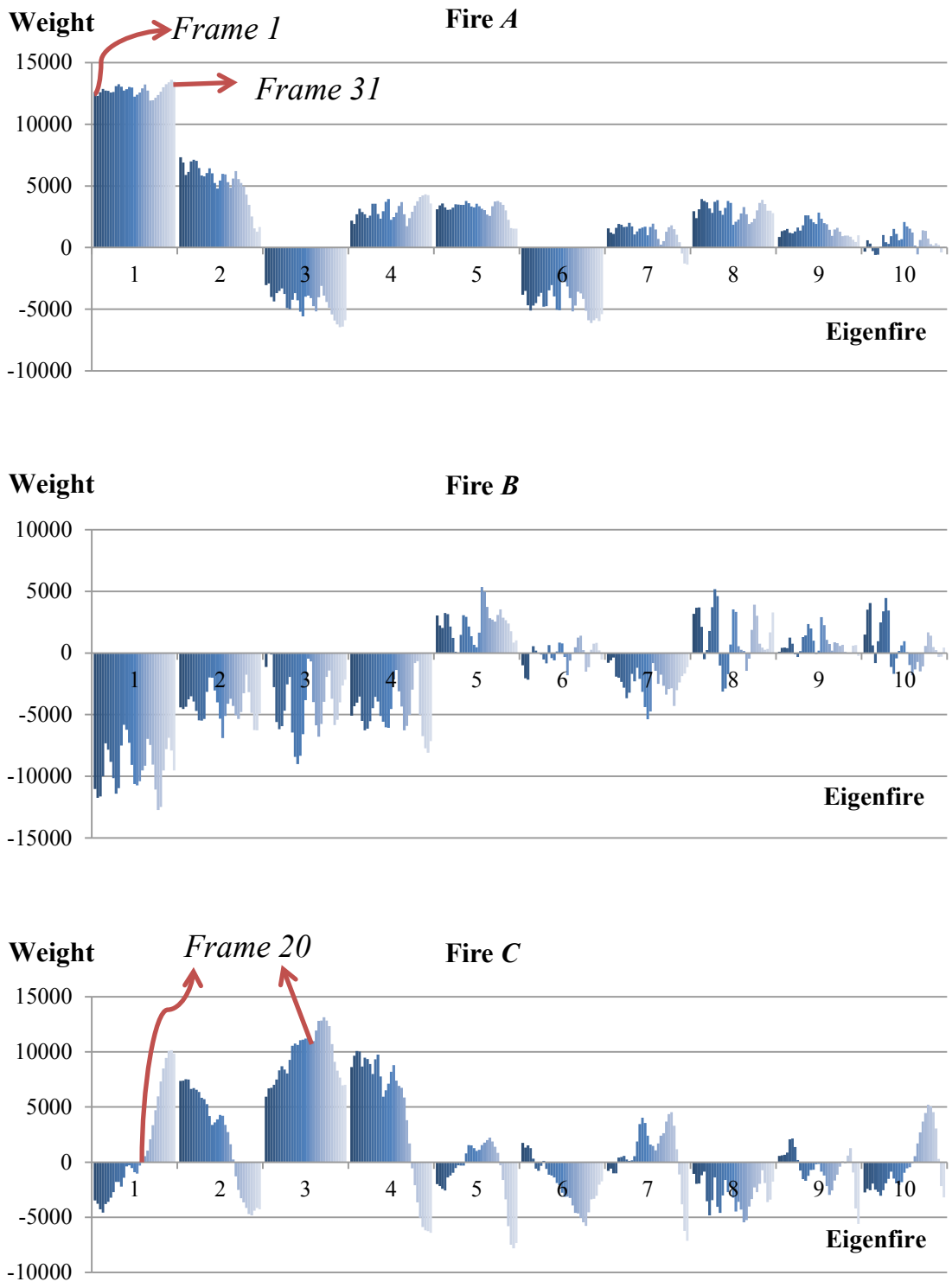


Figure 27: Weights assigned to 31 consecutive frames of our three videos are shown in clustered columns using the first 10 Eigenfires. Darkest color on left side of each clustered column starts with frame number 1 and brightest one ends with frame number 31 on the right side for each Eigenfire

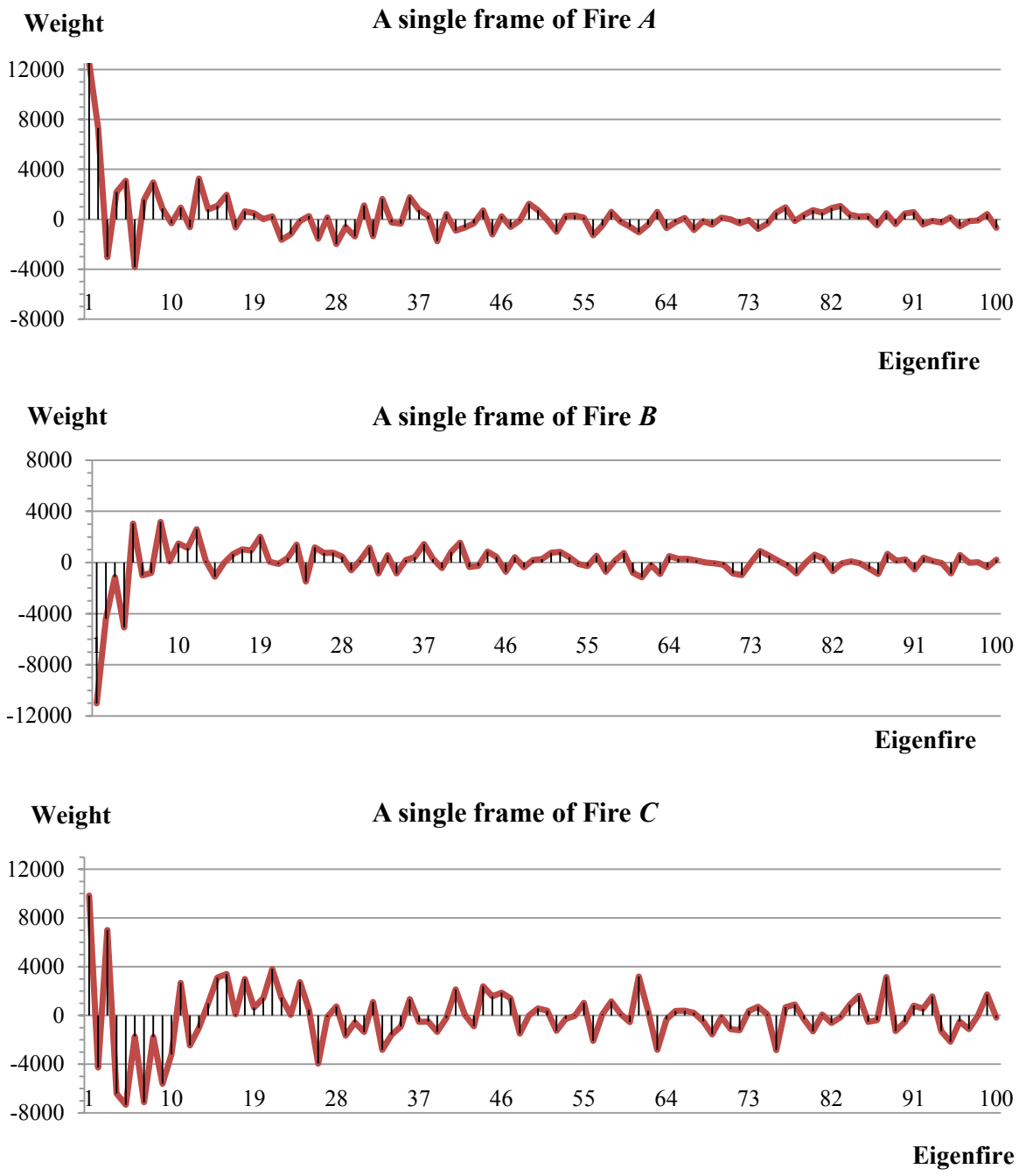


Figure 28: The weights of the first 100 Eigenfires are shown only for a single frame

The information from the paths can also be extensively used either to produce or to validate new animations. For instance, the paths that are in *E* dimensions (maximum 2196 dimensions for our three samples) and obtained from training stage can be deemed as a template. If the animator is interested in synthesizing a new sequence of images from

scratch, then this template will be a good start. Even if the animations are made either randomly or following a particular design (e.g. dynamic textures [15][16]), the analyzed data and paths are still essential sources to prevent exaggerated motions by regular comparisons, or to fix them in order to make more realistic animations.

4.7 Features of Eigenfires

Among the first 30 Eigenfires which clearly express significant features of our samples, the first 10 are mainly used as foundation to synthesize diverse shapes of flames with smooth edges. A short description of a few effective features is provided in Table 1 for their relative principal components. Vague contributions are left blank in this table.

EigenFire #	Feature Description	Positive Weight	Negative Weight
1	Direction of fire (a_1)	Toward left side	Slightly toward right side
2	Direction of fire (a_2)	Toward right side	Slightly toward left side
3	Height and width	Increasing height, decreasing width	Decreasing height, increasing width
4	Opening the flame at the tip (b_1)	Two wide equal branches	Two wide equal branches
5	Shape of the tip of flame	Cone shape	-
6	Volume	-	Larger volume
7	Opening the flame at the tip (b_2)	Three sharp branches	Two sharp branches
8	Hole (c_1)	Two holes in fire	A hole at the tip (two branches)
9	General shape deformation	-	-
10	Hole (c_2)	A hole in the middle	-

Table 1: Features of the first ten principal components, vague contributions are left blank

Although these features are obvious from visualization of Eigenfires (Figure 23), manual reconstruction of images has been performed using the first 10 Eigenfires to validate the information of this table. In our experiments, other sample videos also exhibited similar strong features, easily noticeable in Eigenfires. Therefore, this information will be later used in section 5.2 to synthesize a new unique fire image or to modify a desired frame with manipulations of their weights.

4.8 Image Reconstruction, Compression and Quality Enhancement

Image *reconstruction*, or sometimes named *back projection*, is a process that we restore a projected data, and turning it back into a fire image. Using previously calculated average image Ψ, Eigenfires and part of a weight series loaded from the database, we are able to restore the image I with inverse operations of projection. This is the opposite of what we did to calculate the vector of weights Ω. The reconstruction is given by:

$$I = eigenfire\ \Omega + \Psi$$
$$eigenfire = [eigenfire_1, , \dots, eigenfire_E\ ] \tag{22}$$

A few reconstructed images, which are projected into multiple dimensional fire spaces, are depicted in Figure 29. Since we might use similar reconstructed images later for our algorithms with recognitions, these figures can give you a good idea about how each of our three fire samples will look after being projected back using fewer dimensions. Therefore, a proper number of Eigenfires, as E, can be estimated here for later use in animations. For this purpose, we may allow some negligible noise as long as it does not significantly affect the general shape of the flames. Note that our system allows two different parameters as E to be used later; one for recognition and another for reconstruction. A very important point here is that these optimized numbers of Eigenfires must be suitable for all the three fire samples simultaneously.

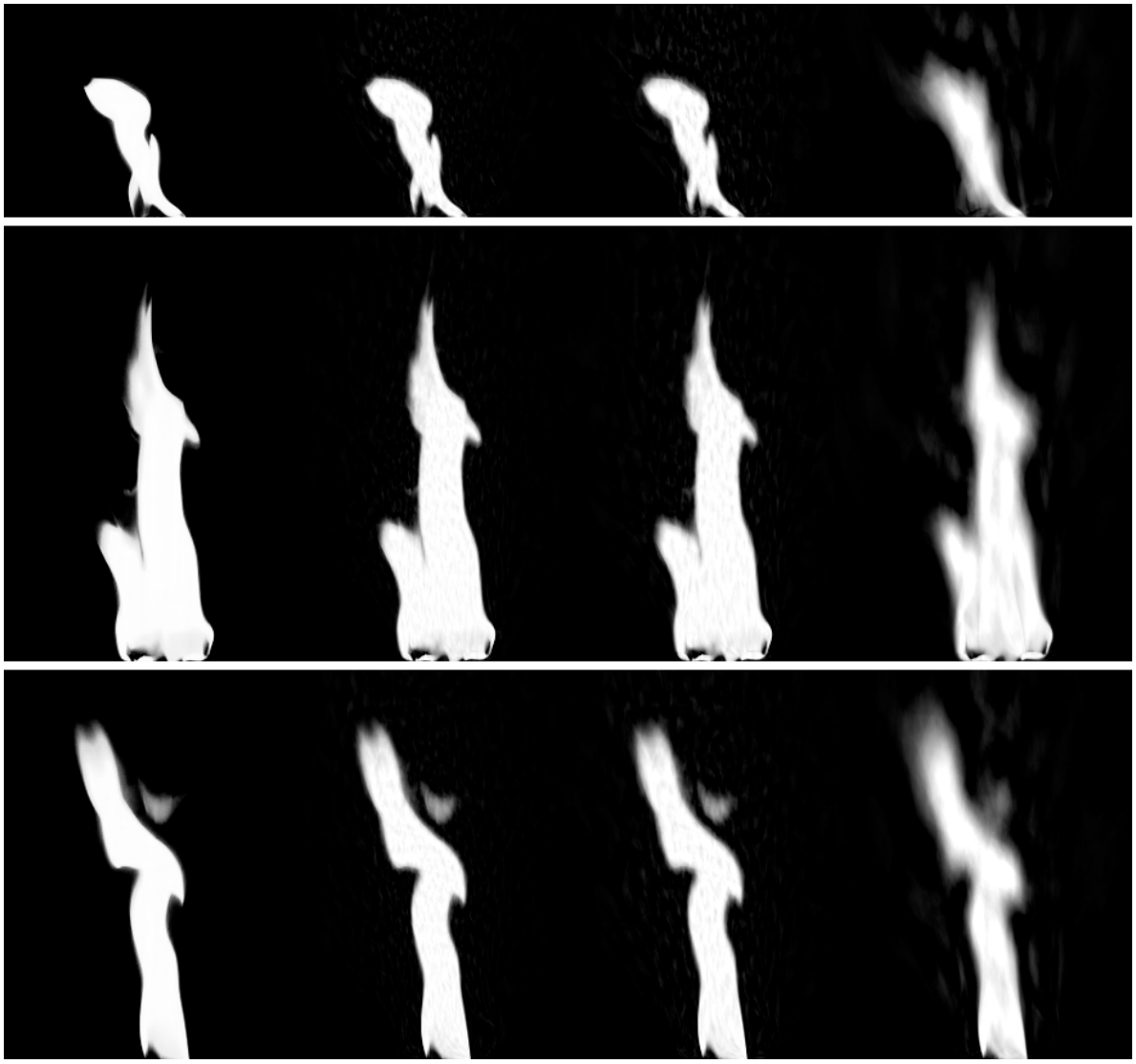


Figure 29: Reconstructing a few images of the database with different dimensional spaces. In each row from left to right, flames reconstructed with: 2196, 1100, 700 and 100 Eigenfires

The experiments suggest that if the flames are reconstructed with a lower number of Eigenfires, then the same Hermite background removal approach can be utilized once more to improve the quality of the reconstructed images considerably. The artifacts usually appear in a few forms, such as layers of various flames with lower intensities, redundant noise outside the contour and some noise inside the contour, as partially visible in Figure 29. Hermite background removal then can easily diminish those artifacts from areas outside the

shape. However, to preserve the accuracy of detections, we try to avoid utilizing Hermite background removal for recognition steps when considering fewer dimensions.

However, the noise inside the boundary of a flame can remain unchanged, because the fire still looks realistic and acceptable, particularly after applying color and performing post-processing to its grayscale fire image. This means we can compress the entire database by using a lower number of Eigenfires and considering synthetic coloring. Figure 30 shows the colorized version of the first and the third columns of Figure 29 with 700 Eigenfires only.

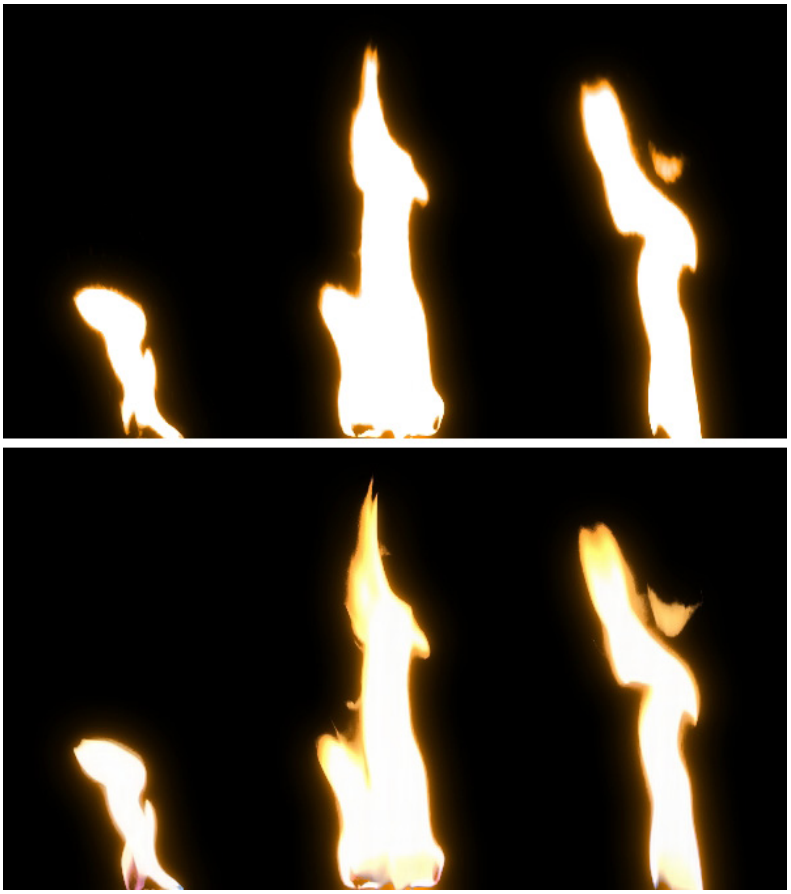


Figure 30: Compression and quality enhancement. First row: Reconstructed flames using 700 Eigenfires (68% compression), and the quality is improved by applying Hermite background removal, a synthetic coloring and bloom effect. Second row: Reconstructed images using all 2196 Eigenfires with original colors of the video

Since we split the binary files of the fire database into packages of a specific length that is defined in the training stage, we may remove unwanted packs at any time. For example, for our main database of three fire sample, a variable of *packSize* = 500 is defined which produces 5 packs for 2,197 images, and are labeled as *Pack_X*, where $X = 1$ to 5. Thus, we may easily ignore, remove or truncate the packages, so that it finally holds the required Eigenfires only.

The same approach can be done virtually via our user-interface to accelerate either recognitions or reconstructions at run-time without touching the packages of the database. The reason is that some animators might be interested in running recognition methods using half of the Eigenfires to speed up generation of animations, but still intend to render and export the final animation with the highest quality using all available Eigenfires.

4.9 Further Experiments on Image Reconstruction

We performed many observations and analyses on Eigenfire which finally led to a few valuable fire modeling and animation methods presented in next chapter. Sharing some of those experiences in this section might be useful even for any other fields of research where the Principal Component Analysis is involved there.

In our initial experiments as part of pre-processing, we removed areas outside the contours by assigning zero values to every pixel without performing background removal. This means we prepared a dataset of flames with sharp edges, and therefore, we were able to reconstruct new flames with some artifacts as numerous layers of sharp edges, particularly by choosing a small training set of images. It should be noted that Eigenfires are highly relevant to our average image, and we can avoid many unexpected artifacts during reconstruction or image synthesis if the flames with smooth edges are considered for the dataset. Our experiments suggest that removing the background and smoke is an essential pre-processing phase for fire samples and lead to more accurate recognitions.

In analysis stage, there were a few questions in our mind that we needed to investigate further. What would be the result of averaging the corresponding weights of two fire images that exist in the training set? Is the result of this projection a new fire image that is somewhat similar to both of them? Figure 31 answers these questions. The average of the weights is not exactly what we might expect. Using all *E*=2,196 Eigenfires of our database, the result appear as a simple blending of both of flames, which is like 50% transparency of first fire + 50% transparency of second fire. Interestingly, as we utilize lower numbers of Eigenfires, the new flame would emerge as a deformation between the two flames, or in other words, a morphing between two complex shapes of flames.

The weights of a new image, Ω', as the mean of two other images, using vector Ω_1 and Ω_2 , are computed as $\Omega' = (\Omega_1 + \Omega_2)/2$.

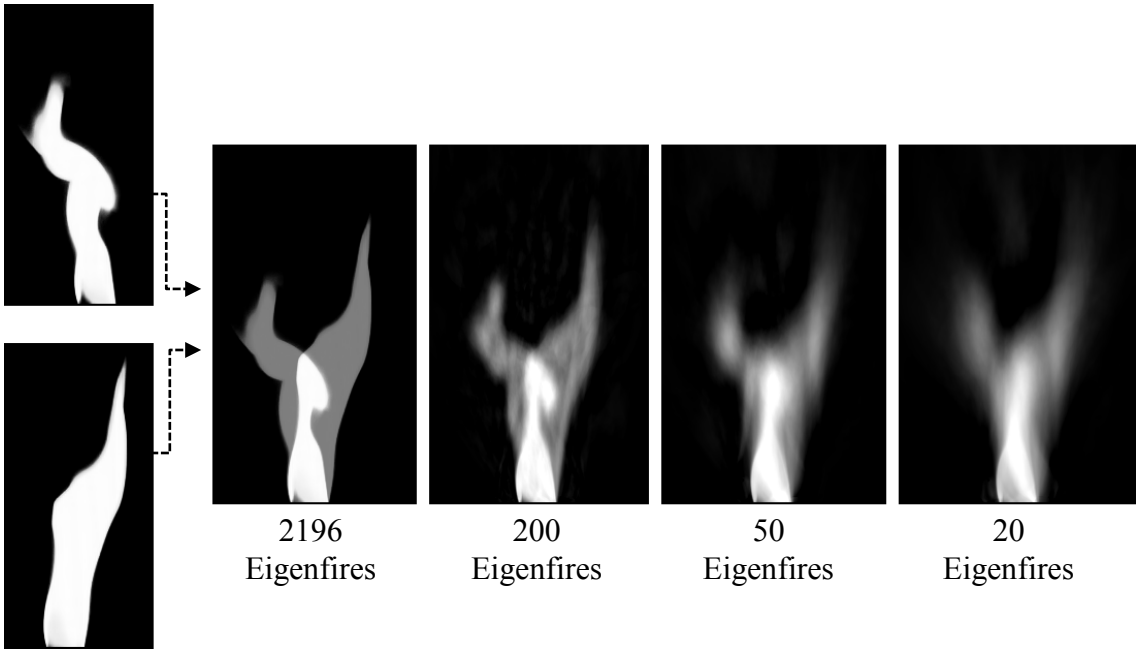


Figure 31: The reconstruction results of averaging the corresponding weights of two flames

Remember visualization of fire images as 3D data points, showed in Figure 24. Considering *E* dimensions, what if we randomly insert a new data point among two or more existing data points that had been already projected into PCA subspace? The result of this

case is almost similar to Figure 31, but this time it is an image constructed of layers of many flames. These features of Eigenfires led to the fact that we can synthesize images as morphing among two or more flames that have very different shapes. This idea will be used in Chapter 5 using one dimensional low-pass filter in fire space.

Chapter 5 Fire Modeling and Animation Using Eigenfires

In this chapter a few new approaches of fire synthesis and animation are introduced, which can improve some of the traditional routines of fire modeling with a realistic-looking fire animation, and especially without acquaintance with physical properties of fluid systems. Although the basic understanding of the chemical and physical nature of fire is valuable for recording better videos and accordingly preparing an impeccable database, it is not required for third-party users and animators.

5.1 Motion Transition

In this section, we describe new methods of fire modeling using our existing fire examples. Assume an artist chooses two ranges of frames from any of three flames, for instance he or she chooses *T1* = [800-880] from fire *B*, and *T2* = [2000-2050] from fire *C*, and plans to connect these separated animations of fire together (*T1* + *T2*). Because the last frame of *T1* belongs to fire *B,* and the first frame of *T2* belongs to fire *C*, then a simple attachment of two frames faces a sudden jump in transition from *T1* to *T2*. How can we construct new frames to fill this gap? There are also other possible scenarios. What if the artist is interested in creating a loop of animation for *T1*? Or a sequence of animations: "*T1*+ *T2*+*T2*+*T1*". The process is called "*Motion Transition*", which means generation of new frames that can provide a smooth or visually convincing transition between two animations. A transition can be to connect one selected range of animation to another range, or to connect the last frame of one animation to the beginning of the same animation or to the one that is first inserted into our motion list, in which the last two are called "*producing loop*". The number of frames for transitions is our "*steps"* that the artist chooses or the algorithm will reach it based on the selected technique, as a main difference from [53]. The following two sections describe our methods for *T1*+*T2* scenario.

Our user interface allows the users to interactively preview the original videos in viewport and insert their desired ranges of animations with a certain name to the motion list. Desired ranges can be marked and saved into the disk as project files of the current database, and later be loaded with their properties and assigned names. For instance, a user may select 50 frames of fire *A* as *T1*, and 100 frames of fire *B* as *T2* to imitate a wind effect, and then go back and continue fire *A* as *T3*. The user may select a transition method at this time among *T1*, *T2* and *T3*, enable producing loop for transition from *T3* to *T1*, and then create the final video, which is absolutely different from the pattern of the original video. The motion list of this example and some user interface options available for motion transitions are shown in Figure 32. "*Low Pass Filter*", "*Direct Bridging*" and "*End Growing Connection*" are three types of transition we purpose in this thesis for animation of fire. User's control on Eigenfires (synthesizing a fire image) is another flexible modeling approach to create or modify a single fire.

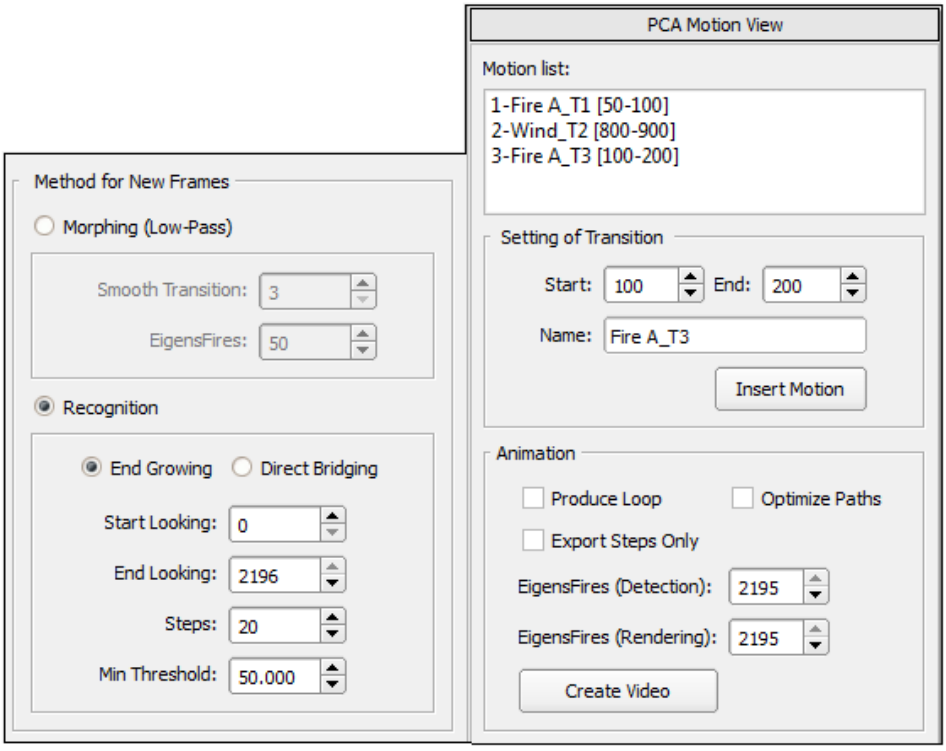


Figure 32: The motion list and some available options for motion transitions.

One of the advantages of our animation methods is generating animations based a scenarios. A scenario or storyboard is illustrated in Figure 33. In this figure, the 20 seconds of normal motion can also be constructed synthetically using loops and motion transitions.

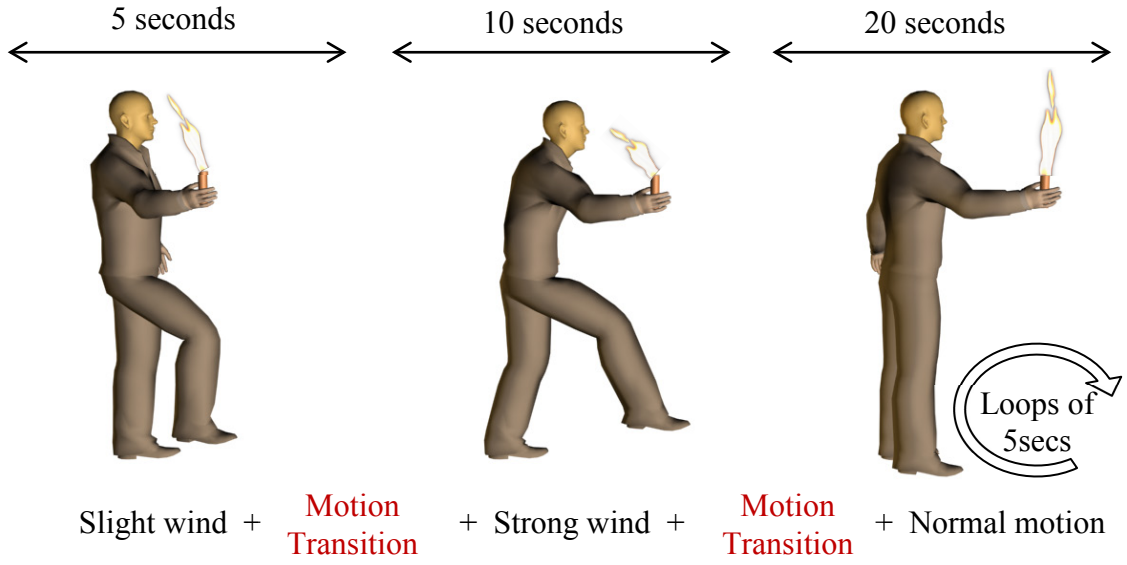


Figure 33: Producing animation of fire based on scenarios using motion transitions and loops

In many video games, the sequences of fire textures need to be displayed infinitely or as long as the player stays in the scene. Hence, our motion transitions can be used to generate a few images to stitch the last frame to the first frame as our "loop" for various flames.

5.1.1 Motion Transition with Low Pass Filter

In this approach, the procedure to transit from one fire to another fire is averaging the weights corresponding to a few frames around the end of *T1* and beginning of *T2*. For the first step, it is required to build a new vector that is composed of vectors of weights $[\Omega_1, \Omega_2, \Omega_3, \dots]$ for all images of *T1* and *T2* respectively. Next, one-dimensional low pass filter with a kernel of *K*=[1/3, 1/3, 1/3] can be applied on this vector to obtain the average of weights for each dimension. It means we are working on a vector of vectors. Since the first

few Eigenfires represent low frequency information of the fire, we benefit from utilizing the first 50 Eigenfires, which are only 0.02% of the total number of our principal components. Therefore, artifacts that we discussed in section 4.9 are avoided, and instead of sharp layers of target images we obtain smooth deformation of flames. As a result, these new vectors of weights ($\Omega_i{}'$) can be projected into the first 50 Eigenfire components. Because the length of the kernel was 3, the averaging outputs of the last two images of *T1* and the first two images of *T2* can be considered, and then be inserted into animation as our transition between *T1* and *T2*. It should be noted that we have to remove the last frame of *T1* and the first frame of *T2* from our new animation, or we should ignore two results out of four morphed images to avoid having two consecutive frames with very similar shapes. The final animation is a morphing between our target images as two distinct shapes of flames. The concept of this technique is illustrated in Figure 34, and three reconstructed frames are presented in Figure 35 using this approach, to fill the gap between *T1* from fire *B* and *T2* from fire *C*.

This technique can be extended using other kernels such as *K*= [1/5, 1/5, 3/5]. These types of unbalanced kernels may be chosen for the end part of *T1*, and then be flipped for the beginning of *T2* to create a different deformation by giving more weights to our target images.

Low-pass filter approach ignores high frequency components and we might notice a change in animation because of the loss of roughness and sharp edges for these new frames. To include fine details, we may randomly copy a few weights that belong to Eigenfires beyond 50, from our target images to our computed average vectors.

User's control on weights for synthesizing fire is a manual adjustment technique that can be used to improve or deform the outputs of this technique. It means the results of low-pass approach can still be customized by animators in just a few clicks. This is discussed at the end of this chapter.

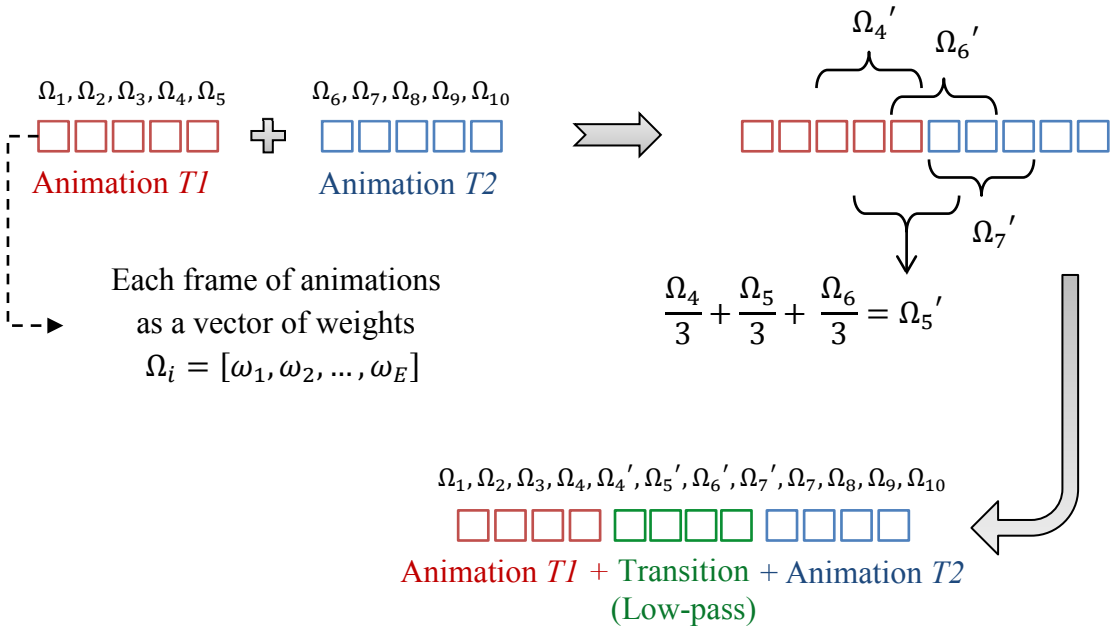


Figure 34: The concept of low-pass filter approach using a kernel of *K*= [1/3, 1/3, 1/3]. Projection of the four new Ω_i' into Eigenfire components are four morphed images

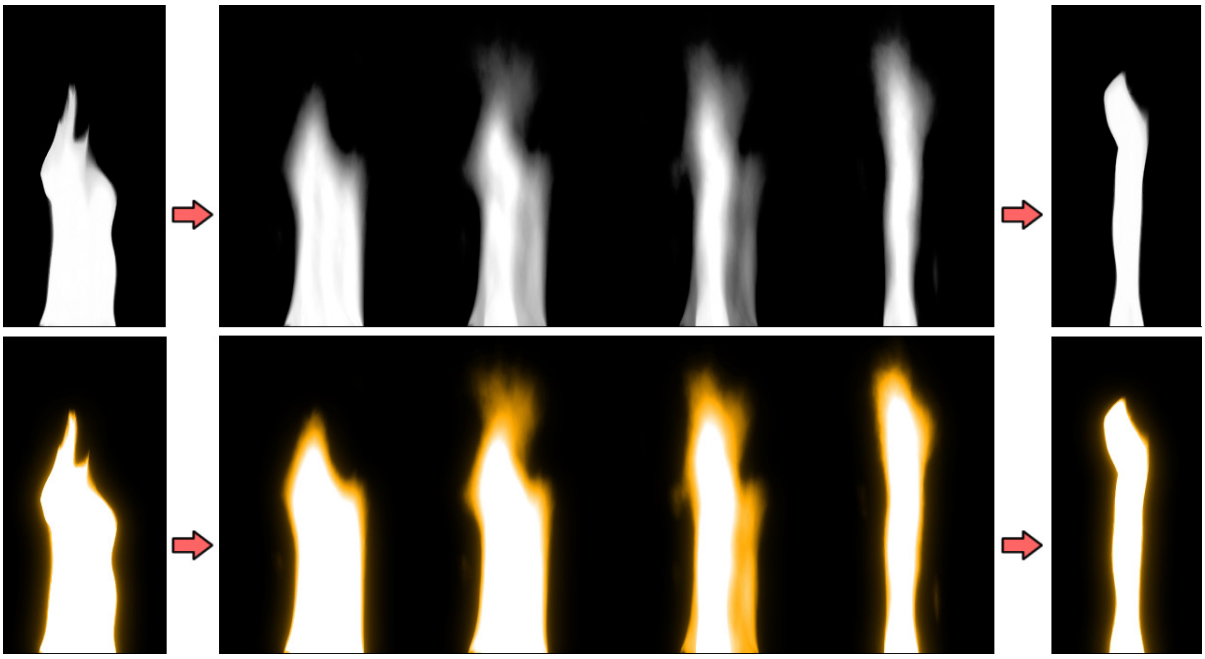


Figure 35: Low pass filter approach. (left) last frame of *T1*, (right) first frame of *T2*, (middle images) four reconstructed images using 50 Eigenfires which appear as morphing. A synthetic color and bloom effect are applied to all images of the second row

5.1.2 Motion Transition with Recognition

In this section, we introduce a few methods based on identification of similarities to model a fire. Like low-pass filter approach, these methods are also filling the gap between our *T1* and *T2* animations, but this time using more number of frames with strong similarities in the entire video. Let the last frame of *T1* be image α, and the first frame of *T2* be image β. Thus, α_j and β_j describe the j-th similar images to initial images, α and β. The purpose is acquiring one frame in each step which is not exactly the same, but can be selected as our next frame. Since finding very similar shapes produces unrealistic animations and might be displayed as slow-motion, defining a minimum threshold θ_{min} is necessary. This θ_{min} should have already been calculated in analysis stage.

We propose two techniques of recognition, "Direct Bridging" and "End Growing Connection" methods. In both cases, the animator chooses a specific number of steps, denoted as *N*, in order to find at most *N* numbers of similar images to be inserted between *T1* and *T2*. Our user interface allows users to select two different *E* as the numbers of Eigenfires for "*rendering*" and "*detections*". The animator might be interested in reconstruction of the final results with the entire available Eigenfires, but still want to accelerate the detections of similarities for our proposed recognition techniques using important Eigenfires only.

5.1.2.1 Direct Bridging

In Direct Bridging method, firstly images of our fire videos are transformed into Eigenfire space if they are not already included as training set in the database. What we receive is a vector of weights Ω for each image that we already visualized as a data point in 3D and with charts. Considering the fact that α and β are two data points in *E* dimensions, connecting them gives us a multi-dimensional line that can be imagined as a bridge. Next, based on the value of steps (*N)* that is set by the user, this multi-dimensional line is divided equally by *N* and the positions are stored as new temporary data points of the bridge. In other

words, we insert *N* virtual new data points by dividing values of weights in *E* dimensions. Reconstructing images with these new data points, which are *N* new vectors of weights, produces unpleasant layers of flames, similar to what we already showed in Figure 31 using all *2,196* dimensions. Therefore, the solution here is looking into existing data points of our database, and then finding closest ones to each of these virtual *N* points on the bridge by minimizing the Euclidean distance between them. It means we are looking for similar flames that are located around the data points on the bridge, and still the criteria should meet the requirement of the permitted minimum threshold θ_{min}.

Figure 36 depicts the concept of this method. A decent example of this technique is presented in Figure 37 that allows animators to quickly build loops for animations by finding a certain number of similar images. Reconstructed images of this figure are colorized by extracting colors from original videos, and finally a bloom effect is applied. The frames belonging to *T1* and *T2* animations are excluded from being detected as similar images of the new path from α to β. The reason is that the number of steps is usually small in this approach, and therefore we can avoid noticeable repetitions of the same flames in this kind of short transitions.

The advantages of this method are the high speed of identifications and flexibility to define the exact number of steps for transitions. The disadvantage here is the possibility of recognizing an image that is closest in distance to a point on the bridge for current step, but it is not close enough to the identified frame of the previous step. The reason is that we are not checking similarities with previous frames on the path, but finding one image in each step which is similar to an image on the bridge. This might reduce smoothness of motions without human verification if a large *N* is chosen for the number of steps. Therefore our second method, End Growing Connection, can solve this issue for higher number of steps.

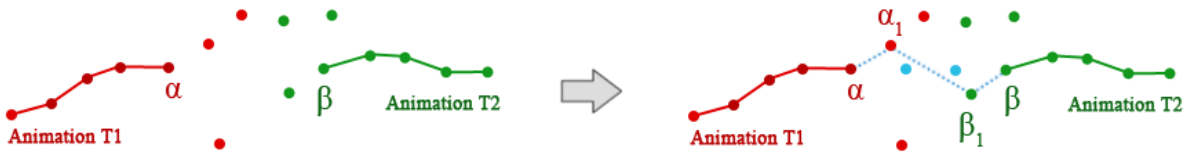


Figure 36: The concept of Direct Bridging method, visualized in 2D. The image on the right shows connection of α to β using this method for *N*=2 steps. Two temporary data points (in blue) are first inserted between α and β to build a bridge, and then we look for the nearest points to each of them that form a path illustrated with blue dotted line

Figure 37: An example of Direct Bridging method to connect *T1* and *T2* animations to simulate a loop of our animations in the same style of fire. (left) Image α, (right) Image β , (middle) three images that are found similar by $\theta_{min} = 50$ in *N*=3 steps

5.1.2.2 End Growing Connection

In End Growing Connection method, the animator chooses a large *N* as the maximum possible number of steps, and then looks for a specific pattern that gradually connects image α to β through finding similar images in between, which are stored as α_j and β_k. The algorithm successfully stops after reaching a point nearest to both α_j and β_k. In other words, we are building a path that can connect maximum *N* data points in *E* dimensions. Remember that each data point is representing one image in PCA subspace. Reconstructing images of this path should generate a short animation to connect *T1* and *T2* animations without any sudden jumping or odd change in speed. The idea here is to suggest an algorithm that can build a short and optimized path suitable for animation of various fire videos.

Algorithm 1 summarizes a simple technique to find a path which is constructed of short distances less than the difference between α and β, and larger than our threshold θ_{min}. Let α and β be the end points of the path we are looking for, and call them two sides of the path. We start from α, then compute the Euclidean distance between α and each available data point (images of the training set). All distances are stored in a list as α_j and become sorted. The last few images of animation *T1* (e.g. the last 7 images) and the images with distances of less than θ_{min}, should be removed from this α_j list. This is to avoid deadlocks, moving backward along the same pattern, slow-motion and fast-forward animations. The same process should be done for β and the sorted β_k list.

Next, an image with the shortest distance from the sorted list α_j is selected as a similar flame to α with respect to the fact that it should also be nearer to the other side, in which the other side is β here. This is performed by computing four distances among α, β, α_j and β_k as presented in Algorithm 1. This similar image will be stored as α_1 temporarily, and we continue the same process for β until reaching β_1. If comparisons of the four distances are convincing for a fire animation that is defined in our algorithm, we can set α_1 and β_1 as our first data points of the path. Otherwise, we repeatedly choose the next corresponding element from the two sorted lists that can make the defined conditions true.

At this level, the same procedure should be repeated but this time for α_1 and β_1 (instead of α and β) until finding α_2 and β_2 in each side of the path. It is like growing two small paths from two sides until joining each other in a shared point (representing an image) to form a larger path. In this approach, we detect maximum *N*/2 similar images for α and *N*/2 for β until we meet one of the following possibilities:

1) A shared image, similar to both α and β, is found: $\alpha_j = \beta_k$
2) No shared image is detected, but the distance between α_j and β_j is small and less than θ_{min} that is determined by the user.
3) All identified images are bigger than θ_{min} after passing through *N* steps.

In the first two cases, the proper path or pattern is constructed and the algorithm stops successfully. In the last case, we should choose a larger step and run the algorithm once more, or employ previous techniques such as Direct Bridging method or morphing with low-lass filter. At the end, the path is also optimized by removing some images from α_j and β_k lists if we encounter dramatic changes of distances. Figure 38 illustrates the concept of this method. It is also required to compare all elements of α_j with all elements of β_k list in each step for successful stop criterion of the algorithm, because the shared image does not necessarily belongs to the same step for both paths of α and β. For example, two sides of the path may join after 3 steps for α and 6 steps for β, when $\alpha_3 = \beta_6$. Thus, all the images that are inserted into the lists after finding a shared image should be removed from that list at the end. Figure 39 illustrates this situation.

Figure 38: The concept of End Growing Connection method, visualized in 2D. The blue dotted line is our optimized path using algorithm 1, and although a maximum *N*=20 steps is chosen the algorithm stops in three steps by finding a shared image ($\alpha_2 = \beta_2$)

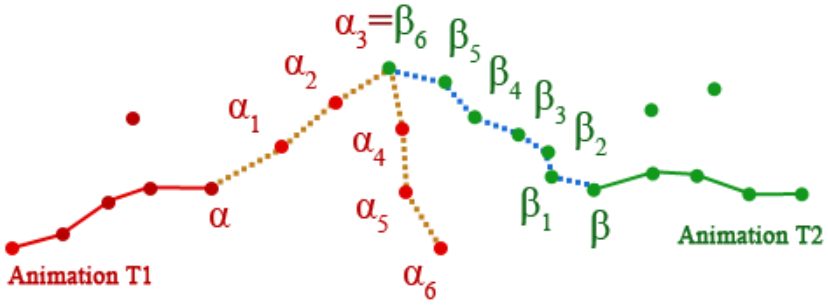


Figure 39: The two dotted paths join at $\alpha_3 = \beta_6$ by meeting a shared image. Elements beyond the shared image, α_4, α_5 and α_6, must be removed from the final path

Algorithm 1: End Growing Connection method, finding an optimized short path with specific small distances suitable for transition from one style of fire to another style

For steps 1 to *N*/2 **do** *// Two images are found in each step.*
    foundStep ⟵ false
    $\alpha_{j\,=[1,\ \ldots,\ 2197]}$ ⟵ Sorted distances of new α to all images of the dataset
    $\beta_{k\,=[1,\ \ldots,\ 2197]}$ ⟵ Sorted distances of new β to all images of the dataset
    j ⟵ 0, k ⟵ 0
    While there exists items in α_j and β_k, and foundStep **do**
        If (any element in list α_j = any element in list β_k) **Then**
            Return *// A shared image is found.*

        // Compute the following distance in E dimensions:
        $D \longleftarrow \| \alpha - \beta \|$, $D_{12} \longleftarrow \| \alpha_j - \beta_k \|$
        $D_1 \longleftarrow \| \alpha - \beta_k \|$, $D_2 \longleftarrow \| \beta - \alpha_j \|$
        shortToα ⟵ false
        shortToβ ⟵ false

        If ($D - D_{12} < \theta_{min}$) **Then**
            If ($D_1 - D \leq \theta_{min}$) **Then**
                shortToα ⟵ true
            If ($D_2 - D \leq \theta_{min}$) **Then**
                shortToβ ⟵ true

            If (shortToα and shortToβ) **Then**
                j ⟵ j + 1, *// Next shortest distance for α*
                k ⟵ k + 1 *// Next shortest distance for β*
            Else
                If (**Not** shortToα) **Then**
                    k ⟵ k + 1
                If (**Not** shortToβ) **Then**
                    j ⟵ j + 1
        Else
            Remove_duplicates()
            $\alpha \longleftarrow \alpha_j$,
            $\beta \longleftarrow \beta_k$,
            foundStep ⟵ true
    End While
End For
Optimize_Path()

Figure 40 shows a few frames from two varied ranges of animations. It is obvious from this figure that our algorithm needs to solve the issue of connecting two animations by filling the gap through a few new frames that gradually forces the flame to bend, rather than to jump in a single frame. Providing these new frames in between using our End Growing Connection method is similar to simulation of an external force (wind) which is illustrated in Figure 41. The same wind effect scenario is generated once more, shown in Figure 42, by considering recognition results from all three fire samples in our database and using a smaller θ_{min} because of fire *A*. This algorithm can be effectively incorporated with artistic creativity to demonstrate some of its exceptional capabilities as well. Figure 43 shows the interesting results of generating a new fire animation which is not already recorded by cameras. This example reveals that animators are not limited to available animations of videos by utilizing our techniques. They can produce such animations with some creativity in an artistic manner by selecting proper α and β.

End Growing Connection method yields very smooth transitions, either for one style of flame or particularly for various combinations of flames. The drawbacks of this technique are unknown number of steps, and failure for very distinct flames if the training set is small. Although we only set one minimum threshold (usually average of thresholds or smallest threshold) to produce transitions for our three fire samples, our system can be easily extended to an adaptive mode for θ_{min} to accept more than one threshold. For example in our samples, a proper θ_{min} for fire *A* is around 500, and for fire *C* is about 1500. The extension then can be applied this way: whenever we are mixing flames (e.g. Figure 42), θ_{min} may adaptively change, which means we should force the algorithm to use the threshold corresponding to that specific form of fire in recognition steps, because we already know which frames belong to which type of fire. Using this approach, more accurate results can be obtained if the recorded flames are very different in size and volume, which was not very perceivable in any samples of our small-scale flames.

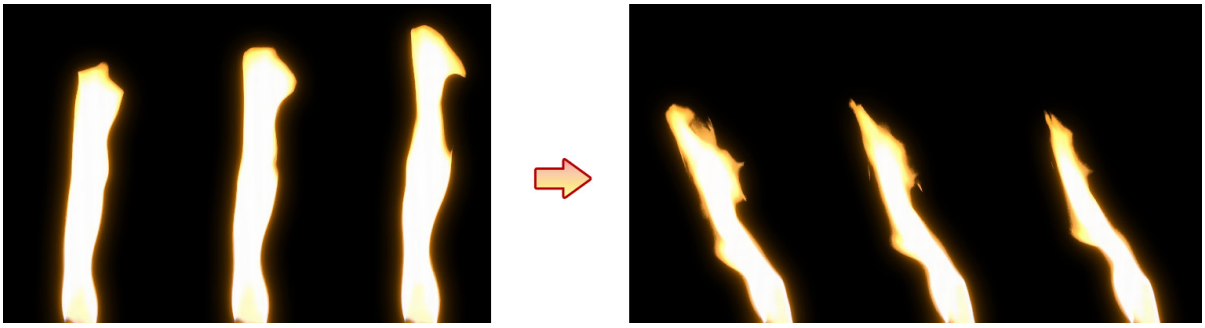


Figure 40: (left) the last three frames of animation *T1*, (right) the first three frames of animation *T2*. End of *T1* is a flame with a straight shape, and beginning of *T2* is a flame with a small slope. A big jumping from *T1* to *T2* is evident

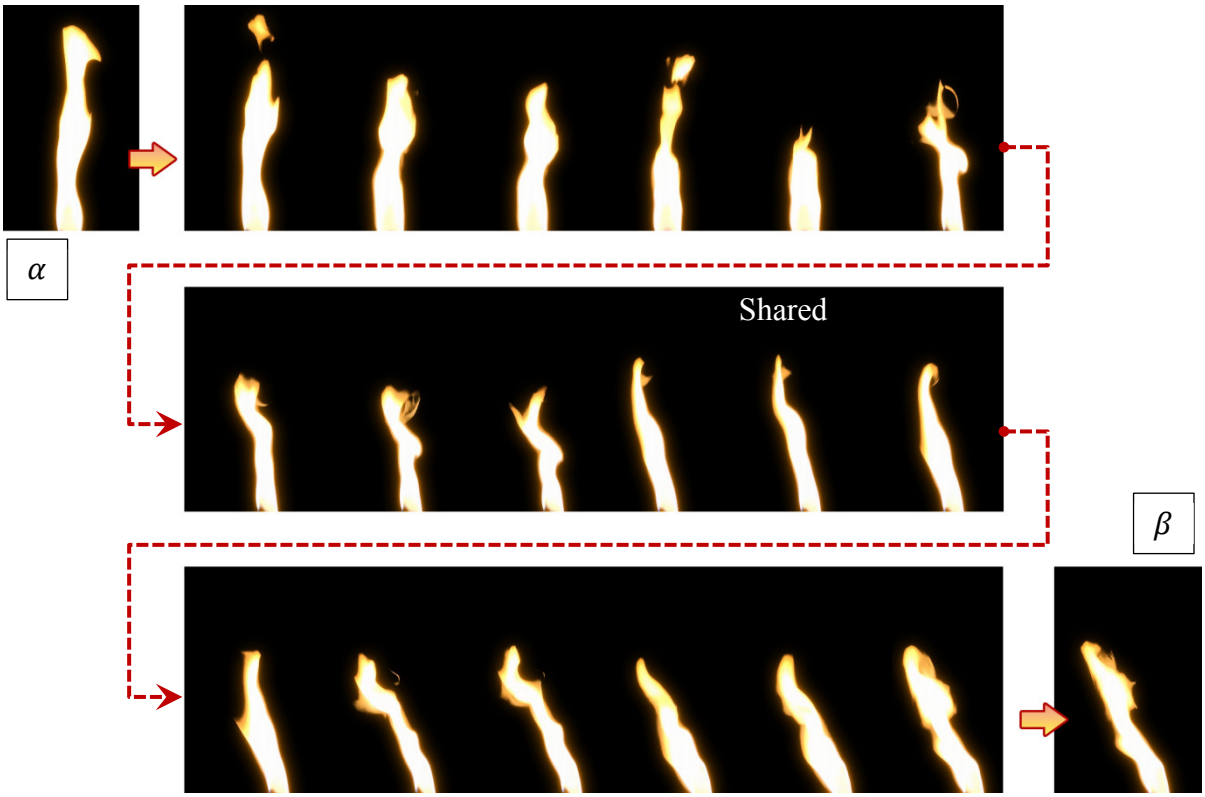


Figure 41: An example of End Growing Connection to simulate an external force (wind). Recognition of similar flames is restricted to fire *C* only for motion transition. With $\theta_{min} = 1500$, the algorithm stops after 18 steps. The image marked as *shared* is where $\alpha_j = \beta_k$

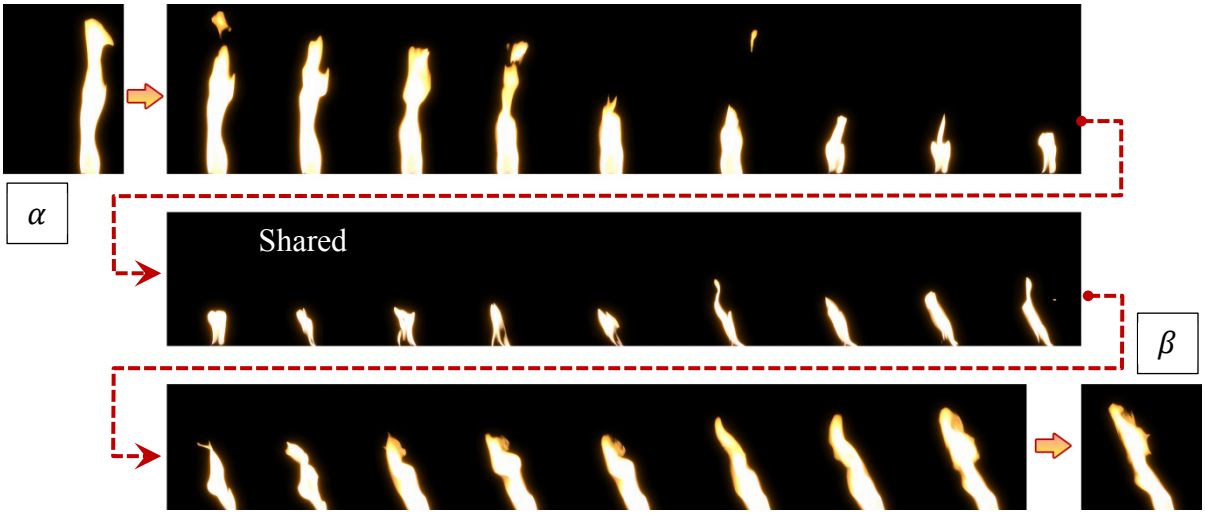


Figure 42: Another example of End Growing Connection method to simulate an external force (wind). This time recognition of similar flames is not restricted and images of all three fire samples (fire *A*, *B* and *C*) are allowed to be selected. With $\theta_{min} = 700$ and in 26 steps, images of fire *A* cause the fire animation to look shorter for a while, gradually expand and bend, due to wind power

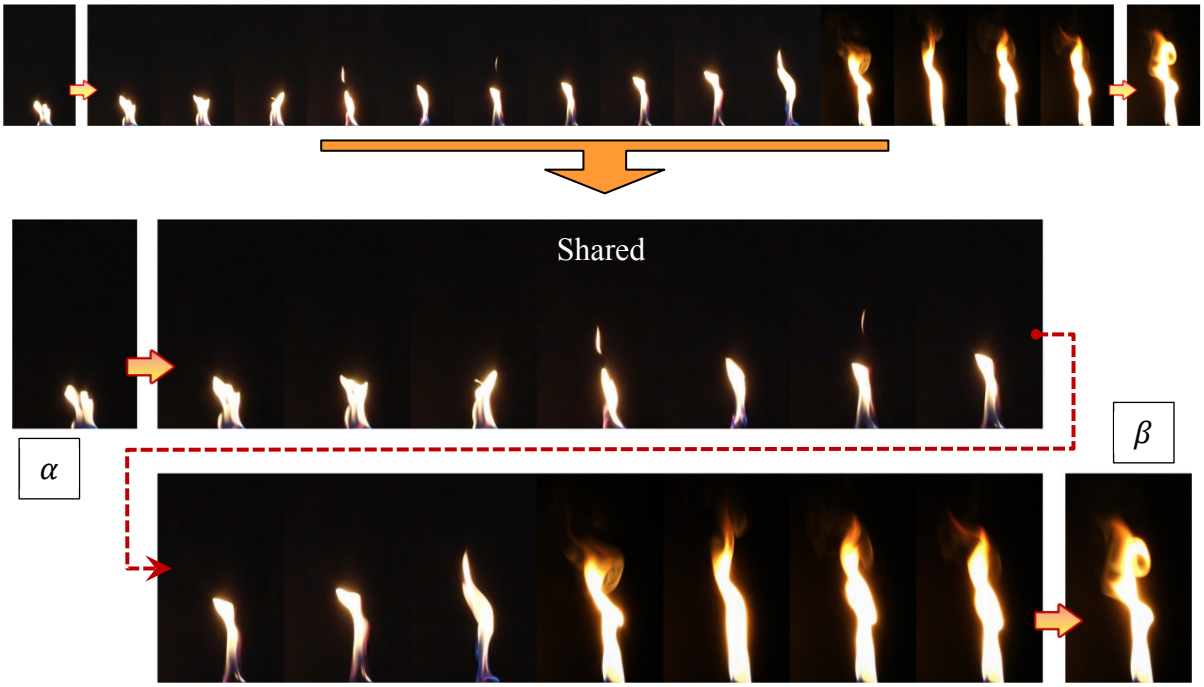


Figure 43: An example of End Growing Connection method to simulate starting a fire. *T1* is only a single frame from fire *A*, and also as image α (left), and *T2* is a regular animation of flame from fire *C*, in which β is the first frame of *T2* (right). Animation of starting a fire is not already recorded and it is not part of our database, but created with some creativity in an artistic manner. The algorithm stops after 14 steps with $\theta_{min} = 400$

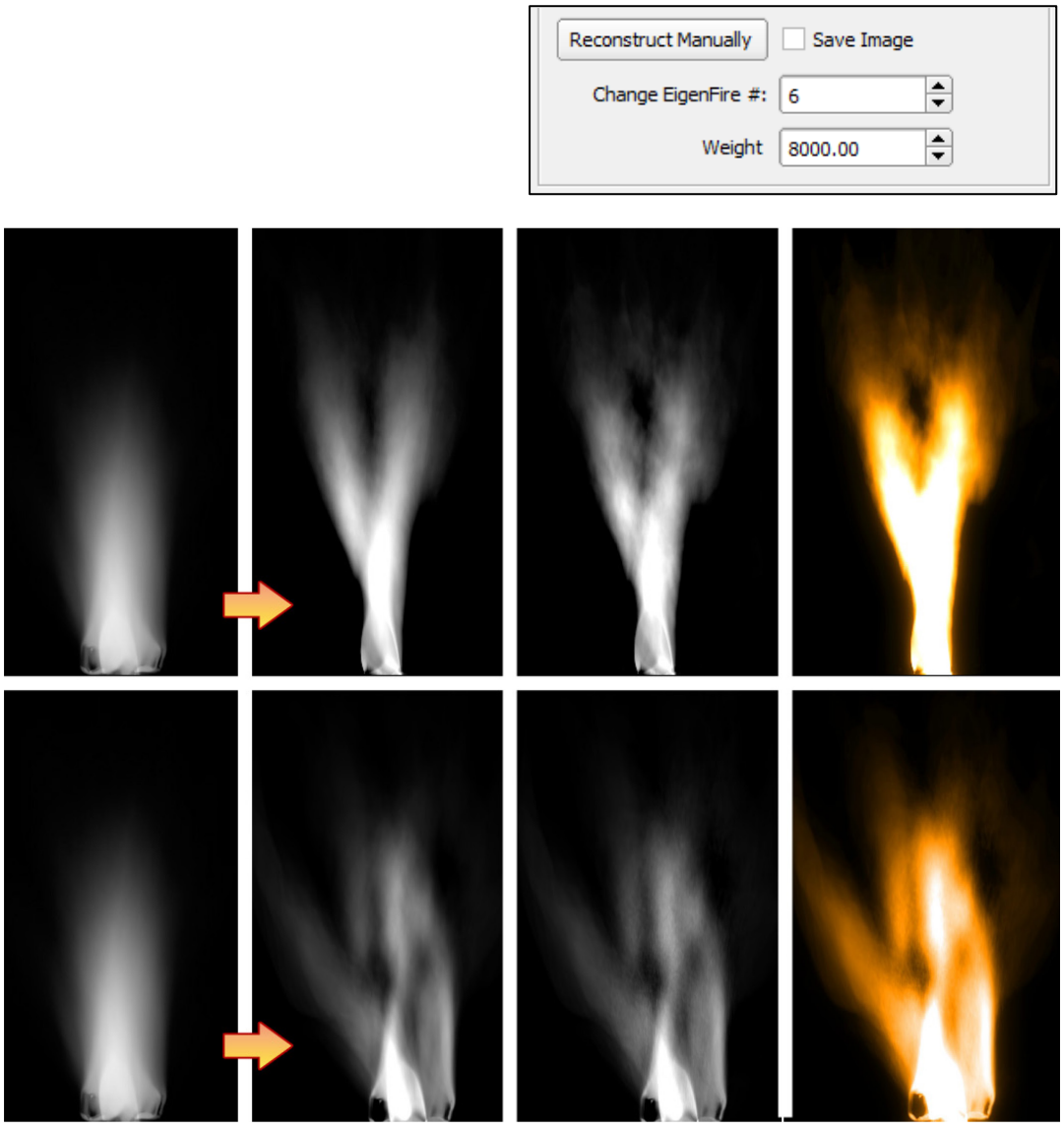
In both of the recognition methods introduced in this section, an image is also rejected if it is already identified in previous steps or it is one of the last few frames of the video before α and after β. Based on our experiments, the next or the previous frame of a candidate frame (α or β) has higher chance of being identified as a very similar frame with the smallest distance. For example, by choosing α as frame 50, frame numbers 51 and 52 or 49 and 48 are probably located in shortest distance to α, which are consecutive frames in our 60fps video. Therefore, by rejecting them, we avoid deadlocks or moving backward along the same pattern, and animations will be more realistic-looking without repetition of the same frames in a short transition video. It is important to mention that our proposed recognition techniques are not limited to what we presented in this thesis and they can be easily extended by any algorithm that is capable of constructing paths in multi-dimensional space, suitable for fire simulations, or other image synthesis methods such as Graphcut [36].

5.2 Synthesizing Fire using Eigenfires and Weights

Considering the fact that motion transition with low pass filter might not always lead to expected result, manual weight adjustment can produce additional details that an animator is looking for. After projecting a specific image into Eigenfire components, the weights of certain Eigenfires can be easily selected and modified via our user-interface, and the new fire is displayed in real-time. Since our first 50 Eigenfires carry major features of the flames, it is not even very tricky to start from scratch as follows:

1) Assigning zero values to weights of all dimensions (Ω).
2) Allocating new values by taking a look at generated Eigenfires of the database (Figure 23), or by referring to the corresponding table of features (Table 1).
3) The new Ω vector then can be projected back to obtain a new fire, which is still resembling a flame but interestingly a mixture of our three samples.

A few examples of this technique are illustrated in Figure 44. First row is created with manipulation of only 8 Eigenfires (0.003% of all dimensions!), and the second row is the result of weight adjustment for only 4 Eigenfires (0.002% of all dimensions!). An average image is received by assigning zero values to all of the weights and choosing a positive or negative value for the weights deforms this mean image.



(Average Ψ) (General shape) (shape + details) (Synthetic coloring)

Figure 44: Creation or modification of a flame using weight manipulation. First row: weights assigned to Eigenfires of 2, 3, 5, 6 and 7 for global shape, and Eigenfires of 50, 80 and 500 for distortions and some details, Second row: weights assigned to Eigenfires of 6, 41, 200 and 2000 only. The final results are shown with a basic synthetic color

For example, based on the weights calculated for our three fire samples, a new image can be constructed with a vector of weights Ω as follows:

$$\Omega = [\ (\omega_1 = 12{,}000 \text{ for Eigenfire \#1}),\ (\omega_2 = 8{,}000 \text{ for Eigenfire \#2}),$$
$$(\omega_3 = -5{,}000 \text{ for Eigenfire \#3}), \dots, (\omega_{50} = 800 \text{ for Eigenfire \#50}), \dots\ ]$$

Assigning small weights to any Eigenfire larger than 50 generates some fine details, having similar effect to applying procedural noises. Based on observations, these small values are suggested to be set as a number less than 10% of the largest value of the weights, in order to prevent unsuitable roughness (e.g. $-1{,}000 < \omega_i < 1{,}000$ for our database). The drawback of manual weight adjustment is the difficulty of creating realistic animations. This feature provides good refinement for users that request further control over the details, and therefore we have not much used it for our sample animations. As a result, we suggest combining this method with previously discussed motion transitions to follow the patterns of real flames first and then the user may deform the desired frames of that specific animation. Thus, we can control fire better in comparison to the single turbulent example of [13], as we know the main features of each database from Eigenfires, and interestingly the synthesized image is extracted from more than one fire.

Chapter 6 Color and Post-processing

6.1 Synthetic Coloring and Original Coloring

Since we convert our RGB fire images to grayscale in pre-processing stage to create a better database of Eigenfires for our training set, we lose the color information. Accordingly, reconstructing images from our Eigenfire database, gives us a similar series of grayscale images. Therefore the color needs to be restored or generated synthetically. Many researchers have been working on image colorization and successfully introduced various methods, such as colorization with manual scribbles [33], local color transfer with reference color images [61] and many more. However, because of simple shapes of our small-scale fire images, we employ our own methods and instead concentrate more on transferring the original color of video to the grayscale fire. There are two options in our system for colorization of our grayscale images: *synthetic coloring* and *original coloring.*

6.1.1 Original coloring

Since many film and visual effects studios are mostly looking for high quality production of fire rather than a few seconds of faster rendering, it is feasible to take advantage of the images of original video. By keeping the original colorful video, the colors can be extracted from the original images with the help of our Eigenfires as a series of 8-bit masks. This is called *original coloring* and performed as Algorithm 2. It should be noted that this algorithm only shows the results for the red channel of the final tinted output, and the same process should be repeated three times for all three channels of RGB images. It also shows how we add a smoke image to our final colorful image, which is discussed further in section 6.3.

Algorithm 2: Original coloring to colorize a grayscale pixel (shown for red channel only)

Inputs:

source:	Grayscale image
original:	Original colorful images of the video
smoke:	synthetic smoke image
threshold:	A threshold between [0, 255] to fade out appearance of colors
destination:	An empty image, or an image with boom effect

Output:

destination:	Final colorized image

```
smokeValue ⟵ 0

If (source[i] > threshold)  Then
        destination[i]  ⟵  original[i]
Else
        // Linearly diminish colors below the value of threshold
        scaleSrc  ⟵  source[i] / threshold
        scaleDest  ⟵  1.0 - scaleSrc

        // Add smoke if it is enabled
        If (Enable_Smoke) Then
                smokeValue ⟵  smoke[i] × scaleDest

        destination[i]  ⟵  saturate(    scaleSrc × original[i] +
                                        scaleDest × destination[i] + smokeValue      )
End If

// Saturation keeps the color values in a valid range
saturate(value): { return  min(value, 0xff) }
```

The first step in Algorithm 2 is a thresholding on our grayscale image. Pixels located beyond this threshold are taken from the original image and copied to destination. Any pixel below this threshold is linearly darkened and composited with bloom and smoke images if they are enabled in the system. This threshold is set around 30 (out of 255) for our samples. The *destination* image is used as both input and output, and it should be an empty image at

the beginning, filled with zero values, if the bloom effect is disabled. Otherwise, it is an image that contains the bloom effect. Because of the optional smoke image, saturation is required, which keeps the color values of each pixel in a valid range (unsigned char in our system). Figure 45 depicts an example of original coloring.

(a) (b) (c)

Figure 45: (a) Reconstructed flame using Eigenfires, (b) Original coloring, (c) Original coloring with bloom effect added

6.1.2 Synthetic coloring

Synthetic coloring is the process of creating a result color in HLS color space (Hue, Luminance, and Saturation), with the luminance/lightness of the source color (grayscale image here) and the hue and saturation of a blend color. The blend color is chosen by animators, and it can be a solid color (e.g. orange color), a colorful gradient picture, or a texture with specific patterns that are with the same size as our source grayscale image. This can be performed as follows:

1) Convert *source* image to HLS color space and store it in *destination.*
2) Convert the chosen blend color or texture into HLS color space.
3) Transfer the hue and saturation channels of the blend image into the *destination*.
4) The *destination* image should be converted back to RGB color space.

This procedure is not only used as a second approach of colorization, but also to tint the bloom effect we will create later. This technique is shown in Figure 46.

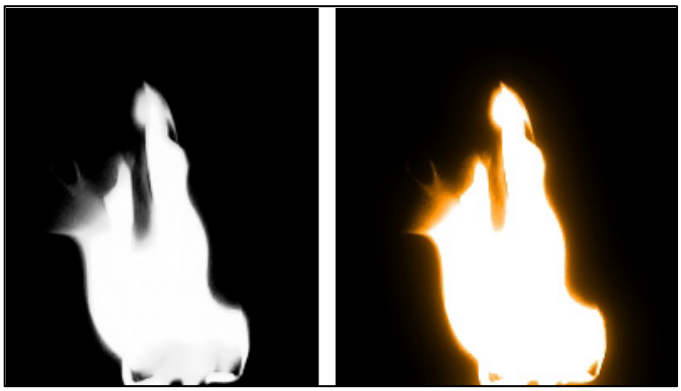


Figure 46: A Synthetic coloring using solid orange color and bloom effect

6.2 Bloom Effect

Bloom effect, sometimes glow or light bloom, is an effect widely used in almost all recent CG animations and video games to reproduce an imaging artifact of physical cameras, due to lenses issues and bright light sources that convolve the image with Airy patterns. Hence, a bloom effect can turn the results into a more realistic fire model, similar to the glow around the flame in our recorded fire samples (Figure 47). To produce a basic bloom effect, the procedure is as follows:

1) Store a copy of our *source* grayscale fire image in *destination*.
2) Apply a Gaussian blur with a large radius on *source* and store in *source_blurred*.
3) Multiply *source_blurred* with a small decimal value between [0, 1] (e.g. 0.4 for 40% transparency), and calculate the sum of this scaled image with *destination*:

$$destination\ (x,y) = 0.4 \times source\_blurred\ (x,y) + destination\ (x,y)$$

4) Repeat step 2 and 3 a few times with smaller radius for Gaussian blur. The number of repetitions is called the number of "*bloom passes*" in our system.

5) Blend *destination* with the *source* image using "*lighten blending*".

6) Colorize *destination* with synthetic coloring technique.

For step 4 and 5, *lighten blending* works flawlessly, in which pixels darker than the pixels of the blurred image are replaced with the pixels of the gray image. In other words, lighten blending transfers some pixels from blurred images to our source gray image for regions outside the contour, and interior part of the flame will be almost unchanged because of high intensities.

The sum of *destination* image from step 5 of bloom effect with our *source* image can be used as the alpha matte to integrate our results into other 2D/3D environments or video editing applications, and to composite it with other backgrounds.

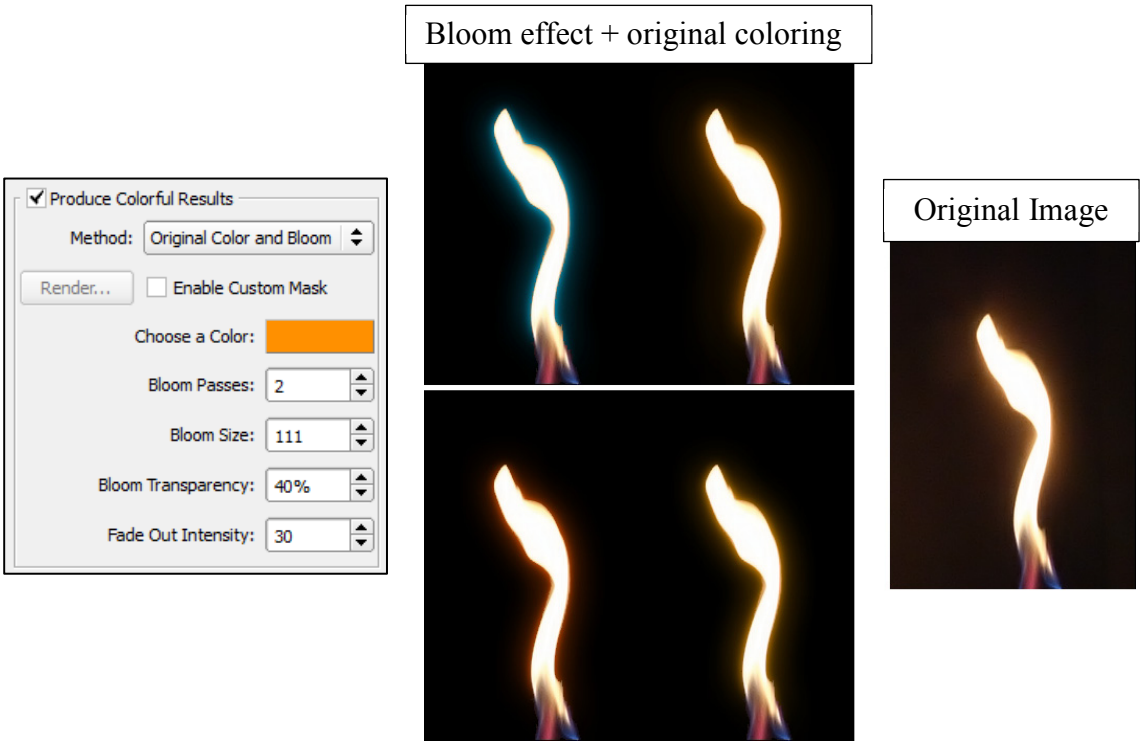


Figure 47: A reconstructed fire with various colors of bloom effect, and original coloring

6.3 Producing Smoke

As we already mentioned, the background and smoke are removed from the training set in pre-processing stage, because it ensures a higher accuracy for recognitions, and consequently much better results for motion transition techniques. For this purpose, we present a technique that allows users to add a layer of smoke to the colorized fire if the original fire contains smoke. In our method, the smoke or soot is obtained from the original video partially through these steps:

1) Convert current frame and next frame of the original video to grayscale images.
2) Compute the subtraction of these two consecutive frames:

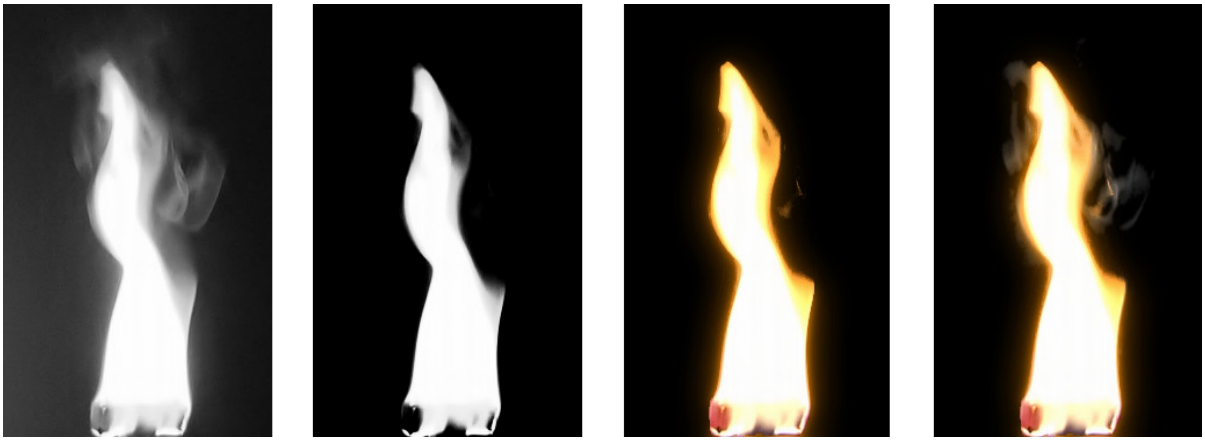
$$Image_{diff} = Image_{current} - Image_{next}$$

3) Perform a binary thresholding, so that any pixels below a user-defined threshold are removed from our $Image_{diff}$ (replaced with zero).
4) Apply a Gaussian blur with a very small radius on $Image_{diff}$.
5) Multiply this image with a color (e.g. dark gray), known as "*multiply blending*":

$$Image_{diff} = Image_{diff} \times color$$

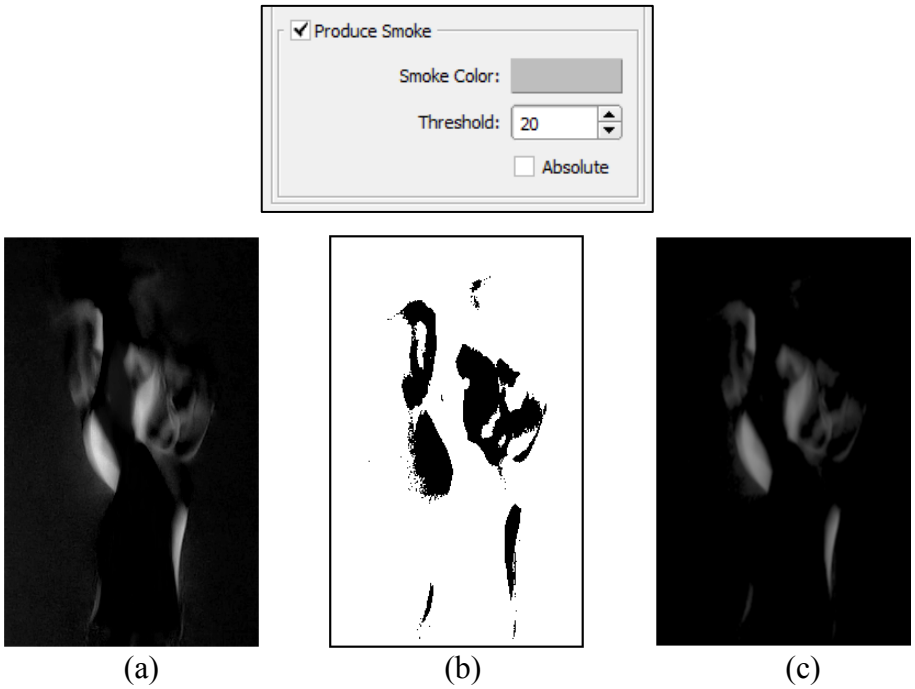
In first step, the current frame is subtracted from the next frame of the original video. If the next frame does not exist, the previous frame will be used. Step 2 can also be calculated with the absolute value of the difference that causes more shared pixels from the next frame appear on the $Image_{diff}$. In both cases, the result must be saturated to be in a valid range by: min(pixel_value, 0xff). Step 5 provides us with the capability of adjusting the color of our synthetic smoke, and to make it darker or brighter. At the end, the updated $Image_{diff}$ is our smoke image, and is used as input of original coloring already shown in Algorithm 2. A synthetic smoke is produced for a frame from fire *B* in Figure 48. Its corresponding smoke image, in addition to $Image_{diff}$ obtained from step 2, 3 and 5 are illustrated in Figure 49. For the transitions, the same approach is performed based on their next frames in the video.

The drawbacks of this technique are that it cannot generate smoke for fire images that did not already contain any smoke, or are combined with too much noise and luminosity.



(a) (b) (c) (d)

Figure 48: Producing smoke. (a) Original image converted to grayscale, (b) reconstructed image (c) Original coloring (d) original coloring with synthetic smoke



(a) (b) (c)

Figure 49: (a) the result of subtraction as $Image_{diff}$, (b) binary thresholding, (c) final smoke image after 5 steps

6.4 Post-processing with Custom Masks

Although we utilized reconstructed images as our masks (background and smoke are excluded), it is possible that a custom mask via a second real-time contour detection be called as an additional colorizing step while generating the results of animations. This custom mask gives more freedom of choice to the users.

There are a few tools available in our system for generation of custom masks, such as video editing, mask view and many filters. For this purpose, animators should first load an Eigenfire database and its corresponding original video, and then apply contour detection and other contour related tools from the filter menu. This creates a real-time mask for the current frame, displayed in the mask view. Whenever the final contour is ready, the mask view must be rendered for the entire video using *render preview* tool. If the artist is interested in calculating a different contour for each type of flame, it is possible to define a range of frames in render preview (Figure 50) to produce a different mask for that desired range. Then the coloring process can be called with the list of custom masks. If this list is empty or not rendered for a certain frame, we refer to the reconstructed image as our mask.

The same custom mask may also be combined with the reconstructed image, instead of replacing it. In our system, it is inserted into the process of original coloring as an extra layer of mask to accept more pixels from the original video based on non-zero regions of the custom mask. Figure 51 shows this application of a custom mask. The masks in (a) and (b) of this figure are inverted for better visualization. As a result, users are capable of ignoring some parts of images in training stage, such as smoke because of better recognitions, and later in animation stage return those deleted pixels into their final output via this custom mask.

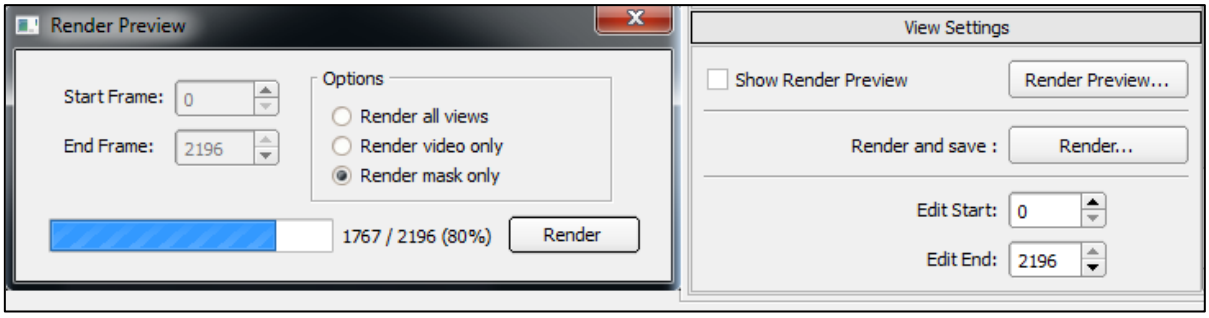


Figure 50: Creating custom masks via Render preview for extracting colors from original video

Figure 51: Application of a custom mask. (a) The reconstructed grayscale image as a *base mask* for both original coloring and bloom effect, (b) the result using the base mask, (c) a custom mask, (d) the result constructed with the base mask for bloom effect, and the custom mask for extra color regions

Chapter 7 Results and Discussions

7.1 Results and Performance

We ran our experiments on a system with Intel Core i7-2600k 3.4GHz CPU and 8GB of memory. We developed our application with C++ language and cross-platform Qt framework for user-interface that equip us with 2D/3D math library and powerful classes for integration with OpenCV and computer graphics libraries. OpenGL 3.3 is also employed for 3D visualizations. The 64-bit implementation allows the users to process very large video files without compression, and virtual memory can take care of required additional memory which might not be available physically on a working system.

For 2,197 fire images of 410x670 pixels, it takes around 1 hour to build a database of Eigenfires including loading data, pre-processing, PCA calculations and preparations of weights via primary projections. More precisely, PCA calculations take about 45 minutes, and projecting all training images into eigen space takes around 20 minutes. However, these calculations need to be performed only once to prepare the initial database which provides faster real-time recognition. The important point here is that the PCA calculations depend on data size considerably. For example, building Eigenfires take 12 minutes only for a dataset of 1,000 images with the same resolutions (8 minutes for PCA calculations and 4 minutes for projections). Similarly, this total time is only around 3 minutes for a dataset of 500 images.

A few effective tricks are explained in next sections to increase the speed of these calculations. Note that except initial PCA calculations and database preparation, all other techniques presented in this thesis are spontaneous meaning they can run at real-time interactive rate. Although we cropped our 720p wide videos to be at 410x670 pixels, we did not resize the fire images to maintain the highest possible resolution for the fire. The cropping was only performed to remove the static background and consequently to avoid redundant processing time for pixels that are not part of the flames.

Our experiments suggest that the optimized number of 1,100 Eigenfires (50% compression) can be considered for both high quality reconstructions and proper detections in motion transitions. A lower number, around 700 Eigenfires, is also sufficient if the purpose is only reconstruction rather than recognition. In this case, our background removal approach can improve the quality of the final result by eliminating unwanted artifacts.

A summary of the performance are shown in Table 2. It shows that by using our optimized number of Eigenfires the steps are almost the same. This table does not include reconstruction times (except for low-pass filter technique, as we have to reconstruct it) and only shows the performance of our algorithms for acquiring a list of frame numbers to be used as our motion transitions. The reason is that animators may take advantage of a few tricks to speed up the run-time processing without reconstructing images from Eigenfires. Next sections discuss this topic in details.

Technique	Steps using Fire	Eigenfires	Total Time	Steps	Time per Step
Direct Bridging (**Figure 37**)	*A, B, C*	2,196	101 ms	3	34 ms
		1,100	65 ms	3	22 ms
End Growing Connection (**Figure 41**)	*C*	2,196	169 ms	18	9 ms
		1,100	104 ms	19	5 ms
End Growing Connection (**Figure 42**)	*A, B, C*	2,196	835 ms	26	32 ms
		1,100	433 ms	27	16 ms
End Growing Connection (**Figure 43**)	*A, B, C*	2,196	448 ms	14	32 ms
		1,100	294 ms	14	21 ms
Low-Pass (**Figure 35**)	-	50	194 ms	4	48 ms

Table 2: Performance statistics of our techniques, excluding reconstruction times

In order to show that our outputs can be easily integrated into video games and 3D environments, we have prepared Figure 52 as a sample of integration into Autodesk 3D Studio Max using camera-facing billboards. The transparency of the background is based on the masks that we produce in pre-processing or post-processing stages, and the amount of blending can be controlled by the animators.

Figure 52: Integration of our outputs into Autodesk 3D Studio Max using camera-facing billboards. The transparency is applied with the corresponding masks

Basically it takes around 465 ms in total to reconstruct per image with all 2,196 Eigenfires, to perform all the post-processing tasks, and to store into the disk. Similarly, all calculations are about 285 ms and 48 ms per image with 1,100 and 50 Eigenfires respectively.

The size of our video files is around 1.7 GB in uncompressed format. The size of the full databases we have built with Eigenfires is also around 2.3 GB in capacity. This size can be reduced into half (1.15 GB) by considering 50% compression and ignoring the remaining 1,100 Eigenfires. Other than our compression method for the database, the generated animations can also be compressed further with third-party codec libraries, such as Xvid and DivX. A screenshot of our system is shown in Figure 53.

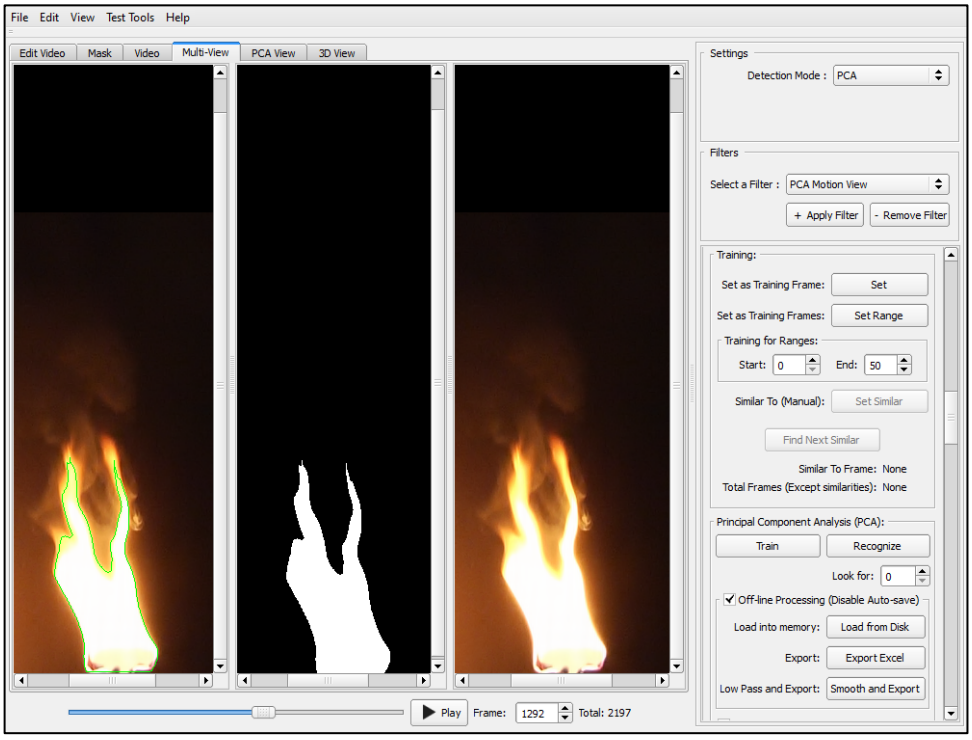


Figure 53: A screenshot of our system and its user interface, displaying contour detection and some PCA settings

7.1.1 Comparisons

Table 3 presents a summary of differences between our approaches and video texture Schödl et al. [53], which we covered in section 2.3.1. An example of video texture is shown in Figure 54 for two turbulent fire videos. In this figure, the transition is performed by synthesizing one or a few more frames using an undesirable crossfading, in which transitions appear as basic blending of images to fool the eyes.

Video Texture by Schödl et al. [53]	Our Techniques
The system is based on L_2 distances	The system is based on distances in PCA subspace for video files of at least 60 fps or higher
Intended to be used for various video samples under similar lightning conditions with fixed camera angles	We concentrated on combining different fire samples under various lightning conditions with cluttered backgrounds
No fire synthesis (It can be extended by Graphcut algorithm [36] and dynamic textures [15])	A unique fire can be synthesized quickly from the same database we built, which might not necessarily exist in the original videos
Loops and transitions for turbulent fire samples are based on undesirable crossfading or blending, and only among frames of one type of fire video. However, original images can be used for other non-turbulent scenes	Loops and transitions for turbulent fire samples are based on either image synthesis or original images from a big database of various fire samples, recorded with high frame rates
Transitions are very short and only responsible to connect any two frames that are optimized or look more similar to each other	Transition can be a long animation. Our transitions can act as new procedural fire animations by themselves, and are capable of connecting any two sequences that the user requests
No compression	The original videos and database can be compressed, and therefore animation stage can perform faster

Table 3: Comparison with Video Texture, Schödl et al.

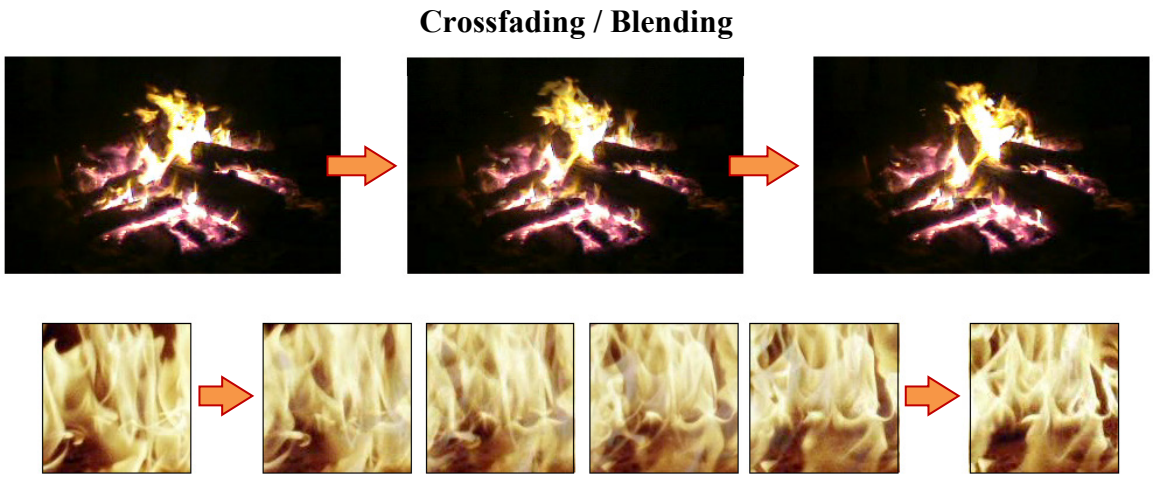


Figure 54: The samples of transition with undesirable crossfading, using video texture approach. Reprint from samples of Schödl et al. [53] and Kwatra et al. [36]

7.1.2 Speed-up Tricks

For professional industries, keeping the original video is not a major issue. This provides many opportunities that can be taken into account as our acceleration tricks either for real-time processing or initial database calculations. As a summary, the following operations are capable of speeding up the process:

- Using a modified copy of the original video for the animation stage.
- Resizing training images to a lower resolution for the training stage.
- Splitting the database into packages of a few thousand images for training stage.

Modified Video:

As we mentioned, we do not necessarily need to reconstruct the final images by projecting into fire space if the aim is rendering the results with the highest quality without compression or the results are only extracted from our training set. It is important to remember that in our samples the recognitions were performed among fire images that were already part of our training set (Fire *A*, *B* and *C*) and all their corresponding weights have

been saved in the database. This way, we are capable of comparing the weights and distances in real-time and finding image numbers suitable for our motion transitions. Hence, another copy of the original fire video can be prepared to accelerate the process. In this modified version, both pre-processing and post-processing tasks (e.g. background is darkened, etc.) should be performed completely, prior entering the animation stage. When the user obtains a list of frames based on Direct Bridging or End Growing Connection, the corresponding frame can be fetched from the modified video, which is already processed, instead of reconstructing them from the database every time or executing the same commands in real-time over and over. The images of Figure 43 are created using this method by changing the brightness of original videos and displaying actual frames directly. This means avoiding redundant projection times, and sticking to the time table listed in Table 2.

Resizing Training Images:

In recent film productions, the videos are usually captured in Full HD, 1080p format and mostly uncompressed, which requires huge amount of resources and processing. However, the training set does not need to comply with the same video resolutions. It can be reduced to a proper size by preserving an acceptable quality, much smaller than the full resolution, and it still can offer appropriate outputs for recognitions in animation stage. Accordingly, we may refer to the original video for the final images, same approach as the modified video. Similar resizing techniques are employed in [12][6] for hand gesture detections in real-time. No matter how large the resolution of the input video is, the small images (detected hands) of less than 200×200 resolution would be used for hand gesture comparisons.

Splitting the Database:

The initial calculation time of building a database of Eigenfires is relevant to the size of the training set. Our suggestion is to split very large videos into proper packages and then

creating separate smaller databases that can be used independently. For example, we recorded three various flames of 6 minutes length each. Now, 2 minutes from each of the videos can be combined to create three separate databases of 6 minutes length, instead of a large 18 minutes database. Although this approach can act as a limitation on some occasions, the one-time processing time of the initial database decreases significantly, and accordingly it may become a precious factor where the timing is critical.

7.2 Discussion of Result Enhancement

There are a few problems and suggestions that need to be taken into account, in order to achieve superior results out of our techniques:

- Issues related to combining solid fuels and burning objects
- Advantages of gaseous fuels
- Advantages of high speed cameras

If a new fire is going to be generated as a combination of variety of flames and fuels, it is recommended to deliberate about the purpose of this merging before recording any video. For instance, consider a scenario that the aim is mixing two distinct videos, one with a solid burning object and another one with a gaseous fuel. What would be the logic behind the transition from that solid fuel to gaseous fuel? One scenario might be: an animator is interested to show a shot that all of a sudden a large burning object shatters into tiny bits, and then those pieces or particles would be rendered in 3D modeling applications as part of the visual effects.

In order to improve the quality of the results obtained from fire synthesis using weight adjustment, the first 50 Eigenfires should be generated without any visible fuel as much as possible. Usage of gaseous fuels is one appropriate option. This guarantees that the users would receive a very smooth morphing, and no odd artifacts (ghostly mixture of burning objects) appear inside the flame. Otherwise, the transition from one burning object to

another type of burning object should bear a justifiable scenario, such as replacing them with 3D models of the solid objects. However, there is another option here which is cropping the videos right above the surface of the burning object and keeping the flames only.

Comparison of our sample videos recorded at both 30 fps and 60 fps reveal some subtle but important differences between most of the consecutive frames. The differences of shapes convinced us to drop the usage of videos with 30fps. Accordingly, more accurate results can be obtained from high-speed cameras that provide 1920x1080 full HD resolutions at frame rates of at least 300 fps, such as Phantom HD Gold, Photron FASTCAM SA2 or recently Sony NEX-FS700 camera. Although these cameras can record up to 2,000 fps at HD resolution, we do not suggest capturing higher than 300 fps, as it will dramatically increase the processing time of training stage, and there will be very little variation between the shapes of consecutive frames in fire, which we ignore with thresholding anyways.

7.3 Drawbacks and Limitations

Despite all the advantages that our methods offer, there are a few limitations in our system that should be taken into account. A few drawbacks of our techniques are as follows:

- End Growing Connection method performs better than our other motion transitions for various fire samples. However, it might fail to find the best shared image if the chosen dataset is very small.
- Our system does not automatically choose the optimized approach for motion transitions, and it is therefore the responsibility of the animators to select the correct approach or to rerun the sample with another approach in case of failure.
- Issues and artifacts of combining solid fuels as we discussed in this chapter.
- Problems of PCA calculation time for very large datasets

7.4 Discussion of 3D Extension

Since our current research supports 2D modeling of flames, many available image-based techniques can be employed to reconstruct 3D fire from our current two-dimensional results with the least amount of change. Some of them have been there for a while but have not been taken into account seriously in recent simulations of fire. We discuss those techniques here that we feel can actually extend our research.

Hasinoff and Kutulakos [24][25] present flame sheets and density sheet decomposition for reconstruction of semi-transparent scenes and flames, from a small set of input views that are recorded simultaneously. Density Sheets are able to reconstruct semitransparent scenes without many artifacts using as few as two input views, exactly suitable for our setup because we prepared our database of fire with two cameras installed in 90 degrees for all the captured samples. They compare the quality of interpolated views of Density Sheet solution (using two cameras) with other methods such as the multiplication solution, and an algebraic method based on fitting Gaussian blobs to the density [26]. It seems this approach is perfectly applicable to our current final results.

Hasinoff introduced another method in [26] by parameterizing the fire with blobs and improving the reconstruction with stochastic resampling. As a result, the 3D captured fire can be embedded into new virtual environments or mixed with other videos.

A sparse view tomographic approach [31] may also be applied for reconstructing a volumetric model from multiple images by placing more cameras around the fire . Although one or maximum two views can be used to generate new flames, it seems to be more successful with more cameras. One of the interesting features of this approach is fast processing of multi-video sequences for animation purposes using a linear system.

Krüger and Westermann [35] perform physically-based 2D simulation on GPU for two slices, and extrude those slices to 3D using particles or texture based approaches to create volumetric effects like smoke, fire or explosions. A similar approach can be used in our system for 3D visualization, considering the fact that our 2D simulation can be obtained

from our procedural approaches using each camera views instead of their physically-based 2D simulation of two slices.

There are a few ideas that can be considered for integration of the mentioned methods with our system. One idea can be using one of the views for PCA calculations, finding a list of motion transition for that view, and then referring to the corresponding frame in the second view to ensure that the animations of two input views are simultaneous and belong to the same shot. Without a doubt, further optimization and comparisons are required to obtain the best realistic fire animation using 3D reconstructions, which is beyond the topics of this thesis.

Chapter 8 Conclusions and Future Work

8.1 Conclusions

We explored fire behavior using PCA and introduced a few novel approaches of motion transitions and loops to model fire from lab controlled sample videos. This process is performed by looking for similar shapes of flames and extending the pattern of fire procedurally through PCA, or by synthesizing new flames. Our methods also allow larger number of frames to be identified for transitions among various chaotic samples of fire without tricking the eyes and without simple blending. Prominently it is not based on unrealistic random noises, but based on real-life fire motions. We introduced "Eigenfires" as a way of representing the features of real fire samples that provide simpler fire synthesis approaches. We visualized fire in such a way that its main characteristics and motions can be examined further, either in 2D/3D spaces or in higher dimensional spaces using graphs. Such features are: volume, direction, overall shape, shape of the tip of a flame, separation of flames, etc. We also proposed a few techniques, such as Hermite background removal, colorization and contour detection, to improve the quality of the final images and to be able to combine various fire samples. By decreasing the PCA dimension used for final image reconstruction, the data compression is obtained as a byproduct, in which we can compress the database up to 78% of the entire size by choosing only most important Eigenfires.

8.2 Summary of Contributions

In summary, our contributions can be itemized as follows:

- Developed a cross-platform system for both fire analysis and animation purposes.
- Proposed a few pre-processing techniques to remove the background of fire videos under various lighting conditions.
- Proposed "Eigenfires" as a way of representing the main features of real fire samples using PCA.

- Analyzed a combination of small-scale fire samples in PCA subspace.
- Developed a few techniques to synthesize a new fire by combining Eigenfires.
- Developed novel techniques of "Motion Transitions" to automatically generate realistic-looking fire animations by repeating or extending a given sequences of fire footage through selecting desired ranges of various video clips.
- Developed a few algorithms for synthetic colorization and compression of the original video.

8.3 Future Research

There are many features that may extend this work for a wider number of users:

- **Increasing the dataset:** Future work aims to increase the database by covering more types of fire, including large-scale fire if it is allowed by safety regulations.
- **3D Extension**: The current 2D work can be converted to a 3D fire system.
- **GPU Extension:** Parts of the system can be implemented on GPU to be used in any real-time interactive environments by freeing the CPU resources.
- **Further motion transitions**: The proposed motion transition algorithms can be easily extended. Many path finding algorithms can be incorporated with our system to generate new animations.
- **Further analysis of motions**: Motion patterns of the fire in PCA space can be analyzed further to help construction of a new flame animation from scratch. There are some signs of sine wave patterns that regularly occur every couple of seconds in various dimensions. Further statistical analysis of this information may improve the system by rejecting improper frames that are found as motion transitions.
- **Adaptive thresholding**: The dedicated thresholds of motion transitions may be combined for various types of flames, or be adaptively changed based on the current size or volume of the flames.

- **Auto selection of algorithms**: Currently it is the responsibility of the animators to choose a proper technique for their desired motion transition. This mechanism can be improved by a prediction procedure, or by a quick testing and processing the results once to find out whether they fail, and then another method can be performed automatically. It would be very useful, particularly for basic users, to see such functionality.
- **User interface extensions**: The user-interface can be improved to be more intuitive in a touch-based computer environment, such as tablets. On the desktop, the GUI can also be enhanced for basic users by a few capabilities such as displaying and previewing specific animation ranges and drag-and-drop feature to insert each piece into a time-line similar to video editing applications.

References

[1] Adabala, N. & Hughes, C. (2004), A Parametric model for real-time flickering fire. In *Proceedings of Computer Animation and Social Agents* (CASA).

[2] Amarasinghe, D. & Parberry, I. (2011), Towards fast, believable real-time rendering of burning objects in video games. In *Proceedings of the 6th International Conference on Foundations of Digital Games,* ACM, New York, NY, USA, pp. 256-258.

[3] Bajwa, I.S.; Naveed, M.S.; Asif, M.N. & Hyder, S.I. (2009), Feature Based Image Classification by using Principal Component Analysis. *Journal of Graphics Vision, and Image Processing,* vol. 09, no.2, pp. 11-17.

[4] Beaudoin, P.; Paquet, S. & Poulin, P. (2001), Realistic and controllable fire simulation. *Graphics interface 2001*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, pp. 159-166.

[5] Bhat, K. S.; Seitz, S. M.; Hodgins, J. K. & Khosla, P. K. (2004), Flow-based video synthesis and editing. In *ACM SIGGRAPH 2004 Papers* (SIGGRAPH '04), Joe Marks (Ed.). ACM, New York, NY, USA, pp. 360-363.

[6] Birk, H.; Moeslund, T. B. & Madsen, C. B. (1997), Real-time recognition of hand alphabet gestures using principal component analysis. In *Proceedings of the 10th Scandinavian Conference on Image Analysis*, Lappeenranta, Finland, pp. 261-268.

[7] Bridson R. (2008), Fluid Simulation for Computer Graphics. A K Peters/CRC Press, chapter 1.

[8] Chenebert, A.; Breckon, T.P. & Gaszczak, A. (2011), A non-temporal texture driven approach to real-time fire detection. In *Proceedings of 18th IEEE International Conference on Image Processing (ICIP)*, pp. 1741-1744.

[9] Chiba, N.; Muraoka, K.; Takahashi, H. & Miura, M. (1994), Two-dimensional visual simulation of flames, smoke and the spread of fire. *The Journal of Visualization and Computer Animation* 5, pp. 37-53.

[10] Choi J.; Park, H.& Park J. (2011), Hand shape recognition using distance transform and shape decomposition. In *Proceedings of 18th IEEE International Conference on image processing (ICIP),* Brussels, Belgium, pp. 3605-3608.

[11] Crane, K.; Llamas, I. & Tariq, S. (2007), Real Time Simulation and Rendering of 3D Fluids. Nguyen, H., ed., *Addison-Wesley*, chapter 30.

[12] Dardas, N.H. & Petriu, E.M. (2011), Hand gesture detection and recognition using principal component analysis. *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA)*, pp. 1-6.

[13] Deepu, V.; Madhvanath, S. & Ramakrishnan, A.G. (2004), Principal component analysis for online handwritten character recognition. *In Proceedings of the 17th International Conference on Pattern Recognition. ICPR*, vol. 2, pp. 327- 330.

[14] De Witt, T.; Lessig, C. & Fiume, E. (2012). Fluid simulation using Laplacian eigenfunctions. *ACM Trans. Graph*. vol. 31, no. 1, article 10, pp. 1-11.

[15] Doretto, G.; Chiuso, A.; Soatto, S. & Wu, Y. (2003), Dynamic textures. *International Journal of Computer Vision*, vol. 51, no. 2, pp. 91-109.

[16] Doretto, G. & Soatto, S. (2003), Editable dynamic textures. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol.2, pp. 137-142.

[17] Douglas, D. & Peucker, T. (1973), Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization, University of Toronto Press*, vol. 10, no. 2, 112-122.

[18] Fedkiw, R.; Stam, J. & Jensen H.W. (2001), Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '01). ACM, New York, NY, USA, pp. 15-22.

[19] Feldman, B. E.; O'Brien, J. F. & Arikan, O. (2003), Animating suspended particle explosions. In *ACM SIGGRAPH 2003 Papers* (SIGGRAPH '03), New York, NY, USA, pp. 708-715.

[20] Fuller, A. R.; Krishnan, H.; Mahrous, K.; Hamann, B. & Joy, K. I. (2007), Real-time procedural volumetric fire. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, pp. 175-180.

[21] Hamadache, M.; Kim J. & Lee D. (2012), Principal component analysis for 3D-manipulator robot control system. *Electrotechnical Conference (MELECON), 16th IEEE Mediterranean*, pp. 395-398.

[22] Hanek, R. & Beetz, M. (2004), The Contracting Curve Density Algorithm: Fitting Parametric Curve Models to Images Using Local Self-Adapting Separation Criteria. *Int. J. Computer Vision*, vol. 59, no. 3, pp. 233-258.

[23] Harris, M. (2004), Fast Fluid Dynamics Simulation on the GPU. In *GPU Gems*, Addison-Wesley, pp. 637-665.

[24] Hasinoff, S. & Kutulakos, K. (2007), Photo-Consistent Reconstruction of Semitransparent Scenes by Density-Sheet Decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 870 -885.

[25] Hasinoff, S. W. & Kutulakos, K. N. (2003), Photo-Consistent 3D Fire by Flame-Sheet Decomposition. In *Proceedings of the Ninth IEEE International Conference on Computer Vision,* vol. 2, pp. 1184-1191.

[26] Hasinoff, S.W. (2002), Three-Dimensional Reconstruction of Fire from Images. *Master's thesis, Dept. of Computer Science*, Univ. of Toronto.

[27] Hewitt, R. (2007), Seeing With OpenCV - A Five Part Series. SERVO Magazine, T&L Publications, Inc.

[28] Hong, J.-M.; Shinar, T. & Fedkiw, R. (2007), Wrinkled flames and cellular patterns. In *ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, vol. 26, no. 47.

[29] Hongliang, L.; Qing, L. & Sun'an, W. (2012), A novel fire recognition algorithm based on flame's Multi-features Fusion. *International Conference on Computer Communication and Informatics (ICCCI)*, pp. 1-6.

[30] Horvath, C. & Geiger, W. (2009), Directable, high-resolution simulation of fire on the GPU. In *ACM SIGGRAPH 2009*, ACM, New York, NY, USA, no. 41, pp. 1-8.

[31] Ihrke, I. & Magnor, M. (2004), Image-based tomographic reconstruction of flames. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, pp. 367-375.

[32] Kass, M.; Witkin, A. & Terzopoulos, D. (1988), Snakes: Active contour models. *International Journal of Computer Vision*. vol. 1, no. 4, pp. 321-331.

[33] Kawulok, M. & Smolka, B. (2010), Competitive image colorization. *17th IEEE International Conference on Image Processing (ICIP),* pp.405-408.

[34] King, S.A.; Crawfis, R.A. & Reid, W. (1999), Fast Volume Rendering and Animation of Amorphous Phenomena. In *Volume Graphics*, chapter 13, pp. 229-242

[35] Krüger, J. & Westermann, R. (2005), GPU simulation and rendering of volumetric effects for computer games and virtual environments. In *Proceedings Eurographics*, pp. 685-693.

[36] Kwatra, V.; Schödl, A.; Essa, I.; Turk, G. & Bobick A. (2003), Graphcut textures: image and video synthesis using graph cuts. In *ACM SIGGRAPH Papers (SIGGRAPH'03)*, New York, NY, USA, 277-286.

[37] Lamorlette, A. & Foster, N. (2002), Structural modeling of flames for a production environment. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM SIGGRAPH 2002, New York, NY, USA, pp. 729-735.

[38] Lata, Y. V.; Tungathurthi, C.K.B; Ram Mohan Rao, H.; Govardhan, A. & Reddy., L.P. (2009) Facial Recognition using Eigenfaces by PCA. *International Journal of Recent Trends in Engineering*, vol. 1, no. 1.

[39] Li, W.; Mao .K; Zhou, X.; Chai T. & Zhang, H. (2009), Eigen-flame image-based robust recognition of burning states for sintering process control of rotary kiln. In *Proceedings of the 48th IEEE Conference on Decision and Control, held jointly with the 28th Chinese Control Conference. CDC/CCC*, pp. 398-403.

[40] Masiero, A. & Chiuso A. (2006), Non linear temporal textures synthesis: a monte carlo approach. In *Proceedings of the 9th European conference on Computer Vision* (ECCV'06), pp. 283-294.

[41] Melek, Z. & Keyser, J. (2002), Interactive Simulation of Fire. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, pp. 431-432.

[42] Min, K. & Metaxas, D. (2007), A combustion-based technique for fire animation and visualization. *The Visual Computer*, vol. 23, pp. 679-687.

[43] Nguyen, D. Q.; Fedkiw, R. & Jensen, H. W. (2002), Physically based modeling and animation of fire. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, (SIGGRAPH '02), New York, NY, USA, pp. 721-728.

[44] Oh, C.-M. & Lee C.-W. (2008), Bayesian shape recognition using Principle Component Analysis and Modified Chain Codes. *International Conference on Control, Automation and Systems, ICCAS*, pp. 2138-2141.

[45] Olano, M. (2005), Modified noise for evaluation on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (HWWS '05), New York, NY, USA, pp. 105-110.

[46] Pegoraro, V. & Parker, S. G. (2006), Physically-Based Realistic Fire Rendering. In *Eurographics Workshop on Natural Phenomena*, E. Galin and N. Chiba (editors), pp. 237-244.

[47] Perlin, K. (1985), An image synthesizer. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '85). ACM, New York, NY, USA, pp. 287-296.

[48] Perlin, k. & Hoffert, E. M. (1989), Hypertexture. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '89). ACM, New York, NY, USA, pp. 253-262.

[49] Perry, C. H. & Picard, R. W. (1994), Synthesizing Flames and their Spreading. In *Proceedings of the Fifth Eurographics Workshop on Animation and Simulation*, pp. 1-14.

[50] Quanhua, C.; Zunxiong, L. & Guoqiang, D. (2008), Facial gender classification with eigenfaces and least squares support vector machine. *Journal of Artificial Intelligence*, vol. 1, no. 1, pp. 28-33.

[51] Rasmussen, N.; Nguyen, D. Q.; Geiger, W. & Fedkiw, R. (2003), Smoke simulation for large scale phenomena. In *ACM SIGGRAPH 2003 Papers*, New York, NY, USA, pp. 703-707.

[52] Reeves, W. T. (1983). Particle systems-a technique for modeling a class of fuzzy objects. In *Proc. of ACM SIGGRAPH* '83, pp. 359-375.

[53] Schödl, A.; Szeliski, R.; Salesin, D. H. & Essa, I. (2000), Video textures. *In Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH'2000),* pp. 489-498.

[54] Sirovich, L. & Kirby, M. (1987), Low-dimensional procedure for the characterization of human faces. In *Journal of the Optical Society of America A4*, pp. 519-524.

[55] Smith, L. I. (2002) A tutorial on principal components analysis. Cornell University, USA.

[56] Stam, J. (2003), Real-Time Fluid Dynamics for Games. In *Proceedings of the Game Developer Conference*.

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

[58] Stam, J. & Fiume, E. (1995), Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (SIGGRAPH '95), ACM, New York, NY, USA, pp. 129-136.

[59] Stam, J. & Fiume, E. (1993), Turbulent wind fields for gaseous phenomena. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, (SIGGRAPH '93), ACM, New York, NY, USA, pp. 369-376.

[60] Suzuki, S. & Abe, K. (1985), Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32-46.

[61] Tai, Y.-W.; Jia, J. & Tang, C.-K. (2005), Local color transfer via probabilistic segmentation by expectation-maximization. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR 2005, vol. 1, pp. 747- 754.

[62] Takahashi, J.; Takahashi, H. & Chiba, N. (1997), Image Synthesis of Flickering Scenes Including Simulated Flames. *IEICE transactions on information and systems*, vol. 80, no. 11, pp. 1102-1108.

[63] Tan, J. & Yang, X. (2009), Physically-based fluid animation: A survey. *Science in China Series F: Information Sciences* vol. 52, pp. 723-740.

[64] Turk, M. & Pentland, A. (1991), Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71-86.

[65] Turk, M. & Pentland, A. (1991); Face recognition using eigenfaces. In *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 586-591.

[66] Tzimiropoulos, G.; Mitianoudis, N. & Stathaki, T. (2007), Robust Recognition of Planar Shapes Under Affine Transforms Using Principal Component Analysis. *Signal Processing Letters, IEEE*, vol. 14, no. 10, pp. 723-726.

[67] Umenhoffer, T.; Szirmay-Kalos, L. & Szijártó, G. (2006), Spherical billboards and their application to rendering explosions. In *Proceedings of Graphics Interface*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, pp. 57-63.

[68] Vanzine, Y. & Vrajitoru, D. (2008), Pseudorandom Noise for Real-Time Volumetric Rendering of Fire in a Production System, in *Volume Graphics*, pp. 129-136.

[69] Wei, X.; Li, W.; Mueller, K. & Kaufman, A. (2002), Simulating fire with texture splats. In *Proceedings of the conference on Visualization, IEEE Computer Society*, Washington, DC, USA, pp. 227-235.

[70] Xi, P.; Lee, W.-S.; Frederico, G.; Joslin, C. & Zhou, L. (2006), Comprehending and transferring facial expressions based on statistical shape and texture models. In *Proceedings of the 24th Computer Graphics International*, Springer-Verlag, Berlin, Heidelberg, pp. 265-276.

[71] Xi, P.; Lee, W.-S. & Shu, C. (2007), A Data-driven Approach to Human-body Cloning Using a Segmented Body Database. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications, IEEE Computer Society*, Washington, DC, USA, pp. 139-147.

[72] Xu, C. & Prince, J.L. (1998), Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 359-369.

[73] Yuan, L.; Wen, F.; Liu, C. & Shum H. Y. (2004), Synthesizing dynamic texture with closed-loop linear dynamic system. *European Conference on Computer Vision (ECCV)*, pp. 603-616.

[74] Zuo, L.; Wang, Y. & Tan, T. (2002), Personal handwriting identification based on PCA. In *Proceedings of SPIE Second International Conference on Image and Graphics*, pp. 766-771.

Related Publications by the Author

Nikfetrat, N. & Lee, W.-S. (2013), Fire Visualization using Eigenfires. *Intelligent Computer Graphics 2012*, Springer Berlin / Heidelberg, vol. 441, pp. 189-207.