EMPIRICAL COMPARISON OF THREE EXISTING METHODS FOR SIMULATING EMBERS IN COMPUTER-GENERATED FIRE

By

CLIFFORD LEE WIGGS, JR.

Bachelor of Science

Oklahoma Christian University

Edmond, Oklahoma

1997

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2006

EMPIRICAL COMPARISON OF THREE EXISTING METHODS FOR SIMULATING EMBERS IN COMPUTER-GENERATED FIRE

Thesis Approved:

Dr. Blayne Mayfield
Thesis Adviser

Dr. John P. Chandler

Dr. M. H. Samadzadeh

Dr. A. Gordon Emslie
Dean of the Graduate College

PREFACE

This study was conducted to provide an empirical comparison of three different implementations to model the paths of embers in the realm of computer-generated fire. This study provides a general background for the three methods being examined, but does not attempt a formal derivation or proof for the methods. The audience for this paper should have a level of knowledge equivalent to the degree of Bachelor of Science in Computer Science.

I thank my advisor, Dr. Blayne Mayfield. His guidance on the discovery of this topic and patience with his *'prodigal student'* was invaluable to the completion of this research.

I also thank my four children. For the majority of their lives they've seen '*Daddy'* disappear each evening to work on his '*paper*'. They will not know what to do with all the extra attention they will be getting, as my evenings are suddenly free.

Finally and mostly, I thank my wife. Her loving encouragement (and at times a swift kick) gave me the motivation to work through the hard times to completion. Her willingness to take on my share of household and family duties allowed me the free time to live in front of a keyboard. Last, but not least, a patient ear to listen to endless issues and revelations in jargon she will never understand.

I would also like to give special acknowledgement to Dr. Matt Finn in the Department of Mathematics at Imperial College London. His email correspondence and Java implementation provided the missing clue to move from theory to implementation on the Lattice Boltzmann Model.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

LIST OF EQUATIONS

Chapter I

1. Introduction

The realm of computer-generated fire has been an active one the last several years. Realistic computer-generated images have long been an interesting topic in theory, but in recent years the increases in technology have lead to this becoming more main stream. Since Hollywood and advertisers have picked up on the idea, computer-generated images are everywhere. There are even full-length motion pictures that are 100% computer-generated.

As part of this movement, more items are required to be computer-generated than have ever been before. One of these items is fire. There has been lots of research in the last several years into how to make computer-generated fire look and move in a realistic manner. Most research seems to fall into three general areas: propagation modeling, flame modeling and smoke modeling. Propagation modeling refers to how a fire spreads to begin burning in a new area and also how the fire stops burning in an area where all available combustible fuel is burned away. Flame modeling refers to the modeling of the actual flame of a fire. This covers the size, shape and color of the flame as well as the movement of flame when it is animated. Smoke modeling refers to the modeling of any gaseous output of the fire, usually smoke and soot.

1.1 Previous Research in Modeling Computer-Generated Fire

Nielsen and Madsen present a very nice and comprehensive review of the physics behind both fire and smoke visually [Nielsen and Madsen, 1999]. They review several different techniques used to depict a fire. They implement all the theory into a physically accurate ray tracer for fire and smoke that includes lighting and shadowing.

Lee et al. describe how to simulate the animation of propagation of fire across an arbitrarily complex polyhedral surfaces [Lee et al., 2001]. This allows them to model the propagation of fire across something as simple as a tabletop or as complex as a human form. Figure 1 is an example of fire propagation produced from their research. Their simulation also takes into account the effects of wind fields and the slope of the polyhedral mesh.

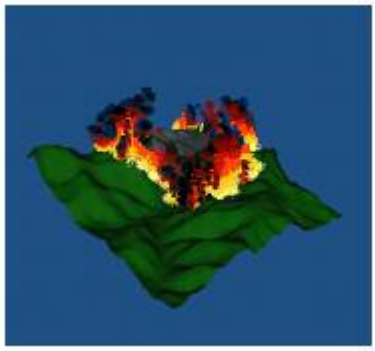
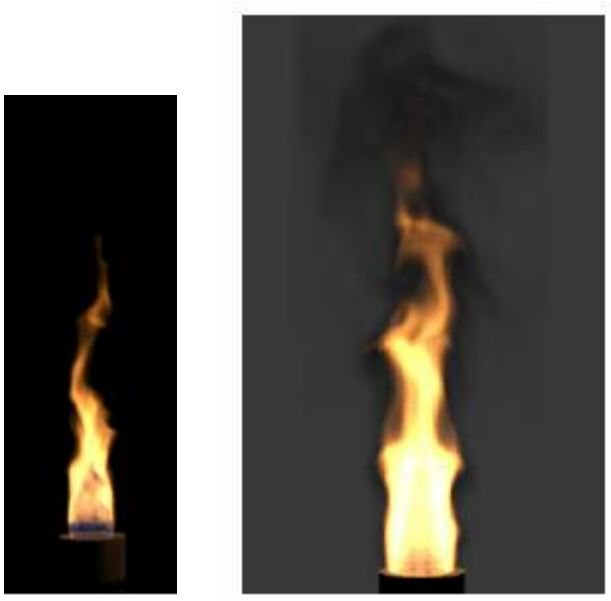


Figure 1: Example of Fire Propagation [Lee et al., 2001]

Nguyen at al. present a voxel-based approach at modeling flame [Nguyen et al., 2002]. As shown in Figure 2a, their research includes modeling the color differentiations of

flame via what they termed the blue core. The blue core is the hottest section of a flame located closest to the fuel source and the point of combustion. They also modeled smoke and soot rising off of the flame, as shown in Figure 2b.



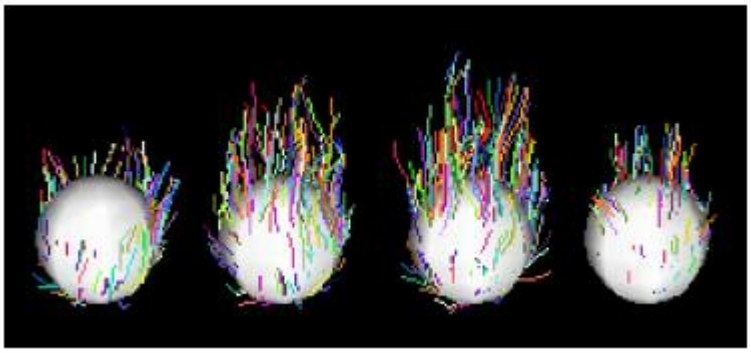
(a) (b)

Figure 2a-b: Examples of Blue Core and Rising Soot [Nguyen et al., 2002]

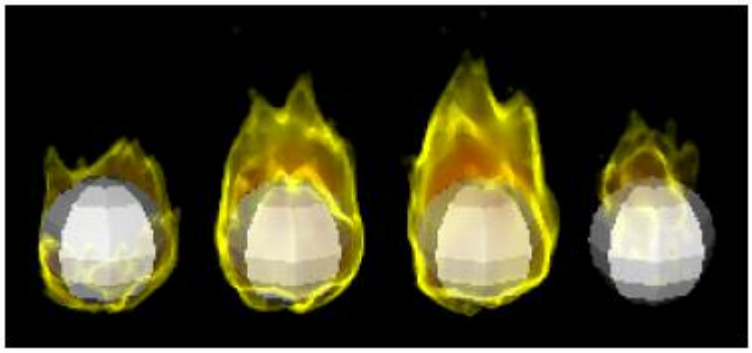
Wei et al. also describe a model for flame and smoke [Wei et al., 2002] [Wei et al., 2004]. They utilize the Lattice Boltzmann Model to control the movement of both their flame and smoke. The Lattice Boltzmann Model is discussed in detail later in this research. Their approach also uses texture splats to render both flames and smoke. A texture splat is a pre-rendered, two- dimensional bitmap. Usually several texture splats are overlapped to provide the final rendered image.

Beaudion et al. present a model of fire as a set of flames [Beaudoin et al., 2001]. Their implementation treats a single flame as a primitive called a flame skeleton. Each flame

skeleton is implemented as a deformable link of vertices rooted on the burning object. Figure 3 from their research shows some examples of flame skeletons and the rendered flame based on those primitives.



(a)



(b)

Figure 3a-b: Examples of Flame Skeletons – Un-rendered and Rendered [Beaudoin et al., 2001]

Barrero et al. researched the turbulent effects of a flame as it interacts with fresh combustible gasses [Barrero et al., 2000]. While the previous examples of flame modeling deal with the overall structure and movement of flame, this research deals very specifically with the boundary between flame and non-flame.

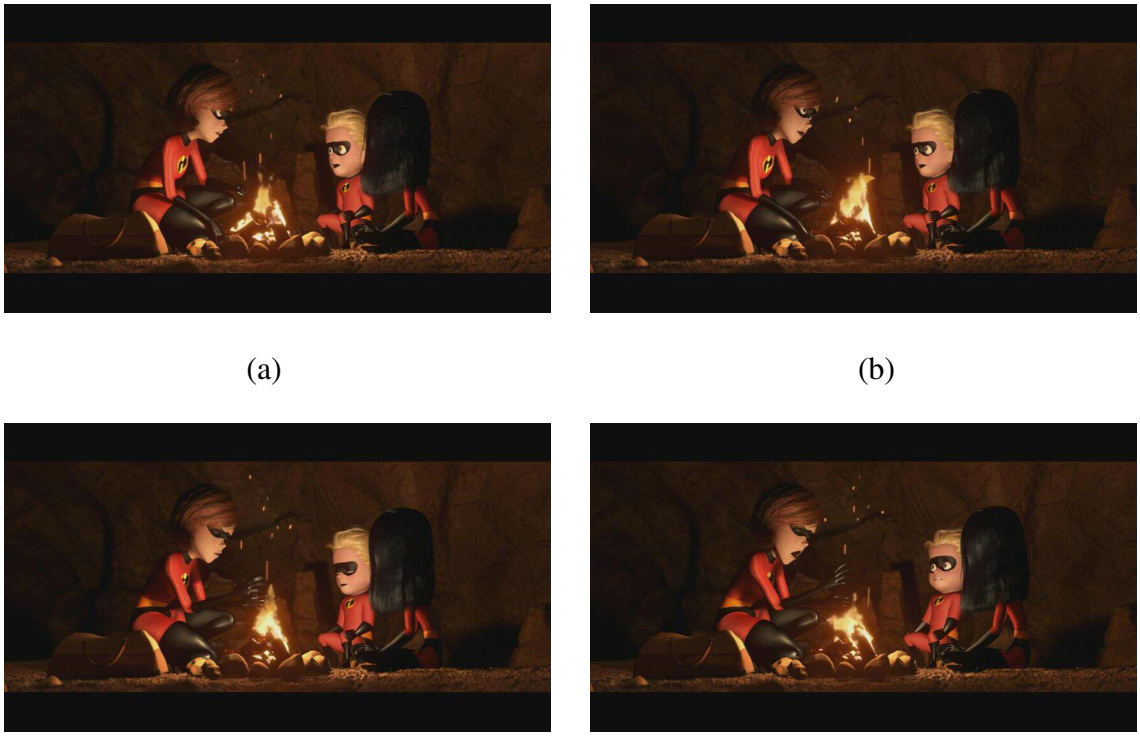
Yoshida and Nishita focused their research on modeling smoke [Yoshida and Nishita, 2000]. They accomplish this by using a primitive that they termed as a MetaBall. Their research includes modeling swirling smoke and the movement of that smoke around arbitrary boundary objects.

1.2 Embers are Missing

The above examples are just some of the research in the realm of computer-generated fire. Among all such research, there seems to be a common aspect missing. None of them address the concept of floating embers, a necessary part of any fire. An ember, in this context, refers to any piece of burning material that breaks free and floats away on the rising gasses of a fire. In common vernacular, it is a spark off a fire.

There have been some attempts to simulate embers. One recent attempt, shown in Figure 4, is the campfire scene in Pixar's animated feature 'The Incredibles' [Incredibles, 2004]. In this researcher's opinion, the ember simulation was effective, but unrealistic. The embers rise from the base of the fire at an unnaturally high rate of speed. Then once a certain height is reached, the embers begin a slow erratic random walk until they are extinguished. There must be a more realistic method.

After researching various methods, three stood out as showing promise: the Lattice Boltzman Model, 3D Fractal Trees and Particle Systems. Each of them presents a unique approach for creating a realistic representation of an ember's path.



(a) (b)

(c) (d)

Figure 4a-d: Screenshots of Embers from 'The Incredibles' [Incredibles, 2004]

1.3 Lattice Boltzmann Models

The first method selected for consideration is the Lattice Boltzmann Model. Research found that this model excels at simulating fluid flow and fluid based systems. Wei, Zhao, et al. have used this method to simulate bubbles and feathers blowing in a breeze, the flames of a fire, and smoke rising from a fire [Wei et al., 2002][Wei et al., 2003][Wei et al., 2004][Zhao et al., 2003]. At its most basic, the heated rising air from a fire is nothing more than a moving fluid. It is a natural assumption that this method would be very effective at simulating the actual path of an ember.

1.4 3D Fractal Trees

The second method selected for consideration is 3D fractal trees. Fractal trees have been used to model various plant life, such as ferns, bushes, and trees. They have also been used to model items such as streams, rivers, blood vessels, and magma from volcanic eruptions. While observing a fire from a fixed point of view over a period of time the pattern of ember paths are determined by a 'controlled randomness'. As an ember rises, at each moment in time there are particular paths it may take, and certain paths it takes more frequently. The culminations of all possible paths over time resemble a construct very similar to fractal trees. For this implementation the process will be reversed and start with a fractal tree that an ember can randomly traverse. Based upon observations of real embers, this method should be very effective at simulating the look of those embers.

1.5 Particle system via third-order polynomial paths

Particle systems are effective at modeling objects using small primitives with defined rules governing their behavior. While observing individual embers of a real campfire, it appears that the embers repeatedly followed the paths of various 3rd order polynomials. It is a natural method to use a particle system to simulate the embers.

Chapter II of this thesis provides more detail about each of the methods selected for this research. It reviews background information as well as implementation details.

Chapter III describes the software that was written to implement each of the methods.

Chapter IV describes the methodology that was used to implement, test, and compare each method. It also describes the results of the comparisons.

Chapter V presents the conclusions draw from the comparisons of the three methods in chapter III.

Chapter VI suggests directions for future studies related to this research.

Chapter II

2. Literature Review

2.1 Lattice Boltzmann Model

The Lattice Boltzmann Model (LBM) is a very effective method for simulating fluid-based systems. In the realm of computer fire simulations, Wei, Zhao et al have used LBM to generate flames as well as smoke [Wei et al., 2002][Wei et al., 2004][Zhao et al., 2003].

2.1.1 Review of LGA

LBM evolved from cellular automata. One type of cellular automata that is used in the simulation of fluid dynamics is called *lattice gas automata*, or LGA [Wei et al., 2004]. A general understanding of LGA assists in explaining LBM. In LGA, individual macroscopic particles exist within a discrete lattice and move with discrete time steps.

2.1.2 LGA Lattice

The *lattice* is implemented as a collection of *nodes* and *links* between those nodes. All of the links between nodes have an equal length. Thus the distance between a node and any of its *neighbors* is constant and considered one unit of measurement. Particles exist within the node and always have unit velocity in the direction of one of the neighboring nodes. This means that during a single time step each particle will move into the next

node. It never will end a time step on the link between nodes. The one exception to the unit velocity requirement is particles that are at rest. Since these particles have a velocity of zero, they also end each step within a node. There are additional requirements that each node can contain no more than one particle at rest and no more than one particle traveling to each of its neighbors. Figure 5 illustrates this requirement via specific examples. As shown in Figure 5a, node A contains no particles while node B contains a single particle that is at rest; node B could not contain two particles at rest. In Figure 5b, there is a particle at rest in node A and another particle in node B with velocity, indicated via an arrow, along the link toward node A; there cannot exist two particles traveling from node B to node A. Figure 5c shows a case that may not be readily apparent. The requirement is that at most one particle can exist on the link exiting *from* a node. Thus it is possible to have a single particle traveling from node A to node B, as well as a particle traveling from node B to node A at the same time. As these particles travel from A to B and B to A, they will pass each other without interacting in any way. As discussed below, collisions only happen within a node, not on the links between nodes. Finally, Figure 5d represents the case where a node A has a particle at rest, a particle traveling to node B, and a particle traveling to node C; this is valid, but difficult to depict graphically as the moving particle obscures the particle at rest.

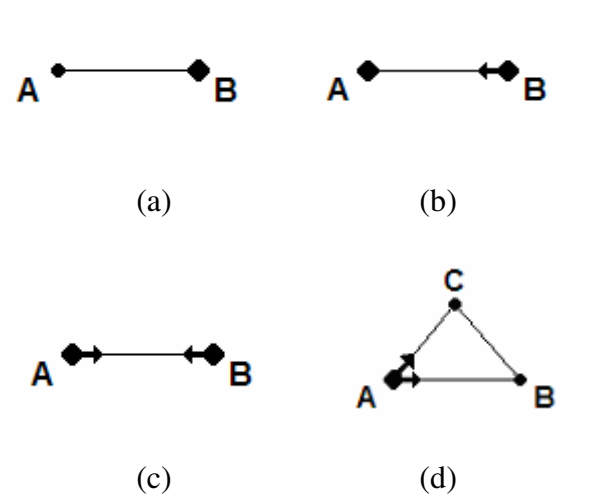


Figure 5a-d: Examples of LGA Nodes and Particles

Taking the above into consideration, the presence or absence of a particle traveling out of a node in the direction of a specific link can be indicated via a single bit value. A bit value of true indicates a particle exists in that node traveling in the direction of that link. Thus, each node contains a number of bits equal to the number of links to other nodes, plus a bit to indicate a particle at rest.

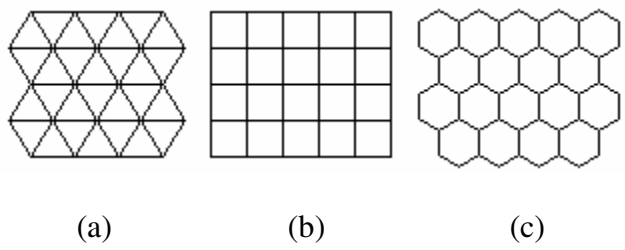


Figure 6a-c: Possible LGA Lattice Structure

The overall structure of the lattice can vary as long as the restriction that all nodes be equidistant from their neighbors is met. Thus the LGA lattice could be two-dimensional

or three-dimensional. Some possible lattice designs are shown in Figure 6. These examples show that the LGA lattices could be triangular, rectangular or hexagonal in design. Any regular shape that can tile a plane and is equilateral between nodes would suffice. Not all lattice geometries produce desirable results. The hexagonal lattice shown in Figure 6c is an example. This lattice is less dense than a triangular lattice such as that shown in Figure 6a. There are fewer nodes within the same area covered by the lattice. Having fewer nodes directly affects the precision and amount of detail that the lattice can provide. Additionally, in lattices like the hexagonal design, the velocity direction of a particle cannot remain constant. In Figure 6a and 6b, a particle can travel from one node to the next and then continue traveling in the same direction. In Figure 6c, after a particle travels from one node to the next, it must alter its velocity direction, as there is no link continuing in that direction. To continue moving the particle would have to choose a new path, which introduces an element of randomness into the design. Based on the research of [Frisch et al., 1986], the 2D triangular lattice, Figure 6a, is used because it ensures macroscopic isotropy in the gaseous behavior. This means that the visible gaseous behavior will be the same regardless of how the lattice is oriented or the direction the gas is flowing within the lattice.

2.1.3 LGA Propagation and Collision

Now that the structure and parts of LGA have been discussed, the mechanics of how LGA functions can be more clearly understood. LGA is an iterative process. Each time step consists of the same two-step process until the simulation is stopped. The first of the two steps is called *propagation*. During the propagation step each particle moves from one lattice node to the next based upon its discrete velocity. As discussed before, since a

particle has unit velocity it will always move to a neighboring node; it will never stop between nodes. The particle retains its velocity direction. The propagation step occurs for every particle simultaneously. Particles that are traveling to other nodes pass each other on a link without interaction. However, as particles enter a node at the same time they collide. This is the second of the two steps and is called *collision*. When two particles collide, their velocity vectors deflect them away from each other, thus causing them to move down different links in the lattice in the next propagation step. The collision must satisfy the laws of conservation of mass and momentum. This means that the collision can neither create nor destroy particles, and the net velocity of all particles at a node must be equivalent in the pre and post collision states. For example, if two particles with opposite velocity collide, they cannot both come to rest; instead, they must continue moving in new directions. Additionally, if a particle collides with another particle that is at rest, then the moving particle must stop and the non-moving particle must begin moving. As with propagation, all particle collisions occur simultaneously. After the collision step, the two-step process repeats in the next time step with a new propagation step.

Figure 7 (developed from examples provided by the Section Computational Science research groups in the Computing, System Architecture and Programming Laboratory of the University of Amsterdam) demonstrate this process in a visual manner [SCS, 2004]. Each line represents the link between two nodes as in Figure 5c. A node exists at the intersection of each line. A particle is displayed as a dot at a node. An arrow indicates the velocity vector of the particle. This velocity is always in the direction of one of the

lattice links. A particle without an arrow indicates a particle that is at rest and has zero velocity. Figure 7a is the initial state before the first propagation step of this example.

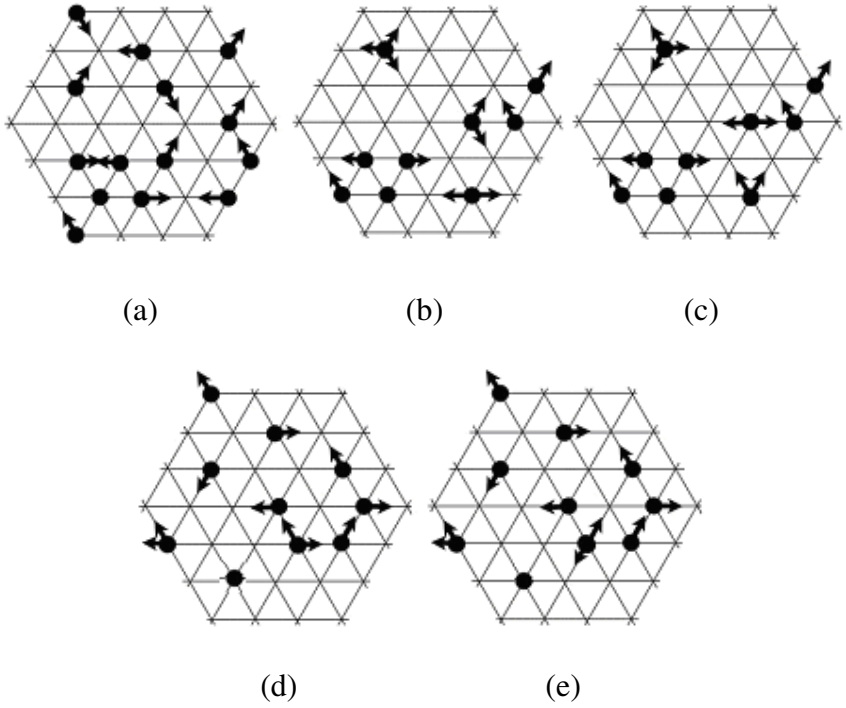


Figure 7a-e: Sample LGA Propagation and Collision Process [SCS, 2004]

Figure 7b represents the lattice after it has gone through one propagation step. All the particles moved according to their velocity direction. In some cases this results in multiple particles entering the same node. In other cases particles move off the edge of the lattice and, in this example, are no longer part of the simulation.

Figure 7c represents the lattice after it has gone through one collision step. All the particles are in the same location as in Figure 7b, but their velocity vectors have been updated. The number of particles that collide at a single node and the directions from

which they entered the node determines how they 'bounce' off each other and will exit the node in one of the discrete directions. This completes a single time step.

Figures 7d and 7e represent the next time step. Figure 7d represents the lattice after it has gone through another propagation step with each particle moving in the direction of its velocity vector. Figure 7e represents the lattice after its second collision step. The lattice continues through the process of propagation/collision until the simulation is stopped.

2.1.4 Problems with LGA

A major drawback to LGA is that it inherently includes statistical noise in the simulation [SCS, 2004][Wei et al., 2004]. This noise is inherent in the discrete design of LGA and prevents a smooth fluid flow. This effect is very similar to the way a curve appears jagged when plotted onto a discrete grid, e.g. pixels on a monitor. The discrete resolution of the lattice is not enough to represent smooth fluid flow. One way to reduce this effect and provide smoother results is by choosing a denser lattice and a smaller time step. For the examples in Figure7, this could mean inserting additional nodes into the center of each equilateral triangle and reducing the time step from 1.0 second to 0.5 second. This is similar to using greater pixel density on a monitor to smooth out the noise (jaggedness) of the curve. This process of adding nodes and reducing time step length increases resolution and leads to smoother results, but also increases computation time and memory requirements for the simulation. If the model requires returning to the original lattice density and time step magnitude, one can average the results of the denser lattice. Each macroscopic node actually would be the average of several microscopic nodes. Each larger time-step would be the average of the values for the smaller time-steps that were

added. Instead of a single macroscopic particle on a lattice, the model would now deal with the average value of several microscopic particles that exist in the microscopic nodes. A single bit would no longer suffice to represent a particle; a real value would be used instead. This averaged value is called a *particle density*, since it represents how dense the averaged area is with particles. This is very similar to the process of anti-aliasing or super-sampling to produce a smoother curve on a monitor. Each screen pixel is divided into multiple smaller sub-pixels to increase the resolution of the curve. Then the values of the sub-pixels must be averaged back into the original pixel size and resolution.

2.1.5 LGA Evolves into LBM

The process of super-sampling is very time and resource intensive. The *Lattice Boltzmann Model*, LBM, was developed to accomplish this in a more efficient manner [Chen and Doolean, 1998]. The specifics will be discussed below, but an overview of the differences between LGA and LBM follow. Instead of going through the process of super-sampling particles in a LGA to obtain a particle density, LBM works directly with the particle density. Thus, instead of a Boolean value to represent whether a particle exists in a node link, LBM uses a real number to indicate how many particles are traveling along the node links. LBM is still an iterative 2-step process of collision and propagation, however these are modified since the particles are no longer discrete.

2.1.6 LBM Lattice

The LBM lattice still can be implemented in 2D or 3D with the same structures possible in an LGA lattice. The implementation of LBM for this comparison will be a 3D rectangular lattice. The rectangular shape is easier to represent in memory as an array of arrays of arrays of node structures, i.e. a three-dimensional array. Experimentation for this research shows that a lattice size of 10 x 10 x 20 gives a good balance between visual accuracy and computational costs.

A lattice node exists at the center of each *cell* in the 3D grid. Each node in the lattice has a discrete number of paths leading to the nodes in neighboring cells. Each cell is surrounded and touched by a total of 26 other cells. This defines 26 possible paths out of a node to its neighboring nodes. For LBM there is an additional path to be considered. It is a reflexive path that leads from a cell back to itself. This represents the possibility of a particle being at rest and staying within the same cell. Thus, a node can be a neighbor to itself, which gives a total of 27 possible links along which a particle density may travel to during propagation and 27 possible neighbors for a cell.

In LGA all neighboring nodes are equidistant from each other. In LBM, the 27 neighboring nodes are not all equidistant. This causes an issue because propagation requires each packet density to travel completely into the next node. This is addressed by accomplishing two things. First, the main lattice is divided into four *sub-lattices*, enumerated via the use of the variable q. Each of these sub-lattices will be a grouping of the links that have the same vector magnitude to the neighboring node. Second, the

magnitude of the velocity for the packet densities is modified to match the distance to the neighboring node. By doing these two things the packet densities within each sub-lattice are able to travel completely from one node to the next during a single time step. The sub-lattice $q = 0$, shown in Figure 8a, represents the particle densities that are at rest. The velocity vector in this sub-lattice has a magnitude of zero. The sub-lattice $q = 1$, shown in Figure 8b, represents the links to the six nearest neighboring nodes. The velocity vectors in this sub-lattice have a magnitude of one. The sub-lattice $q = 2$, shown in Figure 8c, links to the twelve second-nearest neighboring nodes. The velocity vectors in this sub-lattice have a magnitude of $\sqrt{2}$. Finally, the sub-lattice $q = 3$, shown in Figure 8d, represents the links to the eight farthest neighbors. The velocity vectors in this sub-lattice have a magnitude of $\sqrt{3}$.

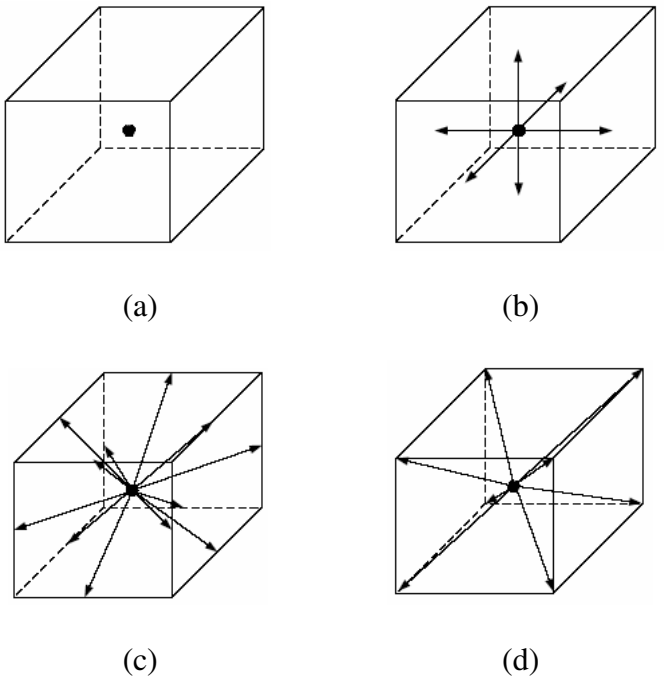


Figure 8 a-d: LBM Sub-Lattices for 3D Rectangular Lattice [Wei et al., 2004]

It is not required that an implementation of LBM use all four of the sub-lattices. They can be combined in different ways to build different 3D lattices. One of the most common is to combine sub-lattices zero, one and two. This is referred to as D3Q19, indicating three dimensions and nineteen links/neighbors for each node. Based on the research of [Wei et al., 2004], this model is a good balance between realism and computational costs. It is the lattice chosen for this research.

2.1.7 LBM Collision and Propagation

Just like LGA, LBM is an iterative method where each time step consists of propagation and collision. The exact order of these sub-steps is not important as long as they follow each other repetitively, so in LBM the collision step is usually performed before the propagation step. In LBM these steps are represented symbolically using an equation notation. These equations appear to be standard across all LBM research. The basis for these equations if called the *particle density distribution function*, represented by $f_{qi}(x,t)$. The dual variables *qi* serve to identify a specific link within a sub-lattice, where *q* identifies the sub-lattice and *i* enumerate the links within that sub-lattice. For example, if $q = 1$ and $i = 0$, this might indicate the link in Figure 8b which leads directly up. The variable *x* indicates the location of a specific node within the lattice. In the case where the lattice is represented via a three dimensional array, *x* would be the triplet to identify a specific node/cell within the array. The variable *t* indicates the time-step. This is used within the formulaic notation to indicate when the simulation moves from one time step to the next, but is not implemented as an actual variable during implementation. Thus

$f_{qi}(x,t)$ represents the real-value of the packet density on path i of the sub-lattice q, leading out of node x at time t. The initial state, $f_{qi}(x,0)$ is defined later in this paper.

$$f^{new}_{qi}(x,t) - f_{qi}(x,t) = \Omega_{qi}(x,t)$$

Equation 1: LBM Collision formula

$$f_{qi}(x+e_{qi},t+1) = f^{new}_{qi}(x,t)$$

Equation 2: LBM Propagation formula

The standard form for the collision and propagation formulas are given as Equations 1 and 2. Some discussion is required to relate this formulaic representation to an actual implementation. As in LGA, collisions only occur within nodes, not on the links between nodes. Equation 1 is the collision formula and uses the particle density distribution function, $f_{qi}(x,t)$, to represent the pre-collision state of the particle density distribution. Equation 1 also introduces two new functions. The first is labeled $f^{new}_{qi}(x,t)$ and represents the post-collision state of the particle density distribution. The second new function is labeled $\Omega_{qi}(x,t)$ and is called the *collision operator*. In both of the new equations the variables q, i, x, and t are the same as in the particle density distribution function. The collision formula says that there exists a collision operator that represents the difference between the pre-collision and post-collision states of the particle distribution. The exact implementation of the collision operator has been an area of research and the implemenation used for this research will be discussed later.

Equation 2 is the propagation formula. It uses the post-collision version of the particle density distribution function, $f^{new}_{qi}(x,t)$, introduced in the collision formula. This represents the propagation that occurs after collison. The end result of the propagation formula is that each particle density travels along it's post-collision link to the next neighboring node. This is indicated by the term $f_{qi}(x+e_{qi},t+1)$. The term $t+1$ indicates that these values are used in the next time step of the simulation. The variable e_{qi} is the velocity vector along the link i in the sub-lattice q. The magnitudes of the velocity vector were discussed above and are defined by the sub-lattice. The direction of the velocity vector is defined by the qi pairing. For example, if $q = 1$ and $i = 0$ represents the link going up, then e_{qi} would be the vector (0,0,1). The exact values for e_{qi} are dependent on the implementation of the 3D lattice. While the formulaic value of e_{qi} must match the velocity required to reach the next node in a single time-step, for an implementation the most important thing is that the term $x+e_{qi}$ represents the location of the post-propagation node within the lattice. Thus, if the lattice is implemented as a three-dimensional array then e_{qi} is the offset triplet to move from the current node to the appropriate neighboring node.

To review, during a time step, t, the particle density on every link in every node goes through the collision operator, $\Omega_{qi.}$ This generates an interim particle density distribution function, $f^{new}_{qi}(x,t)$. The interim particle density distribution function then goes through the propagation step as every particle density steps to the appropriate cell and updates the original particle density distribution function, $f_{qi}(x,t)$, with the values for the next time step.

2.1.8 LBM Collision Operator

The collision operator, $\Omega_{qi,}$ must be selected to satisfy local conservation of mass and momentum. As in LGA, this means that particle density can be neither created nor destroyed. It only can be moved from one link to another. It also means that the overall magnitude of the sum of velocities must remain the same. Based on the research of [Chen and Doolean, 1998], the collision operator can be represented by Equation 3.

$$\Omega_{qi}(x,t) = (-1/\tau)*(f_{qi}(x,t) - f^{eq}_{qi}(\rho,\mu))$$

Equation 3: LBM Collision Operator

This implementation of the collision operator uses the pre-collision state of the particle density distribution function, $f_{qi}(x,t)$, and introduces the use of several new variables and a new function. The new function is called the *equilibrium distribution function* and is represented as , $f^{eq}_{qi}(\rho,\mu)$. The first new variable ρ is the macroscopic density of the current cell and will be defined later. The second new variable μ is the velocity of the current cell and will be defined later. The idea behind this function is that all systems tend toward a steady state or a fully relaxed state over time. A rolling car will eventually come to a stop. Water disturbed by a splash will eventually become calm. A swinging pendulum will eventually come to a stop. These are all examples of a system moving toward its steady state. The equilibrium distribution function calculates what the steady state is for a particular node and its links. (The definition of this function will be discussed later.) The collision operator then calculates the difference between the current state and the equilibrium state. However, since it is undesirable to reach the equilibrium

state in a single time step, a new variable is introduced. The *time relaxation factor*, τ, controls how quickly the node approaches its equilibrium state. The larger the value of τ, the more time steps that are required to reach equilibrium.

The value for the time relaxation factor is calculated based on the requirements for the fluid system being modeled. Every fluid has a determinable *viscosity*. The viscosity of a liquid is a measure of its resistance to flow or change when under force. Honey is more viscous than water, because it resists change more. The relationship between viscosity and the time relaxation factor is given in Equation 4. The viscosity of air at sea level at 20° C is 1.511×10^{-5} N-s/m^2 [Toolbox, 2005]. For the purposes of this research this value is sufficient for the viscosity and will yield a time relaxation factor that is very close to one-half.

$$v = (1/3) * (\tau - 1/2)$$

Equation 4: Viscosity as a function of the Time Relaxation Factor

One of the missing pieces for the collision operator is the macroscopic density or mass of the particle density within a lattice node. As shown in Equation 5, this is the sum of the fluid densities located on every link for that node. The macroscopic density is calculated as needed for every node as it goes through its collision sub-step.

$$\rho = \Sigma_{qi}(f_{qi}(x,t))$$

Equation 5: Macroscopic Density of an LBM Node

Another missing piece for the collision operator is the velocity of the particle density within a lattice node. As shown in Equation 6, this is calculated from the macroscopic density, the packet density distribution function, and the velocity vector for that sub-lattice link. Within a lattice node, some particles will be flowing in one direction and others will be flowing in a different direction. The end result of Equation 6 is a vector that represents the overall trend of movement within that node.

$$\mu = (1/\rho) * \Sigma_{qi} (f_{qi}(x,t) * e_{qi})$$

Equation 6: Velocity of Particle Density of an LBM Node

The final piece needed to use the collision operator is the equilibrium function. Previous research by [Muder,1995] shows that $f^{eq}_{qi}(\rho,\mu)$ can be represented by the linear formula given in Equation 7. This function makes use of the macroscopic density, velocity and velocity vector as defined above. It also introduces four new constant coefficients, A_q through D_q. The values for these coefficients are dependent on the specific lattice geometry selected. These values must be selected to ensure the laws of conservation of mass and momentum within the lattice cell. The values given by [Muders, 1995] that satisfy this constraint for the D3Q19 lattice structure are shown in Figure 9.

$$f^{eq}_{qi}(\rho, \mu) = \rho * (A_q + B_q * (e_{qi} \bullet \mu) + C_q * (e_{qi} \bullet \mu)^2 + D_q * \mu^2)$$

Equation 7: LBM Equilibrium Distribution

	Sub-lattice 0	Sub-lattice 1	Sub-lattice 2
A_q	1/3	1/18	1/36
B_q	0	1/6	1/12
C_q	0	1/4	1/8
D_q	-1/2	-1/12	-1/24

Figure 9: LBM Equilibrium Distribution Coefficients for D3Q19 [Muders, 1995]

2.1.9 LBM Initial State

The equilibrium distribution, Equation 7, also determines the initial state of every node link in the 3D lattice. The initial velocity, μ, and mass, ρ, are dependant on the specific properties for the fluid system being modeled. These values are used to evaluate Equation 7. For this research the initial velocity will be 0.0, indicating no initial movement or wind. The mass of air at sea level at 20° C is 1.205 kg/m^3 and will be sufficient for the purpose of this research [Toolbox, 2005].

2.1.10 LBM Boundary Conditions

One final issue of the propagation step that needs to be discussed is how to handle the boundary conditions for links that lead out of the current lattice. Previous research has identified various ways to handle these boundary conditions [Wei et al., 2004]. While there are others, four common ways to handle these distribution paths are:

1. Let them flow out of the simulation completely, never to return.
2. Have them re-enter the cell they are leaving along the same path in the opposite direction. (i.e. bounce-back)
3. Have them re-enter the simulation on the opposite side of the lattice.
4. Have them re-enter a neighboring cell along a path that is a mirror image of the current path. (i.e. bounce-forward)

For this research the first option will be used on the top of the lattice to represent the area of space directly above a campfire. The four sides of the lattice will be implemented using the second option. Finally, the bottom of the lattice will use the third option. The fourth boundary type will not be used in this research.

2.1.11 LBM Fluid and Ember Movement

Everything discussed so far is a way to model the movement of fluid within the 3D lattice, but it does not actually create movement. To inject movement into the lattice some of the lattice cells along the bottom of the simulation will have fixed values for velocity and mass to simulate the rising heat of the fire and thus causing currents in the heated air. During the collision step, these fixed values are fed into the equation for the equilibrium function instead of the calculated value for those lattice cells.

The frequency and positioning of these cells can create an area in which minimal fluid is flowing. This is similar to the calm area of water that exists immediately behind a waterfall. This effect can lead to particles that will hover around this area until they flow

into the main current of the simulation. As discussed before, the base of the lattice for the implementation for this research will be a 10 x 10 grid. Figure 10 indicates which of those nodes will be used to inject movement into the lattice. Experimentation showed that this configuration gave a nice visual effect.

The fixed values selected for the velocity and mass of these injection nodes must be selected carefully. As stated in the research by [Wei et al., 2004] and supported by experimentation in this research, if the velocity and mass vary too much from that of the equilibrium distribution then it will lead to numerical instability and failure of the simulation, as the packet densities will tend towards infinity. Experimentation for this research showed that a mass of 2 and a velocity vector of (0,0,0.1) provided sufficient results.

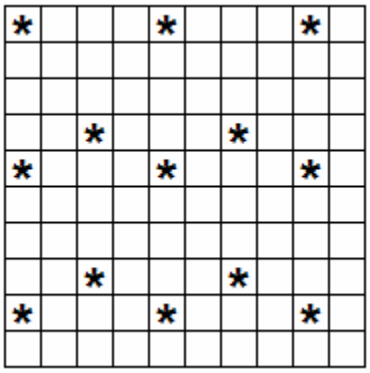


Figure 10: Sample Placement of Nodes in LBM Lattice to inject movement

All the pieces are defined to simulate fluid in an LBM, however the effect of this fluid on embers has not been discussed. The research of [Wei et al., 2004] provides a method to accomplish this. The embers exist in 3D Cartesian space. This easily can be mapped to the 3D array used to implement the lattice by the use of a mathematical rounding

function. For example, if the ember exists at the Cartesian point (13.2, 14.6, 18.4) it is mapped to exist within the 3D array element (13, 15, 18) and thus be affected by that lattice node. During each time step, the position of each ember will be modified by the average velocity vector, Equation 6, for the node cell the ember is currently in. This velocity vector can be added to the embers current position resulting in movement. Embers will be created at random times with random initial positions at the bottom of the lattice. An ember that moves off the edge of the lattice will be extinguished.

For simplicity in this research an ember will be assumed to have no mass and no effect on the fluid dynamics within the lattice. This simplification is based on previous research by [Wei et al., 2004] where items as large as feathers were assumed to have no mass with respect to the lattice fluid flow. If an object with a large mass, such as a large balloon, were to be modeled it would have to be tied back to the LBM lattice where it would provide drag on the fluid velocity and act as an internal boundary for the fluid to flow around. This increases the complexity of LBM calculation and is not appropriate for this research.

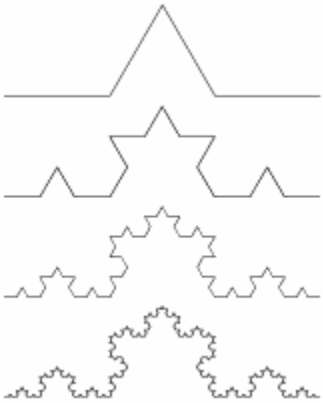
2.2 3D Fractal Tree

Fractal graphics typically are generated via a relatively simple recursive function, but can generate results that are very complex. They exhibit the concept of self-similarity, where a smaller portion of the shape resembles the whole.

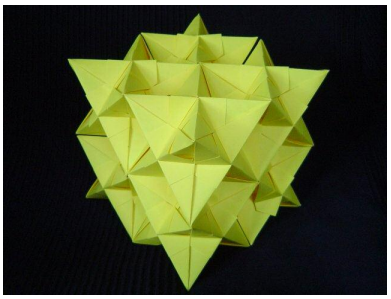
One way of generating fractals is by a method called the *Iterated Function System*, IFS. An IFS fractal is made up of several copies of itself. Each copy is sometimes slightly modified by scale or position. Thus, the concept is to start with a shape or point. Then during a single iteration, a portion of the object is replaced or modified with a copy of itself that has been modified via a fixed transformation or rule. In some IFS there are multiples of these transformations and a randomly selected one is implemented during each iteration [IFS 2006].

2.2.1 Koch Curve

A classic example of a structured fractal, shown in Figure 11a, is the Koch Curve. The basis for this fractal is a straight-line segment. In the first iteration, this line segment is modified to be the top line in Figure 11a. Two legs of an equilateral triangle replace the middle third of the line segment. During the second iteration, the same transformation is performed to all four-line segments that now exist. The end result of this replacement is shown as the second line in Figure 11a. The fractal now contains four smaller versions of the previous step and a total of sixteen equal-length line segments. During the next iteration, each of these sixteen line segments would be modified again by the same transformation. This process of iterations can be continued as many times as is desired. The first four steps of this process are shown in Figure 11a [Loy, 2002].



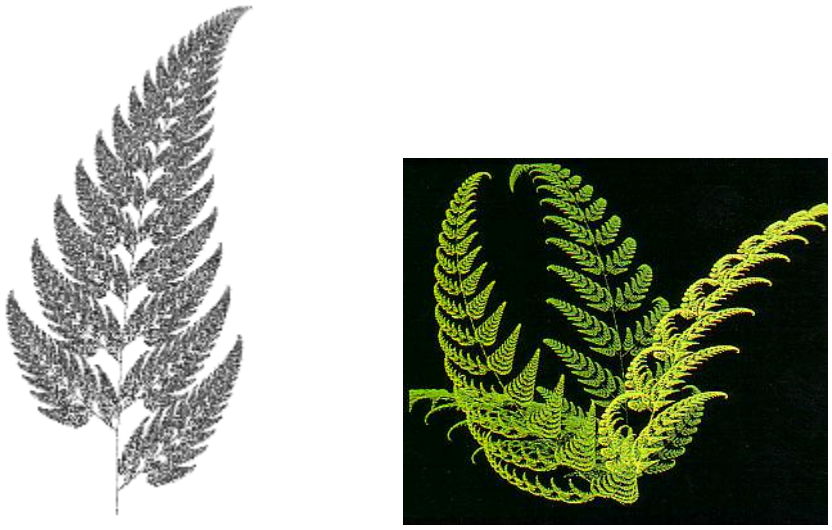
(a)



(b)

Figure 11a-b: Examples of Koch Curve Fractals [Loy, 2002][Kosmulski, 2002]

Fractals are not limited to dealing with line segments. Any fractal can also be implemented using a three-dimensional object as their generator. Figure 11b displays the Koch Curve if it used a regular tetrahedron as its basis [Kosmulski, 2002]. The iterative transformations are slightly different because of the third dimension. In a single iteration, each equilateral triangular surface of the fractal has a new smaller regular tetrahedron placed on it. The midpoints of each side of the equilateral triangle form the three points for the base of the new tetrahedron. In the basis there are four surfaces and thus four new tetrahedrons added to the fractal. After the first iteration, there are twenty-four equilateral triangular surfaces and twenty-four new smaller tetrahedrons would be added. Figure 11b has gone through only two iterations.



(a) (b)

Figure 12a-b: Examples of Barnsley's Fern Fractals [Nicholls, 1998][Fern, 2006]

2.2.2 Barnsley's Fern

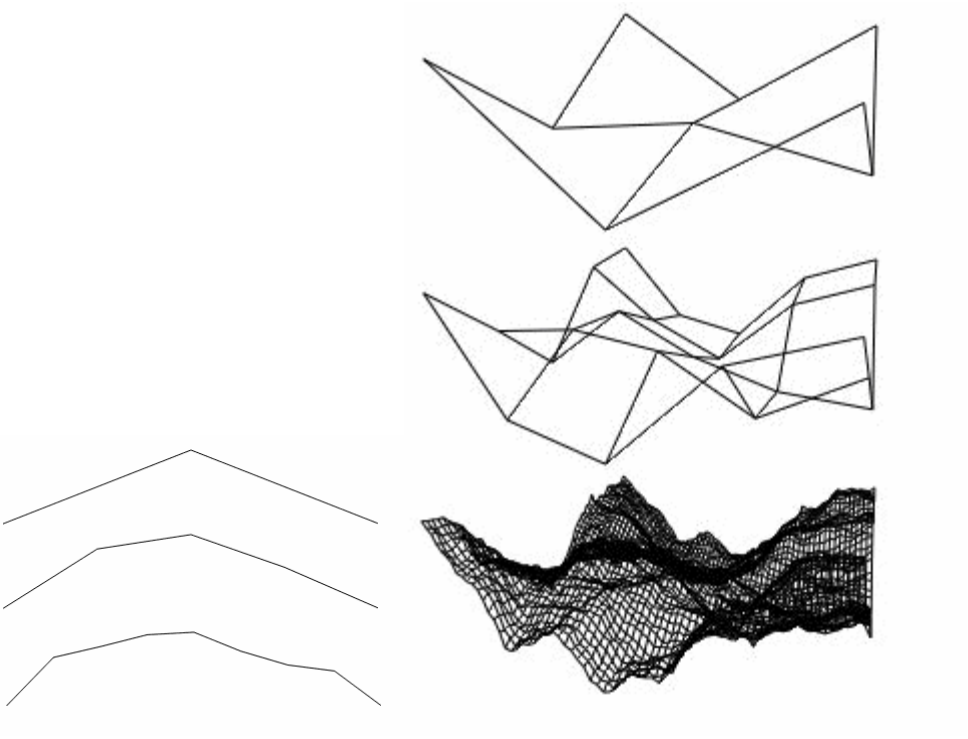
Another classic example of a structured fractal, shown in Figure 12a is Barnsley's Fern [Nicholls, 1998]. Each leaf of the fern exhibits self-similarity with the entire structure. This is also an IFS fractal, but is typically generated in a different way. This approach is referred to as a *chaos game*. The basis for this fractal is a single point in a 2D Cartesian plane. As given in Figure 13, there are four transformation rules. Each of these transformation rules modifies the positioning of the point in a defined manner. During a single iteration, one of the transformation rules is randomly selected based on the weights in Figure 13. After the point's position is plotted onto the plane, it is modified by the selected transformation rule and moved to a new position on the 2D plane. For the first several iterations no pattern will be visible. The fractal will appear to be a random

collection of points. After several more iterations boundaries will begin to appear. Finally the shape shown in Figure 12a will begin to take form.

Rule #	Prob %	Transformation Rule
1	1%	$X' = 0$ $Y' = 0.16 * Y$
2	7%	$X' = 0.2 * X - 0.26 * Y$ $Y' = 0.23 * X + 0.22 * Y + 1.6$
3	7%	$X' = 0.15 * X + 0.28 * Y$ $Y' = 0.26 * X + 0.24 * Y + 0.44$
4	85%	$X' = 0.85 * X + 0.04 * Y$ $Y' = 0.04 * X + 0.85 * Y + 1.6$

Figure 13: Transformation Rules for Barnsley's Fern [IFS, 2006]

Figure 12b shows a variation of a fern that exhibits another way two-dimensional fractals can be implemented in three dimensions [Fern, 2006]. In this example each frond curves gently into the third dimension.



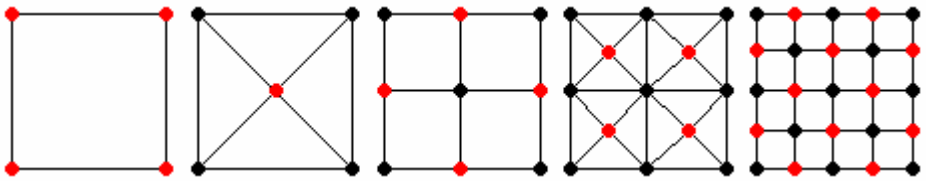
(a) (b)

Figure 14a-b: Examples of Midpoint Displacement Fractals [Martz, 1997]

2.2.3 Terrain Generating Fractals

Not all fractals are structured. Non-structured fractals are constructed using a random factor directly in their transformation. A classic example, shown in Figure 14a is terrain generation via midpoint displacement [Martz, 1997]. Similar to the Koch Curve in Figure 11, the basis for this fractal is a line segment. In the first iteration, the segment is divided into two pieces at its midpoint. The position of this midpoint is then shifted by a limited random amount. In the second iteration, there are two line segments. Each of those line segments is divided into two pieces at their midpoints and the positions of those midpoints are modified by limited random amounts. This process continues as long as desired. Figure 14a shows the first 3 iterations of a sample fractal. The magnitude of

the allowable range for the random value for the midpoint displacement affects how smooth or jagged the resulting terrain will be. Many implementations start with a large allowable range and decrease it during each iteration.



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

Figure 15a-e: Examples of Meshes for Diamond-Square Algorithm [Martz, 1997]

Figure 14b shows that non-structured fractals also can be generated in three dimensions. The *diamond-square algorithm* works on a plane in much the same way that midpoint displacement works on a line [Martz, 1997]. This fractal mimics natural surfaces well. The basis for this fractal is a quadrilateral in 3D Cartesian space where the four corners of the quadrilateral are coplanar. Shown in Figure 15a, this quadrilateral consists of four vertexes connected by four line segments. The iterations in this method consist of two steps. In the first step of the first iteration, a new vertex is added at the midpoint of the quadrilateral. The height of this new vertex is the average of the heights of the four vertexes in its bounding quadrilateral plus a limited random value. If this new vertex were connected to the four corners of the quadrilateral that contains it, then it would form four triangles. As shown in Figure 15b, these triangles are actually one half of a diamond. In this context a diamond can also be thought of as a square rotated 45 degrees. This first step is called the *diamond step* because it creates these diamond shapes. In the second

step of the first iteration a new vertex is added to the midpoint of each of the four sides of the quadrilateral. The height of each new vertex is the average of the three vertexes that make up the half diamond it exists in plus a limited random value. Shown in Figure 15c, if the vertex added during the diamond step is connected to each of these four new vertexes then it creates a rectangular mesh of nine vertexes in three rows and three columns. The top image in Figure 14b is an example of what a sample fractal might look like at this point. This step turns the diamonds back into squares and is called the *square step*. The rectangular mesh of 9 vertexes can also be viewed as four quadrilaterals, which can share edges. In the diamond step of the second iteration, a vertex is added to the center of each of these quadrilaterals. As before, the height of each of the four new vertexes is the average of the heights of the four vertexes in its containing quadrilateral plus a limited random value. As shown in Figure 15d, this creates eight of the half diamond shapes and four full diamond shapes. In the square step of the second iteration, a new vertex is added to the midpoint of each of the twelve segments of the four quadrilaterals. The height of each of these twelve new vertexes is the average of the vertexes that make up the diamond that contains it plus a limited random value. There are three vertexes that make up a half diamond. There are four vertexes that make up a full diamond. Shown in Figure 15e, connecting the vertexes added in this iteration completes a quadrilateral mesh of 25 vertexes. The middle image in Figure 14b shows what a sample fractal might look like after this second iteration. This process of adding vertexes is continued as long as desired. The bottom image in Figure 14b shows a sample fractal after several more iterations.

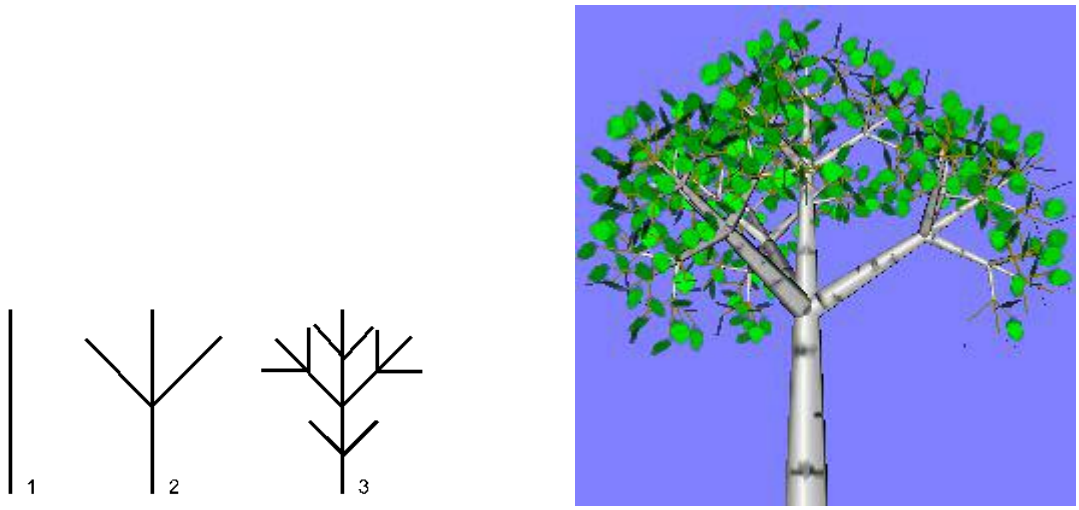
2.2.4 IFS Fractal from a Different Point of View

The fern fractal in Figure 12a could be viewed as a collection of line segments. A single line segment forms the base of the fern branch and ends where the lowest frond joins the branch. This is an intersection of three line segments. The first is the base of the branch already discussed. The second is the base of the frond and leads off to the side. The third continues to form the next section of the fern branch. The end of each of these line segments ends in an intersection of line segments. This process continues until the tip of each frond leaf is reached, where the line segment ends with no intersection. Imagine an ant starting at the bottom of the line segment that forms the base of the main fern branch. It could travel along that line segment until it reached the intersection at the end of the line segment. Once there it would have to use a random weighted value to select a different line segment to traverse. It would then continue traveling down this new line segment until it reached the next intersection. This process would repeat until the ant reached the end of a line segment with no intersection at its end. This is the approach that will be taken by this research. Instead of an ant, an ember will traverse the line segments that make up a fractal. The possible paths that the ember could take will be defined by the structure of the fractal. The actual path taken will be based on random decisions at each intersection.

2.2.5 Tree Fractal

The implementation for this research will be a structured fractal in three dimensions. The fractal selected for this implementation is commonly used to represent the trunk and branches of a tree. The image in Figure 16a shows a two-dimensional example of this type of fractal. The construction of this fractal is similar to the Koch Curve. The basis of

the fractal is simple a line segment. In the first iteration, two smaller line segments are added, as shown in Figure 16a2. In the second iteration, two smaller line segments are added to each of the four line segments in Figure 16a2. Figure 16a3 shows the result after this second iteration. Figure 16b shows an example of how this type of fractal can be extended into three-dimensions. This example has added texture and leaves to give the appearance of a real tree. This type of fractal is used to give a system of controlled randomness in which the fractal defines the possible paths that the embers can travel.



(a) (b)

Figure 16a-b: Example of 2D and 3D Fractal Tree [Demidov, 2001]

2.2.6 Fractal Tree Implementation

The traditional approach at generating a fractal tree through iterative replacement is not appropriate for this research. This approach requires the entire tree fractal to be pre-generated and would generate many paths that would not be traversed by the current ember. The approach of this research is similar to the example above. The entire tree

fractal is viewed as a collection of line segments. The current position of the ember is stored in four structures. The first structure is the direction of the current segment being traversed and is called *UnitVector*. *UnitVector* is a 3D unit vector, i.e. a vector with magnitude of 1. The second structure is the magnitude for *UnitVector* and is called *Magnitude*. *Magnitude* describes the length of the current line segment being traversed and is an integer. *Magnitude* could be stored within *UnitVector* as just a single segment vector, but storing them separately allows their values to be accessed without additional calculations. The third structure is called *BasePt* and is a 3D point in Cartesian space that acts as the base for the current line segment. A vector only describes a direction, to turn it into a line segment a point is needed to act as one end of the line segment. Adding the vector to the point describes the other end of the line segment. *UnitVector*, *Magnitude* and *BasePt* collectively describe the current line segment being traversed. The fourth structure helps define a specific position on that line segment and is called *Step*. *Step* is an integer with a value from 0 to the value of *Magnitude*. As the ember travels along the line segment, *Step* will increase in value. These four structures are used in Equation 8 to calculate the current position of an ember. The structure *CurrentPt* in Equation 8 is a 3D point in Cartesian space.

$$CurrentPt = BasePt + (UnitVector * Step / Magnitude)$$

Equation 8: Current Position on Fractal Tree

The initial value used for *BasePt* is a random value within a 100 by 100 square centered around the Cartesian origin and existing within the XY Plane where Z equals zero. The size of this square is determined by the desired size of the base of the fire in this

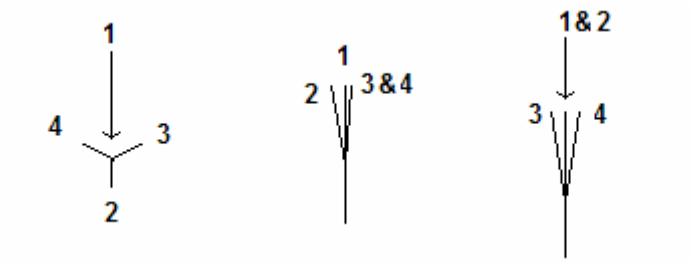
simulation. As the fractal tree is generated, the base point is updated to be the base of the current line segment.

The initial value used for *UnitVector* is (0,0,1). This value ensures that the tree expands in the direction required for this research, i.e. in an upward direction. As the fractal tree is generated, this segment vector is rotated to indicate the direction of the current tree segment.

The initial value used for *Magnitude* is 28. As the fractal tree is generated, this value will decrease by a fixed delta value with each new tree segment. For this research a delta value of one was selected. Once the magnitude reaches a minimum value the end of the tree has been reached and the ember can be extinguished. The absolute minimum for the magnitude is zero, but setting a larger minimum value allows the structure and size of the tree to be controlled with greater detail. For this research a minimum magnitude of zero was selected. All values used for the initial magnitude, the magnitude delta and the minimum magnitude were selected via experimentation to provide a tree of sufficient size for this research.

The initial value used for *Step* is zero as the ember begins at the end of the line segment. The initial value for *CurrentPt* is calculated via Equation 8. During each time step, the ember will step along the line segment until it reaches the line segment's end. Increasing the value of *Step* by one and recalculating *CurrentPt* via Equation 8 accomplishes this. Once *Step* has increased to the point where it equals *Magnitude*, the ember is at the

intersection at the end of the current line segment and a new line segment must be selected. In the process of selecting the new line segment several things happen. *BasePt* is updated to be the end of the current segment and thus the beginning of the next line segment. *Magnitude* is decreased by its fixed delta. *Step* is reset to zero, as the ember has not progressed along the new line segment yet. The final step is to calculate a new *UnitVector* by selecting one of the available paths. In this simulation there will be four possible paths as depicted in Figure 17.



(a)XY Plane (b)YZ Plane (c)XZ Plane

Path	Probability	Rotations (Degrees)
1	14.29%	(0,0)
2	28.57%	(10,0)
3	28.57%	(10,120)
4	28.57%	(10, -120)

(d)

Figure 17a-d: Possible Paths for Next Segment when Traversing Fractal Tree

To control the shape of the tree a bias is applied to the selection process. The percentages used for this research are indicated in Figure 17d. These values for the bias were selected to discourage the ember from traveling in a straight line. Thus, instead of each path being

equally likely to be selected, the upward option is half as likely to be selected as any of the other three paths.

Each of the four paths is calculated from the current value of *UnitVector* via two rotations. The first rotation causes the path to deviate from continuing in the same direction by rotating around an orthogonal vector. The initial value for this orthogonal vector is (0,1,0), i.e. the Y-axis, and based on the approach described below does not have to be calculated or stored beyond this initial value. The second rotation is around the original *UnitVector*. The exact values of these rotations, shown in Figure 17d, were selected via experimentation to control the shape of the overall fractal tree.

2.2.7 Rotations via Matrix Algebra

To perform the rotations required to calculate the new line segment *matrix algebra*, *affine transformations* and a *transformation matrix* are used. The basic concept behind this approach is that a 3D point will be represented as a 4 x 1 matrix and will be processed through a 4 x 4 Transformation Matrix via standard matrix multiplication. The result will be another 4 x 1 matrix representing the transformed point. This is represented in Figure 18 where the point (X, Y, Z) is transformed to the new location (X′, Y′, Z′).

$$
\begin{aligned}
X' &= a_{11} * X + a_{12} * Y + a_{13} * Z + a_{14} * 1 \\
Y' &= a_{21} * X + a_{22} * Y + a_{23} * Z + a_{24} * 1 \\
Z' &= a_{31} * X + a_{32} * Y + a_{33} * Z + a_{34} * 1 \\
1 &= a_{41} * X + a_{42} * Y + a_{43} * Z + a_{44} * 1
\end{aligned}
\qquad
\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \bullet \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}
$$

(a) (b)

Figure 18a-b: Transforming a 3D Point through a Transformation Matrix

The power behind this method is in the 4 x 4 Transformation Matrix. All the standard graphics transformations can be represented via this 4 x 4 matrix: translation, scaling, rotation, skewing, projection, etc. The standard equations for some of these 3D transformations and their transformation matrix are given in the following figures. Figure 19 shows the matrix that will perform scaling with respect to an Axis via the variables: *a ,b ,c*. Figure 20 shows the matrix that will perform translation of a 3D point via the vector (*a, b, c*). Figures 21, 22, and 23 show the matrixes that perform rotation around the z-axis, y-axis and x-axis by *a* degrees. Those are the most common transformation matrixes. The transformation matrixes for skewing, projection, etc. can be found via many resources such as the website for Geometric Transformations in the Computing and System Sciences department at Taylor University [Toll, 1999].

$$X' = aX \qquad \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$
$$Y' = bY$$
$$Z' = cZ$$

(a) (b)

Figure 19a-b: Transformation Matrix for Scaling

$$X' = X + a \qquad \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$
$$Y' = Y + b$$
$$Z' = Z + c$$

(a) (b)

Figure 20a-b: Transformation Matrix for Translation

$$X' = \cos(a)X - \sin(a)Y$$
$$Y' = \sin(a)X + \cos(a)Y$$
$$Z' = Z$$

(a)

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(b)

Figure 21a-b: Transformation Matrix for Rotation around z-axis by *a* degrees

$$X' = \cos(a)X + \sin(a)Z$$
$$Y' = Y$$
$$Z' = -\sin(a)X + \cos(a)Z$$

(a)

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(a) & 0 & \sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(b)

Figure 22a-b: Transformation Matrix for Rotation around y-axis by *a* degrees

$$X' = X$$
$$Y' = \cos(a)Y - \sin(a)Z$$
$$Z' = \sin(a)Y + \cos(a)Z$$

(a)

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & \sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(b)

Figure 23a-b: Transformation Matrix for Rotation around x-axis by *a* degrees

Once all the required transformations are represented via 4 x 4 matrixes, they can be multiplied together to result in a single 4 x 4 matrix that represents all the required transformations. The specifics for multiplying two 4 x 4 Matrixes together are given in Figure 24. This is where the computational power of a transformation matrix is displayed. Suppose an application needs to perform a series of complex transformations on multiple 3D points. The complex transformation might be a rotation followed by a translation, then another rotation, and finally scaling. Rather than performing these four

steps for every point, they are setup as 4 x 4 matrixes and multiplied into a single 4 x 4 matrix. Then this single 4 x 4 matrix can transform each point in a single step. It is computationally more efficient.

$$c_{ij} = \sum_{a} (c_{ia} * c_{ai}) \qquad \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

(a) (b)

Figure 24a-b: 4 x 4 Matrix Multiplication

To utilize this in generating the fractal tree, begin with a *master transformation matrix* as a 4 x 4 identity matrix as shown in Figure 25. Then at the end of each segment, the transformation matrixes for the two rotations required for the randomly selected path can be multiplied into the master transformation matrix. This gives a master transformation matrix that contains all the rotations required to transform from the initial *UnitVector* into the appropriate value for the *UnitVector* of the current line segment. Thus, at the end of each segment the new *UnitVector* can be obtained by taking the vector (0,0,1) and processing it through the master transformation matrix via the process in Figure 18.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 25: 4 x 4 Identity Matrix

2.3 Particle System via Third-Order Polynomial Paths

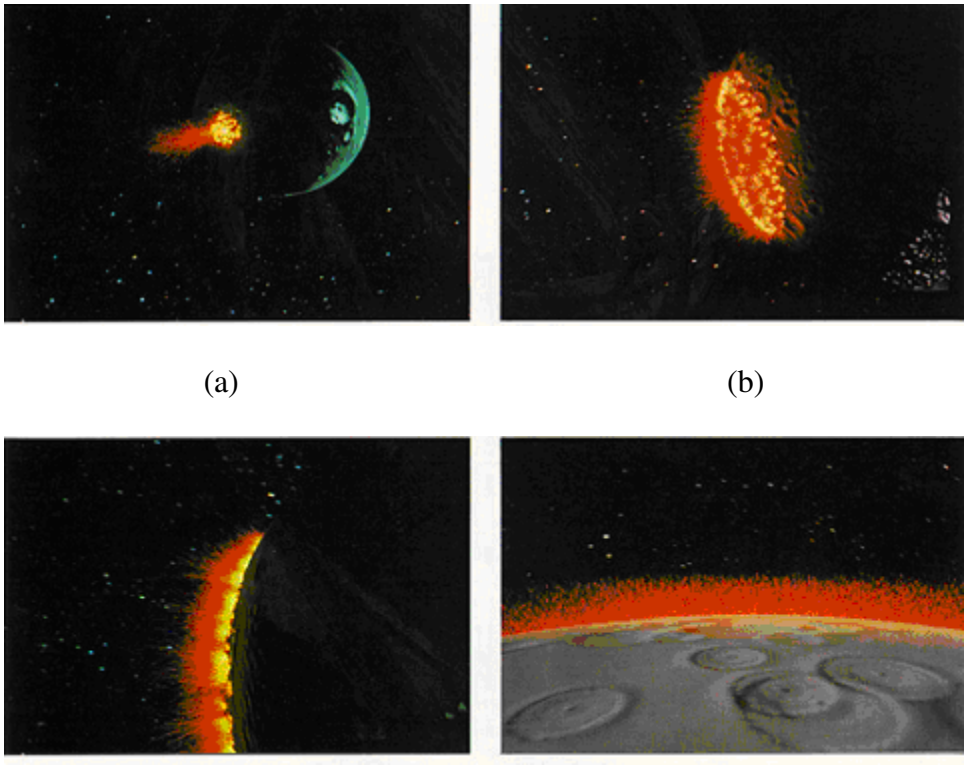
The final method to discuss is a particle system. Particle systems are very good at simulating and modeling 'fuzzy' items such as clouds, smoke, fire, tornados, even flocks of birds [Reeves, 1983][Reynolds, 1987].

A particle system consists of individual, tiny elements, called particles. Each of these particles exists independently, but they can influence the behavior of other particles in the population. A particle system is a step-based, iterative method. The same steps are performed during each time step. During each time step, particles can be added to the population, move their position, change an attribute about themselves, or can be removed from the population. A single particle is created and added to the particle population based upon predetermined rules. While that particle exists it has a defined behavior and may influence the behavior or attributes of its neighbors. For example, suppose a particle system were simulating dust particles and gravitational attraction in space. Two particles would attract towards each other based on their relative mass and the distance between them [Wagon, 1998]. As more particles cluster together they have a larger mass and thus affect the position of more distant particles. In addition, the color of the particle could be affected by how other particles are clustered around it, becoming brighter the more particles cluster around it. A particle may be visually rendered as a point, line, sprite image or even another particle system. After a predetermined amount of time or upon meeting some predetermined criteria, the particle would be removed from the population.

The particle system controls the population of particles as they go through their life cycle based on these predetermined rules. Collectively the particles simulate objects without hard boundaries such as clouds, water, wind and explosions. They can also model behavior such as the path of a flock of birds, a herd of cattle, a school of fish or even a crowd of humans [Martin, 1999][Reeves, 1983][Crowd, 2002].

2.3.1 Genesis Bomb Particle System

One of the first and more famous particle systems is the explosion of the 'Genesis' bomb in the movie 'Star Trek II: The Wrath of Khan' as shown in Figure 26. This effect actually was generated by two separate particle systems. The first particle system controlled the overall shape of the explosion as it moved across the planet's surface. It contained no visible rendering, but spawned multiple copies of the second particle system. The behavior of this system was set up to build a series of concentric circles that simulate the explosion engulfing a planet. The second particle system represented individual smaller explosions. Each particle in this system would vary its color and position to simulate an explosion rising from the point of impact and then falling back to the surface [Reeves, 1983].



(a) (b)

(c) (d)

Figure 26a-d: 'Genesis' bomb Particle System from 'Star Trek II: The Wrath of Khan' [Khan, 1982]

2.3.2 Parametric Polynomials

The particle system for this research will be much simpler. The embers begin life at random intervals with a random position along the 2D plane that represents the base of the fire. Throughout the life of an ember, it traverses a path described by a random three-dimensional polynomial. After a predetermined random period of time or upon reaching an upper or outer boundary the particle ceases to exist.

The polynomials used in this research that control the path of the ember are the three-dimensional parametric equations as shown in Equation 9.

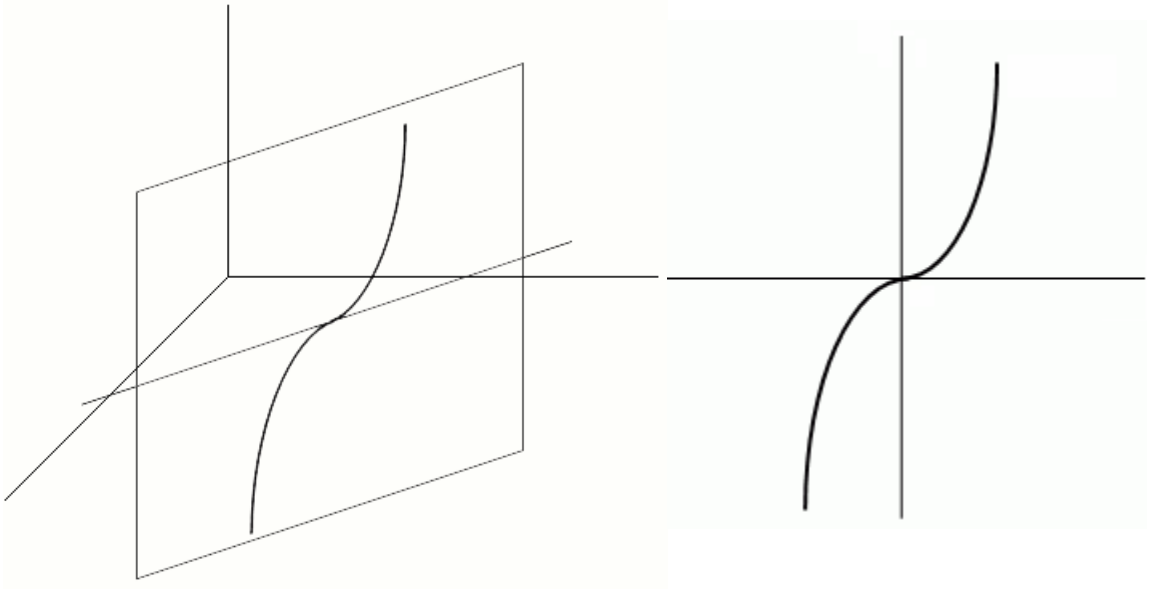
(a) $X(t) = tm_x + c_x$

(b) $Y(t) = tm_y + c_y$

(c) $Z(t) = at^3 + bt^2 + ct + d$

Equation 9a-c: Parametric Equation for Third-Ordered Polynomial

The parametric variable, *t*, represents time. The result of these three equations describes the position of a particle in 3D Cartesian space at any given moment.



(a) (b)

Figure 27a-b: Example of 2D Plane described by Equation 9

The parametric equations *X(t)* and *Y(t)* describe a line in the XY Plane via the standard slope-offset form of a line. The slope of the line is m_y / m_x. The offset of the line is described by the point (c_x,c_y). These equations indicate the position of the particle in

space with respect to time. When a particle is created random values in the range of -10 to 10 are generated for each of the coefficients: m_x, c_x, m_y and c_y. This range was selected arbitrarily. As shown in Figure 27a, this line also describes a 2D Plane in 3D Space that is orthogonal to the XY plane. Each particle exists entirely within the 2D plane described by its coefficients. As shown in Figure 27b, the line described by *X(t)* and *Y(t)* describes the horizontal axis of the 2D plane, while the value of *Z(t)* describes the vertical axis.

The parametric equation Z(t) is the standard form of a third order polynomial. This will represent the height of a particle from the ground. The coefficients in *Z(t)* control the depth of the dip and can be used to give a realistic representation of an ember. When a particle is created it generates random values in the range of -10 to 10 for the coefficients: *b, c* and *d*. This range was chosen arbitrarily. The coefficient *a* is given a value in the range 0 to 10. Forcing the value of *a* to be positive causes the value of *Z(t)* to approach negative infinity as the value of *t* approaches negative infinity. Additionally, as the value of *t* approaches positive infinity the value of *Z(t)* will also approach positive infinity. This ensures that a particle will travel in an upwards direction, rather than being created in the sky and falling towards the ground. Since *Z(t) is* an odd-ordered polynomial, the particle may hover or dip back towards the ground before ultimately rising into the sky. An example of this for a=1,b=0, c=0 and d=0 is shown in Figure 28a. An example of this for a=2,b=0, c=0 and d=0 is shown in Figure 28b.

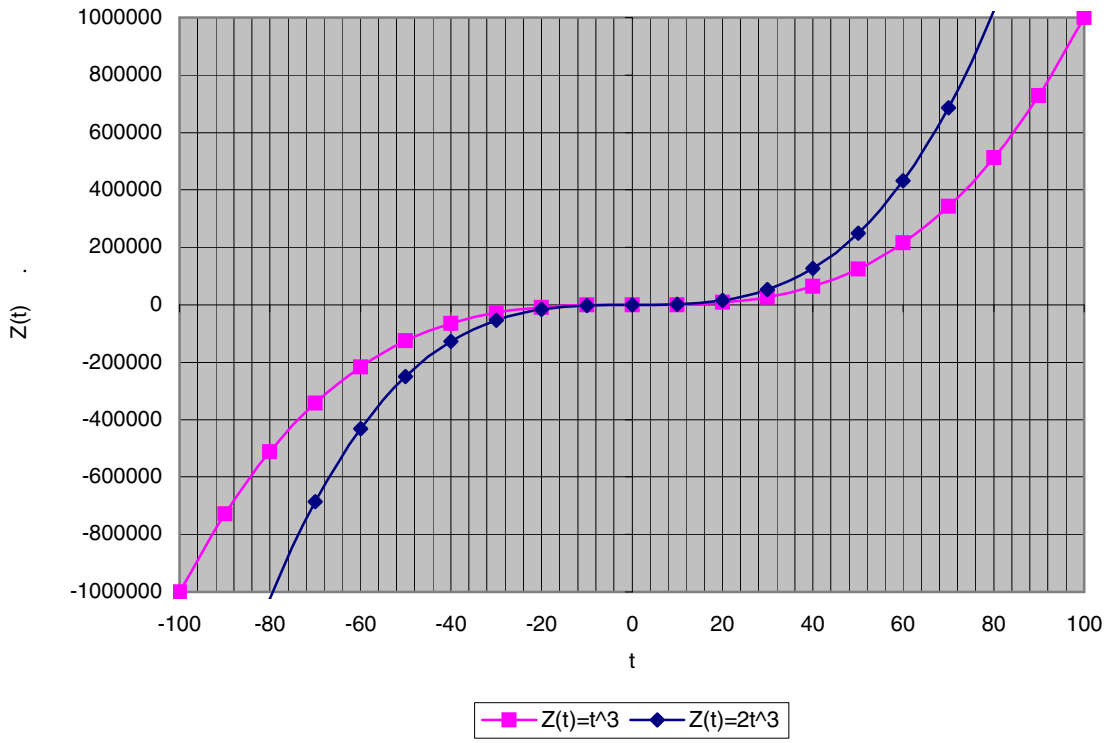


Figure 28: Example of (a) $Z(t) = t^3$ and (b) $Z(t) = 2t^3$

The initial value of *t* has to vary based on the other coefficients for Equation 9c. For example, the bottom-most point displayed in Figure 13a has a value for *t* of -100. In Figure 28b, a value of -100 for *t* is not displayed on the graph, a value of -80 is the first point visible. To address this fact, when each particle is created it is given a *t* value of -10 and is immediately iterated with an increase of 0.02 until *Z(t)* crosses a threshold value that causes it to be displayed in the view port. For this research that threshold value is -170. During each time step of the simulation the value of *t* will be increased by 0.02 and its new position is calculated. Once *Z(t)* reaches an upper threshold, the particle has exited the current view port and is removed from the particle system. For this research the

upper threshold of the view port is 200. Additionally, if *t* reaches a value of 10, it is removed from the particle system. This defines the maximum life for a particle.

Chapter III

3. Software Implementation

To collect the empirical data required the three methods described in Section 2 were implemented. By this researchers preference, they were implemented as a Win32 desktop application using Microsoft Visual Basic .NET 2005 under the 2.0 .Net framework. The application interface has two main sections: a control center and view ports. Each of these is described in a sub-section below.

3.1 View Ports

The view ports provide a way to view the simulation being generated. As shown in Figure 29, the view ports (left hand side of application) consist of a series of five views. The first view is labeled MultiView and is made up of half sized versions of the other 4 views. The next three views are labeled XY Plane, YZ Plane and XZ Plane. As expected these are the orthogonal views of the image being generated. It should be noted that in this implementation the z-axis travels in an 'Up/Down' direction instead of its traditional orientation. The final view is labeled Camera and is a rotate-able view. This view also allows for a static representation of a fire to be displayed and for the coordinate axis to be displayed.

3.2 Display Options

The Display Options tab of the control center controls various options available to configure the view ports. Each section of the MultiView view port can be toggled on or off. The camera view port can be configured to display the Axis and Fire or to not display them. The positional of the camera can also be controlled via three values that control the amount of rotation.

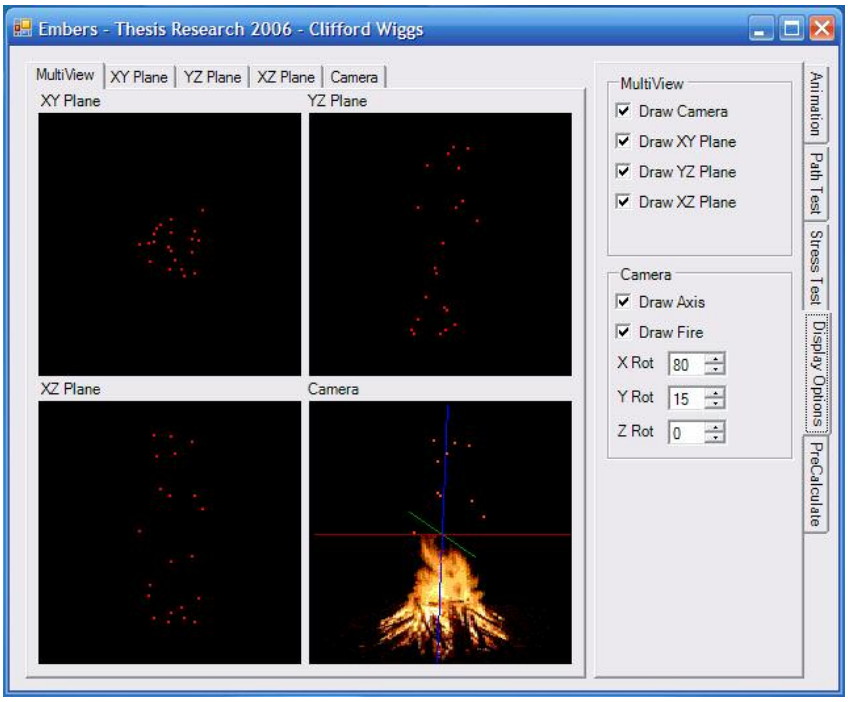


Figure 29: Software Implementation – Display Options

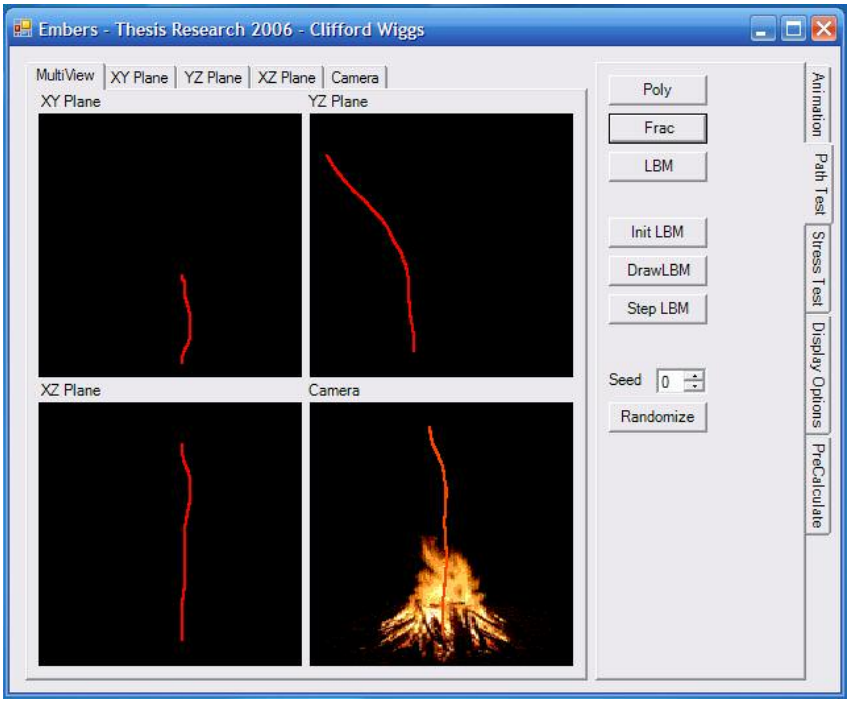


Figure 30: Software Implementation – Path Test

3.3 Path Test

The Path Test tab of the control center allows you to generate a single ember of each type. The path the ember will take through out its lifetime is displayed as a line. Figure 30 shows an example of a 3D Fractal ember that was generated by pushing the button labeled 'Frac'. The buttons 'Poly' and 'LBM' generate embers of the other two types examined in this research.

The 'Init LBM', 'Draw LBM' and 'Step LBM' buttons allow you to directly control the LBM Lattice. This tab also allows you to seed the random number generator for reproducible results.

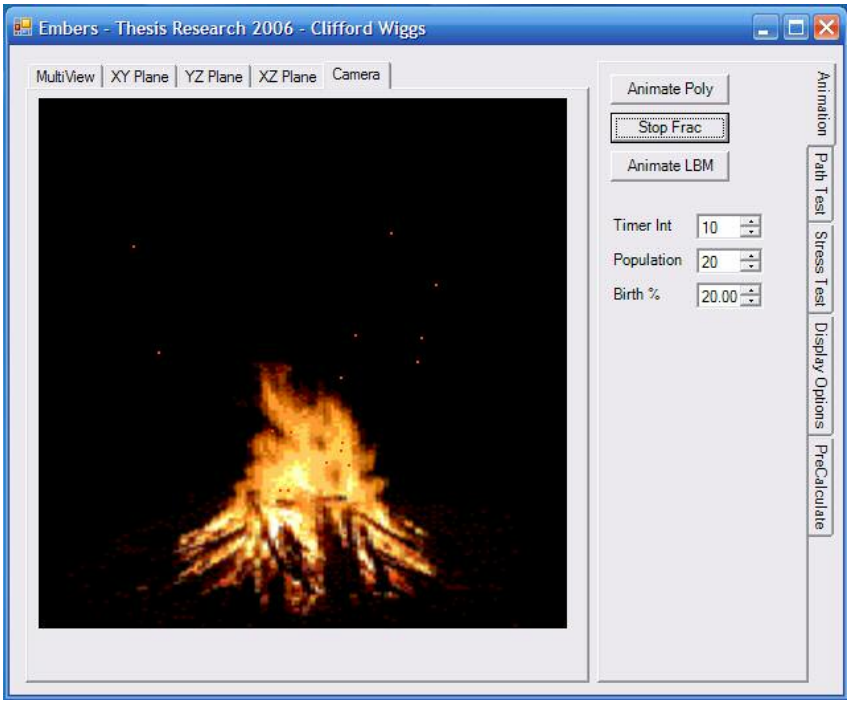


Figure 31: Software Implementation – Animation

3.4 Animation

The Animation tab of the control center allows you to view an animation of each of the ember types, displayed in Figure 31. The animation is begun via one of the three buttons starting with the label starting 'Animation'. The label of that button will then change to read 'Stop' and is used to end the animation.

The animation is controlled by the other three fields on this tab. The 'Timer Int' value is the interval between screen updates and controls the speed of the animation. The 'Population' value controls the maximum number of embers that are allowed to exist at one time. The 'Birth %' value controls the chance that a new ember will be introduced into the simulation.

3.5 Stress Test

The Stress Test tab of the control center, displayed in Figure 32, allows you to generate data on how much time is required by each ember type. There are two sections in this tab: manual and automated. The manual section will perform a single test and display the results in the 'Results (ms)' textbox. The automated section will run a series of the same test performed in the manual section and report the results via a comma delimited file.

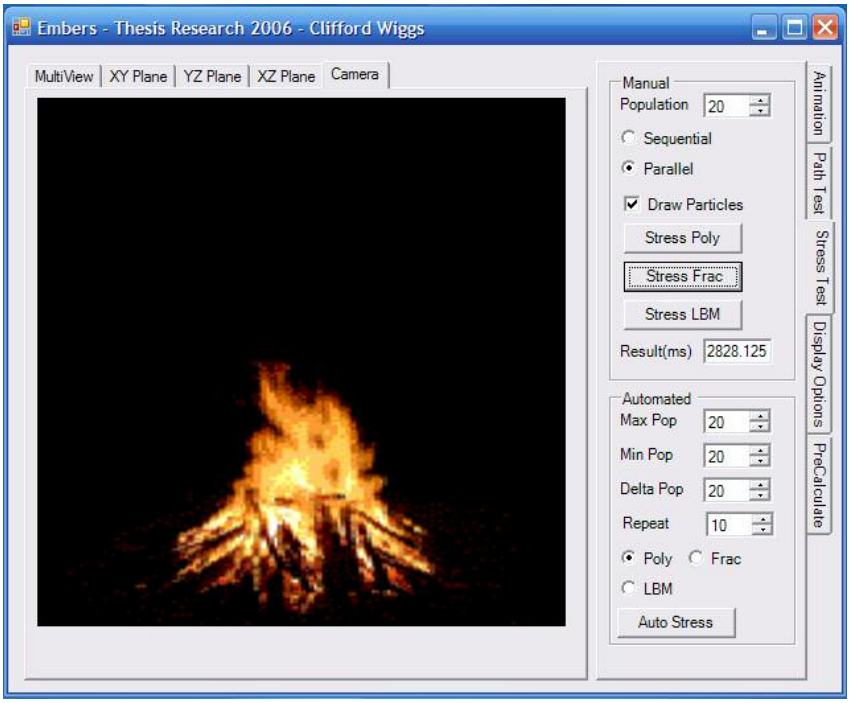


Figure 32: Software Implementation – Stress Test

There are a few options in the manual section which control the test being generated. The 'Population' value controls how many embers will be used in the test. How these embers are used is controlled by the radio buttons 'Sequential/Parallel'. A value of 'Sequential' means that the embers are generated one at a time. The next ember is not generated until the current ember is extinguished. A value of 'Parallel' means that the

entire population of embers are generated at a single time. The test completes when the last ember is extinguished. The next checkbox controls whether the embers will be rendered during the test. This allows you to remove the time spent rendering from the test. Finally the test is started via the 'Stress' buttons.

The automated section also has some options that control its behavior. The 'Max Pop', 'Min Pop' and 'Delta Pop' values control the population size of each test run. The automated test begins with a population of 'Max Pop' and then decreases for each test by 'Delta Pop' until the value of 'Min Pop' is reached. This is similar to how a traditional 'For' loop works. The 'Repeat' value controls how many times the test is performed before the next population size is tested. The radio buttons 'Poly', 'Frac' and 'LBM' control which type of ember is tested. Finally the 'Auto Stress' button begins the automated series of tests.

3.6 PreCalculate

The PreCalculate tab of the control center, displayed in Figure 33, allows you to view an animation of each ember type without any type of delay that might be caused by the calculation of an ember's position. This allows you to view the three ember types at the same speed by pre-calculating all the ember's positions. This is particularly helpful in the case of the LBM embers which require massive calculations for each ember during each time-step.

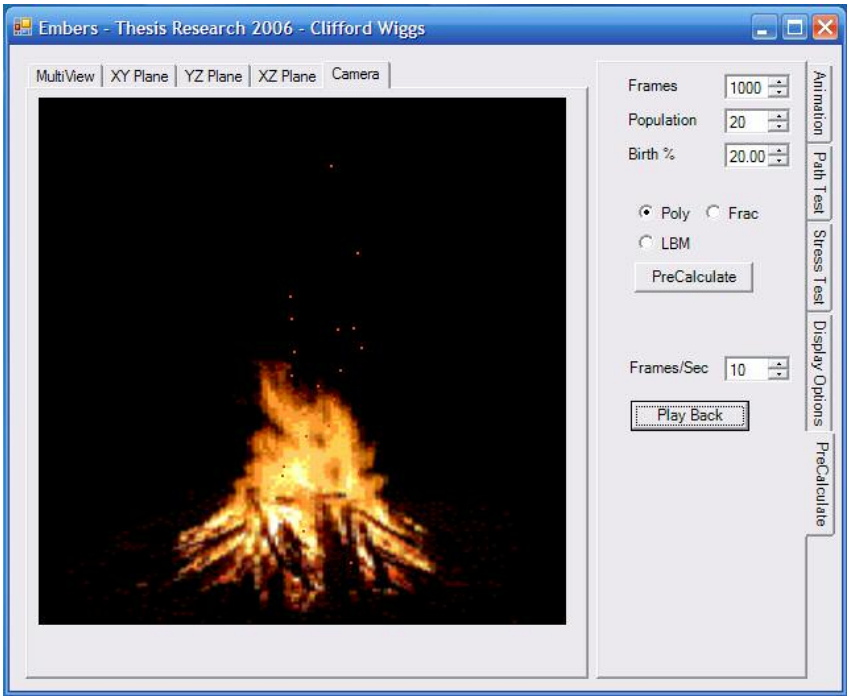


Figure 33: Software Implementation – PreCalculate

There are several items which control the animation being pre-calculated. The 'Frames' value controls how many time-steps or frames of animation will be pre-calculated and cached. The values for 'Population' and 'Birth %' function the same as in the animation tab described above. The radio buttons ' Poly', 'Frac' and 'LBM' control the type of ember to be used in the animation. The 'PreCalculate' button generates the cached frames and informs you when they are complete. The 'Frames/Sec' value controls the playback speed of the cached frames. The 'Play Back' button allows you to start and stop the animation of the cached frames.

Chapter IV

4. Methodology and Results

The three methods described were compared using three metrics. First is a comparison of the amount of memory required per time step as the number of concurrent embers is increased. Next is a comparison of the amount of time required to generate particles as the number of concurrent embers was increased. Last is a subjective visual comparison of which method best visually depicts the path of embers. The conclusions drawn from these results will be discussed in the next section.

4.1 Memory Requirement Metrics

The first metric is the amount of memory required for each method. The amount of memory required consists of three classifications. The first is the amount of memory required to hold the current state of each ember. The second is memory requirements that are common to all embers, i.e. the LBM lattice. The last is the memory required to calculate a single iteration, i.e. the local variables in the scope of a CalculateNextPostion method. The test results for each method are shown in Figure 34.

	Ember Attributes	Common Attributes	Implementation
LBM	18	8000	18
3D Fractal	68	0	44
Particle System	60	0	0

Figure 34: Memory Requirements in Bytes by Classification and Method

To calculate this metric the implementation of each method was examined. Each instance of a constant variable was counted as zero bytes. Each instance of a variable of an integer type was counted as two bytes. Each instance of a variable of a real-value type was counted as four bytes. The total amount of memory required for each method is calculated by adding these values together. The second and third classifications are independent of the number of concurrent embers, however the first is not. As the number of embers is increased it will add more to the memory requirements. Using these values, the amount of memory required to generate concurrent embers was calculated. The results are displayed in Figure 35.

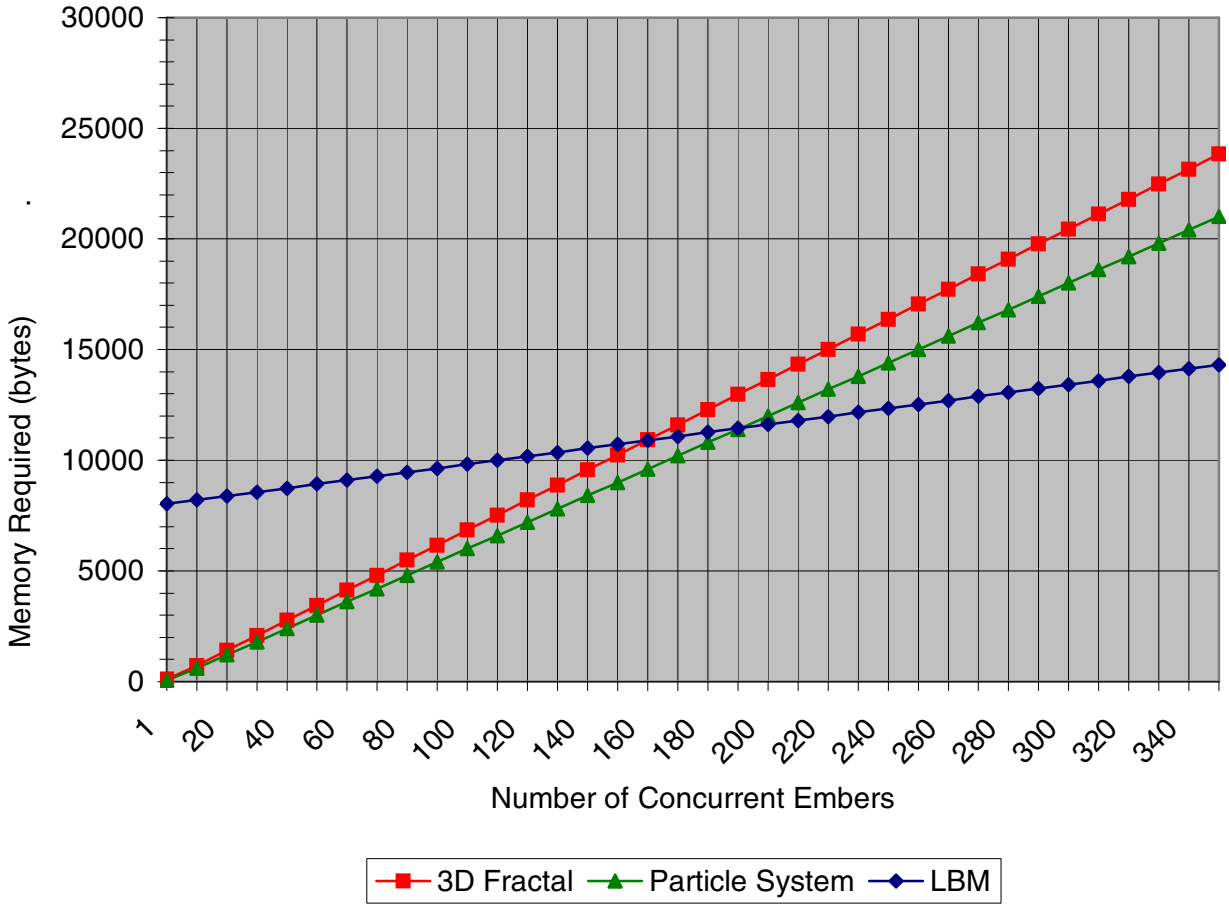


Figure 35: Total Memory Required versus Number of Concurrent Embers

4.2 Time Requirement Metrics

The second metric is related to the time required to calculate each method. Each instance of an ember has an indeterminable live cycle, i.e. each will take a different number of time steps before it reaches the end of its life. For example, a particle in the LBM method might hover for a random amount of time before it is caught in a current of the fluid flow and moves to the boundary of the lattice. Thus instead of comparing the time required to generate a few embers, hundreds of embers were generated to provide an average of these

short-lived and long-lived embers. Additionally, every data point is an average of ten test runs. There were two methods used in this section of the comparison. In the first method the embers were generated sequentially. A single ember is generated at a time. The next ember is not generated until the previous ember has completed traversing its path. A single ember generates too quickly for a comparison to be meaningful. This method allows a comparison of the amount of time required to generate embers in each method. To calculate the amount of time required for each method, an increasing number of sequential embers were generated. None of the embers were rendered visually, only the positioning of the ember was calculated. This was done to ensure that only the method itself and not the rendering logic was measured. The amount of time required for all embers to complete their life cycle was captured and charted in Figure 36.

Because of the amount of time required to update the LBM lattice, the values for the LBM method are drastically different than the other two methods. Thus it has been graphed on a separate axis to the right of the chart and the scale has been compressed by a factor of 625. While it visually appears to have a similar slope to the other two methods, because of the compressed scale, it actually has a much higher slope.

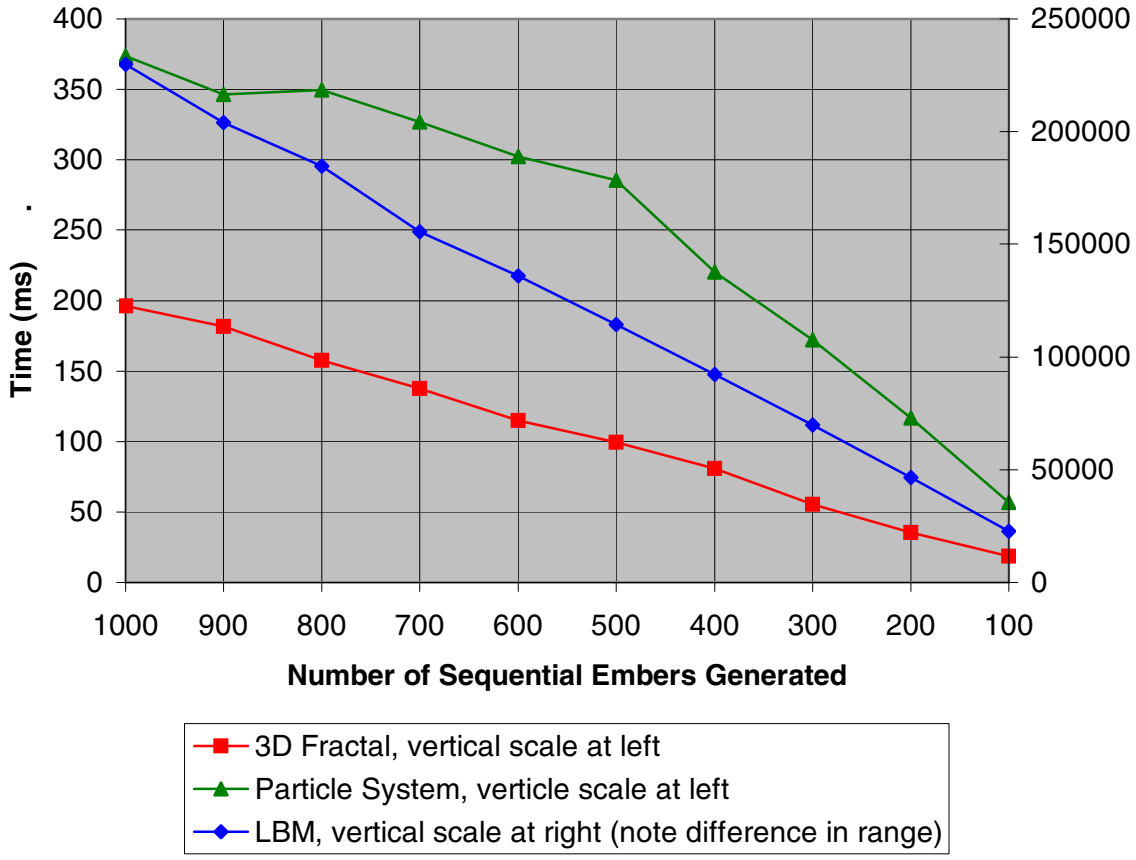


Figure 36: Total Time Required for Sequentially Generated Embers versus Number of Embers Generated

The second method of the second metric is identical to the first method except that all the embers are generated concurrently instead of sequentially. This allows another point of view to compare to time required by each method to generate embers. The amount of time required for all embers to complete their life cycle was captured and charted in Figure 37. Because of the amount of time required to update the LBM lattice, the amount of time required for the LBM is drastically different than the other two methods. Thus it has been graphed on a separate axis to the right of the chart. The scale of the secondary axis has been compressed by a factor of ten. Thus while the LBM method appears to have

a smaller slope than the other two methods, it is actually larger. For comparison purposes, Figure 38 is an expanded view of Figure 37 which has the same range as Figure 36.

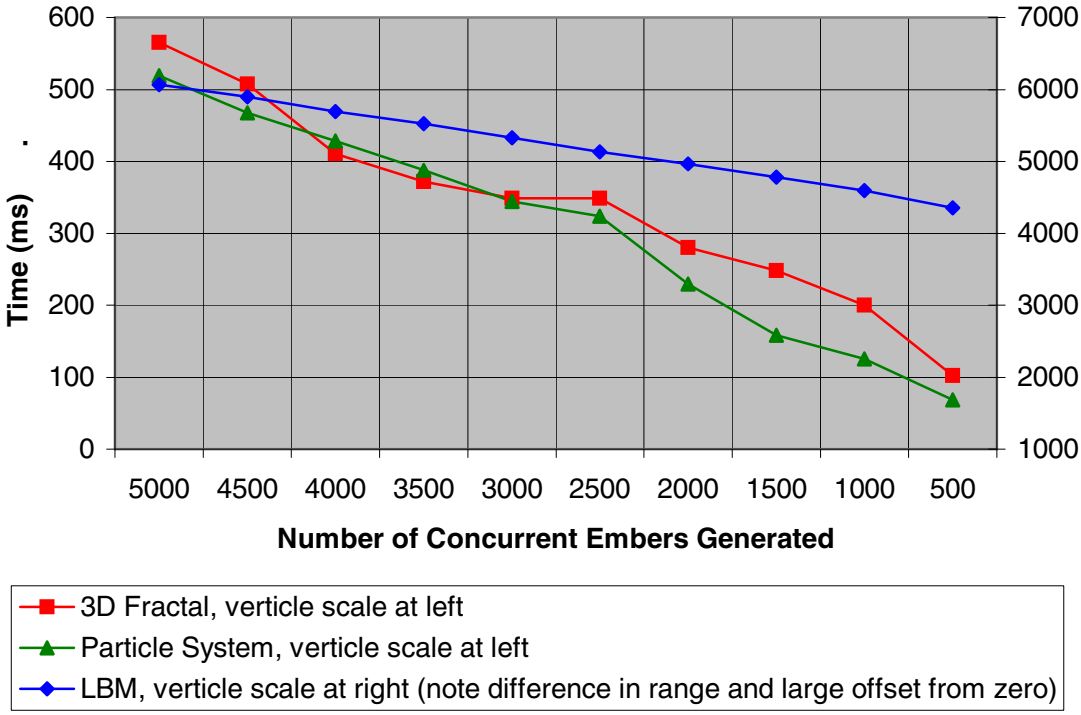


Figure 37: Total Time Required for Concurrently Generated Embers versus Number of Embers Generated – Full Range

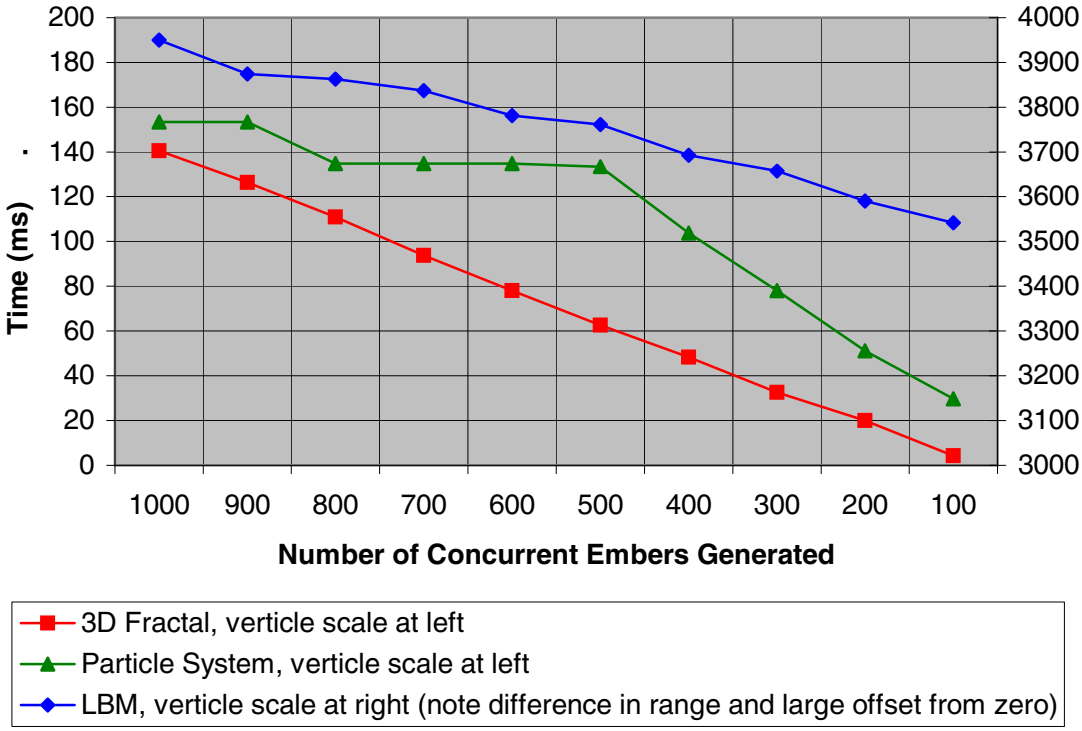


Figure 38: Total Time Required for Concurrently Generated Embers versus Number of Embers Generated – Restricted Range

4.3 Visual Comparison

The third metric is a subjective comparison of which method best visually represents the paths of embers. The electronic copy of this document contains embedded video in Figure 39. The positions of the embers for this comparison were pre-calculated. Thus allowing the comparison of the three methods to have a constant frame rate of 50 frames per second.

This is a subjective measurement. However, the LBM method gives a more fluid and realistic representation of a fire's embers. The embers movement is erratic, but realistic in

the way that thermal updrafts behave. The 2D Fractal method is the second best at realistic representation. The underlying structure of the 3D tree is a little too apparent in the paths the embers take. The particle system method gives the least realistic representation. The embers tend to congregate around the origin of the polynomials coordinate system. This creates an unrealistic 'bunching' effect for the embers.

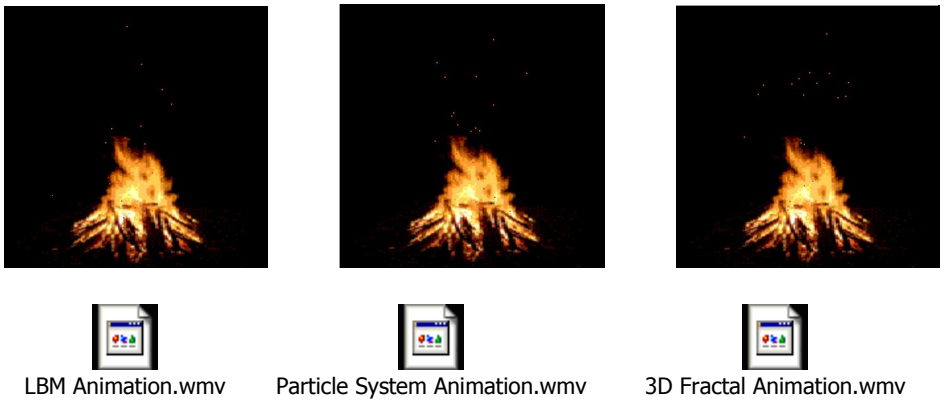


Figure 39: Sample Video of Pre-Rendered Animation for Subjective Visual Comparison

Chapter V

5. Conclusions

Based on the three metrics captured, each method has a situation in which it excels.

For small numbers of particles LBM takes much more memory than the other methods. This is due to the large amount of memory required to store the lattice. However each LBM ember shares the same lattice, so as the number of concurrent particles increases, it begins to take less memory than either of the other two methods. It requires between 150 and 190 concurrent embers before LBM becomes the better choice. Below that value the particle system is a slightly better choice over the 3D fractal method.

Based on execution time, again LBM is much more computationally expensive. Again this is due to the LBM lattice and the time intensive equations required to update it. Either of the other two methods is preferable to the LBM method regardless of the number of concurrent embers. However the fractal method performs slightly better.

Finally, on the visual effect, LBM is the clear winner. Its fluid and realistic motion resembles real embers rather than a simulation of embers. The particle method produces embers which hover around the $t = 0$ area of the polynomial. The fractal method

produces embers, which too closely resemble the structure of the underlying tree structure.

The ranking of all three methods is summarized in Figure 40. Each metric was given a ranking from 1 (best) to 3 (worst). Based on these rankings, if memory, execution time, and visual effect are equal in importance, then the 3D Fractal is the best compromise. However, if the images are pre-rendered instead of being generated in real-time, then LBM generates the best visual effect by far.

	Memory	Execution Time	Visual
LBM	3	3	1
3D Fractal	2	1	2
Particle System	1	2	3

Figure 40: Ranking Comparison of All Methods by All Metrics

Chapter VI

6. Future Work

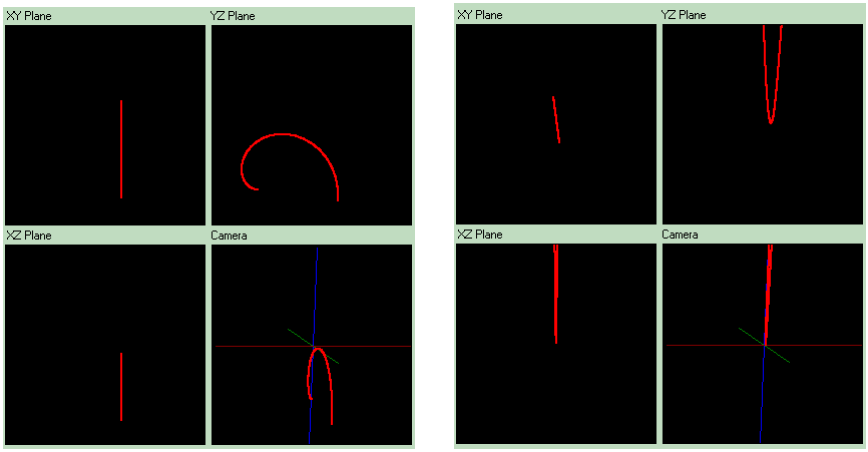
There are several suggestions that are independent of the method used to simulate the embers. These simulations were all conducted in a windless environment. The inclusion of a gentle breeze or a violent wind would greatly affect the outcome of any of these methods. The embers could also be used to impact the environment they reside in. An ember that falls to the ground on a flammable substance could generate a second fire, which would generate embers of its own. Finally, these methods for embers could be implemented along with existing methods for flames and smoke to generate an all-encompassing simulation.

There are also additional research items that could be conducted on each of the methods individually.

LBM research is rich with variety and depth; other variations and implementations could give different visual results. This research used a purely software driven LBM. LBM can also be implemented directly on video hardware, which would improve its computation cost [Li et al., 2003]. LBM can also be used to simulate fluid flows around stationary or moving objects. Examples of this would be embers from a house fire hitting the ceiling of

a room and scattering, embers in a forest fire scattering around a falling tree, or embers deep within a fire floating among the logs before reaching open air.

There are an infinite number of configurations for 3D fractal trees. Modifying any of the parameters would produce a different tree and thus different paths for the particle to traverse. In addition to modifying the length and angle delta of each branch, a different bias for path determination could be introduced. An undesirable example is shown in Figure 41a with a 100% bias for a specific path decision.



(a) (b)

Figure 41a-b: Samples of Undesirable Paths for (a) 3D Fractal and (b) Particle System Methods

In the particle system, the method of following a 3rd order polynomial could be improved with additional restrictions on the random values generated for its attributes. These attributes control the shape of the path for the particle. For some values of *a, b, c*, and *d* the path is too wide to be realistic. Figure 41b shows one such extreme where only a

portion of the path is visible to the view port. The rest of the curve is not displayable. These values could also contribute to the ‘bunching’ effect seen in this research.

References

[Barrero et al., 2000] Daniel Barrero, Mathias Paulin, Rene Caubet. "Simulating Turbulent Combustion", Computer Graphics & Geometry vol. 2:4 pp. 75-99, Winter 2000.

[Beaudoin et al., 2001] Philippe Beaudoin, Sebastien Paquet, Pierre Poulin. "Realistic and Controllable Fire Simulation", Canadian Man-Computer Communication Society 2001 pp.159-166.

[Chen and Doolean, 1998] S. Chen and G.D. Doolean. "Lattice Boltzmann method for fluid flows". *Annu. Rev. Fluid Mech.*, 30:329-364, 1998.

[Crowd, 2002] Crowd Dynamics Ltd., "The Science of Crowd Dynamics", URL: http://www.crowddynamics.com/index.htm, date created: 2002, date accessed: July 2006.

[Demidov, 2001] Evgeny Demidov, "Generation of 3D fractals for Web", Institute for Physics of Microstructures RAS, URL: http://www.ibiblio.org/e-notes/VRML/Web3D/Web3D.htm, date modified: May 2001, date accessed: July 2006.

[Fern, 2006] Image, URL: http://www.psych.utah.edu/stat/dynamic_systems/Content/examples/E42_Manual/Eco-of-Mind_CNS/images/Fern-Fractal-3D.jpg, date created: unknown, date accessed: July 2006.

[Frisch et al., 1986] U. Frisch, B. Hasslacher, and Y. Pomeau. "Lattice-gas automata for the Navier-Stokes equations". *Physical Review Letters,* 56(14):1505-1508, April 1986.

[IFS, 2006] IFS, "Iterated Function System", Wikipedia, URL: http://en.wikipedia.org/wiki/Iterated_function_system**,** date created: unknown, date modified: July 2006, date accessed: July 2006.

[Incredibles, 2004] The Incredibles (film). Pixar Animation Studios, Walt Disney Pictures, 2004, Time: [01:14:07]

[Khan, 1982] Star Trek II: The Wrath of Khan (film). Paramount, June 1982, Time: [*00:45:16*]

[Kosmulski, 2002] Michał Kosmulski, “Modular Origami – Fractals, IFS”, URL: http://hektor.umcs.lublin.pl/~mikosmul/origami/fractals.html, date created: , date accessed: July 2006.

[Lee et al., 2001] Haeyoung Lee and Laehyun Kim and Mark Meyer and Mathieu Desbrun. "Meshes on Fire", Computer Animation and Simulation '01 pp.75-84.

[Li et al., 2003] Wei Li, Xiaoming Wei, and Arie Kaufman. “Implementing Lattice Boltzmann Computation on Graphics Hardware”. The Visual Computer, vol. 19, nos. 7-8, pp.444-456, Dec. 2003.

[Loy, 2002] Jim Loy, “The Koch Curve”, URL: http://www.jimloy.com/fractals/koch.htm, date created: 2002, date accessed: July 2006.

[Martin, 1999] Allen Martin, “Particle Systems”, Worcester Polytechnic Institute, URL: http://www.cs.wpi.edu/~matt/courses/cs563/talks/psys.html, date created: 1999, date accessed: July 2006.

[Martz, 1997] Paul Martz, “Generating Random Fractal Terrain”, URL: http://www.gameprogrammer.com/fractal.html, date created: unknown, date modified: 1997, date accessed: July 2006.

[Muders, 1995] D. Muders. “Three-dimensional parallel lattice Boltzmann hydrodynamics simulations of turbulent flows in interstellar dark clouds”. PhD thesis, University at Bonn, August 1995.

[Muzy et al., 2003] Alexandre Muzy, Eric Innocenti, Jean-Francois Santucci, and David R. C. Hill. “Optimization of Cell Spaces Simulation for the Modeling of Fire Spreading”, SIGSIM 2003 pp.289.

[Nguyen et al., 2002] Duc Quang Nguyen, Ronald Fedkiw, Henrik Wann Jensen. “Physically Based Modeling and Animation of Fire”, SIGGRAPH 2002 pp.721-728.

[Nicholls, 1998] David Nicholls, “Fractal Ferns”, http://www.home.aone.net.au/byzantium/ferns/fractal.html, date created: 1998, date accessed: July 2006.

[Nielsen and Madsen, 1999] T. E. Nielsen and Soren Trautner Madsen. “Modelling, animation, and visualization of fire”, Master's thesis, University of Copenhagen, Denmark, April 1999

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

[Reynolds, 1987] Craig W. Reynolds. “Flocks, Herds, and Schools: A Distributed Behavior Model”, Computer Graphics 21:4 pp. 25-34, 1987 (SIGGRAPH 87)

[Reynolds, 1995] Craig Reynolds, "Boids: Background and Update", Reynolds Engineering and Design, URL: http://www.red3d.com/cwr/boids/, date created: 1995, date modified: 2001, date accessed: July 2006.

[Riddle, 2006] Larry Riddle, "Koch Curve", Agnes Scott College, URL: http://ecademy.agnesscott.edu/~lriddle/ifs/kcurve/kcurve.htm, date created: unknown, date modified: May 2006, date accessed: July 2006.

[SCS, 2004] Section Computational Science (SCS) - Lattice Boltzmann Research, "Lattice Boltzmann Model Theory", University of Amsterdam, Computing, System Architecture and Programming Laboratory (CSP), URL: http://www.science.uva.nl/research/scs/projects/lbm_web/lbm.html, date created: unknown, date modified: October 2004, date accessed: July, 2006.

[Toll, 1999] Bill Toll, "Geometric Transformations", Taylor University, Computing and System Sciences Department, URL: http://www.css.taylor.edu/~btoll/s99/424/res/mtu/Notes/geometry/geo-tran.htm, date created: unknown, date modified: 1999, date accessed: July 2006.

[Toolbox, 2005] Engineering Toolbox, "Air Properties", URL: http://www.engineeringtoolbox.com/air-properties-d_156.html, date created: 2005, date accessed: July 2006.

[Wagon, 1998] Science Joy Wagon, "The Universal Law of Gravitation", Oswego City School District Regents Exam Prep Center, URL: http://regentsprep.org/Regents/physics/phys01/unigrav/default.htm, date created: 1998, date accessed: July 2006.

[Wei et al., 2002] Xiaoming Wei, Wei Li, Klaus Mueller, and Arie Kaufman. "Simulating Fire with Texture Splats". *IEEE Visualization 2002* , pp. 227-237

[Wei et al., 2003] Xiaoming Wei, Ye Zhao, Zhe Fan, Wei Li, Suzanne Yoakum-Stover, and Arie Kaufman. "Blowing in the Wind". *ACM SIGGRAPH / EUROGRAPHICS 2003*, pp. 75-85

[Wei et al., 2004] Xiaoming Wei, Wei Li, Klaus Mueller, and Arie Kaufman. "The Lattice-Boltzmann Method for Gaseous Phenomena". *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 2, March/April 2004, pp. 164-176

[Yoshida and Nishita, 2000] Satoru Yoshida, Tomoyuki Nishita. "Modeling of Smoke Flow Taking Obstacles into Account", Eighth Pacific Conference on Computer Graphics and Applications (PG'00) p. 135, 2000.

[Zhao et al., 2003] Ye Zhao, Xiaoming Wei, Zhe Fan, Arie Kaufman and Hong Qin. "Voxels On Fire".*IEEE Visualization* 2003, pp. 271-279

Appendix A: Implementation Source - 3D Fractal Particle (fracpar.vb)

```
' Class FracPar
' Author - Clifford Lee Wiggs
' Date - July 2006
' Language - VB.Net 2.0
'
' This class represents a single particle.
' It uses a 3D Fractal tree to control its path.

Public Class FracPar
      'Current location of the particle in 3D space
    Private x As Double
    Private y As Double
    Private z As Double

      'Start of current branch being traversed
    Private xRoot As Double
    Private yRoot As Double
    Private zRoot As Double

      'End of current branch being traversed
    Private xTarget As Double
    Private yTarget As Double
    Private zTarget As Double

      'Stepping varibale for current location on current branch
    Private t As Integer

    'Length of current branch
    Private maxT As Integer

      'Offset in 2D Plane for based of tree that is applied to
      'the current location
    Private dx As Integer
    Private dy As Integer

      'Translation Matrix used to track all rotations required
      'from root to current branch
    Private RotationMatrix(3, 3) As Double

    'is this Paricle currently 'alive'?
    Private Alive As Boolean

    'Age fo the current particle - counts down to zero
    Private Age_Steps As Integer

      'Kill the current particle
    Public Sub Kill()
```

```
        Alive = False
    End Sub

    'Mark the current particle as alive
    Public Sub Birth()
        Alive = True
    End Sub

    'Query if the current particle is alive
    Public Function isAlive() As Boolean
        isAlive = Alive
    End Function

    'Age the current particle and kill when appropriate
    Private Sub getOlder()
        Age_Steps -= 1
        If (age_steps <= 0) Then Kill()
    End Sub

      'Accessors for the current point
      'Applies the offset before returning.
    Public Function getX() As Double
        getX = x - dx
    End Function
    Public Function getY() As Double
        getY = y - dy
    End Function
    Public Function getZ() As Double
        getZ = z - 160  'Fixed offset for base of fire
    End Function

      'Initializer for the particle
    Public Sub Init()
      'Initial position and length of the first(root) branch.
        t = 0
        maxT = 28

            'Range for age is dependant on length from root
            'to leaf in the tree
        'Max possible age for an ember is 28+27+26+....+2+1
            '= 28*29/2 = 406 (distance to leaf)
        'Pick a random minimum age... say 100
        Age_Steps = CInt(Int(300 * Rnd()) + 100)

            'Initial root is located at the origin and aligned with z
axi
        xRoot = 0
        yRoot = 0
        zRoot = 0
        xTarget = 0
        yTarget = 0
        zTarget = maxT

            'Choose random offset from origin
        dx = CInt(Int(100 * Rnd()) - 50)
        dy = CInt(Int(100 * Rnd()) - 50)
```

```
        'Yes - we start alive
    Alive = True

        'Initialize the rotation matrix to the idenity
    SetMatrixIdentity()

        'Take first step.
    MoveToNextStep()
End Sub

'Sets RotationMatrix to a 4x4 Identity
Private Sub SetMatrixIdentity()
    Dim i, j As Integer
    For i = 0 To 3
        For j = 0 To 3
            If (i = j) Then     'Diagonal is one
                RotationMatrix(i, j) = 1
            Else    'all else is zero
                RotationMatrix(i, j) = 0
            End If
        Next
    Next
End Sub

'Calculates a Rotation matrix for parameters
'Prepends (multiplies on the left hand side) of the Rotation matrix
Private Sub PrependRotation(ByVal a As Double, ByVal b As Double)

    'Convert degrees to radians
    a = a * Math.PI / 180
    b = b * Math.PI / 180

    'Temporary matrix for calculations
    Dim tmpMatrix(3, 3) As Double
    Dim tmpMatrix2(3, 3) As Double

    'Set identity
    Dim i, j As Integer
    For i = 0 To 3
        For j = 0 To 3
            If (i = j) Then
                tmpMatrix(i, j) = 1
            Else
                tmpMatrix(i, j) = 0
            End If

        Next
    Next

    'Populate tmpMatrix with a rotation around z
    'Do this first because we are prepending, so the last
        'rotation of this step is prepended first
    tmpMatrix(0, 0) = Math.Cos(a)
    tmpMatrix(1, 0) = -Math.Sin(a)
    tmpMatrix(0, 1) = Math.Sin(a)
    tmpMatrix(1, 1) = Math.Cos(a)
```

```
        'Merge this matrix with the existing matrix
        MergeMatrix(tmpMatrix2, tmpMatrix, RotationMatrix)

            'Reset to Identity
        For i = 0 To 3
            For j = 0 To 3
                If (i = j) Then
                    tmpMatrix(i, j) = 1
                Else
                    tmpMatrix(i, j) = 0
                End If

            Next
        Next

        'Populate tmpMatrix with a rotation around x
        tmpMatrix(1, 1) = Math.Cos(b)
        tmpMatrix(2, 1) = -Math.Sin(b)
        tmpMatrix(1, 2) = Math.Sin(b)
        tmpMatrix(2, 2) = Math.Cos(b)

        ''Merge this matrix with the existing matrix
        MergeMatrix(RotationMatrix, tmpMatrix, tmpMatrix2)

    End Sub

    'Utility function to multiple two matrices leaving result in a 3rd
    Private Sub MergeMatrix(ByVal a As Double(,), ByVal b As Double(,),
ByVal c As Double(,))
        Dim i, j As Integer

        For i = 0 To 3
            For j = 0 To 3
                'a(0, 0) = b(0, 0) * c(0, 0) + b(0, 1) * c(1, 0)
                  '+ b(0, 2) * c(2, 0) + b(0, 3) * c(3, 0)
                'a(0, 1) = b(0, 0) * c(0, 1) + b(0, 1) * c(1, 1)
                  '+ b(0, 2) * c(2, 1) + b(0, 3) * c(3, 1)
                'a(1, 0) = b(1, 0) * c(0, 0) + b(1, 1) * c(1, 0)
                  '+ b(1, 2) * c(2, 0) + b(1, 3) * c(3, 0)
                'a(2, 3) = b(2, 0) * c(0, 3) + b(2, 1) * c(1, 3)
                  '+ b(2, 2) * c(2, 3) + b(2, 3) * c(3, 3)
                a(i, j) = b(i, 0) * c(0, j) + b(i, 1) * c(1, j)
                  + b(i, 2) * c(2, j) + b(i, 3) * c(3, j)
            Next
        Next

    End Sub

    'Utility function to multiply 1x4 matrix by 4x4 matrix
      'returning result in 1x4 matrix
    'Note: the 4th element is ignored
    Private Sub PerformRotation(ByRef ux As Double, ByRef uy As Double,
ByRef uz As Double)
        Dim tx, ty, tz As Double

        tx = ux * RotationMatrix(0, 0) + uy * RotationMatrix(1, 0) + uz
* RotationMatrix(2, 0) + RotationMatrix(3, 0)
```

```
        ty = ux * RotationMatrix(0, 1) + uy * RotationMatrix(1, 1) + uz
* RotationMatrix(2, 1) + RotationMatrix(3, 1)
        tz = ux * RotationMatrix(0, 2) + uy * RotationMatrix(1, 2) + uz
* RotationMatrix(2, 2) + RotationMatrix(3, 2)

        ux = tx
        uy = ty
        uz = tz
    End Sub

      'Advances the particle one step in the simulation and
      'calculates position.
    Public Sub MoveToNextStep()
          'Kill particle when it reaches a leaf
        If (maxT = 0) Then
            Kill()
        Else
                  'Take a step along the current branch
            t += 1

                  'Calculate new position on that branch
            x = xRoot + (xTarget - xRoot) * t / maxT
            y = yRoot + (yTarget - yRoot) * t / maxT
            z = zRoot + (zTarget - zRoot) * t / maxT

            If (t = maxT) Then
                'Reached the end of this branch, find the end of
                  'the next branch

                maxT -= 1   'Make the next branch smaller

                        'Current position is the end of the
                        'current branch.
                x = xTarget
                y = yTarget
                z = zTarget

                        'Randomly select next branch direction
                        'and update rotation matix
                Dim tmpX, tmpY, tmpZ As Double
                Dim rndI As Integer
                rndI = CInt(Int(7 * Rnd()) + 1)
                'Update rotation matrix
                If (rndI = 1) Or rndI = 4 Then
                    PrependRotation(0, 10)
                ElseIf (rndI = 2) Or rndI = 5 Then
                    PrependRotation(-120, 10)
                ElseIf (rndI = 3) Or rndI = 6 Then
                    PrependRotation(120, 10)
                Else
                    'continue straight ahead
                End If

                'Perform rotation on a unit value vector and add
                  'to the target
                tmpX = 0
                tmpY = 0
```

```
                tmpZ = 1
                PerformRotation(tmpX, tmpY, tmpZ)

                        'Calculate the end of the next branch based
                        'on the rotated unit vector
                xTarget += (tmpX * maxT)
                yTarget += (tmpY * maxT)
                zTarget += (tmpZ * maxT)

                        'Root of the next branch is the end of the
                        'current branch
                xRoot = x
                yRoot = y
                zRoot = z

                        'Have not traveled along the new branch yet
                t = 0

            End If
            getOlder()
        End If
    End Sub

End Class
```

Appendix B: Implementation Source - Polynomial Particle System Particle (polypar.vb)

```
' Class PolyPar
' Author - Clifford Lee Wiggs
' Date - July 2006
' Language - VB.Net 2.0
'
' This class represents a single particle in a particle system.
' It uses a 3D 3rd order polynomial to control its path.

Public Class PolyPar
      'Current location of the particle in 3D space
    Private x As Double
    Private y As Double
    Private z As Double

      'Offset in 3D space that is applied to the current location
    Private ddx As Double
    Private ddy As Double
    Private ddz As Double

    'parametric variable for position equations.
    Private t As Double

    'x(t) = mx*t+bx
    'Slope and Offset
    Private mx As Double
    Private bx As Double

    'y(t) = my*t+by
    'Slope and Offset
    Private my As Double
    Private by As Double

    'z(t) = az * t^3 + bz * t^2 + cz * t + dz
    'Polynomial Coefficients
    Private az As Double
    Private bz As Double
    Private cz As Double
    Private dz As Double

    'is this Paricle currently 'alive'?
    Private Alive As Boolean

    'Age fo the current particle - counts down to zero
    Private Age_Steps As Integer
```

```
  'Kill the current particle
Public Sub Kill()
    Alive = False
End Sub

'Mark the current particle as alive
Public Sub Birth()
    Alive = True
End Sub

'Query if the current particle is alive
Public Function isAlive() As Boolean
    isAlive = Alive
End Function

  'Age the current particle and kill when appropriate
Private Sub getOlder()
    Age_Steps -= 1
    If (age_steps <= 0) Then Kill()
End Sub

  'Accessors for the current point
  'Applies the offset before returning.
Public Function getX() As Double
    getX = x + ddx
End Function
Public Function getY() As Double
    getY = y + ddy
End Function
Public Function getZ() As Double
    getZ = z + ddz
End Function

  'Initializer for the particle
Public Sub Init()
  'Choose random offset from origin
    ddx = (100 * Rnd() - 50)
    ddy = (100 * Rnd() - 50)
    ddz = (100 * Rnd() - 50)

        'Choose random slope
    mx = (20 * Rnd() - 10)
    my = (20 * Rnd() - 10)

        'Choose random offset
    by = (20 * Rnd() - 10)
    bx = (20 * Rnd() - 10)

        'Choose random coefficients
    az = (10 * Rnd()) 'Must be positive
    bz = (20 * Rnd() - 10)
    cz = (20 * Rnd() - 10)
    dz = (20 * Rnd() - 10)


  'Advance the particle until it enters the current viewport.
    Alive = True
```

```
        'temp value to keep the particle alive while it is advanced
          Age_Steps = 30000
        'inital point that will usually be outside the viewport
          MoveToStep(-10)
        '170 is fixed minimum value for our viewport
          Do While Alive And z + ddz < -170
              MoveToNextStep()
          Loop

          'Range for age is dependant on 'when' the particle
              'enters the viewport
          'Constants below were selected via experimentation to
              'produce desired visual result
          Dim tmp_t As Integer = -1 * t / 0.02
          Age_Steps = CInt(Int(0.6 * tmp_t * Rnd()) + 1.4 * tmp_t)
    End Sub

    'Forces the particle to a particular 'step' and recalculates
position
    Private Sub MoveToStep(ByVal newT As Double)
        t = newT
        x = mx * t + bx
        y = my * t + by
        z = az * t ^ 3 + bz * t ^ 2 + cz * t + dz
    End Sub

      'Advances the particle one step in the simulation and
      'calculates position.
    Public Sub MoveToNextStep()
        MoveToStep(t + 0.02)
        If (z + ddz > 200) Then 'outside of the viewport
            Kill()
        End If
        getOlder()
    End Sub
End Class
```

Appendix C: Implementation Source - Lattice Boltzmann Model (LBMPar.vb)

```
' Class LBMPar
' Author - Clifford Lee Wiggs
' Date - July 2006
' Language - VB.Net 2.0
'
' This class represents a single particle in a particle system.
' It uses a 3D Lattice-Boltzmann Model (LBM) to control its path.

Public Class LBMPar
      'Current location of the particle in 3D space
    Private x As Double
    Private y As Double
    Private z As Double

      'Current location(cell) of the particle in LBM Lattice
    Private tx As Integer
    Private ty As Integer
    Private tz As Integer

    'is this Paricle currently 'alive'?
    Private Alive As Boolean

    'Age fo the current particle - counts down to zero
    Private Age_Steps As Integer

    'Shared Variables for size of LBM Lattice
    Public Const max_x As Integer = 10
    Public Const max_y As Integer = 10
    Public Const max_z As Integer = 20

      'Shared variables for LBM Lattice and temp lattice for
calculations
      'All partices exist on the same lattice
    Private Shared f(max_x, max_y, max_z, 18) As Double
    Private Shared f_new(max_x, max_y, max_z, 18) As Double

    'Unit Vectors for each path (qi)
    Private Shared ex() As Integer = {0, 0, 0, 1, -1, 0, 0, 0, 0, 1, -
1, 1, -1, 1, -1, 0, 0, 1, -1}
    Private Shared ey() As Integer = {0, 0, 0, 0, 0, 1, -1, 1, -1, 0,
0, 1, 1, -1, -1, 1, -1, 0, 0}
    Private Shared ez() As Integer = {0, 1, -1, 0, 0, 0, 0, 1, 1, 1, 1,
0, 0, 0, 0, -1, -1, -1, -1}

    'Used for Bouncing off Ground during propogation
    'Only needs to be defined where ez() = -1
```

```
    Private Shared BounceBack() As Integer = {0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 7, 8, 9, 10}

    'Coefficients for feq (the equilibrium distribution function)
    Private Shared A() As Double = {1 / 3, 1 / 18, 1 / 18, 1 / 18, 1 /
18, 1 / 18, 1 / 18, 1 / 36, 1 / 36, 1 / 36, 1 / 36, 1 / 36, 1 / 36, 1 /
36, 1 / 36, 1 / 36, 1 / 36, 1 / 36, 1 / 36}
    Private Shared B() As Double = {0, 1 / 6, 1 / 6, 1 / 6, 1 / 6, 1 /
6, 1 / 6, 1 / 12, 1 / 12, 1 / 12, 1 / 12, 1 / 12, 1 / 12, 1 / 12, 1 /
12, 1 / 12, 1 / 12, 1 / 12, 1 / 12}
    Private Shared C() As Double = {0, 1 / 4, 1 / 4, 1 / 4, 1 / 4, 1 /
4, 1 / 4, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 /
8, 1 / 8, 1 / 8, 1 / 8}
    Private Shared D() As Double = {-1 / 2, -1 / 12, -1 / 12, -1 / 12,
-1 / 12, -1 / 12, -1 / 12, -1 / 24, -1 / 24, -1 / 24, -1 / 24, -1 / 24,
-1 / 24, -1 / 24, -1 / 24, -1 / 24, -1 / 24, -1 / 24, -1 / 24, -1 / 24}

      'Values obtained from
    'http://www.engineeringtoolbox.com/air-properties-d_156.html
    Const Viscosity As Double = 0.00001511 'Viscosity of Air at 20C
    Const Rho_0 As Double = 1.205 'Density of Air at 20C

      'Inital Density for Inlet cells
      'If this value is too large it will cause instability
      'in the lattice.
    Const Rho_Delta As Double = 2

      'Calculated relaxation time
    Private Shared tau As Double = (3.0 * Viscosity) + 1.0 / 2.0

      'Utility function to expose the lattice.
    Public Shared Function Get_f(ByVal x As Integer, ByVal y As
Integer, ByVal z As Integer, ByVal i As Integer) As Double
        Get_f = f(x, y, z, i)
    End Function

      'Kill the current particle
    Public Sub Kill()
        Alive = False
    End Sub

    'Mark the current particle as alive
    Public Sub Birth()
        Alive = True
    End Sub

    'Query if the current particle is alive
    Public Function isAlive() As Boolean
        isAlive = Alive
    End Function

      'Age the current particle and kill when appropriate
    Private Sub getOlder()
        Age_Steps -= 1
        If (age_steps <= 0) Then Kill()
    End Sub
```

```
      'Accessors for the current point
      'Adjusts to fir viewport before returning
    Public Function getX() As Double
        getX = (x - max_x / 2) * 10
    End Function
    Public Function getY() As Double
        getY = (y - max_y / 2) * 10
    End Function
    Public Function getZ() As Double
        getZ = z * 16 - 150
    End Function

      'Initializer for the particle
    Public Sub Init()

      'We are alive!
        Alive = True

        'Random initial location in Lattice
        tx = CInt(Int((max_x - 1) * Rnd())) + 1
        ty = CInt(Int((max_y - 1) * Rnd())) + 1
        tz = 1 'Start one step above the inlet nodes

        'Emperical tests show that age is usually between 200
            'and 400 when exiting the lattice
        'Choose a random age within this range
        Age_Steps = CInt(Int(200 * Rnd())) + 200

        'Initial 'world' location is same as lattice location
        x = tx
        y = ty
        z = tz
    End Sub

      'Advances the particle one step in the simulation and
      'calculates position.
    Public Sub MoveToNextStep()
        If (tx < 0 Or tx > max_x Or ty < 0 Or ty > max_y Or tz < 0 Or
tz > max_z) Then
            Kill()      'Can not exist outside of lattice
        Else
            Dim rho, vx, vy, vz As Double
            Dim i As Integer

      'Calculate Rho(density) for the current lattice cell
            rho = 0.0
            For i = 0 To 18
                rho += f(tx, ty, tz, i)
            Next

      'Calculate Velocity Vector for the current lattice cell
            vx = 0.0
            vy = 0.0
            vz = 0.0
            For i = 0 To 18
                vx += f(tx, ty, tz, i) * ex(i)
                vy += f(tx, ty, tz, i) * ey(i)
```

```
                vz += f(tx, ty, tz, i) * ez(i)
            Next
            vx /= rho
            vy /= rho
            vz /= rho

      'Move the current position by the amount of the current velocity
            x += vx
            y += vy
            z += vz

                'Calculate the lattice location from the current
point
            tx = Math.Round(x)
            ty = Math.Round(y)
            tz = Math.Round(z)

            getOlder()
        End If
    End Sub

      'Shared method for advancing the lattice one time step
    Public Shared Sub Lattice_Step()
        collision()
        propagate()
    End Sub

      'Initalize the lattice
    Public Shared Sub Init_Lattice()
        'initialise mass and momentum
        'i.e density and velocity
        'Set all packet distributions to equilibrium distribution
            'with zero velocity and constant density (1)
        Dim x, y, z, i As Integer

        'When velocity is zero, then f_eq is simplified to just a() *
rho
        For z = 0 To max_z
            For y = 0 To max_y
                For x = 0 To max_x
                    For i = 0 To 18
                        f(x, y, z, i) = A(i) * Rho_0
                    Next
                Next
            Next
        Next

        'Throw away first few cycles to provide stability for
      'flow from inlet cells
        Lattice_Step()
        Lattice_Step()
        Lattice_Step()
        Lattice_Step()
        Lattice_Step()
    End Sub

      'Performs the collision step in the LBM lattice
```

```
    Private Shared Sub collision()
        Dim x, y, z, i As Integer 'indexes to lattice
        Dim rho As Double 'density
        Dim vx As Double 'x-velocity
        Dim vy As Double 'y-velocity
        Dim vz As Double 'z-velocity
        Dim vv As Double 'v squared for pre-calculation
        Dim ev As Double 'e dotproduct v for pre-calculation
        Dim f_eq(18) As Double 'equilibrium distribution for all paths

        'Calculate for a single cell at a time and for all cells
        For z = 0 To max_z
            For y = 0 To max_y
                For x = 0 To max_x
                    'In the cases of an 'inlet' node, we have a
                        'constant velocity and density
                    If (z = 0) And ((x Mod 4 = 0) And (y Mod 4 = 0)) Or
(((x + 2) Mod 4 = 0) And ((y + 1) Mod 4 = 0)) Then
                        rho = Rho_Delta
                        vx = 0
                        vy = 0
                        vz = 0.1
                    Else

                'Calculate Rho - density for current cell while
iterating
                        rho = 0.0
                        For i = 0 To 18
                            rho += f(x, y, z, i)
                        Next

                'Calculate Velocity Vector for current cell while
iterating
                        vx = 0.0
                        vy = 0.0
                        vz = 0.0
                        For i = 0 To 18
                            vx += f(x, y, z, i) * ex(i)
                            vy += f(x, y, z, i) * ey(i)
                            vz += f(x, y, z, i) * ez(i)
                        Next
                        vx /= rho
                        vy /= rho
                        vz /= rho
                    End If

                  'Calculate f_eq (equilibrium distribution values
                  'for this cell)
                    vv = vx * vx + vy * vy + vz * vz 'Vector squared
                    For i = 0 To 18
                        ev = vx * ex(i) + vy * ey(i) + vz * ez(i)
                        ' Dot Product
                        f_eq(i) = rho * (A(i) + B(i) * ev + C(i) * ev *
ev + D(i) * vv)
                    Next

                  'Calculate Omega (collision distribution) and
```

```
                    'populate f_new (temp lattice)
                      For i = 0 To 18
                          f_new(x, y, z, i) = f(x, y, z, i) - (1 / tau) *
(f(x, y, z, i) - f_eq(i)) 'Omega

                          'No such thing as a negative density (ie
                          vacuum)
                          If (f_new(x, y, z, i) < 0) Then
                              f_new(x, y, z, i) = 0
                          End If
                      Next
                  Next
              Next
          Next
      End Sub

        'Performs the propagation step in the LBM lattice
      Private Shared Sub propagate()
          Dim x, y, z, i As Integer         'indices for lattice
          Dim nx, ny, nz, ni As Integer     'New indicies for lattice

          'For every cell and path
          For x = 0 To max_x
              For y = 0 To max_y
                  For z = 0 To max_z
                      For i = 0 To 18
                          'Calculate the new x,y,z,i - ie which cell
                          'the packet distribution is moving to
                          nx = x + ex(i)
                          ny = y + ey(i)
                          nz = z + ez(i)
                          ni = i

                          If (z = 0 And ez(i) < 0) Then
                          'Bottom of Simulation moving down
                              'Bounce Back off Ground
                              'Stay at same z height
                              'With a new 'i' direction
                              ni = BounceBack(i)
                              nz = z

                          'Calculate periodic boundary for corners and
sides
                              If (nx < 0) Then nx = max_x
                              If (nx > max_x) Then nx = 0
                              If (ny < 0) Then ny = max_y
                              If (ny > max_y) Then ny = 0
                          ElseIf (z = max_z And ez(i) > 0) Then
                          'Top of Simulation
                          'Float off into the air
                          'At 'Outlet' Nodes we simply leave the
simulation
                              'Don't put this value anywhere
                              Continue For
                          Else 'Inner part or sides of simulation
                              'Periodic Boundary Conditions
```

```
                            'As we move out, reenter on the opposite
side
                            'We keep the same i, but get a new x,y,z
                            If (nx < 0) Then nx = max_x
                            If (nx > max_x) Then nx = 0
                            If (ny < 0) Then ny = max_y
                            If (ny > max_y) Then ny = 0
                            If (nz < 0) Then nz = max_z
                            If (nz > max_z) Then nz = 0
                        End If
                        'Store the post-collison packet distribution
                        'back into the main lattice in its
                        'post-propagation location
                        f(nx, ny, nz, ni) = f_new(x, y, z, i)
                    Next
                Next
            Next
        Next

    End Sub
End Class
```

VITA

Clifford Lee Wiggs, Jr.

Candidate for the Degree of

Master of Science

Thesis: EMPIRICAL COMPARISON OF THREE EXISTING METHODS FOR SIMULATING EMBERS IN COMPUTER-GENERATED FIRE

Major Field: Computer Science

Education: Graduated from Turner High School, Kansas City, Kansas in May 1993; received Bachelor of Science degree in Mathematics and a Bachelor of Science degree in Computer Science from Oklahoma Christian University of Science and Arts, Oklahoma City, Oklahoma in May 1997. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 2006.

Name: Clifford Lee Wiggs, Jr. Date of Degree: December 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: EMPIRICAL COMPARISON OF THREE EXISTING METHODS FOR SIMULATING EMBERS IN COMPUTER-GENERATED FIRE

Pages in Study: 90 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: The purpose of this study was to implement computer-generated embers using three existing methods and then compare them. While there exists research related to fire propagation, flame rendering, and smoke generation, there is nothing that discusses embers. This study focuses specifically on the generation of an ember's path when it is animated. The three methods selected were the Lattice Boltzmann Model, a 3D fractal tree, and a particle system that uses third-order polynomials. The three methods were implemented and then compared using three different metrics: memory requirements, time requirements (computational cost), and a subjective visual representation.

Findings and Conclusions: A different methods performed best in each of the three metrics. The particle system had the lowest memory requirements. The 3D fractal tree had the least computational cost. The Lattice Boltzmann Model had the best visual representation. Over all, when rendering embers in a real-time environment, the 3D fractal tree is the best choice. However, if pre-rendering your images, then the Lattice Boltzmann Model is the best choice.

Adviser's Approval: Dr. Blayne Mayfield