**BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA**
**FACULTY OF MATHEMATICS AND COMPUTER SCIENCE**
**SPECIALIZATION COMPUTER SCIENCE**

# DIPLOMA THESIS

# Simulating Fire in Unity 3D

**Supervisor**
**Assoc. Prof. Rareș Florin Boian, PhD**

*Author*
*Neamțiu Ovidiu Zaris*

2025

**UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA**
**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**
**SPECIALIZAREA INFORMATICĂ**

# LUCRARE DE LICENŢĂ

# Simularea Focului în Unity 3D

**Conducător științific**
**Assoc. Prof. Rareș Florin Boian, PhD**

*Absolvent*
*Neamțiu Ovidiu Zaris*

2025

ABSTRACT

This paper presents an innovative approach to the simulation of dynamic, procedurally generated flames in a real-time virtual environment. The project makes use of Unity3D's capabilities to represent fire in a realistic and captivating way by using custom algorithms and shaders. The main contributions to this project have been in the design of the procedural flame generation and control system, which entails the construction of an intricate flame simulation with a degree of randomness, simulating the erratic behavior of actual fire.

A number of different, yet connected modules, each focusing on a different component of the behavior of the flame, are used to create the flame simulation. The flame's form, "flicker" effect, color gradient, expansion as it spreads, and mobility across a surface are some of these characteristics. The flame's shape is dynamically controlled through a custom shader that manipulates the vertices of a mesh based on noise, creating the illusion of a flame's natural movement.

The system also features an approach for simulating flame spread. As the flame moves, it leaves behind a trail of new flames, simulating the spread of fire. The direction of the flame's movement is randomly generated, yet constrained to remain within a defined surface area. To ensure that the flame stays within this boundary, a raycasting mechanism that detects when the flame is about to go off the table is employed, adjusting the flame's direction accordingly.

This project has explored a relevant approach to visualising and simulating realistic flame behaviour in a three-dimensional environment. Using Unity and shader programming, the system is capable of producing flames that not only flicker and move in a manner consistent with real-world observations but also propagate across surfaces, thereby mimicking the spread of fire. The system's flame objects are also responsive to the viewer's perspective, thanks to the inclusion of billboard functionality. The project is a demonstration of the creative potential inherent in game development and computer graphics.

# Contents

# Chapter 1

# Introduction

## 1.1 Objectives

The primary goal is to develop a system capable of rendering realistic flame behavior and propagation in a 3D environment using the Unity game engine. This involves creating a dynamic, visually appealing flame effect using C# and shader programming, with a particular focus on achieving convincing flame flicker and movement. By enabling flame objects to move and spread across surfaces, the technology also aims to simulate the spread of fire.

## 1.2 Structure

The structure of this paper unfolds as follows:

Chapter 1, the introduction, outlines the objectives, relevance, and motivation of the research.

Chapter 2, concerned with the state of the art, provides a comprehensive review of the current flame simulation techniques and recent developments, with an emphasis on different noise functions and their application in simulations. Various simulation techniques and uses of game engines are discussed.

Chapter 3, detailing the framework and methodology, offers an in-depth look at the theoretical foundations of the project. It details the dynamics of flame movement, propagation, and evolution, as well as camera-facing flames by means of billboarding techniques.

Chapter 4, describes the interactive panel, elemental simulation, and customization capabilities of the flame dynamics for game development. The potential applications of these capabilities are also discussed.

The paper concludes with Chapter 5, presenting the conclusions, contributions, implications and applications of the study. Future work and final remarks are also

included in this chapter.

## 1.3 Relevance and motivation

This project's relevance stems from its potential uses in a variety of fields. The developed system can be employed in virtual reality simulations, video game development, architectural visualization, and even fire safety protocols.

The project can increase the visual richness and interactivity of digital environments and provide a realistic representation of fire behavior and spread, making the user experience more engaging and credible. For instance, in video games, this could increase player engagement and satisfaction. In architectural visualizations or virtual reality simulations, it could provide a more accurate representation of fire scenarios, which could be beneficial for safety planning and disaster management.

This study also paves the way for more research and development outside of these immediate uses. It may be extended to represent numerous fire forms and adjust to diverse environmental conditions. With further optimization and enhancements, it could contribute to the development of more advanced fire simulation systems in the future. Thus, the relevance and benefits of this project extend from enhancing current digital experiences to providing simulations aimed to test the safety of real world environments.

# Chapter 2

# State of the art

## 2.1 Introduction

This chapter provides an exploration of the current state of the art in flame simulation. Emphasis lies on real-time techniques suitable for interactive applications. Fundamental principles of 3D graphics and rendering form the core of these simulations. These principles lead to a discussion on various simulation techniques and the role of noise functions, such as Perlin noise, in creating realistic, organic patterns.

Different methodologies for simulating fire and similar volumetric phenomena within a 3D environment will be analyzed, shedding light on the strengths and limitations of each approach. Furthermore, the significant role of game engines in providing widespread access to powerful, real-time 3D graphics capabilities receives acknowledgment, thus broadening the application spectrum of these simulations.

### 2.1.1 General Theory

#### Fundamentals of 3D Graphics

3D graphics involve the creation and manipulation of objects in a three-dimensional space. These objects are typically represented as a collection of polygons, often triangles, which can be positioned and oriented in any way within the 3D space. To create a realistic representation, these objects are often textured, which involves mapping a 2D image onto their surfaces.

#### Rendering Pipeline

The process a graphics system uses to turn a 3D scene into a 2D image is known as the rendering pipeline. Vertex processing, basic assembly, rasterization, and pixel processing are typically involved in this process. Each of these phases is essential in defining how the scene will ultimately look.

#### Shaders

Shaders are small programs that run on the GPU and control the rendering process at various stages. Vertex shaders manipulate the properties of vertices, such as their position and color, while fragment shaders determine the color of individual pixels. Shaders can be used to create a wide range of effects, from simple lighting and texturing to complex procedural patterns like the flame simulation in this project.

#### Lighting models

Lighting models are algorithms that simulate the way light interacts with objects in a scene. They take into account factors like the angle and color of the light sources, the characteristics of the surfaces of the objects, and the viewing position. Common lighting models include Lambertian reflection for diffuse surfaces and Phong reflection for specular highlights.

#### Texture Mapping

The process of putting a 2D picture or texture to a 3D object is known as texture mapping. This can be used to enhance an object's detail without increasing the complexity of its geometry. In the context of flame simulation, texture mapping can be used to apply a noise pattern to the flame, creating a more realistic appearance.

In the following sections, these principles will be applied to the specific context of flame simulation, exploring how they can be used to create a convincing representation of fire in a real-time simulation.

## 2.2 Simulation Techniques

### 2.2.1 Fracturing

Fracturing techniques are commonly used in computer graphics to simulate the breaking or shattering of objects. These techniques can also be applied to simulate complex volumetric phenomena such as fire. The basic idea behind fracturing is to divide an object into smaller pieces, which can then be individually manipulated to create the illusion of the object breaking apart.

One approach to fracturing is the use of Voronoi diagrams, which can partition space into regions based on distance to a set of seed points. Each region corresponds to one piece of the fractured object. This approach can produce visually realistic results, as the shapes of the pieces can be controlled by the choice of seed points.

In the context of fire simulation, fracturing techniques could potentially be used to generate the shapes of individual flames or sparks. For example, a large fire could

be represented as a collection of smaller flames, each corresponding to a region in a Voronoi diagram. This approach could allow for more detailed and realistic simulations, as each flame could be individually animated and rendered.

A study by Thomas and Zhang (2023) [TZ23] explored the use of real-time fracturing techniques in video games, comparing these to traditional pre-fracturing methods in terms of visual realism and performance. They found that while real-time fracturing could produce more visually realistic results, it also placed greater demands on computational resources. This highlights the need for careful optimization in the development of real-time simulations. The techniques discussed by Thomas and Zhang, such as the use of Voronoi diagrams for fracturing, could potentially be applied to the procedural generation of fire shapes, providing a possible direction for future research.

### 2.2.2 Flame Rendering with the GPU

One of the significant advancements in the field of fire simulation is the use of GPU (Graphics Processing Unit) and CUDA (Compute Unified Device Architecture) for real-time flame rendering. This technique was proposed by Wei Wei and Yanqiong Huang in their paper "Real-time Flame Rendering with GPU and CUDA" (Wei and Huang, 2011) [WH11].

The authors propose a method of flame simulation based on the Lagrange process and chemical composition, which is non-grid and overcomes the problems associated with grid-based methods. The turbulence movement of the flame is described by the Lagrange process, and chemical composition is added into the flame simulation, which increases the authenticity of the flame.

For real-time applications, the paper simplifies the Euclidian Minimum Spanning Tree (EMST) model, which is used for modeling the chemical composition of fire. The authors also utilize GPU-based particle systems combined with OpenGL VBO and PBO unique technology to accelerate the speed of vertex and pixel data interaction between CPU and GPU. This results in a significant increase in the frame rate of rendering, achieving fast dynamic flame real-time simulation.

Furthermore, the paper presents a strategy to implement flame simulation with CUDA on GPU, which achieves a speed up to 2.5 times the previous implementation. This method has significant implications for computer animation and virtual reality, providing a more efficient and realistic simulation of fire.

This technique is relevant to the proposed model as it provides a method for real-time simulation of fire, a volumetric phenomenon, using advanced computational techniques. The use of GPU and CUDA for rendering not only increases the speed and efficiency of the simulation but also enhances the realism of the rendered flame.

## 2.3 Noise Functions in Graphics and Simulations

The study "A Survey of Procedural Noise Functions" by A. Lagae and colleagues [LLC+10] offers an extensive analysis of various procedural noise functions utilized in the field of computer graphics. It includes a comparison of Perlin Noise with several other noise functions, highlighting their unique characteristics and applications.

### 2.3.1 Perlin Noise

Perlin Noise, developed by Ken Perlin in 1985 [Per02], is a gradient noise function that has been widely adopted in both academic and commercial settings. It generates noise by calculating a pseudo-random gradient at each of the eight closest vertices on an integer cubic lattice, followed by a splined interpolation. The pseudo-random gradient is determined by hashing the lattice point and using the outcome to select a gradient. The interpolant is a quintic polynomial, ensuring a continuous noise derivative. Despite its simplicity and speed, Perlin Noise is weakly band-limited, which can lead to aliasing and loss of detail.

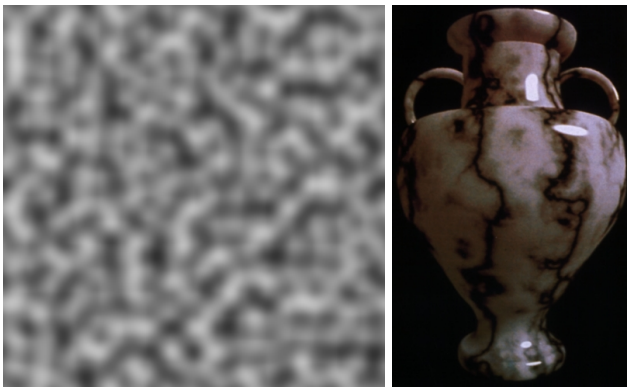


Figure 2.1: Perlin Noise. (a) Perlin's popular noise function, the first procedural noise function. (Image source `http://kitfox.com/projects/perlinNoiseMaker/`). (b) Perlin's famous marble vase, one of the first procedural textures created using Perlin noise. (Image source: `https://redirect.cs.umbc.edu/~ebert/691/Au00/Notes/procedural.html`).

### 2.3.2 Wavelet Noise

Cook and DeRose proposed a new type of noise called wavelet noise in 2005 [CD05]. Their inspiration stems from noticing that Perlin noise is prone to problems with aliasing and details loss because it is only weakly band-limited. Wavelet noise represents noise as a quadratic B-spline surface, and is almost perfectly band-limited.

This noise is produced by taking an image filled with random noise, downsampling it, upsampling it again, and subtracting the result from the original image. This leaves the band-limited part of the noise that is not representable at half-size.

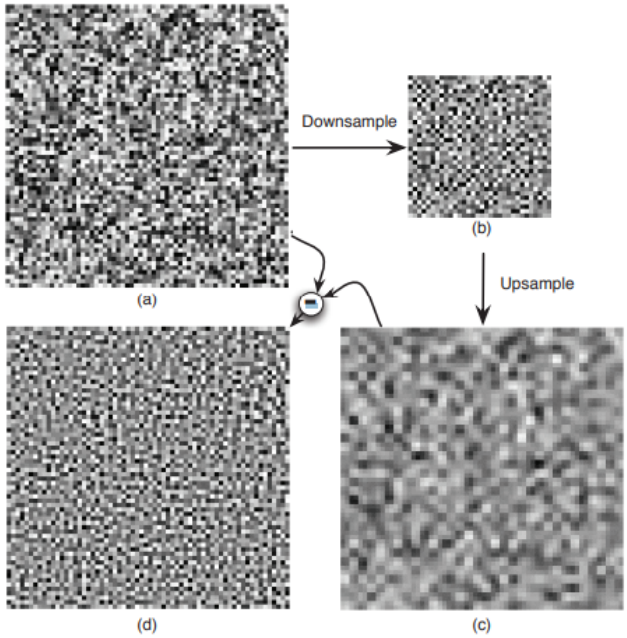



Figure 2.2: Wavelet noise generation. (a) Image R of random noise. (b) Half-size image R↓. (c) Half-resolution image R↓↑. (d) Noise band image N = R - R↓↑. (Image source: [CD05]).

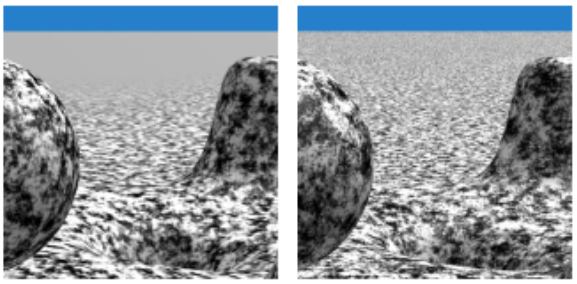



Figure 2.3: Comparison between images produced using (a) Perlin noise and (b) wavelet noise. Image (a) showcases the optimal balance between detail and aliasing achieved through the best practices use of Perlin noise; it's notable how much high-frequency detail in the far distance is absent. (Image source: [CD05]).

### 2.3.3 Anisotropic Noise

In 2008, Goldberg et al. [GZD08] introduced a new concept known as anisotropic noise. They noticed that the noise functions available at the time only supported

isotropic filtering, which necessitated a compromise between aliasing artifacts and loss of detail. To address this, they proposed a new noise function that supports high-quality anisotropic filtering.

Anisotropic noise works by dividing the frequency domain into oriented subbands to generate noise textures. These bands are not only narrowly band-limited in scale but also have a preferred orientation. The creation of anisotropic noise is based on steerable filters that partition the frequency domain. These filters have several key properties that are vital for noise generation. They define a subband that is tightly localized in scale and orientation, implement an invertible transform, and are steerable in orientation. This allows for the exact recovery of a signal from its decomposition into subbands and avoids interpolation artifacts when blending the subbands for noise filtering.

Furthermore, the precomputation of noise subbands at a single scale allows for efficient scaling and offers a balance between storage requirements, rendering speed, and visual quality. The technique's adaptability extends to generating detailed surface noise through 2D texture mapping and parametric distortion compensation.

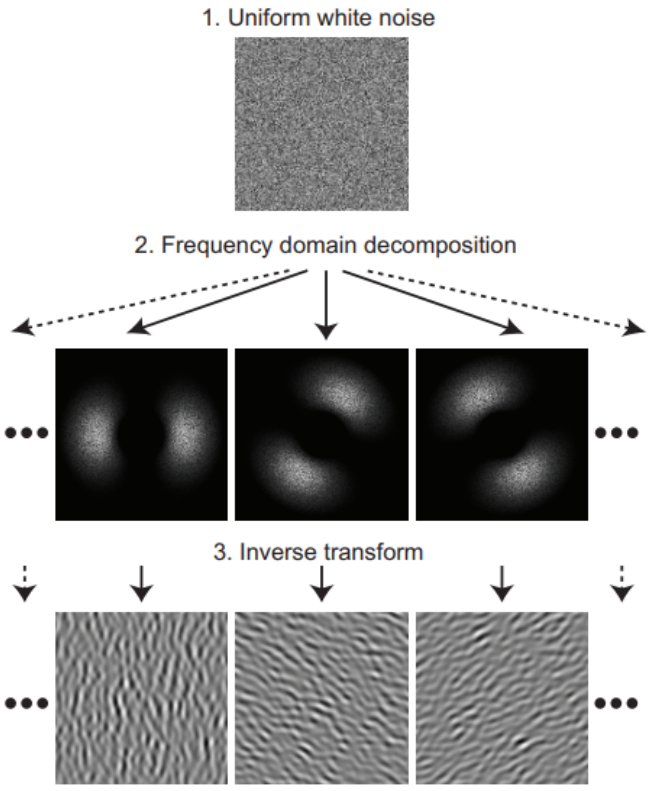



Figure 2.4: Anisotropic noise generation. Illustration of spectral noise generation. The frequency domain decomposition has three orientations. Three oriented subbands at the same scale and their corresponding spatial domain images are shown, which are stored as textures. (Image source: [GZD08].)

While Perlin Noise is appreciated for its speed and simplicity, it has limitations in terms of band-limiting. Wavelet Noise and Anisotropic Noise offer solutions to these limitations, with Wavelet Noise being almost perfectly band-limited and Anisotropic Noise supporting high-quality anisotropic filtering.

## 2.4 Use of Game Engines in Simulations

Game engines, traditionally used in the gaming industry, have found a new application in the realm of simulations. This technique involves leveraging the advanced capabilities of game engines to create realistic, interactive, and immersive simulation environments.

Game engines offer a plethora of benefits when used in simulations. Firstly, they provide a high degree of realism, thanks to their advanced graphics rendering capabilities. This allows for the creation of visually appealing and detailed simulation environments, enhancing the user's immersion and engagement. Secondly, game engines come with built-in physics engines, which can simulate real-world physics accurately. This is crucial in many simulation scenarios, such as flight or vehicle simulations, where accurate physics modeling is key. Lastly, game engines often include tools for artificial intelligence (AI), enabling the creation of intelligent agents within the simulation.

However, the use of game engines in simulations is not without its challenges. One of the primary concerns is the steep learning curve associated with these tools. Game engines are complex pieces of software, and mastering them requires significant time and effort. Moreover, while game engines are designed for real-time performance, this can sometimes come at the cost of accuracy. In other words, in order to maintain smooth performance, game engines may occasionally simplify or approximate certain calculations, which could potentially impact the accuracy of the simulation.

### 2.4.1 Unity for Training Simulations

The study "A Game-Based Learning Framework for Controlling Cyber-Physical Systems", brought forward by Liu, D. and Frasheri, M. and Vyatkin, V., [LFV20] provides an innovative perspective on utilizing game engines for educational purposes, specifically for understanding Cyber-Physical Systems (CPS). The authors developed a unique game-based framework that allows users to control a CPS, in this case, a drone, to accomplish various tasks within a game environment.

The game engine of choice for this project was Unity3D. The authors' decision to use Unity3D was influenced by several factors. Unity3D is a popular game engine

that is compatible with multiple platforms and has a large user community. Its visual scripting tool allows users to create complex game mechanics without needing extensive programming skills, making it more accessible to users who may not have a strong coding background. Additionally, Unity3D's ability to integrate external libraries was a crucial feature for the implementation of the CPS control algorithms.

The game designed by the authors mimics real-world CPS operations, with the drone's behavior being controlled by algorithms. The game's levels increase in complexity, each introducing new CPS control concepts. The control algorithms for the drone were implemented using an external library, which was integrated into Unity3D. This allowed the authors to utilize existing control algorithms and concentrate on the game-based learning aspect of their project.

The primary aim was to create a learning platform for understanding CPS control. The game provides immediate feedback on the player's actions, reinforcing the learning process. A user study conducted by the authors to evaluate the effectiveness of their game-based learning framework indicated that the game was successful in teaching the basics of CPS control, with players showing improved understanding and performance over time.

The authors successfully created a game-based learning framework for CPS control using the Unity3D game engine. This game, which allows players to control a drone using real-world control algorithms, provides a practical, hands-on learning experience. The game was found to be effective in teaching the basics of CPS control, demonstrating the potential of game engines in educational simulations.

### 2.4.2 Frostbite for Fire Propagation

The paper titled "Dynamic Real-Time Fire Propagation for Networked Multiplayer Games" by Iris Kotsinas and Viktor Tholén [KT22], from Linköping University, presents an approach to simulating fire propagation in networked multiplayer games. The motivation behind their work was to enhance the realism and interactivity of fire effects in games, particularly in the context of multiplayer networked games.

The authors used the Frostbite game engine, developed by DICE, for their implementation. The approach involved two main methods: a level set method and a node graph implementation. The level set method is a numerical technique used for tracking interfaces and shapes, in this case, the fire front. The node graph implementation is a method that uses a graph structure to represent the propagation of the fire.

The fire behavior originated from a generated vector field, which is a mathematical construct that assigns a vector (direction and magnitude) to each point in a space. This vector field was based on level properties such as wind direction and

speed, and fuel availability. The vector field was used for both the level set method and the node graph implementation.

The results of the study showed that the level set method resulted in a successful fire propagation simulation. The vector field created in the pre-processing played a considerable part in the final behavior of the fire propagation. The normal advection operator, which is responsible for moving the level set (the fire front) forward in time, and the vector field operator, which adds the behavior of the fire, worked together to create a realistic fire behavior.

The node graph implementation was also successful, but the networking of this method only took into account the most simple scenario, where the client and server are synced. The current implementation does not support how the client would get the state of the game at mid-simulation.

The authors concluded that while both methods were successful, there are areas for future work. These include improving the heat sampling algorithm, extending the systems into 3D, and exploring indoor simulations with the node graph method.

## 2.5 Recent Developments and Future Trends

### 2.5.1 CFAST, FDS and Smokeview

NIST's (The US' National Institute of Standards and Technology) Fire Research Division is involved in the development of computational tools to analyze fire behavior. These tools include the Fire Dynamics Simulator (FDS), a computational fluid dynamics model, Consolidated Fire and Smoke Transport (CFAST) zone model, and Smokeview, a visualization tool for the output from FDS and CFAST. The ongoing research is aimed at enhancing the capabilities of these tools, with a particular focus on improving the accuracy of predicting burning rates for different types of fuels.

The FDS has proved invaluable in designing fire protection measures for buildings, leading to significant cost savings. It has also been used in forensic investigations to recreate the conditions and behavior of fires. However, there are ongoing challenges in improving the accuracy of the tool, especially in predicting heat release rates in large-scale fires. This involves a complex interplay of factors including local heat feedback, soot emissions prediction, and understanding the thermal properties and behavior of different materials when exposed to fire.

FDS has been expanding its scope from modeling simple plumes and compartment fires to more complex scenarios such as wildfires at the interface of urban and wildland areas. These scenarios present new challenges, such as handling changing external boundary conditions, modeling wind field data, and dealing with complex terrain. An important aspect of modeling wildfires is accounting for ember trans-

port, a significant mechanism of fire spread.

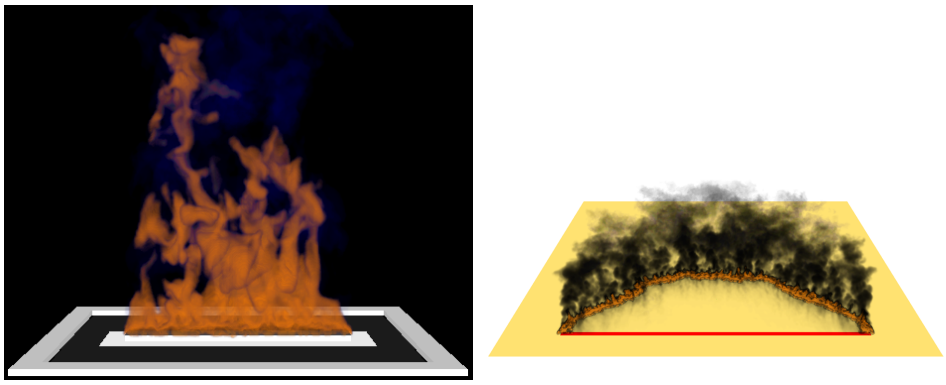


Figure 2.5: (Left) Visualization of soot volume fraction (orange) and CO2 (blue) for the UMD line burner propane flame. (Right) FDS Simulation of Grassland Fires. (Image source: `https://www.nist.gov/programs-projects/advanced-fire-modeling`.)

There is a recognized need for establishing a standard set of validation cases for fire spread over specific materials, such as PMMA (poly methyl-methacrylate), to further improve the tool's prediction capabilities. Additionally, research is ongoing to better predict the radiation heat flux from fires impacting surfaces, which is dependent on various factors like geometry, fuel type, and the radiative properties of the fuel and exhaust products.

Smokeview, which is designed to visualize fire simulations, is also being enhanced with 3D visualization capabilities. This is intended to aid in firefighter training and to provide insights into fire evacuation and compartment fire phenomena. Currently, 3D videos can be generated from a single viewpoint within an FDS simulation.

### 2.5.2 Zibra AI

Zibra AI [AI23] has made significant strides in the field of fire simulation in gaming with the development of Zibra Liquids, a plugin for the Unity engine. This tool harnesses the power of AI-based object approximation to facilitate real-time liquid simulation, including the simulation of fire.

#### The Motivation Behind Zibra AI

The creation of Zibra AI was driven by the need to enhance the realism and interactivity of fire and other liquid simulations in games. Traditional methods of representing fire in games often lack the desired level of realism. Zibra AI sought to

address this gap by leveraging the power of AI and real-time simulation to create a more immersive and dynamic gaming experience.

### The Approach of Zibra AI

Zibra Liquids employs a novel approach to simulate fire. It replaces the traditional representation of fire with a particle system, creating a more realistic depiction of fire. The system also includes the burning of terrain materials, such as grass, further enhancing the realism of the simulation.

Zibra Liquids is equipped with a variety of features to enhance the simulation. It includes three types of colliders: Analytic, Neural, and Skinned Mesh colliders. These colliders allow for more complex interactions with the fire. Analytic colliders are simple shapes like cubes, spheres, capsules, torus, and cylinders. Neural colliders are created from arbitrary static mesh, and Skinned Mesh colliders are created from skinned mesh, implemented as a group of neural colliders, one for each bone.

The system also includes force fields, which can be used to apply force to the liquid in a specific way and in a specific region. As an example, this can be used to create a fountain by putting a force field that pushes liquid up in the body of water, or to create radial gravity.

### The Results

The use of Zibra Liquids has resulted in more realistic and dynamic fire simulations in games. The system also allows for the simulation of various features of fire, enhancing the overall gaming experience. Certain versions of the software support mobile platforms, allow for a particle count limit of 10 million, and include multiple liquid/material types in a single simulation, among other features.

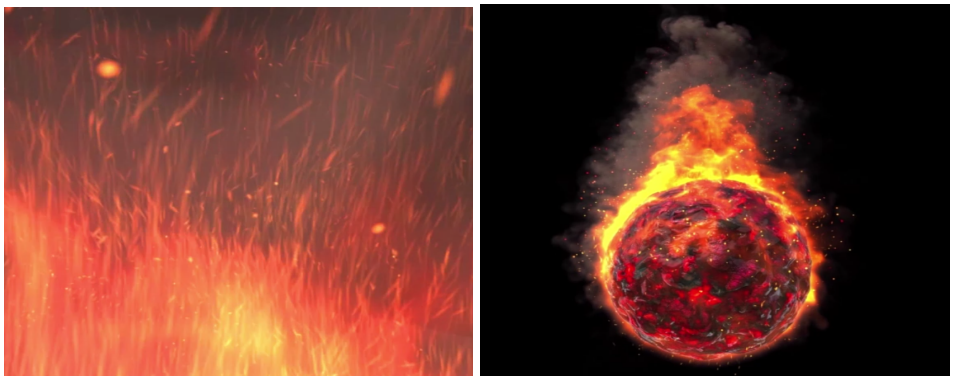


Figure 2.6: Simulation of fire and smoke using Zibra AI.
(Images source:
`https://zibra.ai/zibra-smoke-fire/`)

## 2.6 Conclusions

Fire simulation techniques have undergone substantial evolution, with a myriad of methodologies being devised to accurately depict the intricate nature of fire and flames. These strategies, ranging from particle-based systems to fluid dynamics models, have significantly elevated the quality of fire simulations across a host of applications, such as video games, virtual reality, safety training, and digital film production.

The contribution of game engines in making real-time 3D graphics capabilities widely accessible has been recognized. These engines have served as a platform for the deployment of fire simulation techniques, thereby broadening their reach.

In recent times, the focus has shifted towards enhancing the precision of these simulations. Tools like the Fire Dynamics Simulator (FDS), Consolidated Fire and Smoke Transport (CFAST) zone model, and Smokeview, developed by NIST's Fire Research Division, exemplify this trend. These pieces of software go on to demonstrate that, beyond simulating visually appealing flames for the purpose of immersion and engagement in video games, flame simulators have the potential of providing testing and training environments for fire safety protocols, which can prove to be invaluable in assessing the safety of infrastructure and perhaps even contributing to saving lives.

Despite these advancements, there are still hurdles to overcome in improving the precision of these tools, particularly in estimating heat release rates in large-scale fires. This involves a complicated mix of factors such as local heat feedback, prediction of soot emissions, and understanding the thermal behavior of various materials under fire exposure.

Looking forward, the domain of fire simulation is set for further growth, with ongoing research focused on improving the capabilities of existing tools and devising new techniques. The expansion of FDS's scope to model more complex scenarios, like wildfires at the urban-wildland interface, is a clear example to this trend. As the domain continues to progress, it is expected that fire simulations will become even more accurate and lifelike, thereby expanding their range of applications and potential influence.

# Chapter 3

# Framework and Methodology

## 3.1 Introduction

The simulation of dynamic natural phenomena, such as fire and flames, has been a fascinating and challenging research area in computer graphics and visual effects. Realistic flame simulations have diverse applications in gaming, virtual reality, and film production, enhancing visual fidelity and creating immersive experiences. This chapter focuses on presenting the theoretical foundations and technological implementation of a dynamic flame simulation. It explores the algorithms, architectures, and frameworks used to create a visually appealing and realistic flame effect.

The technological implementation of the dynamic flame simulation relies on a structure that integrates various algorithms and techniques. The Unity game engine, renowned for its real-time rendering capabilities and extensive toolsets, serves as the development platform for this simulation. By utilizing C# scripts and shaders, the behavior and appearance of the flame effect can be controlled and shaped. The simulation architecture is designed for modularity and interaction between different scripts, providing a flexible system.

This chapter further explores the algorithms and techniques employed in the simulation, each playing a vital role in achieving the desired visual effect. The process of generating the hexagonal mesh, which forms the core structure of the flame, is examined. The implementation of the billboard script and the advanced billboard script, responsible for ensuring that the flame always faces the camera in order to enhance visual fidelity, is discussed. Additionally, the utilization of Simplex Noise introduces natural randomness and fluid motion to the flame effect. The algorithms governing flame height control and movement are analyzed, as they determine the growth and behavior of the dynamic flame.

## 3.2 Theoretical Foundations

### 3.2.1 Flame Geometry and Structure

Understanding the geometry and structure of flames is crucial for generating accurate simulations. Flames exhibit distinct shapes and structures, including a core region, outer regions, and other characteristics such as height, width, and curvature.

In order to simulate a realistic flame shape, the model uses two connected modules.

The first one is responsible for drawing a simple hexagon mesh. It does so by first calculating the positions of the six vertices of the hexagon by using the sine and cosine functions to place them around a circle, based on the width and height fields. In order to create the mesh, it then defines four triangles connecting the previously defined vertices. The calculated triangles and vertices are afterwards assigned to the mesh, thus constructing a hexagonal shape.

Figure 3.1: The triangles used to draw the hexagon

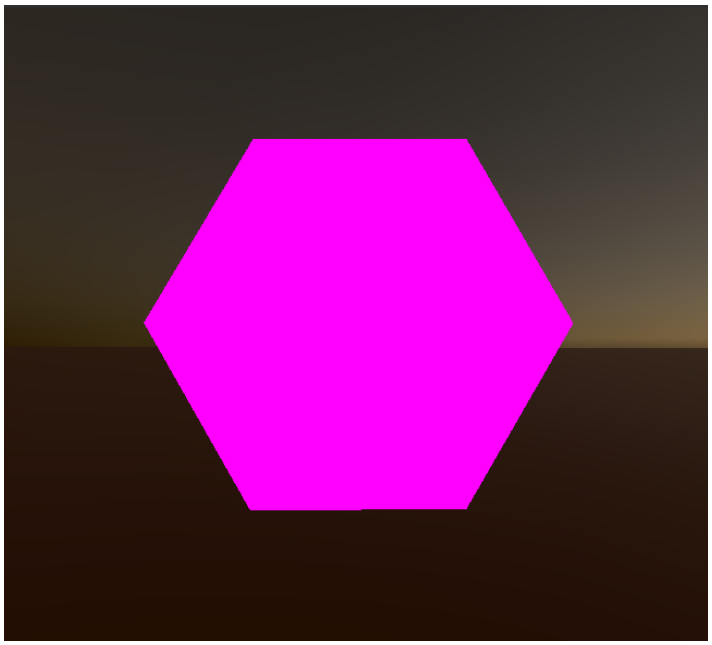
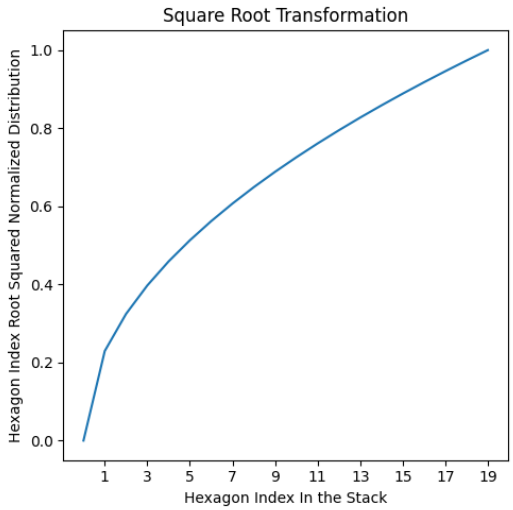
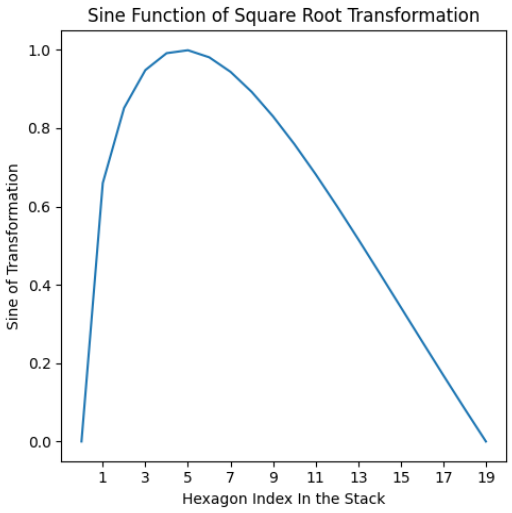


Figure 3.2: Hexagon

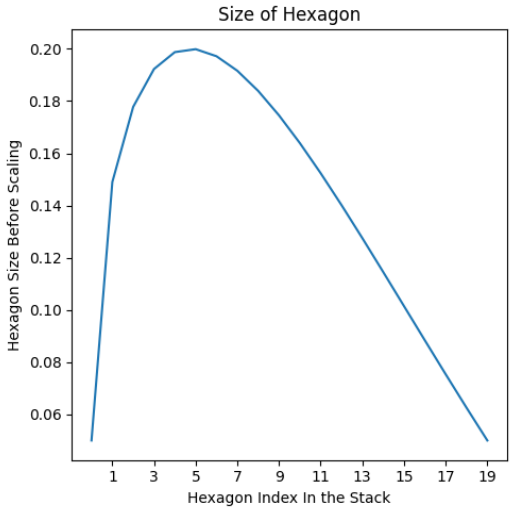
The second module performs the task of creating a vertical column of hexagons to define the shape of the flame. Initially, the script arranges a specific number of hexagons on top of each other, ensuring a constant distance between adjacent hexagons. Subsequently, a normalized distribution is computed based on the hexagon's position in the stack, which can also be interpreted as its height. To bias the distribution towards 1, it undergoes a square root transformation. This biasing is crucial because it allows for the subsequent application of a sine function. The resulting effect is a rapid increase in size, reaching its peak near the middle of the flame, followed by a smooth decrease towards the end. This approach not only facilitates effortless adjustment of the number of hexagons in a flame but also enables the simulation of the flickering effect of a real flame by independently manipulating each hexagon.




(a)




(b)




(c)

Figure 3.3: Plots Representing the Calculation of Hexagon's Sizes

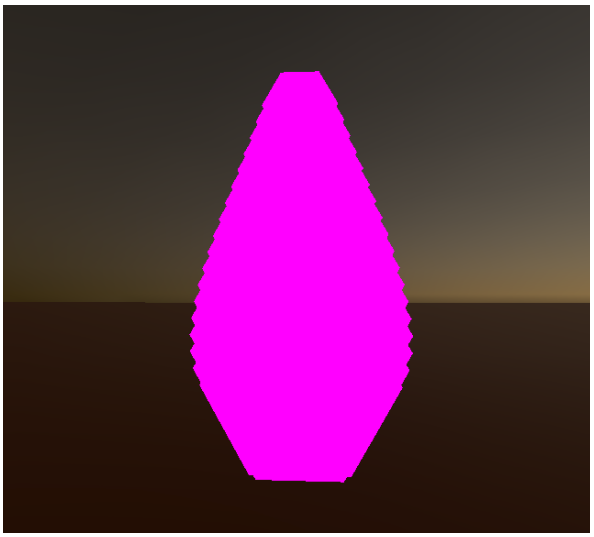


Figure 3.4: Hexagon Stack

### 3.2.2 Flame Shading

In the heart of the flame simulation, there is a core component that dramatically enhances the aesthetic quality and liveliness of the flame: the shader.

The shader utilizes a set of uniforms including flame opacity, noise factor, the vertical position of each of the hexagons comprising the flame, and the flame height. Each of these contribute distinctively to the overall flame behavior, making it possible to achieve a dynamic, visually compelling flame simulation.

#### The Fragment Shader

The fragment shader decides the color and opacity of each pixel of the flame. It calculates the color based on the distance from the center of the texture. By default, the color gradient transitions from orange at the center to red towards the edge of the flame, mimicking the color variations within an actual flame. The shader's uniforms make the color of the flame adjustable, both its inner color, and its outer color. Furthermore, it discards any pixels beyond a specific radius, giving the flame its round, organic shape. It also adjusts the opacity, causing the flame to fade out at its edges, which further enhances the natural appearance of the flame.

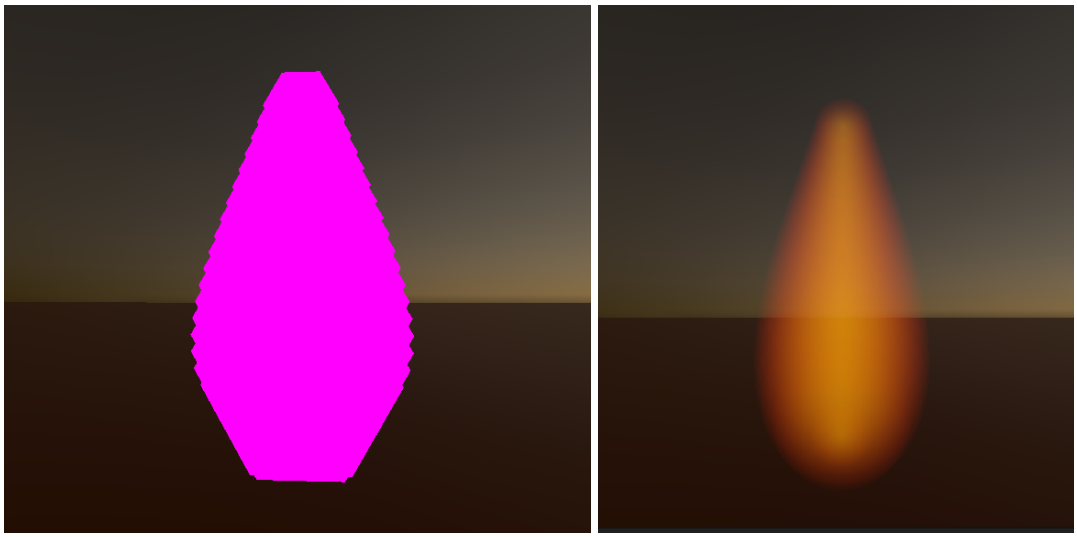


Figure 3.5: The hexagon stack from 3.4, before and after applying the shader.

**The Vertex Shader**

The vertex shader primarily deals with the manipulation of the position of vertices that compose the hexagons comprising the flame, thereby creating natural distortion effect.

It introduces a displacement to each vertex based on a ”noisy” value that is passed in through a uniform, which provides a dynamic, non-uniform offset to the vertex positions. This displacement undergoes further adjustment. It’s dynamically modulated by a smoothstep function, which generates a displacement factor ranging smoothly between 0 and 1 as the vertical position of the vertices increases. This factor ensures the vertex displacement effect gradually intensifies from the base towards the tip of the flame.

The noise comes from a noise-generating algorithm, specifically Simplex Noise, which is employed to regulate the dynamic motion of the stack of hexagons, simulating a flame’s flicker. The simplex noise generator assigns each hexagon a noise value that varies over time. These noise values, once multiplied with a random value assigned per flame, influence the flickering characteristics of each hexagon in the stack, simulating the unpredictable and natural motion observed in real-world flames. The flame flicker effect begins from the central hexagon and is then propagated to the others. A history of noise values for the central hexagon is maintained, and these values are used to set the noise for other hexagons based on their distance from the center. This results in a fluid and seamless movement of the hexagons, each one moving in coordination with the ones adjacent to it, thus creating a realistic flame-like undulation across the hexagonal stack.

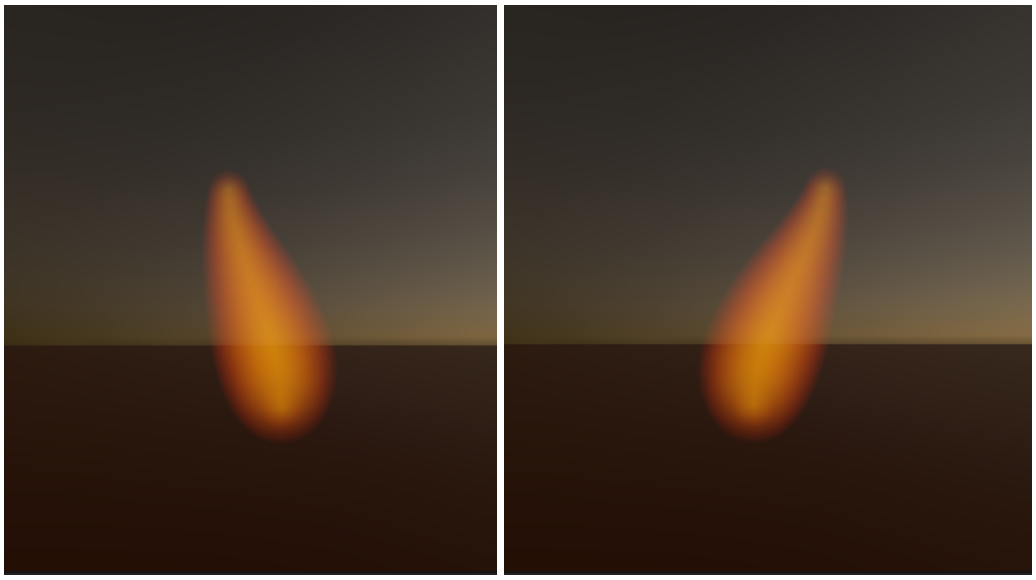


Figure 3.6: Flame Waving Movement.

### 3.2.3 Flame Movement

Realistic flame movement is essential for creating dynamic and visually captivating simulations. An entire module is dedicated solely to the flame's placement and how it moves.

#### Flame Placement

For starters, the flame is placed on the desired surface. This is accomplished by shooting a ray straight down from the position of the flame, leveraging the Physics.Raycast() method of the Unity game engine. The purpose of this is to then detect the collision of the ray with the surface that lies underneath the flame. If the ray hits the desired surface, the position of the flame is shifted accordingly.

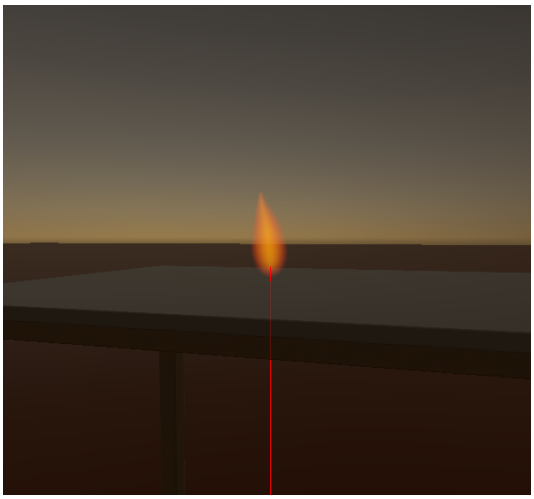


Figure 3.7: Placing the flame by using ray collision detection.

#### Flame Movement Direction

Furthermore, the movement direction of the flame is determined. The first step of this process lies in determining the direction from the flame to the center of the surface that it is burning on. Finally, a random offset, between -45 and 45 degrees, is applied to it, and the resulting direction is assigned to the flame. The process is then repeated every time the flame is about to leave the surface. Detecting whether the flame is about to be out of bounds is done in a similar fashion to placing the flame on the surface. A ray is shot straight down, only this time the movement direction and speed of the flame is taken into account before shooting it. Upon not detecting the correct surface, the flame changes direction. This technique not only gives the flame the realistic effect of random movement, it also ensures that the flame doesn't move away from the surface that it is burning on.

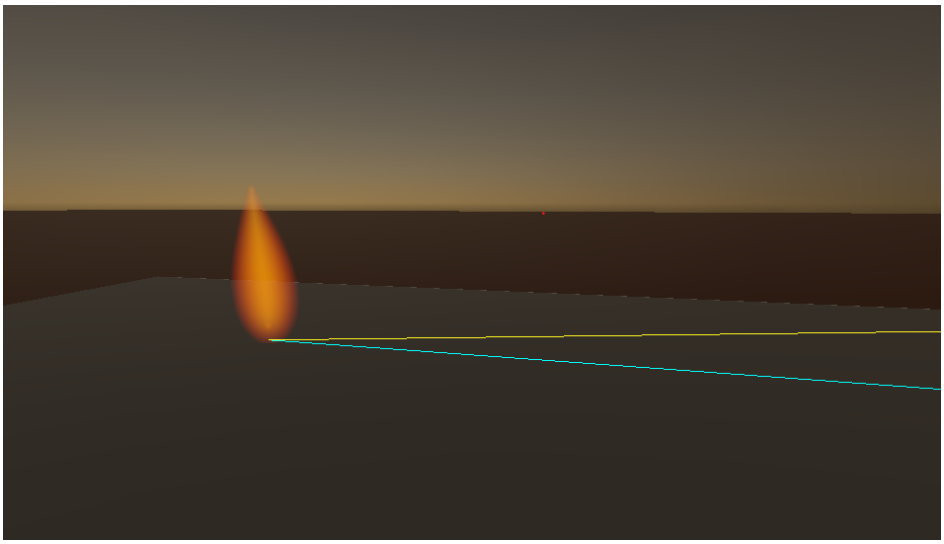


Figure 3.8: Determining a random direction of movement. (Cyan) The calculated direction from the flame to the center of the surface. (Yellow) The final direction, obtained by applying a random offset.

### 3.2.4 Dynamics of Flame Propagation and Evolution

The dynamics of flame propagation and evolution are primarily controlled by two distinct functional modules, which implement variations in flame height and movement.

#### Flame Propagation

In the first module, the mechanism of flame spread across a surface is described. The very first flame has a movement direction and speed, which determine how

fast and in which direction it moves. The flame leaves a trail of similar flames, with a pre-determined and customizable chance for these to grow larger. If they end up being growing flames, they begin expanding continuously to a predetermined size, and start fluctuating in scale afterwards.

#### Flame Evolution

In the second module, the flame growth mechanism is defined. Each flame has a base growth speed and a random growth factor. The base growth speed determines the constant speed at which the flame will grow or shrink, whereas the random growth factor introduces variability in the flame's progression. This growth factor periodically transitions to a constrained random value over a defined duration, creating a dynamic and visually engaging growth pattern. The flame alternates between growing and shrinking, giving the appearance of real flame consuming its fuel.

Together, these two scripts model the complex behavior of flame propagation and growth. The first script controls the flame's spread across the tabletop, producing a dynamic and visually engaging scene, while the second one modulates the growth and shrinking of each flame, showcasing a realistic burning effect.

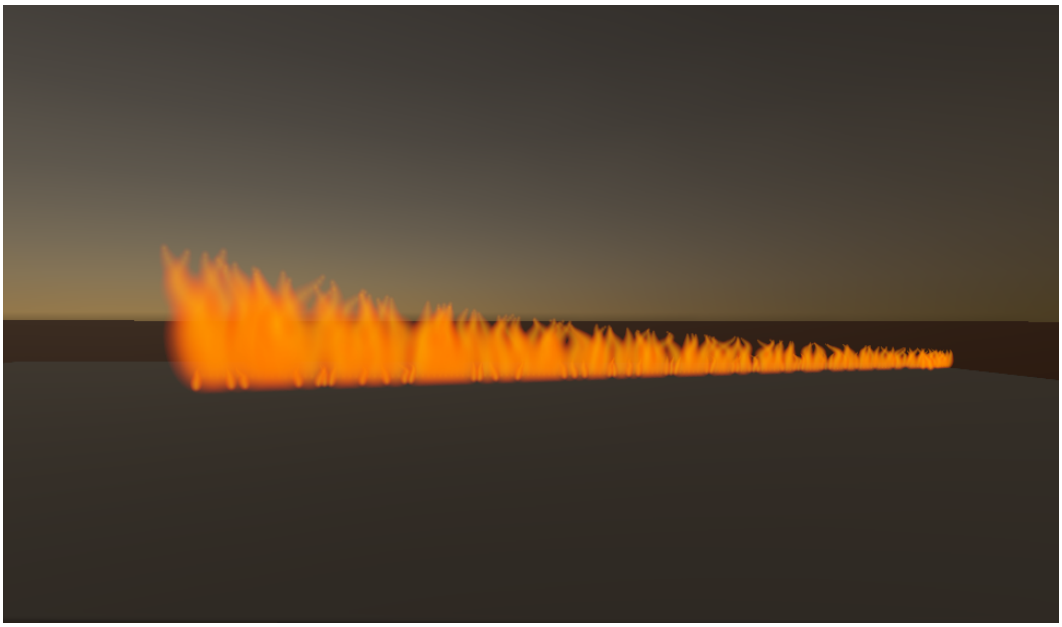


Figure 3.9: The initial flame leading the propagation, while the trailing flames begin to grow.

### 3.2.5 Camera-facing Flames through Billboarding

For optimal visibility and immersion, flames need to constantly face the camera. This is achieved by means of a technique called "billboarding". Billboarding is often

used in computer graphics simulations with the purpose of making a 2D object, in this case, the flames, look like a 3D one. In order to implement this, the direction vector that is pointing from the object to the camera is obtained by subtracting the position vector of the object from the position vector of the camera. In the scope of the proposed model, the y component of the obtained vector is then set to 0, so as to always keep the flames perpendicular to the surface. The resulting vector is then used to adjust the flame's rotation accordingly.

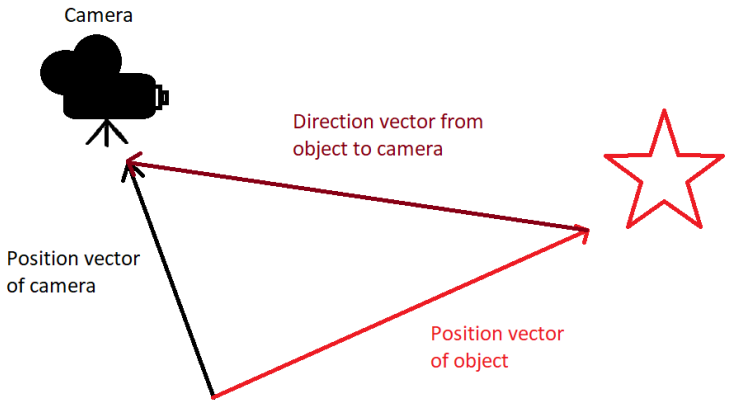



Figure 3.10: The Billboarding Technique

### 3.2.6 Advanced Billboarding

Despite the classic billboarding effect being enough for a variety of situations such as torch or candle flames, applying 2D flames to 3D objects does not work. Placing the flame in the center of the object will cause the flame to cut through it, dispelling the effect of the object burning. This is where the advanced billboarding comes into play.

Similar to the billboard module, this algorithm ensures that the flame always faces the camera. Moreover, it displaces the flame by a specified offset, such that the flame is always between the camera and the burning object, "orbiting" it. This method ensures that the flame always covers the object, and by virtue of the flame being transparent, the object appears to be burning.

Figures 4.3 and 4.4 are examples of this algorithm in action.

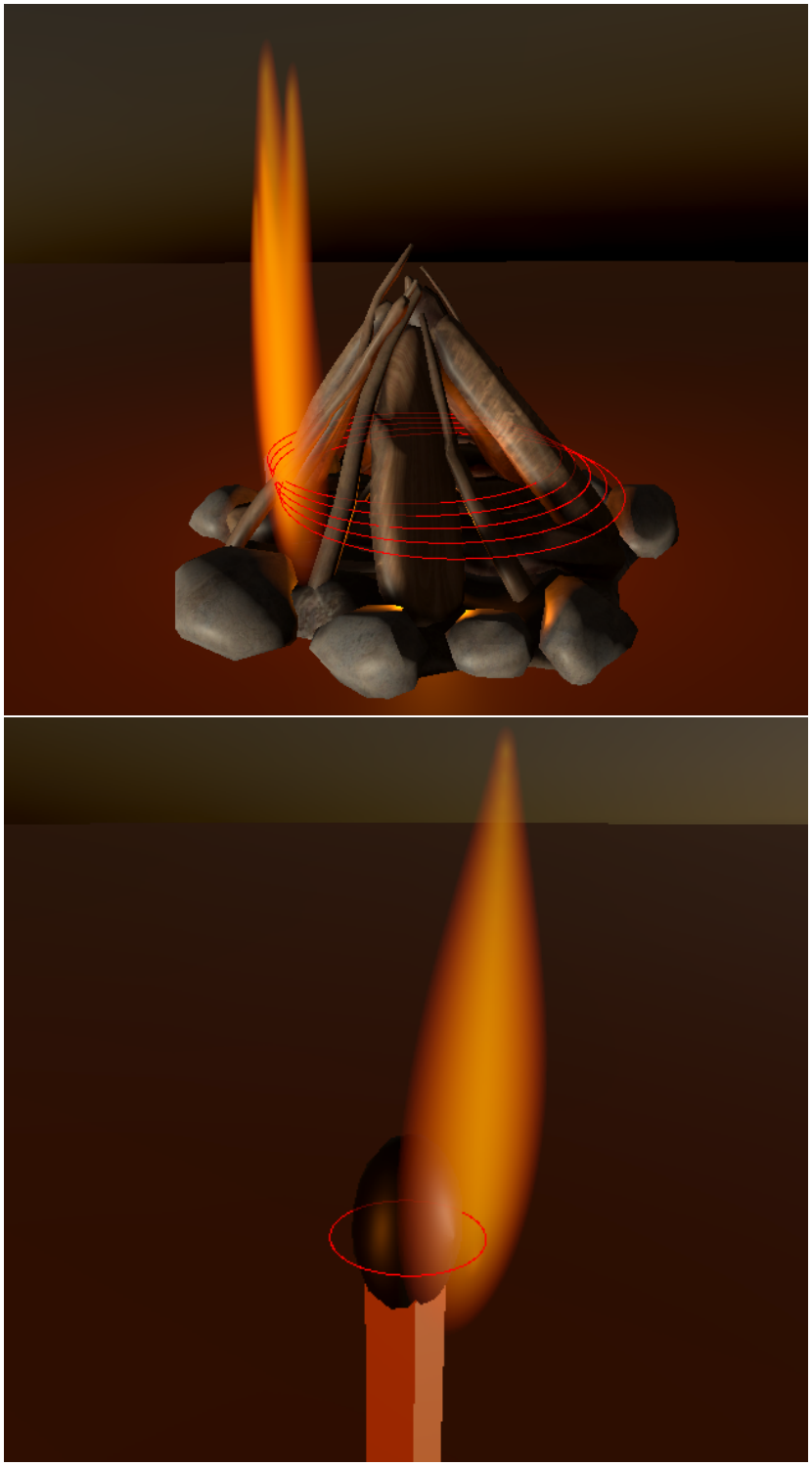


Figure 3.11: Advanced Billboarding for Campfire and Match.

# Chapter 4

# Interactive User Interface and Customizability: Tailoring Flames for Game Development

## 4.1 The Interactive Panel

The interactive part of the flame simulator is a crucial aspect that enhances user engagement and interactivity. Designed with a simple and intuitive layout, the UI component allows users to have a hands-on experience in real-time simulation manipulation, fostering an immersive environment that encourages exploration and creativity.

The sliders facilitate control over the flame's color and behavior. Three color sliders allow users to modify the Red, Green, and Blue (RGB) channels of the flame, offering endless color combinations for a customizable and dynamic flame coloration. This ensures a realistic and stylized visual representation based on the user's choice, enhancing the versatility of the flame simulator for different usage scenarios.

Finally, three more sliders control the propagation speed of the flame, the growth chance of the trailing flames and how large these trailing flames will grow before commencing the height variation protocol. These settings allow users to influence the flame's speed and the likelihood of trailing flames growing, adding a level of interactivity and control over the simulation.


Figure 4.1: The Interactive Panel

## 4.2 Elemental Simulation

One of the simulator's most engaging features is the Elemental Simulation, offering users the opportunity to interact with the simulation in a fun and educational way. This capability elevates the simulator beyond just a visual tool, introducing a layer of real-world scientific context that enhances user engagement and deepens the understanding of flame behaviors.

The Elemental Simulation is controlled through a dropdown menu populated with various combustible elements: Default Fire, Hydrogen, Lithium, Potassium, and Copper. Each option within this menu corresponds to a different burning element, which, when selected, influences the color combination of the flame within the simulation. This mirrors the specific colors these elements emit when burnt in reality, an effect known as flame test in chemistry.

For example, choosing Hydrogen will result in a light blue flame, emulating the real-world appearance of a hydrogen flame. Similarly, selecting Copper produces a green flame, Lithium a crimson red, and Potassium a purple hue, each accurately reflecting their real-world counterparts. The Default Fire option offers a traditional orange flame, providing a familiar and baseline reference for comparison.

By providing the extra layer of simulating different elements burning, the simulator bridges the gap between the artistic freedom of game development and scientific accuracy. With this, the simulator becomes more than just a visual tool, gaining the potential of transitioning towards more educational purposes in the area of chemistry.

The interactive panel as a whole provides developers thinking about adopting this model with the right tools needed in order to test the capabilities of the simulation and reach a conclusion on whether or not they will ultimately use it.

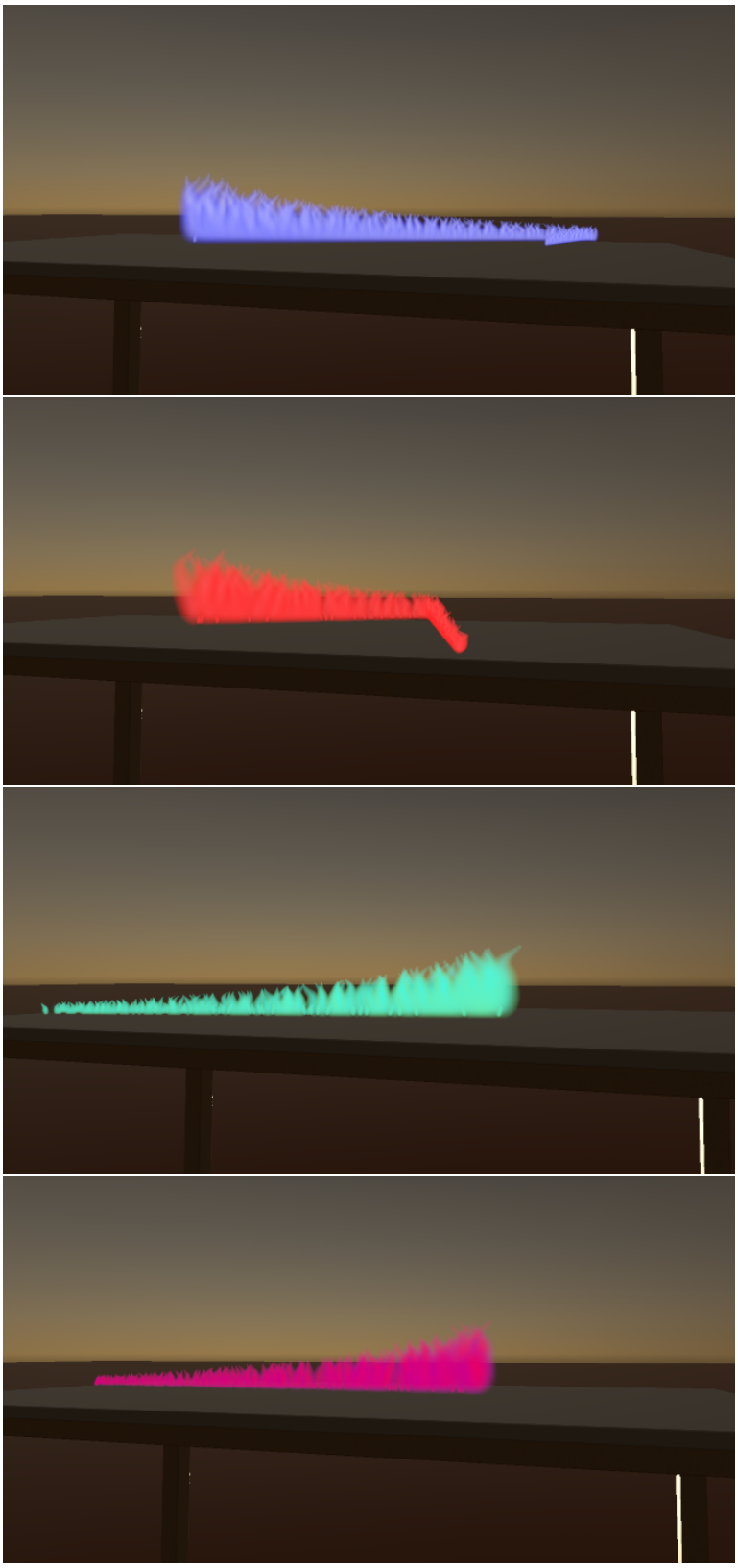


Figure 4.2: Different Elements Burning

## 4.3 Customizabilty of Flame Dynamics for Game Development

Beyond being a visually appealing simulation, another important goal of this flame simulator is to offer a versatile, adaptable tool for game developers. By incorporating extensive customizability features, both through the User Interface, and within the underlying C# scripts that model the behaviour of the flame, the simulator empowers developers to adjust the flame characteristics to suit the specific needs of their games, making it a potent asset for game development projects.

### 4.3.1 Shape

The Unity C# script that handles the drawing of the hexagon stack comprising the flame offers significant potential for developers to create and manipulate the structure.

First, it enables developers to modify the scale of the hexagons as well as the number of hexagons present in the stack. These parameters are freely adjustable, providing control over the physical size of the overall stack size.

Besides having the possibility of modulating the general stack scale, the sizes of the biggest and smallest hexagons in the stack are also adjustable. Every other hexagon's scale will then be made to fit these restraints.

Furthermore, by modifying the computations involved in generating the flame's shape, one can influence the vertical position of the point of maximum width within the stack.

A parameter within the script governs the distance between the centers of each hexagon. Modifying this distance allows for controlling the appearance and compactness of the stack. One thing to note is that providing too great of a distance between the hexagons can lead to visible separation between them, especially at the tip of the flame.

While these aspects are entirely flexible in the code, they can also be found in the Unity Inspector Window.

### 4.3.2 Color

Beyond controlling the inner color of the flame directly through the UI, the code allows potential developers wishing to apply a deeper layer of customization to modulate the outer color. The shader will then apply a smooth fade between the two colors.

### 4.3.3 Movement

The modules dictating the flame's movement and placement are also highly customizable.

As such, parameters like how high above the desired surface the flame is burning, how quickly it changes direction when nearing the edge of the surface or how wide the angle is when picking a new random direction for the flame to move towards, are entirely configurable.

Moreover, this script unlocks the possibility of experimenting with the distance between separate flames, or, to be more precise, how far the leading flame has to travel before spawning a trailing flame.

Finally, the chance that trailing flames will grow further or keep the scale of the leading flame is also entirely up for adjustment.

### 4.3.4 Height

The algorithm that manages the evolution of the flame in terms of height fits the pattern, showcasing flexibility. An important aspect regarding this algorithm is that it only comes into play after the trailing flames reach a certain specified scale.

On the first hand, it provides the possibility of altering the base rate at which trailing flames grow or shrink. Furthermore, a randomness factor is incorporated to introduce variability in the height's evolution speed, enhancing the realistic appearance of the flame. This factor can be tweaked to control the extent of randomness applied to the flame's growth speed.

The height of the flame can be adjusted to oscillate between two limits. These limits can be set to ensure the flame height remains within a desired range.

The script includes an option for the time over which randomness in the flame's growth speed changes. This time can be adjusted to change the rate at which this parameter transitions to its new random value, allowing for smoother or more abrupt changes depending on the desired visual effect.

Lastly, it automatically shifts the direction of the flame's growth (upwards or downwards) based on the current height and the defined limits. This creates an oscillation effect, where the flame grows and shrinks over time.

## 4.4 Potential Applications

This section presents an exploration of how the flames can be integrated across diverse contexts and scenarios, underscoring the adaptability inherent in the proposed algorithm. The aim is to demonstrate the utility that the proposed model provides

for game developers.

The focus lies on placing the flames in distinct environments. Each context provides a demonstration of the flexibility of the algorithm, illustrating its capacity to enhance a broad range of gaming or simulation circumstances.

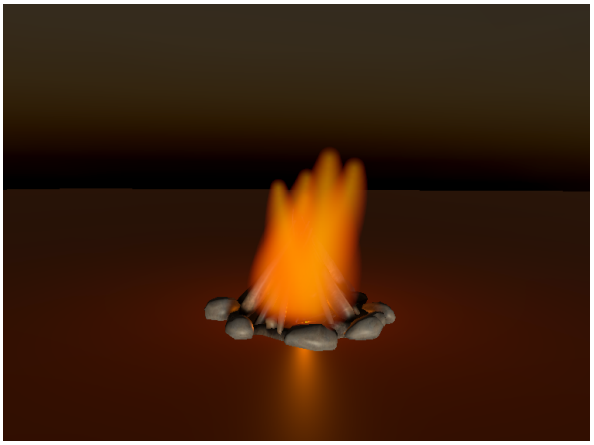


Figure 4.3: Campfire. Model taken from turbosquid.

Figure 4.4: Match. Model taken from turbosquid.

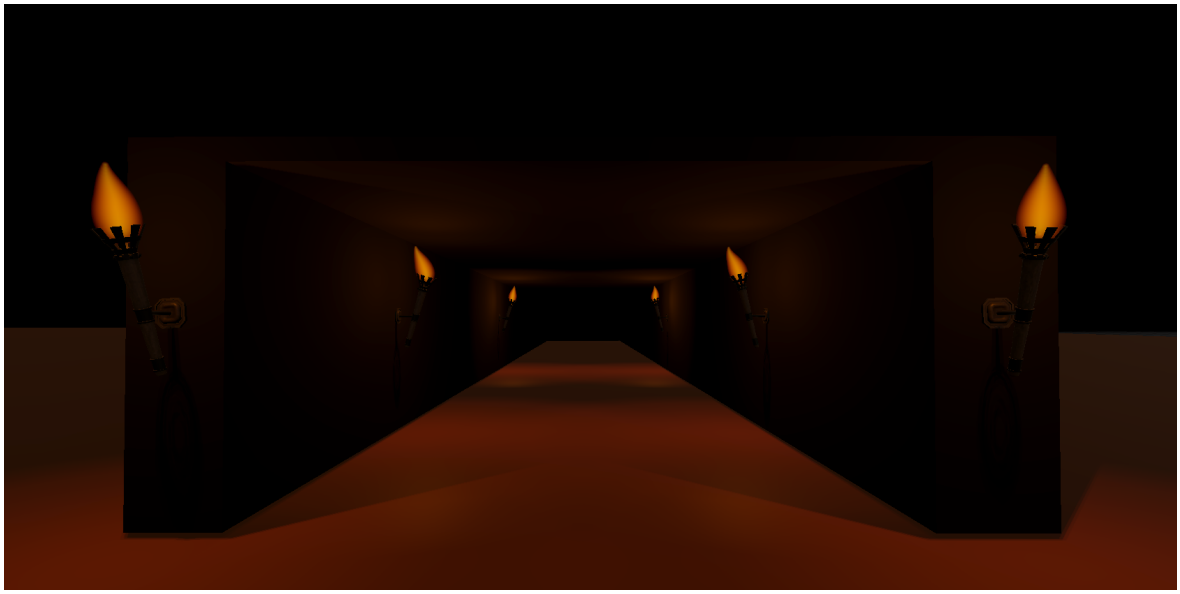


Figure 4.5: Dungeon. Torch model taken from sketchfab.

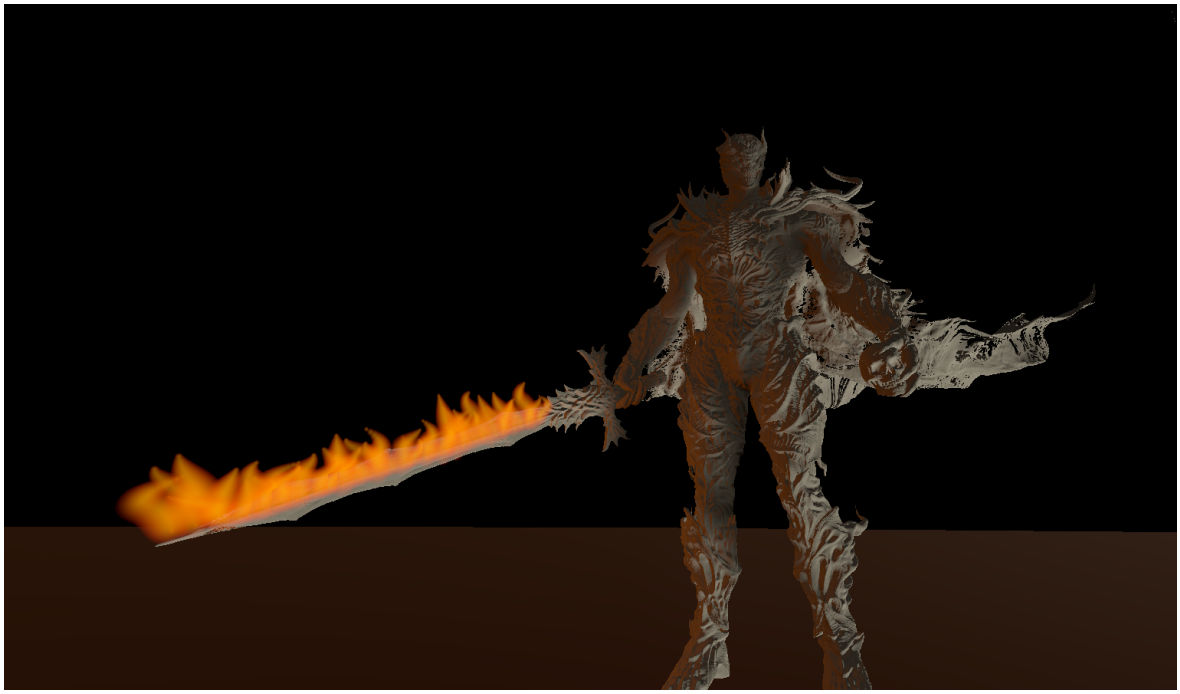


Figure 4.6: Monster with Burning Sword. Model taken from cgtrader.

# Chapter 5

# Conclusions

## 5.1 Summary of Findings

The paper encompasses a detailed investigation into the intricate process of flame simulation. It starts with a thorough review of the existing simulation techniques and how game engines are leveraged, setting the theme upon which the project rests. The deployment of an advanced billboard system, in conjunction with the principles of flame geometry, shading, movement, and propagation dynamics, results in a realistic and flexible flame simulation system. Furthermore, the addition of an interactive user interface and extensive customizability options, particularly designed for game development, amplify the system's adaptability and user-friendliness.

## 5.2 Contributions

The project's contributions are significant.

First, the development of the advanced billboard system enables a more dynamic and realistic flame representation.

Second, the customizable flame dynamics offered by this system present a versatile tool for game developers, facilitating adjustments according to specific requirements.

Last but not least, the manner in which the simplex noise generator applies noise to the hexagons is yet another contribution worth mentioning. A notable improvement is the introduction of a historical record for noise values, enhancing flame transitions' realism. Furthermore, the procedural approach provides variability and randomness in flame behaviour, enhancing the natural appearance.

## 5.3 Implications and Applications

The implications of the flame simulation system for game development are substantial. Its adaptability in terms of flame dynamics manipulation can enhance the realism in games, thereby augmenting the immersive experience for players. Beyond gaming, the simulation system could serve in virtual reality applications for training scenarios such as firefighting, or in digital film and animation. Additionally, the possibility of visualizing the burning of different elements further diversifies the potential use cases of the simulation.

## 5.4 Future Work

While considerable progress has been made, there are potential enhancements for the flame simulation project. Subsequent work could aim to optimize system performance or expand the range of customizable flame attributes. Adding smoke to the simulation can prove to be a significant improvement that would make it a much more valuable prospect in the field of safety planning. Moreover, the system could be modified to simulate other elemental phenomena, thus broadening the spectrum of potential applications.

## 5.5 Final Remarks

The journey through this project represents a blend of theoretical research and practical implementation. Despite encountered challenges, the success of the flame simulation project demonstrates the potential of integrating simulation techniques into game engines. This paves the way towards more realistic and engaging virtual environments.

# Bibliography


[AI23] Zibra AI. *ZIBRA LIQUIDS User Guide 1.5.3*, 2023.

[CD05] Robert L Cook and Tony DeRose. Wavelet noise. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 803–811. ACM, 2005.

[GZD08] Alexander Goldberg, Matthias Zwicker, and Frédo Durand. Anisotropic noise. *ACM Trans. Graph.*, 27(3):1–8, aug 2008.

[KT22] Iris Kotsinas and Viktor Tholén. Dynamic real-time fire propagation for networked multiplayer games. Technical report, Department of Science and Technology, Linköping University, 2022.

[LFV20] D. Liu, M. Frasheri, and V. Vyatkin. A game-based learning framework for controlling cyber-physical systems. In *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*, pages 37–42. IEEE, 2020.

[LLC+10] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and K. Zimmerman. A survey of procedural noise functions. *Computer Graphics Forum*, 29(8):2579–2600, 2010.

[Per02] Ken Perlin. Improving noise. *ACM Trans. Graph.*, 21(3):681–682, jul 2002.

[TZ23] Rachel Thomas and Wenshu Zhang. Real-time fracturing in video games. *Multimedia Tools and Applications*, 2023.

[WH11] Wei Wei and Yanqiong Huang. Real-time flame rendering with gpu and cuda. *I.J. Information Technology and Computer Science*, 1:40–46, 2011.