Using Particle Systems to Simulate Real-Time Fire

Sanandanan Somasekaran

This report is submitted as partial fulfilment
of the requirements for the Honours Programme of the
School of Computer Science and Software Engineering,
The University of Western Australia,
2005

Abstract

In computer graphics, particle systems are a collection of graphical objects or particles, often referred to as "fuzzy objects", that move around within a specified boundary based on a set of predefined rules governing the dynamics. Associated with each particle are attributes that affect its creation, dynamical behaviour, and expiration. Particle systems are the foremost technique used in modelling real-time fire in computer graphics. This work investigates the application of particle systems to the generation of fire.

This work presents a detailed review of particle systems and the how such systems can be used to simulate fire. This will include a discussion on the sensitivities of attributes to type of effect generated for fire. This work will demonstrate that simpler computer models can be used to generate realistic visual effects. Such methods are of interest in the gaming community and other real-time applications.

Keywords: graphics, particle system, physically based, fire modelling
CR Categories: I.3.5, I.3.6, I.3.7

Acknowledgements

First and foremost, I would like to thank God for giving me the health, strength, motivation, and courage to take up this project and complete it. I would like to thank my supervisor, Dr. Karen G. Haines for taking deep interest in my work, guiding me throughout this project, and providing me with a good working environment.

I would like to thank Angeline Loh (Ange) and Anthony Prior (Tone) for their encouragement and help. Without them, the IVEC office environment would not have been a fun and lively environment to work in. Ange, thanks for helping me out with the new algorithm to model the flames more realistically and for those delicious pears and apples and not to forget the sticky date pudding. Thank you Tone for those codes to calculate the frame rate of my fire animation. Thanks for the numerous help you have rendered.

Many thanks to Charles Stan-Bishop and Mark Moss for helping me with the mathematics and physics which got me bogged down in the early stages of development of this project. Thanks guys.

A big thanks to Nick Wood, the Demonstrations Supervisor for SciTech Discovery Centre for demonstrating real-life explosions and fire.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Computational modelling of natural phenomena such as fire, water, clouds, smoke and rain has long been a challenge in computer graphics due to their complex behaviour. While various techniques have emerged in the last three decades to realistically model these phenomena, only few are in use today. This is due to either hardware limitations or complexities involved in calculating the physics engine behind these models which may be computationally intensive or hard to accurately implement. One such technique for modelling these phenomena is particle systems.

Particle systems modelling is a popular physically-based technique used for modelling fuzzy objects such as smoke [21] and explosions [2]. A particle system is comprised of one or more particles, entities or objects that move around within a specified boundary based on dynamics. Graphically particles can be as simple as two-dimensional points to complex models of real-life objects of varying sizes and shapes in three-dimensional space. The behaviour of the particles is spatio-temporal as their attributes change over space and time.

As described by Yang [21], a particle goes through three phases during its lifespan: generation, dynamics and death. The generation phase involves the creation of a particle each iteration, initialising its attributes and generating it randomly within a predetermined boundary. The attributes of the particles are either initialised with fixed or random values. The level of randomness in each attribute is determined by a stochastic process. The key aspect of a particle system is its dynamics. That is, how the particles within the system move and interact with each other and how their attributes change dynamically each iteration. The last phase, death or expiration of a particle, results in the removal of a particle from the system. This occurs when the lifespan of a particle drops to a minimum value that indicates the expiry of the particle.

A typical structure of a particle in a particle system contains the following common attributes described in Table 1.1. In general, the attributes of a particle can be either static or dynamic. Dynamic attributes change in time, while static

No.	Attribute	Description
1	Position	The location of a particle within the three-dimensional boundary.
2	Velocity	Directional vector to decide how fast a particle travels in a given direction.
3	Colour	Sets the red, green, and blue colour values of a particle.
4	Lifespan	How long a particle will exist within the particle system before being removed.
5	Size	Default size of the particle.
6	Opacity	Transpareny of particle in the system.

Table 1.1: This table provides a summary of all the common attributes of a particle seen in a general particle systems model.

attributes remain constant in each iteration. The position of a particle is its current location in three-dimensional space and is constantly displaced in each iteration. The displacement of a particle is determined by its velocity, which includes its speed and direction. In each iteration, a particle's velocity is added to its current position to determine its next position in the system. A particle continues to exist within the system until its energy expires. Thus energy represents a particle's lifespan and decreases in time.

The lifespan of a particle may affect its colour, opacity, and size. A particle's colour can change constantly with each iteration to create an array of effects. A particle's colour can also be combined with its opacity to determine its intensity and transparency. The higher the opacity, the brighter the particle's colour will appear while a lower opacity would result in the particle appearing more transparent. The lifespan of a particle coupled to its opacity causes the particle to appear increasingly transparent until it fades away. The particle's size coupled to its lifespan can be used to represent life. For example, particles with full life are represented as large-sized points or objects but as the particle ages, they shrink in size until they die away. All this just touches on the powerful capability of such systems.

As shown in Figure 1.1, a particle goes through a cycle of changes until it expires. First, a particle is created and all its attributes are initialised. This is plotted on screen based on its given position. The particle's velocity is then modifed by adding acceleration. Next, the updated velocity is added to the particle's current position to derive its new position.

The colour of the particle is also updated either by changing to another colour or combining it with its opacity to derive its new intensity and transparency. Because the particle's opacity is based on its lifespan, and as the lifespan decreases, the value of the opacity is decreased and updated accordingly. Finally, the system checks if a particle has expired. If so, it is removed from the system. Otherwise, it is replotted at the new location.

As explained by Reeves [14], the advantages of particle systems that makes them an ideal technique for modelling natural phenomena are three-fold. Firstly, particles are simple primitives and use the simplest of surface representations. As a result, it takes the same amount of computational time to render many primitives to create complex animations. Secondly, a particle system model is defined procedurally. Thus, the behaviour of particles can be controlled using various functions and parameters. This reduces the amount of time needed by the animator to create a highly detailed model. Finally, unlike surface-based modelling techniques, modelling the dynamics of particles is easy and thus well suited for complex animations.

A major drawback of particle systems is the large amounts of particles are required to animate a scene. Each particle requires a space in the computer's memory. In a typical scene that requires thousands of particles, such as water, a large amount of memory space has to be allocated for the entire scene. Each time a particle is created, it is allocated a space in memory. When the particle dies and as a result, removed from the system, the allocated memory space must be freed. If the allocation and deallocation of memory are not properly managed, the rendering performance is poor.

This project provides a study of particle systems and how these systems can be used to model fire. To better understand particle systems, investigations examined how system parameters can be varied to simulate different types of flames. This involved understanding of how fire behaves. This understanding was applied to create a computer program to simulate resulting effects. Background information on particle systems has been presented in this chapter. This work continues with a review of current techniques used in the area of real-time fire modelling in the next chapter, particularly, particle systems. Chapter 3 next describes particle systems in detail and the core design of this implementation of particle systems to model flames. Then, Chapter 4 details a series of experiements conducted to study how particle systems can be used to aesthetically model fire. Finally, Chapter 5 and 6 discusses the results of the experiments, drawing conclusions, and pointing possible future research.

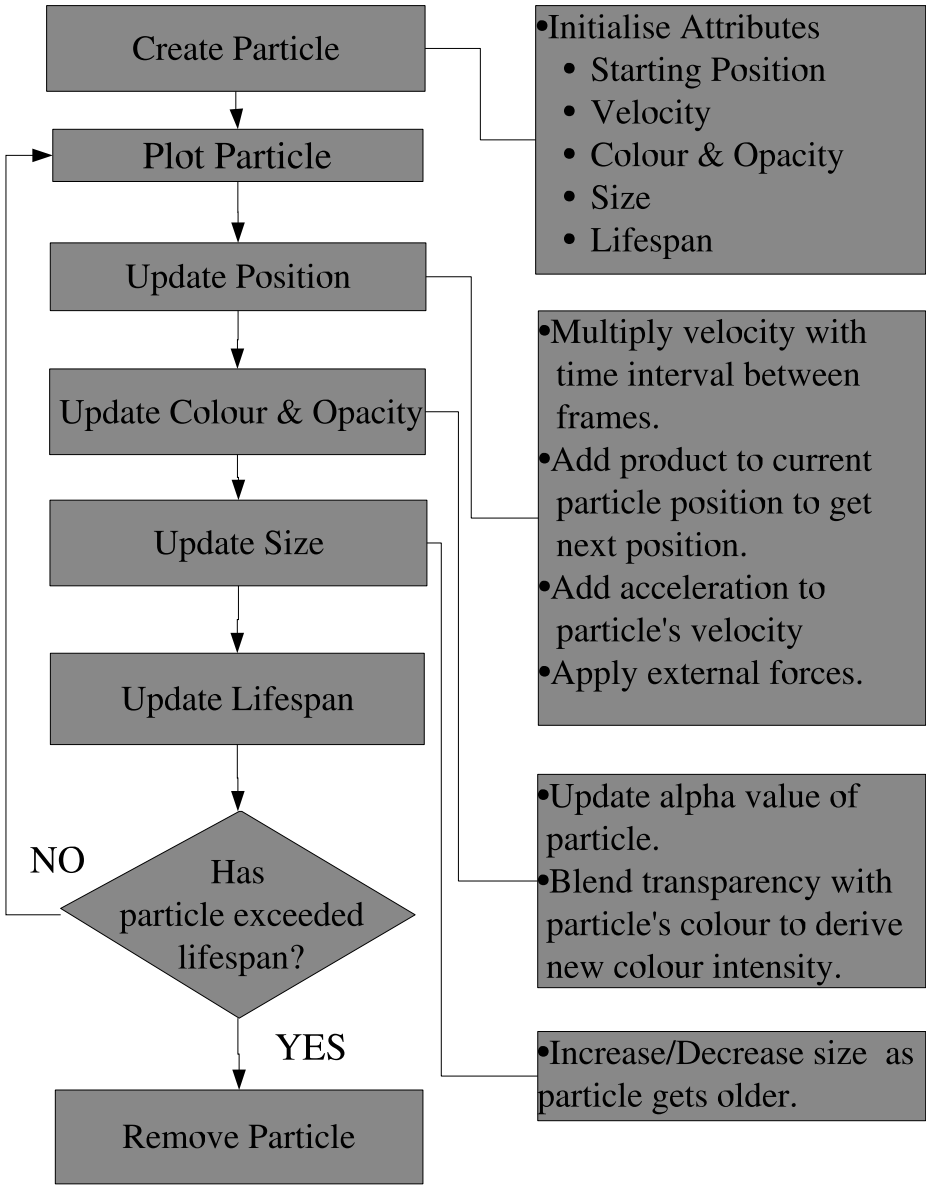

Figure 1.1: This flowchart, adopted from Woodhouse's [20] standard per-frame pipeline, summarises the behaviour of particles in a particle system each iteration. It illustrates a typical life-cycle of a particle in a particle system. Once a particle is created and assigned various attributes, it is plotted on the screen. Subsequently, all its attributes are modified and updated step-by-step and replotted on screen. If the particle exceeds its lifespan, it is removed from the system.

CHAPTER 2

Previous and related work

Many techniques have been researched and implemented "with increasingly successful results" [6] to deliver high levels of realistic visual effects for today's demanding gamers. Eyman [6] subdivides these approaches as attempting to model fire as a texture and as a physical model. The following discussion on the literature review will focus on physically-based modelling methods as the scope of this project is on implementing a physical model of fire.

Physically-based fire modelling techniques, which used mathematical techniques to model the physical characteristics of real fire, have in the last two decades become increasingly important due to its complex and different approaches to animating special effects. Stam and Fiume [17] presented a simple model for the flame of a fire and its spread using diffusion processes. Using incompressible Navier-Stokes equations, Nguyen et al. [12] modelled the expansion that takes place when a vaporized fuel reacts to form hot gaseous products. Lamorlette and Foster [9] used stochastic models of flickering and buoyant diffusion to provide realistic local appearance while applying physics-based wind fields and Kolmogorov noise to add controllable motion and scale. Procedural mechanisms were also developed for animating all aspects of flame behavior including moving sources, combustion spread, flickering, seperation and merging, and interaction with stationary objects.

Presented first in 1983 by William T. Reeves [14], Particle Systems became a very popular physically based modelling technique for fire. Reeves was attempting to find a model that can be used over the conventional methods used in creating special effects for the Genesis Demo sequence from the movie, Star Trek II: The Wrath of Khan [14, 16]. Reeves realized conventional modeling for creating realistic fire was more suited for creating objects with "smooth, well-defined surfaces" and not for fire, which is made of undefinable surfaces [16]. A particle system can be basically described as a collection of tiny objects or points on screen with no definitive shape, structure, size or predictable motion. Reeves called these "fuzzy objects" [14]. Engell-Nielsen and Madsen [5] explains the

concept behind particle systems as "All real-life objects are made up of particles and atoms. Thus the model used to simulate these objects must also contain these elements, namely the particles". These particles have attributes that determine their characteristics within a system from the time they are introduced into a system where they move and change around througout thier lifespan before dying out. Described as "the earliest computer graphics fire model" by Lee et al. [11], particle systems laid a foundation for animating realistic fire effects.

Recently, Lee et al. [11] analysed Reeves's system on difficulties in defining boundaries and the large amount of particles required to animate a fire. Building upon Reeves's work, Lee et al. presented a new method for animating fire over polyhedral surfaces. Fire was represented as an evolving front and flames as particle based in the system. Lee et al.'s work was unique in the sense, they focused on fire propagation techniques. To create rich, complex dynamics for the flame motion, Lee et al. described uses of dynamic wind fields to simulate the movement. The moving flames drop particle-based flames which leave behind charred surfaces.

While Reeves's system presented a simple model for creating special effects sequences, meeting the demands of an highly interactive video game where multiple special effects are being rendered real-time posed a challenge. John van der Burg extended Reeves's work by creating an advanced particle system [18] which addressed this issue. The advanced particle system was not very much different to Reeves's wherein the former applied an hierarchy based particle system discussed by the latter in [14]. It is described as "a collection of two or more single particle systems" in [16]. The underlying concept was particles themselves were particle systems. Burg represented this hierarchy as a class based system. The heart of this system is the particle system class that manipulates the attributes of a particle, which by itself is a class. The former is in turn controlled by a particle manager class that handles the creating, releasing, updating and rendering of all the particle systems [18]. The advantage of advanced particle system is that each particle system inside behaves uniquely. So in the case of modelling fire, one particle system can be used for modelling the flames and another for modelling the smoke or and another for wind.

Burg's advanced particle system was furthered by Miroslav Sabo [16]. Sabo introduced the concept of adding property milestones to a particle's life cycle while keeping the hierarchy based system intact. A property milestone is represented as a value of a particle's attribute at a certain point in time during its lifespan. By adding more control over a particle's property with respect to its age, property milestones allow the values of the parameters of a particle to be adjusted during its lifecycle. Sabo felt in order to achieve realistic visual effects

using particle systems, it was necessary to have very good control over particle properties without compromising the stochastic nature of the particles.

Another issue with Reeves's work was with the particle dynamics. The movement of particles in Reeves's particle system were based on simple equations determining the rate at which particles were introduced into the system and next position of a particle within the system. This led to an often predictable pattern of particle movements, that is, a particle's next position can be determined. Although Reeves considered gravity to simulate an arc-like pattern for the particle flow to add realism, it was still insufficient to model the unpredictable nature of a real flame.

A newly improved particle system, called The Second Order Particle System by IImonen and Kontkanen [8] extended Reeves's work to address this issue. The Reeve's particle system, lacked dynamic forces that could influence the trajectories of particles often resulting in artificial looking animations. However, this was in contrast to Sabo's [16] view that "A key point regarding particle systems is that they are chaotic. Instead of having a completely predetermined path, each particle can have a random element, called a stochastic component, which modifies its behavior. This random element is, in fact, the main reason why particle systems are so good at reproducing realistic effects" [16].

The novel idea behind IImonen and Kontkanen's work is that "forces affect other forces" [8]. The key element described in thier system was the force generators which are "special paticles that exert force that moves other particles". These force generator particles were themselves subjected to the influence of forces leading to a more dynamic and realistic special effect. In a fire simulation demonstrated by IImonen and Kontkanen, vortices were added into the second order particle system. This characterised fire's nature of unpredictable motion, uneven distribution and clustering of the flames [8]. This system contrasts Reeves's system whereby flame particles would have followed a precalculated trajectory. The Second Order Particle System's computational efficiency is a plus point in real-time systems such as video games where special effects have to be convincing real while at the same time not compromise on the computing speed.

While Reeves' particle system was mainly used in modelling environmental effects, Reynolds's [15] took a different approach in its application. Reynolds described an alternative method to simulating the aggregate motion of a flock of birds by use of a distributed behavioural model. Reynolds observed three important characteristics of birds in a flock. First, birds in a flock avoid colliding into one another while in transit. Secondly, the birds match their speed with one another and maintain the same direction as other flock members when flying

and finally, they stick close to each other. The bird objects or flock-members are actually particles that have been replaced with geometrical shape models. These flock-member objects have an orientation, thus, a path that can be predetermined unlike particles which take any random path.

In addition, unlike particles in particle systems, objects in Reynolds's distributed behavioural model have to interact with one another just as birds in real-life do. Given the large number of birds in a flock, it is a tedious process to script complex pathfinding algorithms for each bird without any flaw. Reynolds argued that doing so would result in scripts that may be hard to maintain or modify should any changes be required in the behaviour of the flock. Reynolds presented a method that could address this problem by assuming that a flock is "the result of the interaction between the behaviors of individiual birds". This is done by simulating the behaviour of a single bird and observing its participation in a flock. If the bird model displayed correct flock-member behaviour, multiple instances of that bird model can be created and allowed to participate in a flock. This creates the effect that birds of the same flock display similar characteristics and pattern in their motion.

The works of Reeves's [14], Sabo's [16], Burg's [18], Reynold's [15], and Lamorlette's and Foster's [9] have greatly influenced my work as ideas drawn from their approaches formed the basis of my implementation, which is next discussed.

CHAPTER 3

Implementation of Fire Using Particle Systems

The focus of this project is to create realistic looking real-time fire in computer graphics. This involves the implementation of a computer simulation program that was used to study and model the dynamics and aesthetics of fire to achieve results that appear realistic and can run smoothly in real time. In my implementation of a camp fire model, I have chosen to take the aesthetic modelling approach using particle systems. The technique chosen for modelling a real-time fire is the Reeve's particle system [14].

Real-time modelling of fire can be categorised into scientific models and aesthetic models. While both models attempt to create realistic flames, the purpose, aim, and application of these models vary from each other. Scientific models are driven by research into the dynamics of a specific phenomenon. These models aim to produce accurate representation of fire from its underlying physics, including its thermodynamics. The purpose of scientific models is to gain an indepth knowledge of the behaviour of fire. Scientific models are often used in empirical studies, ecological and environmental studies, and military simulations.

In contrast, aesthetic models are concerned with only the visual representation of fire, that is the appearances of flames on screen. The purpose of aesthetic models is to recreate the visual effects of fire using relatively less computationally intensive techniques and calculations than scientific models. As such, aesthetic models are more suitable for resource constrained computers or time critical applications such as video games. Results will demonstrate that this model can be used to simulate a camp fire as well as a laminar flame model.

3.1 The Model

Modelling the dynamics of fire using particle systems relies on four important factors: the shape, luminiscence, opacity, and the colour of the particles. The overall shape of a fire is modelled by the movements and sizes of the particles. Effects such as flickering and separation seen in turbulent flames depend on the dynamics of the particles. The luminiscence of the particle system simulates the incandescence effects of the flame. Incandescence is the effect of light given off by heated objects. Fire is not a solid matter but a highly heated and energised gaseous phenomenon. Thus, objects on the other side of a fire can be seen through the flames. The opacity factor of the flames determines this translucency effect seen in fire.

The colour of a particle system determines the texture and affects the appearance of a fire. In order to recreate the colour of flames as realistic as possible, photographs of real flames were used as reference. Observations showed a flame's colour depended on the type of fuel and oxidiser used during the combustion process. Carbon-based fuel tended to produce flames with an orange or yellow appearance while gaseous fuel produce flames with blue appearance. For simplicity, only carbon-based simulations were considered. Next, the design of a specific model of a particle system developed is presented.

The particle system implemented has three important structures: the particles, the emitters, and the emitter manager. Particles are represented as tiny objects or points on screen with no definitive shape, structure, size or predictable motion. This non-deterministic nature of the particles is achieved by subjecting each particle's and emitter's attributes to stochastic processes to give it the chaotic behaviour.

Particles are created and introduced into the particle system by emitters. Emitters are random points positioned within a specified boundary. In fact, the emitters themselves can be thought of as particles that are static. Emitters have various properties which contribute to the dynamics of the particles and general appearance of a flame model. The key attributes of an emitter are its position, burst rate or emission rate of particles, initial directional vector of emission, dynamic list of particles, and the energy of the emitter.

The position of the emitters determines where the emitters are created within the particle system. In this implementation of the particle system, emitters are placed within a circular boundary, which mark the fire's radius. Emitters also control the rate of emission of the particles and the general direction in which they are emitted as described in [10]. The emission rate or burst rate, determines the number of particles that are emitted by each emitter. As shown in Figure 3.1,

Figure 3.1: This figure shows the layered and clustered effects of particles emitted at the same rate and time step.

particles can appear to be emitted in layers. This is because, at each rendering cycle, all active emitters emit the same number of particles at the same time. As a result, particles in the same rendering cycle appear clustered together that move in layers. Therefore, to ensure particles are not emitted at a constant rate, which causes clusters of particles to be released at the same time, the burst rate is given a random variance. This results in a particle stream that has a more realistic effect as shown in Figure 3.2.

When an emitter is created, it is assigned a random number of particles which are contained in a dynamic list. This list changes dynamically when particles are added or removed. Figure 3.3 shows the effects of particles travelling in straight lines in the vertical direction. This is because the particles were not emitted in any random direction. This behaviour is not suitable for fire. Therefore, particles are assigned to their respective emitters and emitted at different directions. The initial directional vectors of these particles are determined by two angles of rotation around the pitch (X-axis) and yaw (Y-axis). These two angles of rotation are varied by randomising the values between zero and 180 degrees for each emitter to keep emissions above the ground.

The energy of an emitter determines how long it will stay active in the particle system. Each iteration, the emitter's energy is decreased until it reaches a minimum value which indicates its expiry. An expired emitter is removed from the system along with all the particles that had been assigned to it. An emitter manager keeps track of all the emitters in the system using a dynamic list to store emitters. Each time step, when a new emitter is created, it is automatically added to this dynamic list. The emitter manager also updates the status of the

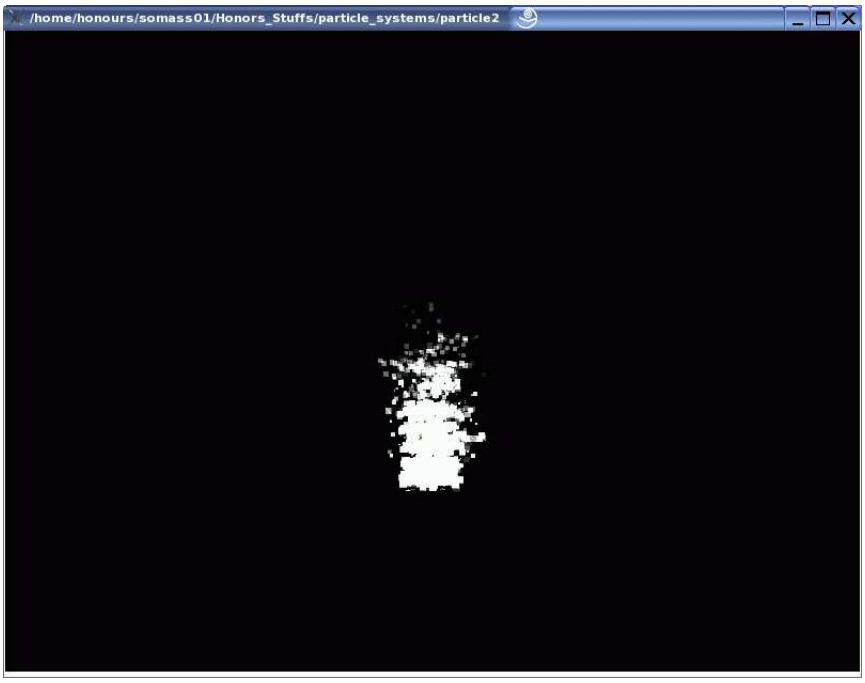

Figure 3.2: This figure shows how the particles are emitted when emitters are given random burst rates. The burst rate of an emitter determines how many particles are emitted by the emitter in each iteration. When the value of the burst rate is randomised for each emitter, particles are emitted in random amounts. As a result, particle streams do not move in clusters or layers as shown above.

No.	**Attribute**	**Description**
1	Position	The location of an emitter within the three-dimensional boundary.
2	Burst Rate	Number of particles emitted in one iteration.
3	Angle of Rotation	Initial direction of particle emission.
4	Dynamic List	Container to hold pool of particles.
5	Energy	Lifespan of emitter in the system.

Table 3.1: This table provides a summary of all the attributes of an emitter in this implementation of the particle system.

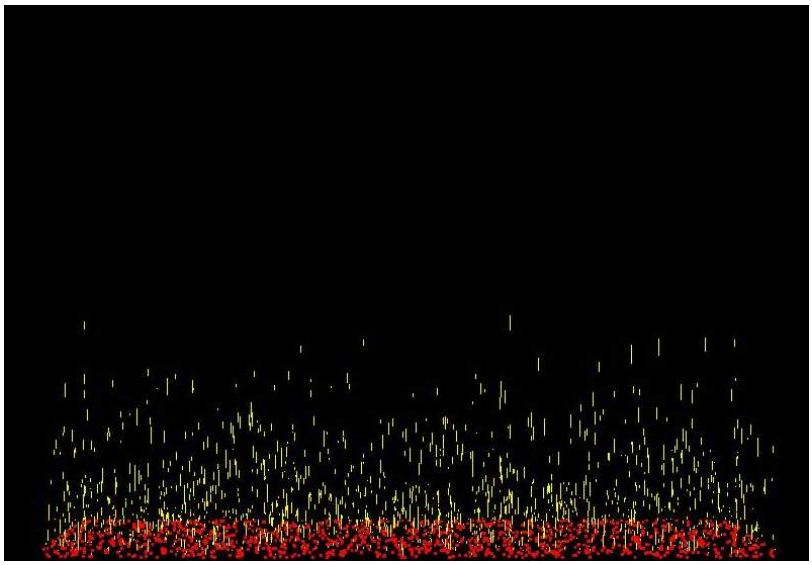

Figure 3.3: This figure shows particles travelling in a single upward direction. The particles in this implementation travel in straight lines in the vertical direction only. This is because the particles are not emitted at random angles set by the parent emitter's angle of rotation.

emitters and removes "dead" emitters in the system. It manages proper memory allocation and deallocation of pointers used in the particle system. A typical hierarchy of a particle system with the three important structures is illustrated in Figure 3.4.

The key parameters of a particle in the fire model was based on the general structure of a particle as described in Table 1.1. These parameters include the position, energy, colour, opacity, velocity, and size of the particles. The starting position of a newly created particle is set to its parent emitter' position and changes throughout its lifespan. The velocity of a particle determines its next position in the system. As with emitters, particles have an energy attribute which decides how long they will exist in the system. In the particle system model developed, a particle which has expired, is simply reset back to its starting point and reincarnated. This is to improve the performance of the system by reducing memory allocations and deallocations when creating and removing objects. In addition, a particle's colour and size are determined by its energy and plays a significant role in modelling the appearance of a flame. As a particle's energy decreases, its size and intensity of its colour decreases as well as its opacity.

In the next section, description of the different approaches taken to model the

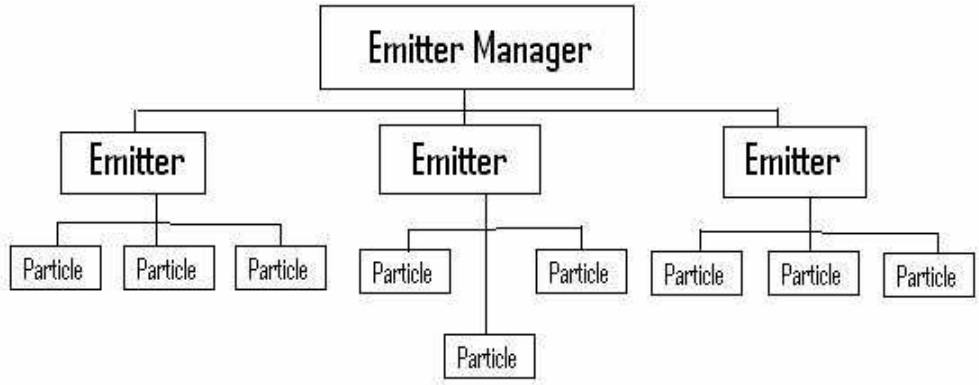

Figure 3.4: This figure illustrates the three layer hierarchy of the current implementation of the particle system. At the highest level is the Emitter Manager which has a dynamic list of emitters that grows and shrinks when emitters are added or removed. The emitters themselves have a dynamic list that contains a pool of particles.

dynamics of the particle system for a real-time fire will be discussed.

3.2 Particle Dynamics

Several approaches were investigated to model the movements of particles to simulate real flames. In general, at a given time *(t)*, a particle's new position is governed by its velocity *v(t)*, acceleration *a(t)*, and iteration (Δt). As previously discussed, the initial directional vector of a particle, is determined by its parent emitter's angle of rotation, which causes particles to be emitted at different directions. Once a particle's direction is derived, its velocity is used to determine how fast the particle will travel in that direction. To calculate the next position of the particle *p(t+Δt)*, the following system of equations were used,

$$p(t + \Delta t) = p(t) + (v(t) * \Delta t) \tag{3.1}$$

$$v(t + \Delta t) = v(t) + a(t) \tag{3.2}$$

$$a(t + \Delta t) = (v(t) - v(t - \Delta t))/\Delta t \tag{3.3}$$

This system of equations used the particle's current velocity *v(t)* and time difference Δt to determine the distance covered by the particle. This distance is

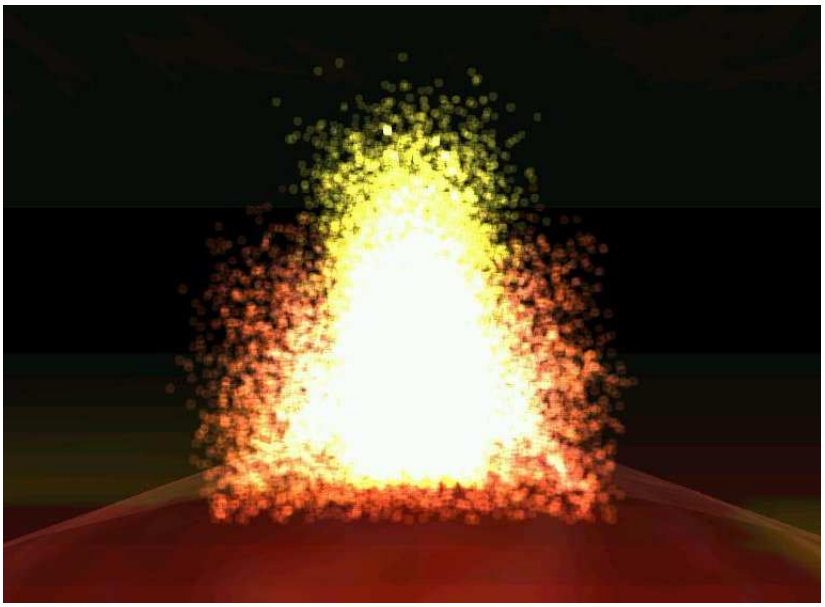

Figure 3.5: This figure shows the result of an early approach taken to model fire. The effects are unrealistic as the particles travel in wild and unrestrained manner.

added to the particle's current position, *p(t)*, to derive its next position, $p(t+\Delta t)$. The change in a particle's velocity is derived by adding its current acceleration value, *a(t)* to its current velocity *v(t)*. A particle's acceleration, *a(t)*, is calculated by dividing the change in the particle's current and previous velocity by the time interval, Δt.

Initial studies used the above system to depict fire. However the results, as seen in Figure 3.5, presented an inaccurate representation of the fire movement. This is because the particles moved rapidly and in any random direction, behaving wildly.

To control the velocity and reduce the chaotic behaviour of the particles, two types of velocity dampeners were used: gravity and drag. Gravity is a uniform force that affect the movements of a particle along the vertical direction (i.e. the Y-axis) by providing a downward pull on the particle. The drag factor, was used to control the movements of the particles along the horizontal axes, (i.e. the X and Z axes).

As seen in Figure 3.6, a second approach examined the flow of particles using the trigonometric functions. The motivation behind this method was to model the movement of the particles in wave-like patterns seen in flames. This method did

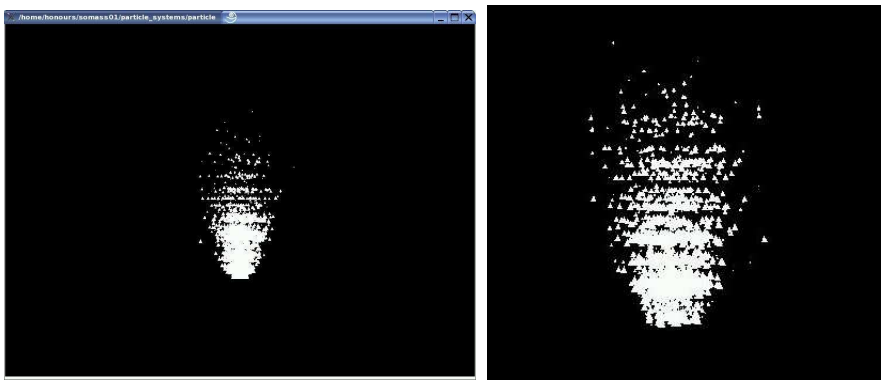

Figure 3.6: This figure shows the effects of particle movements when modelled using sin and cosine trigonometric functions to create wave-like flow. The effects shown above, however, did not prove realistic as the movements appeared too symmentrical and predictable.

not produce effects suitable or fire because the particle flow was too symmetrical and predictable.

A more refined approach was derived from the study of a video capture of fire. Based on these observations, it was determined that "the tongue" of the flames, in the outer rim of the fire, constantly changed its orientation. A randomness was observed in the way the tongue swayed sideways. As explained by Harris [7], this phenomenon occurs because as gases in a fire get hotter, they become less dense than its surrounding air. Thus, the gases move upwards towards a region where the pressure is lower. To simulate this low pressure effect, a set of random points were picked from each axis, to form an array of coordinates in three-dimensional space. These random points were used to simulate points of low pressure. Each iteration, these pressure points were changed randomly to form new sets of coordinates.

When a particle is emitted into the system, it is moved incrementally towards the nearest low pressure point. Thus, the rising particles form sharp peaks as observed in real flames. Initially the low pressure points were stationary co-ordindates. However, the results were not realistic. To rectify the problem, the low pressure points were allowed to move randomly within a specified boundary. This approach is illustrated in Figure 3.8. The flame model generated by this approach is shown in Figure 3.7.

Graphical applications, such as video games, particle systems based fire modelling techniques have incorporated physics-based engines. The motivation to find out whether the results of the previous approach could be further improved

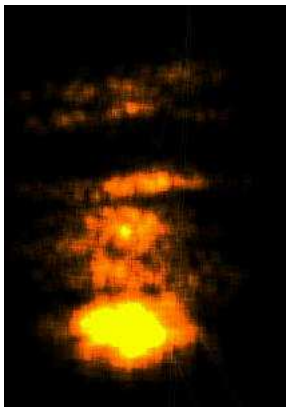

Figure 3.7: This figure shows a flame model generated by approach that simulates particles moving towards low pressure regions.

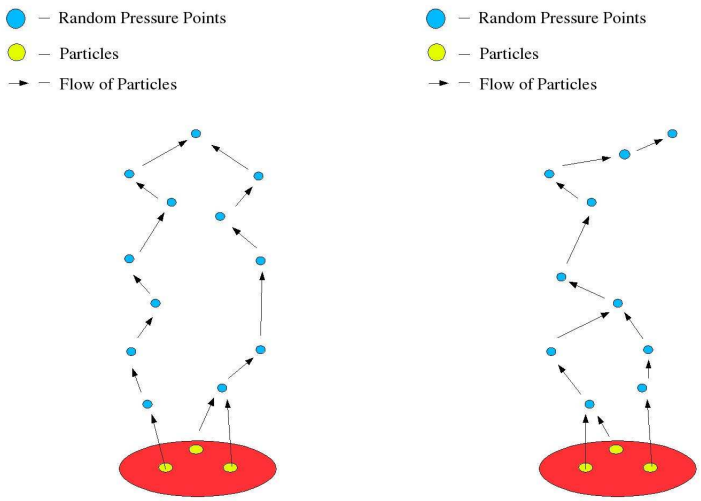

Figure 3.8: This figure shows at each iteration, random points within the boundary of the particle system are selected. When a particle is emitted into the system, it is moved incrementally towards the nearest low pressure point.

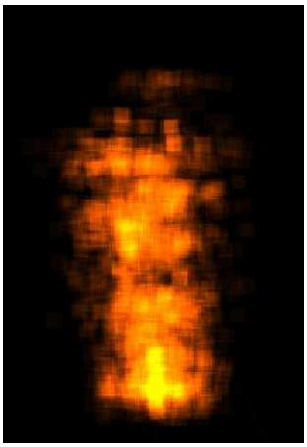

Figure 3.9: This figure shows a fire model was simulated using a simplified version of the thermodynamics algorithm given by Elias [4]. It extends the previous approach where particles are simulated as moving towards low pressure points.

in terms realism led to the implementation of a simple algorithm that simulated the thermodynamics behaviour of flames. The visual effects of this thermodynamics engine is shown in Figure 3.9. This algorithm is a simple model of the thermodynamics behaviour of fire. An abstraction of the algorithm written to implement the thermodynamics as given by Elias [4] is:

$$FUEL + OXYGEN \approx HEAT + WASTE \tag{3.4}$$

This equation states that the combustion of fuel and oxygen, produces heat with waste or smoke being the by-product. Elias's [4] algorithm to model this chemical reaction which is

$$R_p = (O_p * F_p * H_p - E_b) * k \tag{3.5}$$

where R_p is the reaction rate of a particle, O_p is the particle's oxygen value, F_p is the fuel component, H_p is the particle's initial heat or ignition temperature, E_b is the energy barrier of the fuel, and k is the reaction rate constant. Stam observed that in order to sustain fire, reactants, fuel, oxygen, and initial heat need to be present. The absence of any of the reactants causes fire to extinguish. The rate of chemical reaction is dependent on the concentration of the reactants and the temperature. Higher temperature and concentration of the reactants results in a faster reaction rate. The concentration of the reactants is derived by the product of O_p, F_p, and H_p.

In a chemical reaction described by Equation 3.5, the fuel must be supplied with an initial energy or heat, which is high enough to overcome its energy barrier. This energy barrier of a fuel, also referred to as its ignition temperature, is the minimum activation energy required to start the exothermic chemical reaction. This is simulated in the system by subtracting the product of O_p and F_p by E_b to determine whether a positive chemical reaction has started. The result is multiplied with k to derive R_p. The rate constant, k, is the speed of the exothermic chemical reaction. If the reaction rate, R_p, drops below zero, it implies there is no reaction to sustain the fire. The value of R_p is limited by the maximum reaction rate of the chemical reaction. The maximum reaction rate is an arbitrary value that can be pre-defined.

Each iteration, value of R_p is deducted from O_p and F_p to model the depletion of the reactants. If O_p or F_p drops below zero, the particle's energy is set to zero and subsequently the particle is removed from the system.

Another key observation made on the characteristics of fire was that in the event of a turbulence, the body of the fire moved sideways as a whole. That is, individual flames bodies that make up an entire fire did not bend or sway in separate directions but in a single direction. This meant individual particles cannot be emitted in random directions. There were two methods of modelling this behaviour. The first method was described by Lamorlette and Foster [9] who simulated this behaviour by modelling individual flame elements as parametric space curves. For this, a set of points were interpolated to define a flame's spine. The curves change over time using physics-based, procedural, hand-defined wind fields. In considering turbulence, these curves are re-sampled to ensure continuity. However, as this approach was beyond the scope of this project for implementation, an alternate approach was designed to simulate the effects of a moving flame body.

One of the key advantages of particle systems is that particles or objects that have similar behaviour or representations can be modelled as an aggregate. This can be used to eliminate the tedious process of scripting and animating individual objects by defining rules to govern the overall behaviour of the particles. Reynolds [15] demonstrated this advantage by presenting a distributed behavioural model for simulating the movements of birds of the same flock. Using Reynolds's work, the existing three-layer hierarchy of the particle system was extended to a four-layer hierarchy, as can be seen in Figure 3.10. Groups of emitters were merged into a single family of emitters. Emitters of the same family have the same values for their burst rate, direction of emission, energy, and number of particles. Similarly, particles under the same family of emitters share the same velocity, size, and energy. The family of emitters are in turn managed by a manager class

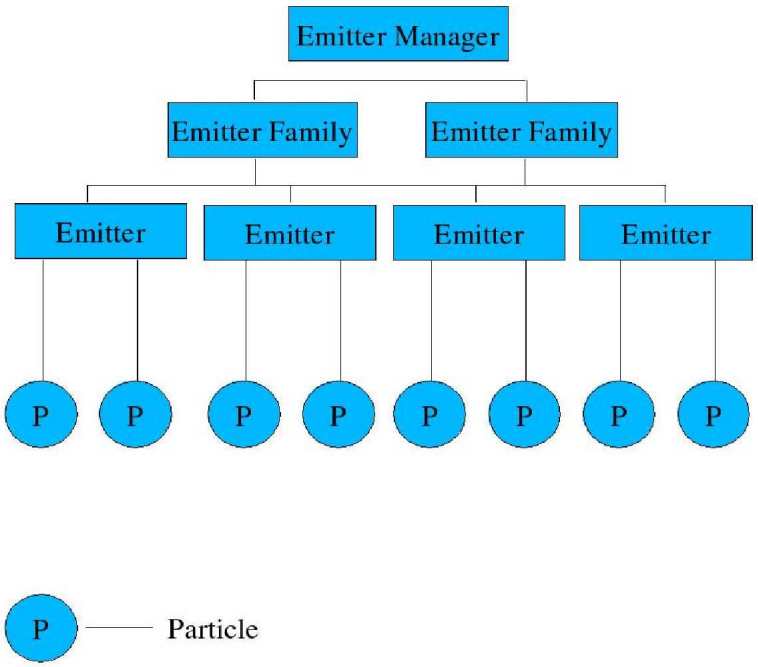

Figure 3.10: This figure illustrates a four-layer hierarchy which extends the previous three-layered hierarchy of the particle system. The emitter manager is at the highest level and has a dynamic list of emitter families that grows and shrinks when emitter families are added or removed. Emitter families each have a dynamic pool of emitters. The emitters themselves have a dynamic list that contains a pool of particles. Emitters of the same family share the same burst rate, direction of emission, energy, and number of particles. Similarly, particles of the same family of emitters have common values for velocity, size, and energy.

which keeps track of them.

CHAPTER 4

Experimental Results

A computer simulation based on the model discussed in Section 4 was successfully developed and implemented using Angel's [1] simulator as a basis. The computer model was written using the C programming language and the OpengGL 1.5 API. The program was compiled using GNU Compiler Collection (GCC) version 3.3.6. Simulation environment used SUSE 9.1 operating system, AMD Opteron 64 bit processor, 1024MB ram, and Ge-Force 6800 256MB Graphical Processing Unit (GPU). Each test ran for 150 seconds. This was empirically determined to be the optimal time for these tests.

For an animation, images are rapidly displayed in succession with "slight changes in the content" [19] to trick the human eye into believing the changes as movement. "The human eye perceives these changes as movement because of its low visual acuity" [19]. Higher the frame rate, in other words, faster the rate of change in the images, the smoother the movement will seem to the human eye while lower rates will seem jerky. As the frame rates in motion pictures and television are 24 frames per second and 30 frames per second respectively, a minimum frame rate to achieve for the particle system would be 24 frames-per-second. Fast frame rates are crucial to video games where speed of game play is very important to gamers. Because the frames-per-second (fps) is the standard unit for measuring performance in animation, a set of codes were written to calculate the frame rate of the particle system.

Testing focused on two main features of the particle system: particle parameters and thermodynamics. Results of these tests are next presented with a discussion to follow in the next section.

4.1 Particle Parameter Testing

Particle parameter testing examined particle attributes. Attributes studied included the optimal particle emission rate, optimal type of the particles, and

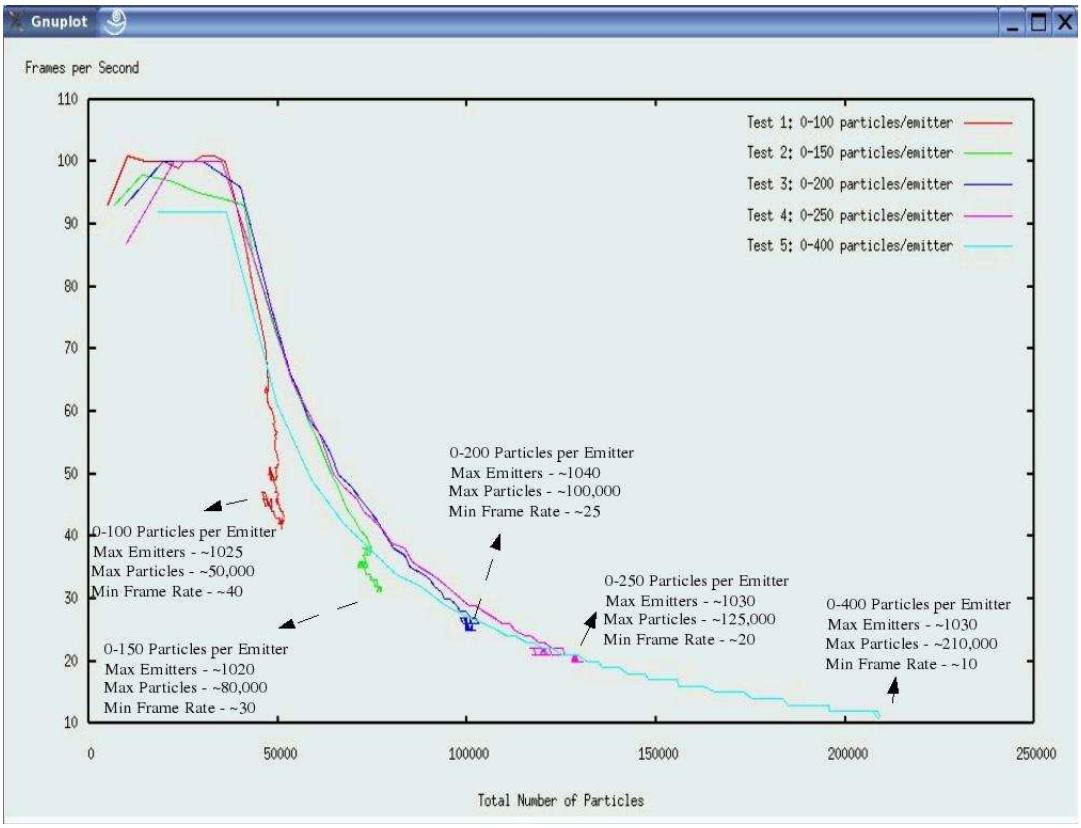

Figure 4.1: This figure displays a graph showing the results of the tests conducted to determine the particle emission rate of emitters and maximum number of particles to introduce into the particle system without sacrificing a smooth frame rate of the animation.

finally the energy, size, and colour of the particles. Experiments first considered the particle emission rate of the system. The aim of this testing is to determine the maximum number of particles the system can have without sacrificing performance. Testing varied the maximum number of particles an emitter may have. Values considered included 100, 150, 200, 250, and 400. During testing, the frame rate and total number of particles in the system were recorded, The results of the tests conducted for the particle emission rate are provided in Figure 4.1.

The next set of tests examined the type of particles emitted. The emitted particles were squares, cubes, triangle-strips, and spheres. These tests examined how object type affected the performance of the particle system. Its predicted that the greater number of vertices within a given particle the slower the performance. Thus, it is expected that squares will have a better performance than

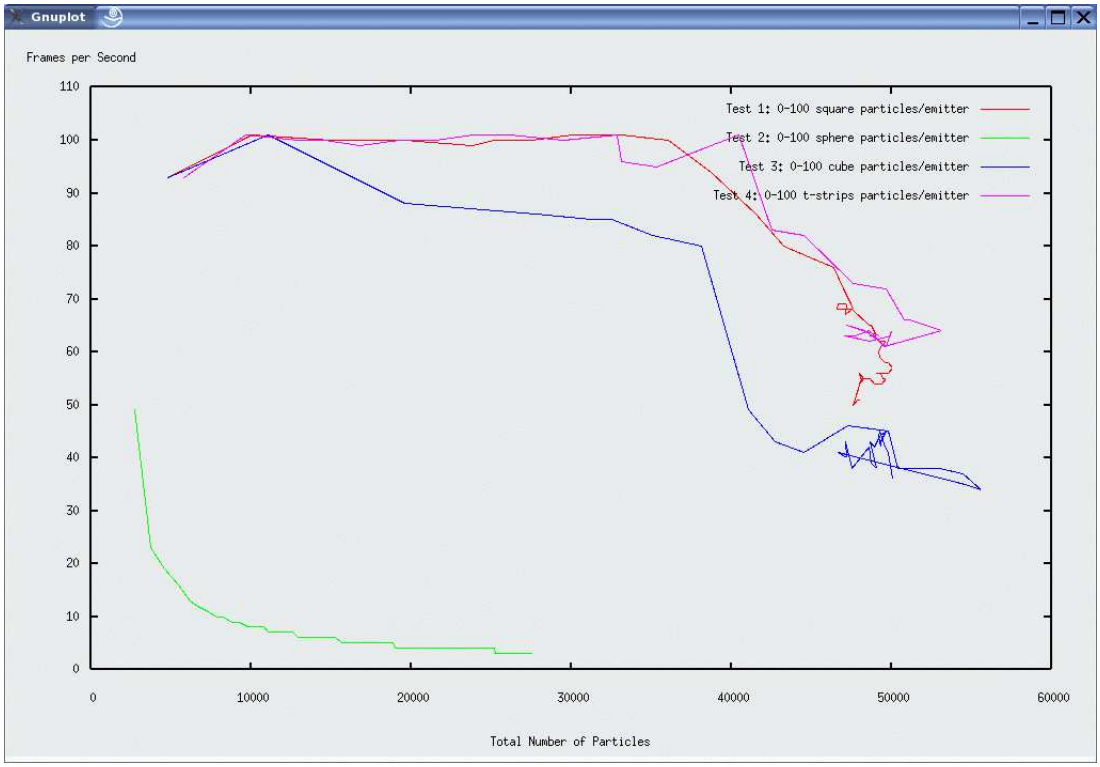

Figure 4.2: This figure displays a graph showing the different frame rates recorded when using spheres, cubes, square particles. Particles emitted as spheres as shown above produced the lowest frame rate. The fastest frame rate was achieved by using square particles followed by cube particles.

the vertex-heavy spheres. Results of these experiments are shown in Figure 4.2.

Particle parameter testing then studied the effects a particle's energy. For these tests, a particle's initial energy, P_e, was initialised using,

$$P_e = ENERGY + (Rand() * 11)/10 \tag{4.1}$$

where ENERGY is an user input for a particle's base energy value and, $Rand()$ was a random number between 0 and 10. During testing, the value for ENERGY was varied between -3.0 and 3.0. This value was dynamically varied during runtime using the simulator. Table 3.1 shows the list of values used for setting the ENERGY parameter of a particle.

Results of the test values, -3.0, and -2.0 for ENERGY are given in Figure 4.3 followed by the outputs for the test values, -1.5 and -1.0, as shown in Figure 4.4.

No.	ENERGY	Range of Particle's Initial Energy
1	-3.00	-3.00 to -2.00
2	-2.00	-2.00 to -1.00
3	-1.50	-1.50 to -0.50
4	-1.00	-1.00 to -0.00
2	-0.50	-0.50 to 0.50
3	-0.30	-0.30 to 0.70
4	-0.20	-0.20 to 0.80
5	-0.10	-0.10 to 0.90
6	0.00	0.00 to 1.00
4	0.10	0.10 to 1.10
4	0.20	0.20 to 1.20
7	0.30	0.30 to 1.30
8	0.50	0.50 to 1.50
9	1.00	1.00 to 2.00
9	1.50	1.50 to 2.50
10	2.00	2.00 to 3.00

Table 4.1: The range is derived by adding the ENERGY value to the randomly generated number between 0.0 to 1.0.

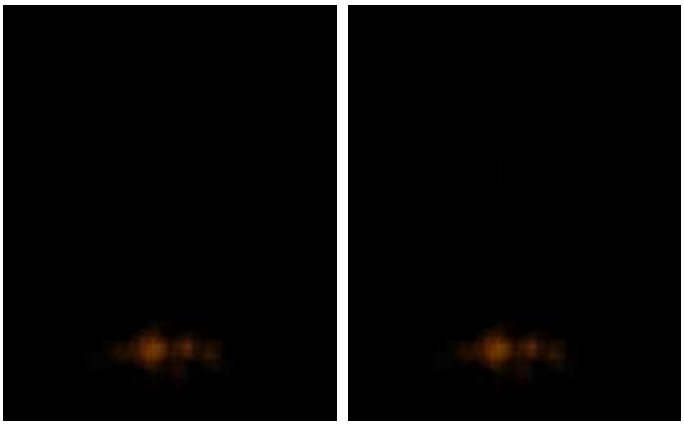

Figure 4.3: This figure shows the result of the flame model when the particles' ENERGY value is set to -3.0 and -2.0. Left: Flame model with particles' ENERGY value set to -3.0. The range of energy values is between -3.0 to -2.0. Right: Flame model with particles' ENERGY value set to -2.0. The range of energy values is between -2.0 to -1.0.

Figure 4.5 shows the effects of the flame models produced when the ENERGY value was tested with -0.5 and -0.3. The results shown in Figure 4.6 are for the ENERGY test values -0.2 and -0.1. The next set of outputs, for tested values 0.0 and 0.1 are given in Figure 4.7. Figure 4.8 shows the output for the test values 0.2 and 0.3. The flame models for test values 0.5 and 1.0 are given in Figure 4.9. The results for the last two test cases are given in Figure 4.10.

A flame's volume is defined by the size of the particles. Whether the shape of a flame appears blocky or smooth depends on the size of the particles as well. Because an optimal particle size is required to achieve the desired volume and shape, testing next examined particle size. These tests investigated a range of particle sizes to derive the best particle size as well as examine what other visual effects could be produced by modifying the parameter. For these tests, the initial particle size,P_s, was determined using

$$P_s = SIZE + (Rand() * 11)/10 \tag{4.2}$$

where SIZE value of a particle's size was limited between -3.00 to 3.00. The simulator allowed testing to change the SIZE dynamically. Table 4.1 shows a list of values used for setting the SIZE parameter of a particle.

Results of the test values, -3.0, and -2.0 for SIZE are given in Figure 4.11 followed by the outputs for the test values, -1.5 and -1.0, as shown in Figure 4.12.

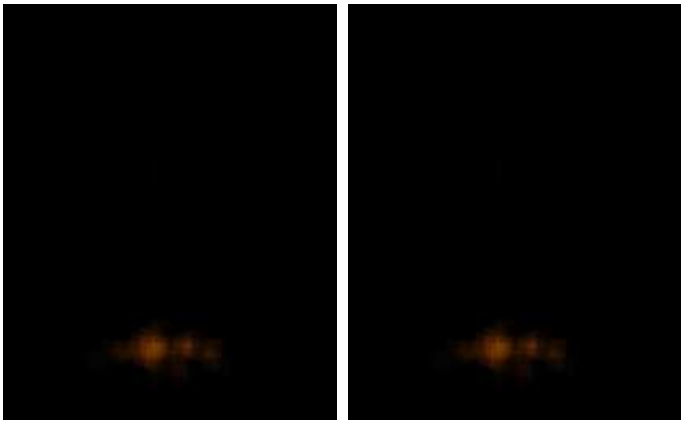

Figure 4.4: This figure shows the result of the flame model when the particles' ENERGY value is set to -1.5 and -1.0. Left: Flame model with particles' ENERGY value set to -1.5. The range of energy values is between -1.5 to -0.5. Right: Flame model with particles' ENERGY value set to -1.0. The range of energy values is between -1.0 to 0.0.

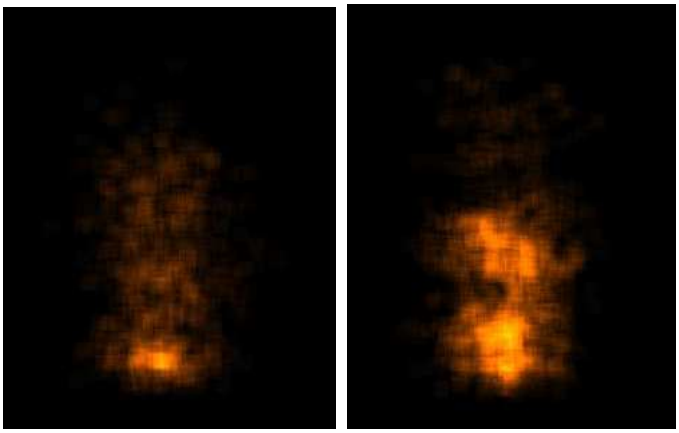

Figure 4.5: This figure shows the result of the flame model when the particles' ENERGY value is set to -0.5 and -0.3. Left: Flame model with particles' ENERGY value set to -0.5. The range of energy values is between -0.5 to 0.5. Right: Flame model with particles' ENERGY value set to -0.30. The range of energy values is between -0.30 to 0.70.

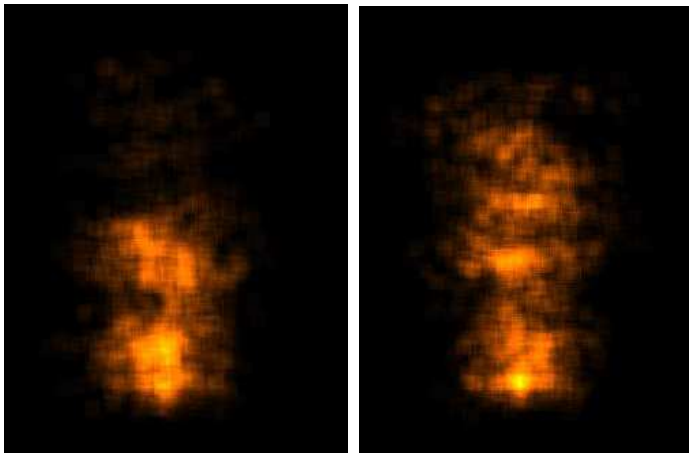

Figure 4.6: This figure shows the result of the flame model when the particles' ENERGY value is set to -0.2 and -0.1. Left: Flame model with particles' ENERGY value set to -0.20. The range of energy values is between -0.20 to 0.80. Right: Flame model with particles' ENERGY value set to -0.10. The range of energy values is between -0.10 to 0.90.

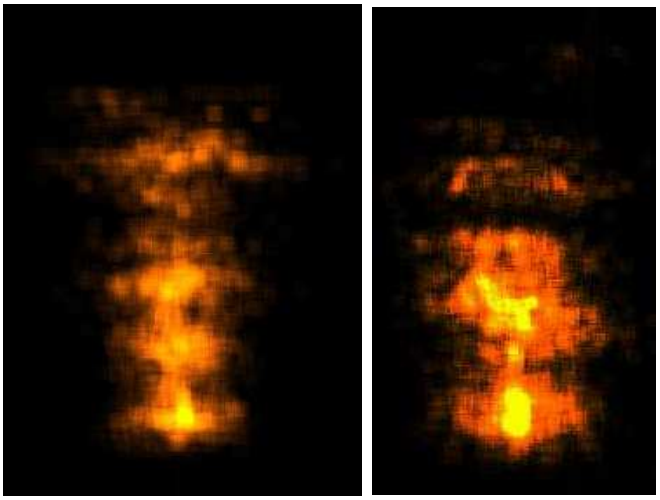

Figure 4.7: This figure shows the result of the flame model when the particles' ENERGY value is set to 0.0 and 0.1. Left: Flame model with particles' ENERGY value set to 0.0. The range of energy values is between 0.0 to 1.0. Right: Flame model with particles' ENERGY value set to 0.1. The range of energy values is between 0.1 to 1.1.

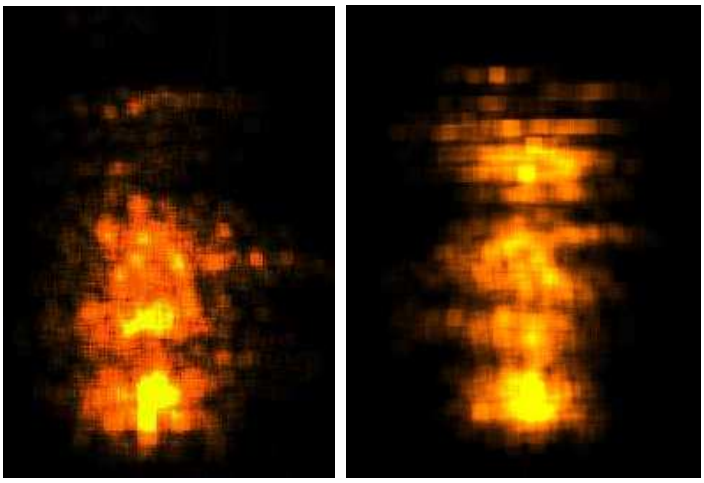

Figure 4.8: This figure shows the result of the flame model when the particles' ENERGY value is set to 0.2 and 0.3. Left: Flame model with particles' ENERGY value set to 0.2. The range of energy values is between 0.2 to 1.2. Right: Flame model with particles' ENERGY value set to 0.3. The range of energy values is between 0.3 to 1.3.

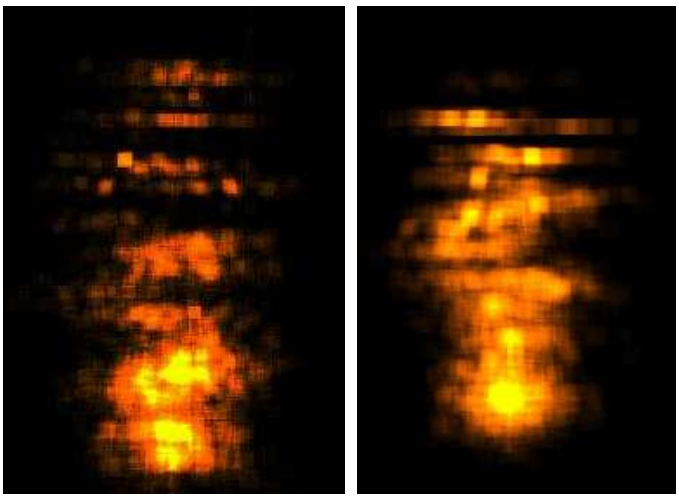

Figure 4.9: This figure shows the result of the flame model when the particles' ENERGY value is set to 0.5 and 1.0. Left: Flame model with particles' ENERGY value set to 0.5. The range of energy values is between 0.5 to 1.5. Right: Flame model with particles' ENERGY value set to 1.0. The range of energy values is between 1.0 to 2.0.

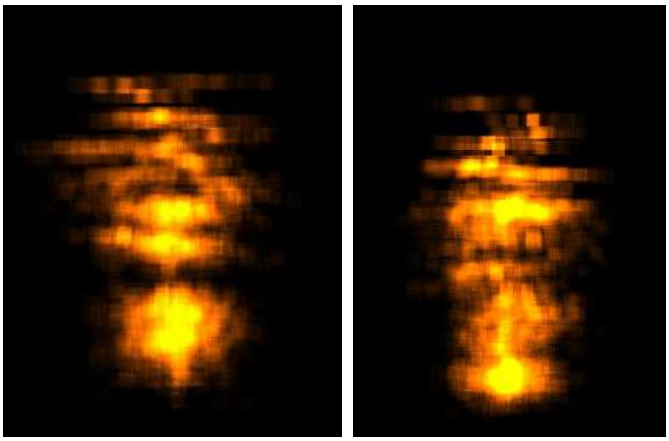

Figure 4.10: This figure shows the result of the flame model when the particles' ENERGY value is set to 1.5 and 2.0. Left: Flame model with particles' ENERGY value set to 1.5. The range of energy values is between 1.5 to 2.5. Right: Flame model with particles' ENERGY value set to 2.0. The range of energy values is between 2.0 to 3.0.

Figure 4.13 shows the effects of the flame models produced when the SIZE value was tested with -0.5 and -0.3. The results shown in Figure 4.14 are for the SIZE test values -0.2 and -0.1. The next set of outputs, for tested values 0.0 and 0.1 are given in Figure 4.15. Figure 4.16 shows the output for the test values 0.2 and 0.3. The flame models for test values 0.5 and 1.0 are given in Figure 4.17. The results for the last two test cases are given in Figure 4.18.

Modelling the aesthetics of a fire is as equally as important as its dynamics. The current parameter testing focused on creating a camp fire which is the result of carbon-based combustion. Therefore, the flames produced are yellow and orange colours. There are two ways of defining the colour of a particle in OpenGL: materials and colours. Color is used to specify the red, blue, and green values of the particles. Materials define how light is reflected off the particles' surfaces and therefore the type of material they are made of. This work focused on colour rather than materials. In particular, because carbon-based flames are yellowish-orange in colour, only the red and green colour components of the particles were varied. The particles' blue colour value was set to 0.0 for all the tests. The values tested for the red, blue, and green colour components of the particle system are given in Table 4.3. The results of the tested colour values are shown in Figures 4.19, and 4.20.

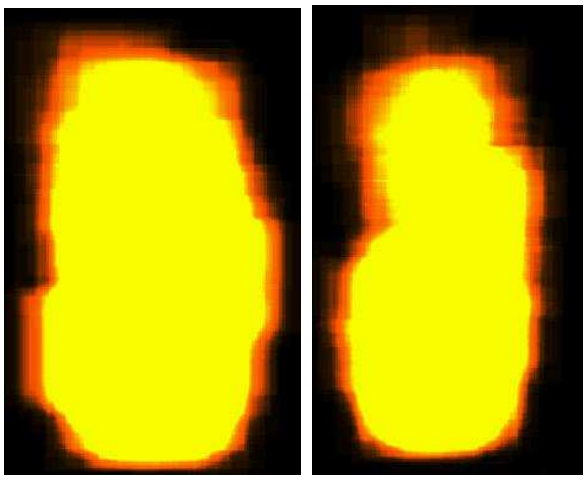

Figure 4.11: This figure shows the results of the flame model when the particles' SIZE values are set to -3.0 and -2.0. Left: Flame model with particles' SIZE value set to -3.0. The range of size values is between -3.0 to -2.0. Right: Flame model with particles' SIZE value set to -2.0. The range of size values is between -2.0 to -1.0.

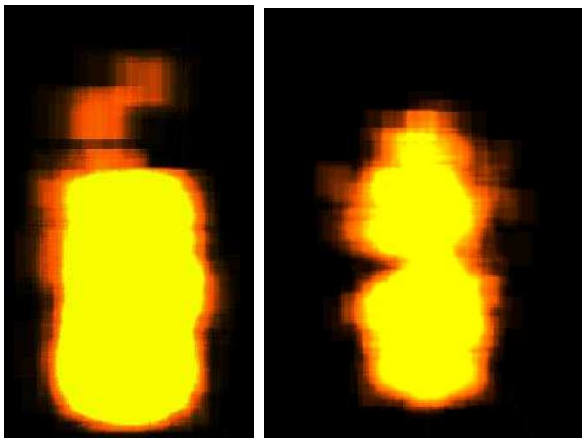

Figure 4.12: This figure shows the results of the flame model when the particles' SIZE values are set to -1.5 and -1.0. Left: Flame model with particles' SIZE value set to -1.5. The range of size values is between -1.5 to -0.5. Right: Flame model with particles' SIZE value set to -1.0. The range of size values is between -1.0 to 0.0.

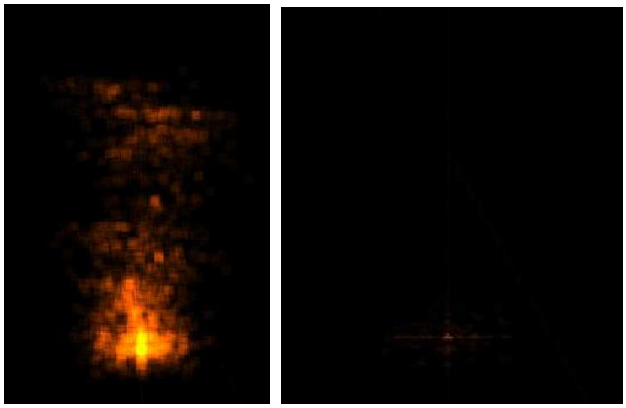

Figure 4.13: This figure shows the results of the flame model when the particles' SIZE values are set to -0.5 and -0.3. Left: Flame model with particles' SIZE value set to -0.5. The range of size values is between -0.5 to 0.5. Right: Flame model with particles' SIZE value set to -0.30. The range of size values is between -0.30 to 0.70.

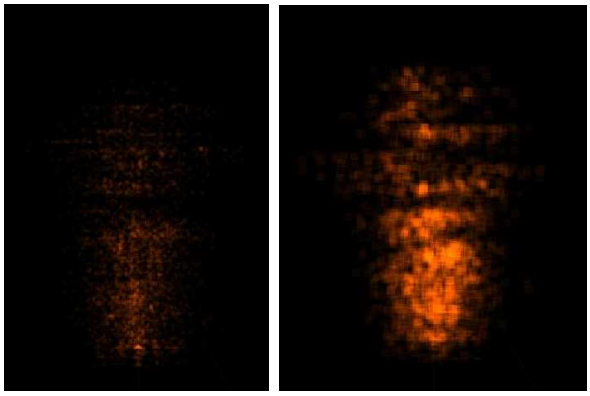

Figure 4.14: This figure shows the results of the flame model when the particles' SIZE values are set to -0.2 and -0.1. Left: Flame model with particles' SIZE value set to -0.20. The range of size values is between -0.20 to 0.80. Right: Flame model with particles' SIZE value set to -0.10. The range of size values is between -0.10 to 0.90.

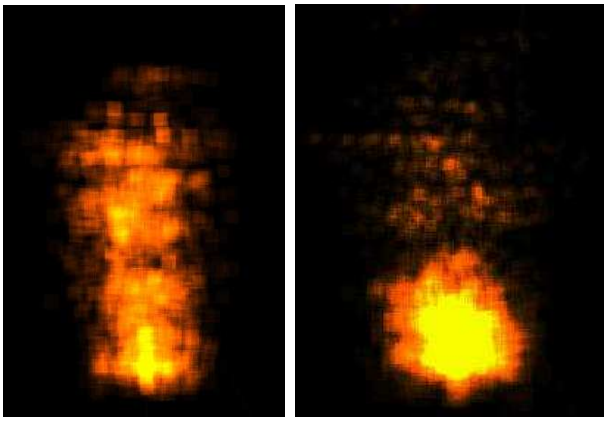

Figure 4.15: This figure shows the results of the flame model when the particles' SIZE values are set to 0.0 and 0.1. Left: Flame model with particles' SIZE value set to 0.0. The range of size values is between 0.0 to 1.0. Right: Flame model with particles' SIZE value set to 0.1. The range of size values is between 0.1 to 1.1.

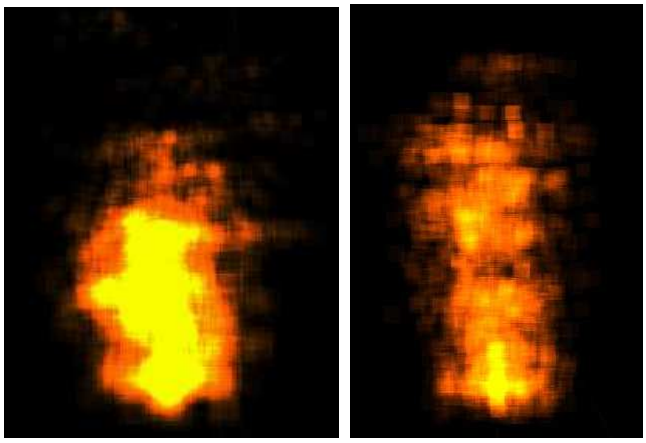

Figure 4.16: This figure shows the results of the flame model when the particles' SIZE values are set to 0.2 and 0.3. Left: Flame model with particles' SIZE value set to 0.2. The range of size values is between 0.2 to 1.2. Right: Flame model with particles' SIZE value set to 0.3. The range of size values is between 0.3 to 1.3.

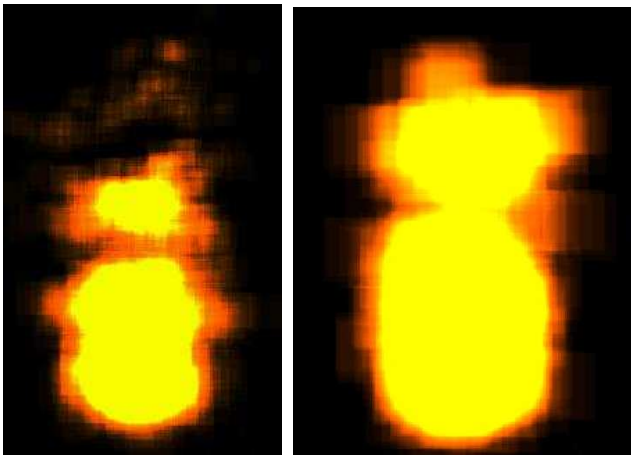

Figure 4.17: This figure shows the results of the flame model when the particles' SIZE values are set to 0.5 and 1.0. Left: Flame model with particles' SIZE value set to 0.5. The range of size values is between 0.5 to 1.5. Right: Flame model with particles' SIZE value set to 1.0. The range of size values is between 1.0 to 2.0.

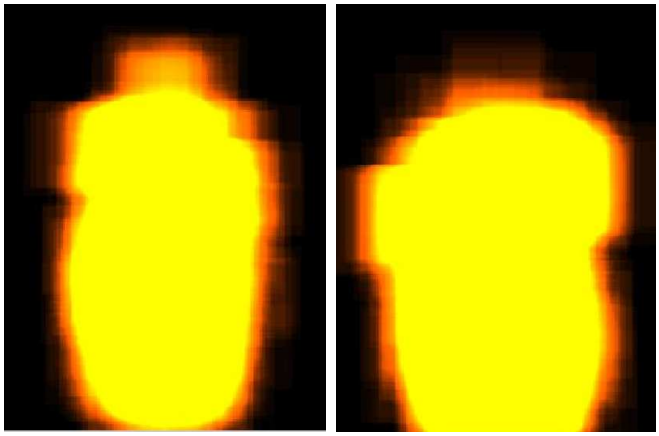

Figure 4.18: This figure shows the results of the flame model when the particles' SIZE values are set to 1.5 and 2.0. Left: Flame model with particles' SIZE value set to 1.5. The range of size values is between 1.5 to 2.5. Right: Flame model with particles' SIZE value set to 2.0. The range of size values is between 2.0 to 3.0.

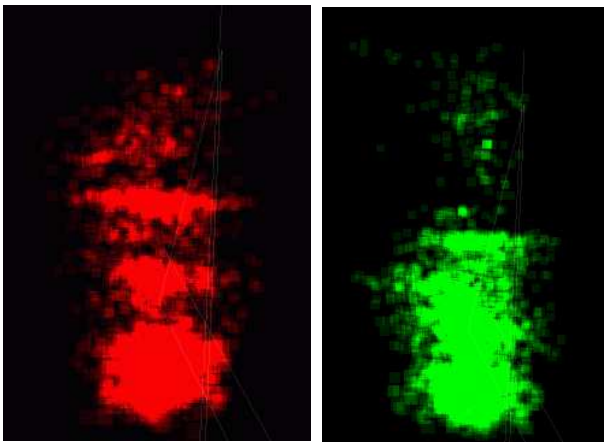

Figure 4.19: This figure shows two flame models. The one shown on the left is of pure red colour while the model on the right is entirely green.

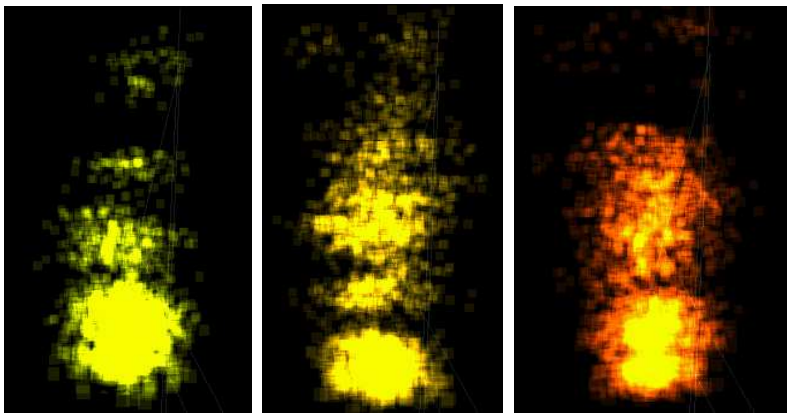

Figure 4.20: This figure shows three flame models with different colours. The flames shown on the left is coloured yellow entirely. Yellow colour was achieved by setting the red and green colour intensity of the particles to full and the blue colour value to 0.0. The flame model in the middle is set with colour amber. Amber colour was derived by setting the red colour component to maximum and the green component with 75% intensity. The flames, shown on the right, were given a yellowish orange colour. This was done by setting the intensity of the red colour component of the particles to full with only 40% green component.

No.	SIZE	Range of Particle's Initial Size
1	-3.00	-3.00 to -2.00
2	-2.00	-2.00 to -1.00
3	-1.50	-1.50 to -0.50
4	-1.00	-1.00 to -0.00
2	-0.50	-0.50 to 0.50
3	-0.30	-0.30 to 0.70
4	-0.20	-0.20 to 0.80
5	-0.10	-0.10 to 0.90
6	0.00	0.00 to 1.00
4	0.10	0.10 to 1.10
4	0.20	0.20 to 1.20
7	0.30	0.30 to 1.30
8	0.50	0.50 to 1.50
9	1.00	1.00 to 2.00
9	1.50	1.50 to 2.50
10	2.00	2.00 to 3.00

Table 4.2: This table provides the range of values tested for the particle's initial size at runtime. The range is derived by adding the SIZE value to the randomly generated number between 0.0 to 1.0.

4.2 Thermodynamics

The thermodynamics engine of the particle system was examined to study the corelation between the energy barrier (E_b), reaction rate constant (k), maximum reaction rate (Max Rate), and the particle's fuel (F_p), oxygen (O_p), and heat (H_p). The objective was to observe how each thermodynamics parameter affected the visual output. The experiment was divided into three parts with each part examining different parameters and effects of the system.

The first part of the experiment was conducted to determine whether any flame was generated in the absence of heat, oxygen, or fuel. Table 4.4 shows the input values set for the fuel, oxygen, and heat. As seen in Table 4.4, the component tested was set to 0.0 to simulate absence. Results of these tests demonstrated that in the absence of heat, oxygen, or fuel, no flames were generated.

The second part of thermodynamic testing, had two-fold objectives. The first was to observe the output of the flames in relation to the reaction rate and the maximum rate of reaction. The set of test values chosen, as given in Table

No.	Red	Green	Blue	End Colour
1	1.00	0.00	0.00	Red
2	0.00	1.00	0.00	Green
3	1.00	1.00	0.00	Yellow
4	1.00	0.75	0.00	Amber
5	1.00	0.40	0.00	Orange

Table 4.3: This table provides the range of values tested for the particle's initial colour at runtime.

No.	$\mathbf{R}_p$	$\mathbf{O}_p$	$\mathbf{F}_p$	$\mathbf{H}_p$	$\mathbf{E}_b$	k	Max Rate	Effects
1	-0.02	2.00	0.00	2.00	2.00	0.01	0.10	No Flames
2	-0.02	0.00	2.00	2.00	2.00	0.01	0.10	No Flames
3	-0.02	2.00	2.00	0.00	2.00	0.01	0.10	No Flames

Table 4.4: This table shows the values used for the first part of the thermodynamics experiment. The fuel, oxygen, heat values of a particle were set to 0.00 individually to observe whether any flames were generated in the absence of a reactant. The reaction rate is determined by the equation 3.5.

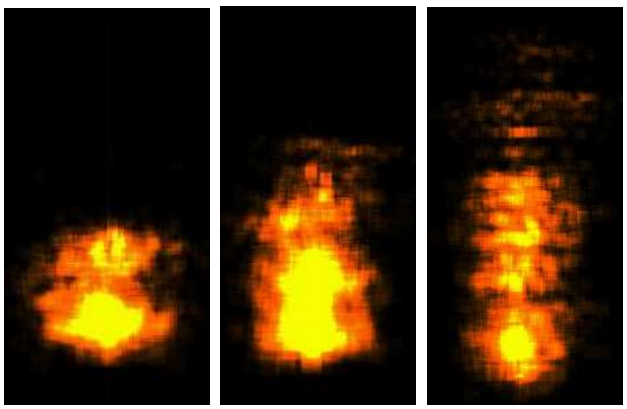

Figure 4.21: This figure shows three models of fire with different heights. Figure on the left shows a flame model of low height. Figure in the middle shows a flame model of medium height. Figure on the right shows a flame model of normal height. For the convenience to the reader, this figure is replicated in 5.2.

4.5, were to study the flames when the reaction rate was lower and higher than the maximum reaction rate. The second objective was to study what effect the reaction rates had the heights of the flames. These values are also shown in Table 4.5. Results of these tests are shown in Figure 4.21.

No.	$\mathbf{R}_p$	$\mathbf{O}_p$	$\mathbf{F}_p$	$\mathbf{H}_p$	$\mathbf{E}_b$	**k**	**Max Rate**	**Effects**
1	0.06	2.00	2.00	2.00	2.00	0.01	0.10	Flames with medium height
2	0.01	2.00	2.00	2.00	-5.00	0.01	0.10	Flames with normal height
3	1.2	5.00	5.00	5.00	5.00	0.01	0.10	Flames with normal height
4	1.25	5.00	5.00	5.00	0.00	0.01	0.10	Flames with normal height
5	0.22	3.00	3.00	3.00	5.00	0.01	0.30	Flames with medium height
6	0.44	3.00	3.00	3.00	5.00	0.02	0.30	Flames with low height
7	0.22	3.00	3.00	3.00	5.00	0.01	0.40	Flames with medium height
8	115.00	5.00	5.00	5.00	10.00	1.00	0.10	Flames with medium height
9	6.00	5.00	5.00	5.00	5.00	0.05	0.30	Flames with low height

Table 4.5: This table shows the values used to determine the corelation between the reaction rate and the height of the flames. The reaction rate is determined by Equation 3.5.

The reation rate is limited by the maximun reaction rate. So if the maximum reaction rate is 0.0, then the reaction rate is also 0.0. The concentration of the reactants is also affected by the reaction rate. If the energy barrier exceeds the reactants' concentration, the net reaction rate is negative. Thus no flames will be produced. The final set of tests on the thermodynamics examined if this was true. The values for these tests are given in Table 4.6. Results for these tests showed that for those values provided in Table 4.6, no flames were produced.

No.	R_p	O_p	F_p	H_p	E_b	k	Max Rate	Effects
1	-12.00	2.00	2.00	2.00	20.00	1.00	0.10	No Flames
2	50.00	5.00	5.00	5.00	-25.00	0.50	0.00	No Flames
3	0.00	3.00	3.00	3.00	2.00	0.00	0.10	No Flames
4	25.00	3.00	3.00	3.00	52.00	-1.00	-0.10	No Flames
5	25.00	3.00	3.00	3.00	52.00	-1.00	-0.50	No Flames
6	0.25	3.00	3.00	3.00	52.00	-0.01	-0.10	No Flames
7	0.25	3.00	3.00	3.00	52.00	-0.01	-0.50	No Flames

Table 4.6: This table shows the values used to study under what conditions flames are not produced. The reaction rate is determined by the equation 3.5.

4.3 Laminar Flame Model

The existing model produces a camp fire. Testing next determined whether this computer model could be extended to simulate a simple laminar flame model. The outcome will then produce a candlelight flame.

Observations showed a laminar flame had relatively low turbulence compared to camp fires. Turbulence in the camp fire model was controlled using the velocity dampener, drag. Drag controls the rate of movements of particles along X and Y axes. The distance a particle moves in the horizontal axes is shortened and its speed is slowed down by the drag factor. The higher the value of the drag factor, the shorter and slower the particle travels in the X and Y axes. Lower values specified for drag causes particles to move rapidly along the horizontal axes causing the flames to behave wildly. To reduce the turbulence seen in the camp fire model, the drag settings were increased till the flow of the particle system was steady and completely upwards. The equations for controlling the movement of the particles using the drag factor is given by

$$P_x(t + \Delta t) = P_x(t) + ((V_x(t) * \Delta t)/DRAG_x) \tag{4.3}$$

$$P_z(t + \Delta t) = P_z(t) + ((V_z(t) * \Delta t)/DRAG_z) \tag{4.4}$$

where $P_x(t+\Delta t)$ and $P_y(t+\Delta t)$ and $P_x(t)$ and $P_y(t)$ are the particle's next and current positions in the X and Z axes respectively. $V_x(t)$ and $V_y(t)$ are the particle's current velocity in the X and Y direction respectively. The time difference is denoted by Δt. $DRAG_x$ and $DRAG_y$ are the user defined drag values for the X and Y directions. As illustrated by the equation, the higher the drag values, the slower the speed of the particle in that direction.

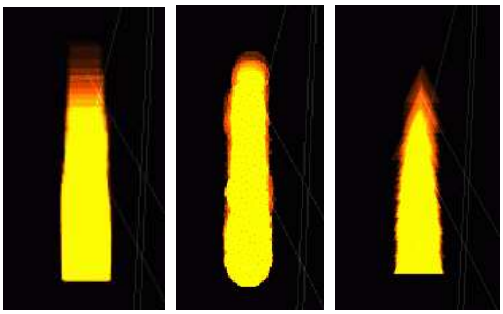

Figure 4.22: This figure shows three different laminar flame models. The flame model shown on the left is created using particles rendered as cubes. The laminar flame in the middle is created using spherical particles. The laminar flame model on the right is created using triangle-strips.

Modelling the shape of a laminar flame required choosing the right type and size of particles. The shape of a typical laminar flame is a thick and broad base converging into a thin and sharp peak. Setting the right size for the particles was based on the results from the particle size experiment. Next, there were three types of particles that could be used for modelling the shape of the laminar flame: three-dimensional triangle-strips, cubes, and spheres. Figure 4.22 shows three laminar flame models each using a different particle type.

In the following section, the outcome of the experiments conducted on the various particle parameters, thermodynamics parameters, and laminar flame model extension will be discussed.

CHAPTER 5

Discussion

The results of the experiment on particle emission rate, as seen in Figure 4.1, show the highest frame rate recorded for the test running a maximum of 100 particles per emitter was 41.04fps. The test introduced a maximum of approximately 51000 particles. Next, the system was set to generate up to 150 particles per emitter. This yielded a minimum frame rate of 30.14fps with a maximum of about 77000 particles. Results of the test generating a maximum of 200 particles per emitter, showed a minimum frame rate of 25.20fps and a maximum of about 102000 particles. A low frame rate of 20.07fps with a maximum particle count of about 130000 particles was recorded when the system was experimented with a particle emission rate of up to 250 particles per emitter. The lowest frame rate was recorded in the final particle emission test, in which the maximum number of particles an amitter generated was up to 400 particles. The results derived were a low frame rate of 11.0 fps upon reaching a maximum particle count of approximately 209000 particles.

These results signify that the frame rate of the particle system animation decreases as the total number of particles in the system increases. The maximum number of particles a particle system can have depends on the particle emission rate. Higher particle emission rate increases the maximum number of particles in the system. Slight fluctuations in the frame rate and the total number of particles can also be seen in Figure 4.1. This is most noticeable when the particle systems reached their respective maximum number of particles. This is because the older emitters that have depleted their energy are removed from the system resulting in a decrease in the total number of particles in the system. This causes the frame rate to go up. But as soon as an emitter is removed, a new one is added to the system which starts to introduce new particles that increases the total number of particles. This in turn causes the frame rate to drop. The rate at which "dead" emitters are removed and new ones are added once the maximum number of particles is reached keeps the particle system well populated with particles. Based on the results on this experiment, the optimal particle emission rates have been deduced to be 0 to 100, 0 to 150, and 0 to 200 particles per

emitter. This means the maximum number of particles the system can introduce is approximately 102,000, beyond which the performance of the animation starts to deteriorate.

The results in Figure 4.2 show that the highest frame rates were achieved when particles were emitted as triangle-strips. The recorded minimum frame rate for this was 61.34fps with a corresponding number of particles of approximately 49800. This was followed by the emission of particles as squares. The frame rate dropped to 50.62fps with approximately 47500 particles. The frame rate when representing cubes as particles yielded a decent frame rate of 34.05fps The lowest frame rate was the result of representing spheres as particles. A value of 3.16fps was recorded for approximately 27400 particles when spheres were used.

Particle type testing show that the type of particles used in the particle system affects the overall performance of the system. Using three-dimensional objects for particle emission is slower than two-dimensional objects in terms of computational speed. This is due to the number of vertices needed to be calculated and drawn by the system. As the number of surfaces of an object increases, the number of vertices required to draw the object increases as well. Three-dimensional objects require many vertices to be drawn on screen. For instance, for 10,000 particles of type squares generated into the system, 40,000 vertices needs to be calculated, given a triangle-strip is defined by three vertices. However, 240,000 vertices would be required if the particles used were of type cubes, each with 24 vertices. Spherical objects, in particular, drastically slow down the animation due to heavy computations involved in calculating the tessellations and curvatures. Higher tessellations cause spherical objects to look smoother, however, this draws a lot of calculations that will hog the system's performance. This is shown in Figure 4.2 where the frame rates dropped below 10fps even before the total number of sphere type particles in the system reached 10,000.

From examining the results of the particle's energy experiment, a key observation was made on the effect a particle's energy had on the appearances of the fire. The observation was the amount of energy particles had affected the overall translucency of the entire fire. The test values, -3.0, -2.0, -1.5, and -1.0, for the ENERGY resulted in no flames being generated, as shown in Figures 4.3 and 4.4. The next test range, -0.5 to 0.5, produced barely visible flame as shown in Figure 4.5. The flames were highly translucent. Over the next three test values -0.30, -0.20, -0.10, the flames generated became gradually brighter in terms of visibility and colour. Test values -0.1, 0.0, and 0.1 produced the best results, as shown in Figures 4.6 and 4.7, in terms of the flame's visual quality and realism. The edges or outer rims of the flames appeared smooth and curvaceous. When the test values of the ENERGY were raised beyond the value of 0.00, the flames

produced were of high intensity and brightness. Furthermore, the edges of the flames appeared rough with jagged edges. The overall lifespan of the particles was also seen to be affected by the energy value of the particle. Particles with lower ENERGY values had shorter lifespan and extinguished sooner than the particles with higher ENERGY. This affected the height of the flames becuase, the longer a particle lives, the farther it will travel. Between Figure 4.6 to Figure 4.10 a significant rise in the heights of the flames can be noticed.

In OpenGL, the alpha property of a rendered object's colour affects its transparency. Objects with higher alpha values appear more opaque while lower alpha values causes objects to appear more transparent. In this implementation of the particle system, the particle's energy attribute sets the value for its alpha property. Particles with a ENERGY value below -0.30 appeared too translucent that the flames produced did not appear realistic. On the other hand, particles with a ENERGY value above 0.1 appeared too bright and intense. The increased brightness and intensity is the result of the accumulation of the alpha values of individual particles drawn on the same screen area. For instance, if two particles each with an alpha value of 50% are drawn on the same screen area, their alpha values are added together resulting in a 100% alpha value. This is an effect of the anti-aliasing feature seen the animated flames that is also giving the current flame model smooth and well-curved edges. A particle's energy is used to set the value of its alpha component. In turn, the alpha component is used for blending the particle's colour with the background. When the ENERGY value of the particles is increased, which increases the alpha component's values of the particles, the blending effects are not very noticeable. Based on this observation, the best values for a particle's ENERGY attribute have been deduced to be between -0.1 to 0.1. The colours of aging particles can be alpha-blended with these optimal energy values to create the effect of particles dying gracefully.

For the particle sizes tests, test values -3.00, -1.5, -1.0, -0.5, 0.5, 1.50, and 2.00 for the SIZE of the particles, produced very unrealistic looking flames. The results of the test values are shown in Figures 4.11, 4.12, 4.13, 4.17, and 4.18. The flames appeared deformed and blocky. When the SIZE value was set to -0.30 there were no visible flames. The visual effects generated for tests values ranging between -0.10 and -0.20 appeared as though particles were emitted as steam or vapour. These effects could be incorporated into scenes that require modelling of water vapour or hot gases rising. Test values ranging between 0.00 and 0.10 for the SIZE produced the best looking flames. The results from the experiment conducted on the particle's size parameter showed that different visual effects can be achieved by modifying the sizes of the particles. These visual effects can be employed in a variety of applications, not necessarily fire. As an example, the results shown in Figures 4.13 and 4.16 can be used simulate images formed on

Figure 5.1: The figure shows a real-life fire. The colour was used as a reference to model the colour of the simulated flame model.

a nightvision goggles or flares in action-based video games.

The final experiment conducted on particle parameters was the colour attribute. The red, green, and blue components of the particles were adjusted to derive a realistic colour for the flame model. The colours derived were red, green, yellow, amber, and blaze orange. Red and green are completely unrealistic as real carbon-based combustion process do not produce flames of such colours. Yellow was not acceptable as the colour appeared too uniform and bright. As a result, the edges of the flames appeared jagged. To reduce the brightness, the green component was reduced to 75% intensity, which resulted in the colour, amber. With amber, the brightness was acceptable but the jagged edges were still visible. The intensity of the green component was further reduced to 40%. The resulting colour appeared yellowish orange in texture (See Figure 4.20). The edges appeared smoother and the colour was able to match that of a real flame as shown in Figure 5.1. The conclusion drawn for the optimal colour values for the red, green, and blue components for the current flame model are 1.0, 0.4, and 0.0 respectively.

The thermodynamics test results in Table 4.4 show that flames are not produced under the following conditions: when the maximum reaction rate is zero or negative, when the particle's fuel, oxygen, or heat is zero or negative, or when there is a negative reaction rate. Three factors influenced negative reactions rates. First factor was when the energy barrier was higher than the reactants' concentration. Second, when the reaction rate constant was negative while the reactants' concentration were positve. Lastly, when the maximum reaction rate is negative and is lower than the reaction rate, the reaction rate is set to the

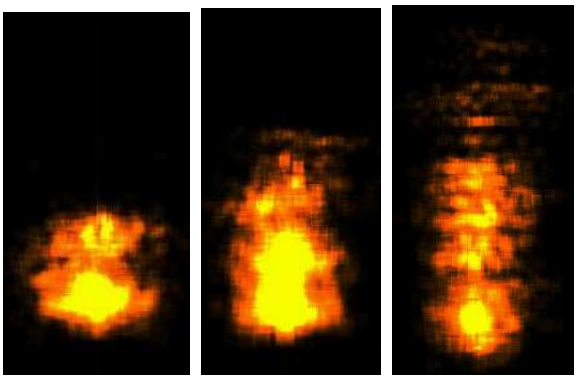

Figure 5.2: This figure shows three models of fire with different heights. Figure on the left shows a flame model of low height. Figure in the middle shows a flame model of medium height. Figure on the right shows a flame model of normal height.

maximum reaction rate.

The results also show that when the reaction rate is lower than the maximum rate of reaction, the fuel and oxygen values of the particles are subtracted by the value of the reaction rate. On the other hand, if the maximum rate of reaction is lower than the reaction rate, the fuel and oxygen values are deducted by the maximum rate of reaction. For the same amount of fuel concentration, the higher the reaction rate, the lower the height of the flames, while the lower the reaction rate, the taller the flames appear.

A relation between the height of the flames and the concentration of the reactants, and the rate of reaction was also observed. Given the same concentration of the reactants, higher the reaction rate, the lower the height of the flames. Lower reaction rates resulted in increased height in the flames. Figures 5.2 show the three different heights recorded in the results.

The results of the thermodynamics experiment show the particle system is able to simulate the absence of a reactant or heat convincingly. The particle system is also able to simulate the effects of different fuels burning. As an example, coal is a fuel with a high energy barrier. Flames produced by coal are relatively short in height as it takes longer to burn as opposed to flames produced by the burning of gaseous or liquid fuels. The current particle system is able to model this behaviour correctly. When the energy barrier is set to a high value, the height of the flame decreases as it lowers the reaction rate. Lower energy barrier increases the reaction rates and as a result raises the height of the flames. Though not an accurate representation of the thermodynamics of real fire, the results of

the thermodynamics experiment, nevertheless, show the current particle systems implementation is able to model different behaviours of flames convincingly.

The laminar flame model was created to explore the capabilities of the current implementation of the particle system. The objective was to find out if the existing camp fire model could be extended to depict other forms of fire models. The results have been convincing and satisfactory. Figure 4.22 shows the three different types of particles used to model a laminar flame. The types of particles used were triangle-strips, cubes, and spheres. The most appealing laminar flame is determined to be the model using triangle-strips as it is able to accurately model the curves and pointed peaks seen in real laminar flames. The laminar flame model using cubes is not realistic as it appears too blocky and is not able to model the curves of a laminar flame accurately. Similarly, spherical particles are unable to model sharp peaks seen in laminar flames.

CHAPTER 6

Conclusion and Future Work

This project has successfully implementated a particle systems modelling technique to simulate fire. To study the behaviour of the particle systems, a computer simulator was written. The use of the computer simulator to empirically study the complexity of the fire model was advantageous because parameters could be varied at runtime. Tests were done using the computer simulator to examine the particle parameters and thermodynamics. Testing also demonstrated that the same particle system can be used to create different flame effects. Specifically, I was able to turn a camp fire into a laminar flame by varying the particle parameters. This work just touches on the powerful capability of particle systems.

This effort also demonstrate that simpler computer models can be used to visualise dynamical systems such as those often found in natural phenomenon. For example, my model was used to simulate fire. The same model can be slightly altered to generate dust, water vapour, bubbles, and snow. This can be done by varying not only the particle parameters but also the coupling parameters to the dynamics required to simulate these phenomenon. Such simpler models are more advantageous when performance is critical such as those required in video games. Although, this work did not investigate all applications for particle systems, it is surmised that particle systems are also useful to scientific applications such as modelling the dynamics of a real forest fire.

Extensions to the particle system can be added to improve the visual quality and realism of the fire model as well as its overall performance. As previously discussed, fire can be modelled either as a texture or as a physical model. This project has only focused on implementing a particle systems model, a physically based modelling technique. A major drawback observed in using this technique is the large amounts of particles required to animate a fire scene. The reason is to give the flames a more volumetric appearance. Each particle requires a space in the computer's memory. In a typical fire scene that requires thousands of particles, a large amount of memory space has to be allocated for the entire scene. Each time a particle is created, it is allocated a space in memory. When the particle dies and as a result, removed from the system, the allocated memory

space must be freed. If the allocation and deallocation of memory are not properly managed, the rendering performance is poor. One way of addressing this problem is to reduce the number of particles needed to animate a scene but without compromising the visual quality of the flame model. Another issue noticed in the current fire model is the lack of realistic colour and texture. An alternate solution to this problem would be to combine a texture modelling technique with the current particle system.

The need for photorealistic quality of flame models in an interactive environment require some level of texturing. "The basic motivation for wanting to model fire as a texture that can be applied to a three-dimensional object is the speed and ease of subsequent use" [6]. Texture modelling involves applying textures or images over polygonal surfaces to increase the details or information of the surfaces without adding more polygons which can significantly affect the computing performance. It is generally divided into image-based and procedural. While image-based modelling involves the traditional approach of mapping of two-dimensional images onto three-dimensional surfaces, procedural modelling uses a function or set of functions to a set of points in order to generate a texture to create natural looking textures [3]. One such procedural modelling technique is the Perlin Noise algorithm.

The Perlin Noise algorithm was created by Ken Perlin [13] while working on a new texture modelling technique for movie Tron in 1983. Categorised as procedural texture modelling, this method is now an industrial standard. Perlin Noise is a random number generator that takes in an integer value as a parameter and returns a random number based on that parameter [4]. These random values are plotted and then interpolated to produce wave graphs which are combined with other similar graphs to produce a more fractal looking image. The benefit of this method as described by Perlin is there is no requirement for source texture images as the algorithm creates random images on the fly unlike traditional techniques. This is useful in modelling the amorphous behaviour of fire which takes no definitive form. Perlin also presented another advantage. Perlin noise algorithm creates a virtual object "carved out of a virtual solid material" [13], giving it a solid look. Using this technique textures can be generated at runtime to "coat" the particles in the system with naturally looking textures that resemble that of fire. This can also be particularly useful in modelling the fire with more volume.

Bibliography

[1] Angel, E. *Interactive Computer Graphics: A Top-Down Approach Using OpenGL*, 3rd ed. Addison Wesley Professional, July 2002.

[2] Bryan E. Feldman, James F. O'Brien, O. A. Animating suspended particle explosions. ACM, ACM, pp. 708–715.

[3] Byrd, C. Hypertexture. Available at: http://www.cs.wpi.edu/ matt/courses/cs563/talks/cbyrd/pres2.html last accessed 15/10/2005.

[4] Elias, H. Perlin noise, January 2000. Available at: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm last accessed 22/04/2005.

[5] Engell-Nielsen, T., and Madsen, S. T. Modelling, animation & visualisation of fire. Master's thesis, University of Copenhagen, Universitetsparken 1 2100 Copenhagen Denmark Att: Theo Engell-Nielsen & Sren Trautner Madsen, April 1999.

[6] Eyman, Y. Rediscovering fire: A survey of current fire models and applications to 3d studio max.

[7] Harris, T. How fire works. Available at: http://science.howstuffworks.com/fire1.htm last accessed 17/07/2005.

[8] IImonen, T., and Kontkanen, J. The second order particle system. *Journal of WSCG 11*, 1 (February 2003).

[9] Lamorlette, A., and Foster, N. Structural modeling of flames for a production environment. Association of Computing Machinery, Inc., ACM, pp. 729–735.

[10] Lander, J. The ocean spray in your face. *Game Developer* (July 1998), 13–19. Available at: http://www.gdmag.com last accessed 28/05/2005.

[11] Lee, H., Kim, L., Meyer, M., and Desbrun, M. Meshes on fire. *In EG Workshop on Computer Animation and Simulation* (2001), 75–84.

[12] Nguyen, D., Fedkiw, R., and Jensen, H. Physically based modeling and animation of fire, 2002), note = Available at: http://citeseer.ist.psu.edu/nguyen02physically.html last accessed 15/09/2005,.

[13] Perlin, K. Making noise, 1999. Available at: http://www.noisemachine.com/talk1/index.html last accessed 22/04/2005.

[14] Reeves, W. T. Particle systems - technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics 17*, 3 (July 1983), 359–376.

[15] Reynolds, C. W. Flocks, herds, and schools: A distributed behavioral model. vol. 21, Association of Computing Machinery, Inc., ACM, pp. 25–34.

[16] Sabo, M. Improving advanced particle system by adding property milstones to particle life cycle.

[17] Stam, J., and Fiume, E. Depicting fire and other gaseous phenomena using diffusion processes. ACM, pp. 129–136.

[18] van der Burg, J. Building an advanced particle system. *Gamasutra* (June 2000). Available at: http://www.gamasutra.com/features/200000623/vanderburg_01.htm last accessed 13/05/2005.

[19] Vanderburg, G. L., and Morrison, M. Tricks of the java programming gurus.

[20] Woodhouse, F. Particle systems: The theory, 2002.

[21] Yang, F. Smoke simulation, November 2002. Available at: http://www.cs.usask.ca/grads/fey399/ProjectReport829.htm last accessed 23/04/2005.

APPENDIX A

Original Honours Proposal

Title: An Investigation of Current Techniques in Animating Real-Time Fire

Author: Sanandanan Somasekaran

Supervisor: Dr. Karen G. Haines

Background

The thesis of this work is that the realism afforded by current computer graphical methods for modelling and animating fire can be improved. To prove this, a physically-based modelling technique and a procedural texture modelling technique will be combined to model a real-time fire. The aim is improve the realism and visual quality of real-time animated fire. This project specifically will focus on a general model for the flame of a fire that can be used in various areas of applications, such as those mentioned above. However, the model should not be taken for an accurate physical representation or depiction of a real fire.

Fire, a natural phenomena, has for ages intrigued and fascinated mankind with its unpredictable, destructive yet controllable nature. It is described as being arguably the most visually impressive natural phenomenon [Eyman 2004]. The need for realistic graphical representation of this natural phenomenon has grown in many areas of application. Simulation of fire is now being used to model the bush fire spread to study the behaviour of forest fires to prevent widespread disasters. Computational models for fire are also being used to simulate burning buildings to design better escape and evacuation routes. Animation of fire has also become increasingly common in the entertainment industry. Movies and video games alike use special effects such as fire to enhance the visual experience of the audience.

However, the computational modelling of fire has been a long standing challenge in computer graphics due to its unpredicatable, complex and amorphous nature. In his paper discussing a simple three-dimensional model for the flame of a fire and its spread, Stam describes flames as the result from the combustion of fuels and oxidisers. When the molecules of these compounds rise to high temperatures, a chemical reaction occurs resulting in flame [Stam 1995]. While many techniques have been researched and implemented with increasingly successful results [Eyman 2004] to deliver high levels of realistic visual effects, there still are many areas in the graphical modelling of fire that still need improvement as will be discussed in the next section.

Graphical modelling of fire can be subdivided into two approaches: texture-based and physically-based modelling. Eyman presents how fire is modelled as a texture as well as using physical methods to simulate its behaviour. Physically-based fire modelling techniques use mathematical techniques and physics-based equations to model the physical characteristics of real fire. Texture modelling techniques involve applying textures or images over polygonal surfaces to increase the details or information of the surfaces without adding more polygons which can signifantly affect the computing performance. Texture modelling techniques are further divided into image-based and procedural. Image-based textures utilise the traditional approach of mapping of two-dimensional images onto three-dimensional surfaces. Procedural textures are generated by adding noise effects into many mathematical expressions.

In the last two decades, physically-based modelling and procedural texture modelling techniques have become increasingly important due to their complex and different approaches to animating special effects. A very popular physically-based modelling and procedural texture modelling technique to model real-time fire are the particle system and Perlin Noise algorithm as discussed next.

Particle system was presented by William T. Reeves in 1983 to model fuzzy objects such as fire, water, smoke and clouds [Reeves 1983]. A particle system can be basically described as a collection of tiny objects with no definitive shape, structure, size or predictable motion. These particles have attributes that determine their characteristics within a system from the time they are introduced into a system where they move and change around throughout thier lifespan before dying out. Described as the earliest computer graphics fire model by [Lee et al 2001], particle systems laid a foundation for animating realistic fire effects.

Several improvements over Reevess particle system have been presented in recent years. Lee et al. presented a new method for animating fire over polyhedral surfaces [Lee et al 2001]. Fire was represented as an evolving front and flames as particle based in the system. Lee et als work was unique in the sense,

they focused on fire propagation techniques. To create rich, complex dynamics for the flame motion, Lee et al. described uses of dynamic wind fields to simulate the movement. IImonen and Kontkanens The Second Order Particle System, extended Reevess system by presenting the concept of adding force generator particles that led added more dynamism to particle movements. These particles were themselves subjected to the influence of other forces. This concept characterised fires nature of unpredictable motion, uneven distribution and clustering of the flames [IImonen and Kontkanen 2003].

The Perlin Noise algorithm [Perlin 1999], created by Ken Perlin in 1983 is capable of modelling surfaces with natural looking textures. The method was used in the movie Tron in 1983 and has become an industrial standard. The algorithm generates random numbers based on integer value inputs. These random numbers are used to plot wave graphs which are combined with other similar graphs to produce fractal looking images. Such images appear as though they have been carved out of a solid object. [Perlin 1999] has demonstrated the technique as capable of creating realistic flame textures.

The next section discusses the areas in current fire animations that need improvement and how particle system and Perlin Noise procedural textures will be used to address these issues.

Aim

While many techniques, such as Perlin Noise algorithms and particle systems have been popular in creating special effects, to our knowledge both these techniques have not been combined together to produce visually stunning flames. The following are four issues that have been indentified in current fire animations that need improvement:

- Lack of volume in the animated flames and the fire seem translucent if not transparent because objects on one side of a flame are visible to objects on the other side.
- Animated flames have unrealistic lifespans.
- Animated flames appears more painted thus giving a very smudged look and globular look (i.e. flame particles look like blobs or round balls).
- Colors, textures and shading of fire are often unrealistic.

The objective of this project is to combine Perlin Noise and particle system to address and improve the four areas mentioned in the previous section. While using particle system can model the shape, behaviour, and dynamics of fire thus addressing the first two issues, Perlin Noise algorithm can create realistic textures and shading to improve the aesthetics of a flame, addressing the third and fourth issues.

Method

The proposed approach for modelling the structure and behaviour of flames of a fire is to first create a generic particle system. This particle system will be a simple model with particle streams moving in a predefined manner. Following that, the system will be improved upon by applying physical equations to model the dynamics of the particle streams. This will involve:

- Learning OpenGL API for graphics programming.
- Learning Maya Animation toolkit to learn about animation and particle systems.
- Researching various implementations and techniques of particle systems.
- Implementing a simple particle system based on examples.
- Improving the particle system by applying physical properties of real flames.
- Experimenting the particle system under various conditions to improve and derive the optimal conditions for smooth performance and realism of the particle system.

The next step will be to apply both image based and procedural textures onto the particles to give the particles a realistic appearance and look and feel. The rationale behind using both these approaches is to determine which texture modelling approach produces better results in terms of realism. While the image based textures are derived from two-dimensional images, procedural textures are generated using the Perlin Noise algorithm. For this, the following set of activities will be undertaken:

- Learning texture loading and mapping techniques using OpenGL API.
- Applying the particles in the particle system with image-based textures.

Activity	From	To
Researching related literatures	08/03/05	01/04/05
Exploring Particle System	18/03/05	07/06/05
Preparing Revised Proposal	19/05/05	25/05/05
Exploring texture handling	28/05/05	08/06/05
Exploring Perlin Noise algorithm	18/06/05	01/07/05
Combining Perlin Noise textures with Particle System	25/06/05	06/07/05
Writing Up Dissertation	01/07/05	01/09/05
Preparing Seminar Presentation	05/09/05	12/09/05
Writing Up Final Dissertation	14/09/05	15/10/05
Preparing Poster	20/09/05	25/10/05

Table A.1: Timetable for list of activities to be undertaken.

- Learning about Perlin Noise algorithm and how to implement it to create procedural textures.
- Applying lighting techniques to enhance the aesthetics of the animated flames.
- Experimenting the Perlin Noise algorithm to derive the optimal quality of textures for smooth performance and realism.

Following the implementation of a working flame model, a simulation environment will be created to test the special effects under different conditions. These conditions vary from using different particle emission rates, particle amounts, particle types, and different textures modelling approaches. This will require:

- Combining Perlin Noise algorithm with the particle system by applying the procedural textures over the particles.
- Testing and observing the differences in performance and realism of the particle system after combining the particle system with Perlin Noise procedural textures.
- Discussing and writing the comparison of the methods.

A timetable summarising the list of activities discussed above is provided in Table A.1.

Software and Hardware Requirements

The rendering of fire effects using particle system and Perlin Noise algorithm would be computationally intensive. There would be a requirement for high performance CPUs and powerful GPUs. The IVEC lab is equipped with the state of the art AMD Opteron processors and Ge-force Nvidia 6800 GPUs to support such graphic intensive computations.

The Alias.com website offers a Learners Edition of Maya animation tool for educational purposes. This software would be used to animate several special effects related to fire. In the course of this project, large files of graphic programs and libraries available on the Internet will be downloaded. For this purpose, a higher quota for Internet access will be required.

References

[1] Yngve.G.D., OBrien.J.F and Hodgins.J.K 2000. *Animating Explosions.* In Proceedings of SIGGRAPH 2000. ACM PressACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM 29-36.

[2] Nguyen, D.Q., Fedkiw, R. and Jensen, H.W. 2002. *Physically Based Modeling and Animation of Fire.* In Proceedings of SIGGRAPH 2002, ACM Press, 721-72.

[3] Perlin, K. 1999. *Making Noise.* http://www.noisemachine.com/talk1/index.html [1999]. Last accessed 14/03/2005.

[4] Eyman. Y. 2004. *Rediscovering Fire: A Survey of Current Fire Models and Applications to 3D Studio Max*, University of Maryland.

[5] Feldman, B. E., OBrien, J. F., Arikan, O., *Animating Suspended Particle Explosions.* The Proceedings of ACM SIGGRAPH 2003, San Diego, California, July, pp. 708-715.

[6] Shah A. 2003. *Modelling Fire and Explosions.* 4th Annual Multimedia Systems, Electronics and Computer Science, University of Southampton.

[7] AliasWavefront, The Art of Maya, *Particle Effects*, 2000. 173-74.

[8] Nguyen H. NVidia, GPU Gems, Programming Techniques, Tips and Tricks for Real-Time Graphics, *Fire in the Vulcan Demo.* Chapter 6, 87-105.

[9] William T. Reeves, *Particle Systems - A Technique for Modeling a Class of Fuzzy Objects*, Computer Graphics 17:3 pp. 359-376, 1983 (SIGGRAPH 83).

[10] Stam, J., *Depicting Fire and Other Gaseous Phenomena Using Diffusion Processes Computer Graphics*, Annual Conference Series, Vol.29, pp 129-136, 1995.

[11] Lamorlette A., Foster N., *Structural Modelling of Flames for a Production Environment*, Proceedings of the 29th annual conference on Computer Graphics and Interactive Techniques, pp 729-735, 2002.

[12] Byrd, C., *Making Noise* [online], Available http://www.cs.wpi.edu/ matt/cou rses/cs563/talks/cbyrd/pres2.html, last accessed 23/04/2005.

[13] H. Lee, L. Kim, M. Meyer, and M. Desbrun., *Meshes on Fire*, In EG Workshop on Computer Animation and Simulation, pp 75-84, 2001.

[14] IImonen, T., Kontkanen, J., *The Second Order Particle System*, Journal of WSCG, Vol.11, No.1, 2003.