

University of Nevada, Reno

# **An Advanced Wildfire Simulator: vFirelib.v2**

A thesis submitted in partial fulfillment of the  
requirements for the degree of Master of Science  
in Computer Science and Engineering

by

Andy Mario Garcia

Dr. Frederick C. Harris, Jr., Thesis Advisor

December, 2018



THE GRADUATE SCHOOL

We recommend that the thesis  
prepared under our supervision by

**ANDY MARIO GARCIA**

Entitled

**An Advanced Wildfire Simulator: vFirelib.v2**

be accepted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE

Dr. Frederick C. Harris, Jr., Advisor

Dr. Sergiu M. Dascalu, Committee Member

Dr. Scotty Strachan, Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

December, 2018

## Abstract

Wildfires can cause severe amounts of damage to wildlife habitat as well as commercial and residential properties. They put at risk the well-being of the environment and the firefighters who combat them to minimize damage and ensure the safety of the public. A fire simulator that is accurate, real-time, and convenient to use can be of great assistance in developing the most efficient and safest plan to stop a wildfire's spread. This thesis presents an advanced wildfire simulator. The application has two components: a web-based application and a 3D virtual reality application. The web-based application offers portability, speed, and ease of use. This has the added advantage in supporting many different platforms and devices. The 3D virtual reality application provides a real-time on the ground scene perspective. Using a GPGPU framework we can calculate faster than real-time results of an entire wildfire simulation employing the fire model developed by Richard C. Rothermel. This is one of the most widely used fire spread models among wildfire simulators. Taking advantage of GPU's massively parallel computational power the simulator can compute a large wildfire simulation in a matter of seconds. The wildfire simulator has been advanced with features such as dynamic wind, moisture, and vegetation fuel which allow users to input changes to the environment that can include rain, water drops by helicopter and or plane, curing or saturation of vegetation, firebreaks created by bulldozers or firefighters, and even controlled burns. On top of those features the ability to enable spotting fire effects due to crown fires has been implemented as well as moving a majority of the sequential preprocessing necessary for the fire simulator onto the GPU thus decreasing overall run time.

## Dedication

I dedicate this thesis to my mother and brother for their overwhelming support.

## Acknowledgments

I would like to thank my adviser, Dr. Frederick C. Harris, Jr., and my committee members Dr. Sergiu M. Dascalu and Dr. Scotty Strachan for their time and suggestions. I'd like to give thanks to Chase Carthan for his help early on with GDAL. I'd like to thank Nolan Burfield with his help in debugging, file I/O, and Unity's game engine. I'd like to thank Roger Hoang and Jessica Smith for creating the foundation which made the advancements on this fire simulator possible, and I would like to thank Rui Wu for his help in debugging, his work on the web service, and his encouragement when I needed it most.

This material is based upon work supported in part by the National Science Foundation under grant numbers IIA-1301726 and IIA-1329469, and by Cubix Corporation through use of their PCIe slot expansion hardware solutions and HostEngine. Any opinions, finds, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or Cubix Corporation.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Dedication</b>	<b>ii</b>
<b>Acknowledgments</b>	<b>iii</b>
<b>List of Tables</b>	<b>vi</b>
<b>List of Figures</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background and Related Work</b>	<b>5</b>
2.1 Fire Science . . . . .	5
2.1.1 Taxonomy of Fire Models . . . . .	6
2.1.2 Fire Shape . . . . .	6
2.1.3 Surface Fire . . . . .	10
2.1.4 Crown Fire . . . . .	12
2.1.5 Spotting . . . . .	13
2.2 Fire Simulators Past and Present . . . . .	14
2.2.1 BEHAVE . . . . .	15
2.2.2 fireLib . . . . .	16
2.2.3 FARSITE . . . . .	16
2.2.4 vFire . . . . .	16
2.2.5 vFireLib . . . . .	17
2.3 Fuel Models . . . . .	18
2.4 GPGPU Computing and Architecture . . . . .	19
2.5 GDAL . . . . .	23
<b>3 Software Engineering</b>	<b>25</b>
3.1 Overview . . . . .	25
3.2 Fire Simulation Requirements . . . . .	25
3.2.1 Functional Requirements . . . . .	26
3.2.2 Non-functional Requirements . . . . .	27
3.3 Use Case Modeling . . . . .	27

<b>4</b>	<b>Implementation</b>	<b>33</b>
4.1	Overview . . . . .	33
4.2	Fire Propagation Models and Their Sequential Implementations . . .	34
4.2.1	Burn Distance Model . . . . .	36
4.2.2	Minimal Time Model . . . . .	38
4.2.3	Iterative Minimal Time Model . . . . .	39
4.3	Parallel Implementation . . . . .	40
4.3.1	Minimal Time . . . . .	41
4.3.2	Iterative Minimal Time . . . . .	42
4.3.3	Burn Distances . . . . .	43
4.4	Other Features . . . . .	43
4.4.1	Spotting . . . . .	44
4.4.2	Pause and Resume . . . . .	45
4.4.3	Moisture, wind, and Vegetation Manipulation . . . . .	46
<b>5</b>	<b>Results</b>	<b>48</b>
5.1	Results . . . . .	48
5.1.1	Overview . . . . .	48
5.1.2	Web Service Architecture . . . . .	52
5.2	Web-Based Visualization Application . . . . .	53
5.2.1	Overview . . . . .	53
5.3	3D Visualization . . . . .	58
5.3.1	Overview . . . . .	58
5.3.2	Fire Integration . . . . .	58
<b>6</b>	<b>Conclusions and Future Work</b>	<b>61</b>
6.1	Conclusions . . . . .	61
6.2	Future Work . . . . .	61
	<b>Bibliography</b>	<b>63</b>

# List of Tables

3.1	The functional requirements of vFireLib.v2 . . . . .	26
3.2	The Non-functional requirements of vFireLib.v2 . . . . .	27
5.1	Execution times without Spotting . . . . .	50
5.2	Execution times with Spotting . . . . .	50

# List of Figures

2.1	Fire Propagation Methods [22] . . . . .	11
2.2	Spotting Fire Phenomenon [21] . . . . .	14
2.3	The Standard 40 Fuel Models [18] . . . . .	20
2.4	GPU SM CUDA Programming [35] . . . . .	22
2.5	GPU Architecture [35] . . . . .	23
3.1	Use Case Diagram . . . . .	28
4.1	Probability of Ignition [21] . . . . .	45
4.2	Dynamic Fuel Load Transfer Table [21] . . . . .	47
5.1	Timings in seconds for all the implementations with spotting. . . . .	49
5.2	Speedup graph found by dividing CPU/GPU running times. . . . .	51
5.3	$\log_2$ based graph of the speedup. . . . .	52
5.4	System Structure (Blue Rectangles–Server; Green Rectangles–Client)	54
5.5	Web-based Client Screenshot, the red color means a cell is burning, the black color means a cell is unburnable, and other colors stand for different vegetation types. The user can start/stop the simulation and also choose cells to set on fire. . . . .	55
5.6	Vegetation Modification Bar. The user can change the vegetation on the 2D grid map by using the bar. When the cursor hovers over the button, detailed information will pop up (this can be seen in the white text on black background). The current version supports eight types of vegetation. . . . .	56
5.7	Vegetation Modification Map. This is an example that some cells (in the middle of the figure) are changed into unburnable. It is clear that these cells stop the fire spreading. . . . .	57
5.8	Wind Modification of Chosen Area. The user can change the 2D wind vectors in each cell. These vectors are represented by arrows. Longer arrows are used to represent more forceful wind and the arrow direction is the same as the cell wind direction. . . . .	57
5.9	The Unity application where the white edges show where the fire edge is at this point in the simulation and the yellow to black coloring of the terrain shows the early to late time of arrival of the fire at that point. . . . .	59

# Chapter 1

## Introduction

Wildfires can cause serious problems for populations and the environment if not controlled. In 1963, a series of wildfires burned 183,148 acres woodland, destroyed 186 homes, 197 outbuildings, and seven people died in the disaster [33]. There are advanced technologies that help firefighters and their suppression efforts as well as keeping them safe. Technologies such as fire shelters, a last line of defense if a wildfire surrounds any firefighter in the field they can deploy this material around themselves and hopefully survive an oncoming fire. Improved wind models help firefighter's determine how to safely combat a fire because fire reacts so strongly with wind variables this can help avoid the situation where a firefighter needs to deploy a fire shelter and help determine where a fire may divert to and how quickly [43]. Even with these advance technologies wildfires can still be a challenge today. On May 3rd, 2016, the wildfire in Fort McMurray, Canada consumed around 2,400 buildings and homes. It forced the biggest wildfire evacuation in Alberta's History [31].

An accurate and real-time wildland fire simulator with a visually appealing interface is a significant area of research [3]. Firefighters can use these tools to estimate how long the wildfire will burn and how it will spread across the given terrain. If this information is provided, suppression efforts and resources can be employed to execute the most efficient, swift and safe plan of action. Educators can use the tools as well to teach their communities about wildfire danger and prevention. For example, a teacher or community leader can simulate wildfires in certain geological relevant locations and provide where to evacuate and how much time they may have before

the wildfire reaches certain areas based on the simulation.

In order to simulate wildfires, fire scientists have developed fuel models and equations to estimate the propagation of fire [4, 32, 36]. The spread of a wildfire is affected by the properties of the environment. Such properties include, but are not limited to: fuel load, fuel type, wind, live moisture, dead moisture, precipitation, canopy cover, crown height, and wind. Incorporating these and other variables into a wildfire simulator can allow for an accurate prediction of where a fire will burn to and how quickly it will arrive there. Research into developing fuel and moisture models is an active field, and these models provide the basis for the properties on which this simulator is based. The ability to realistically simulate wildland fires is desirable to provide fire experts the ability to more accurately predict the impact of their fire-fighting decisions [24, 27]. User imposed manipulations to the wildfire environment include adjusting moisture content to simulate a water drop, adjusting fuel loads where a simulated bulldozed tree line could exist, or reverse spread testing in which a fire is started by firefighters in order to burn the fuel away from the advancing wildfire. Unfortunately, the amount of data required for realistic fire simulations requires a large amount of computation time to produce an accurate simulation [3]. The more accurate and fine-grained the simulation, the longer it takes to process the data. A forest fire is a dynamic entity, therefore a simulators ability to run in real time or even more pertinent to run in faster than real-time is necessary for it to be an effective tool. The more complex and functional a wildfire simulator is, the more useful it is to fire scientists, firefighters, and at risk communities. There are multiple aspects to accurately model the spread of a wildfire. The main four properties which influence the spread of a fire are: base fires, crown fires, fire acceleration, and spotting [29]. Base fire spread is the foundation for all fire spread simulations that occurs along the floor of a forest. Crown fires occur when the forest fire spreads into the tops of the trees. A crown fire may be passive or active. An active crown fire is one which contributes to the overall spread of the fire. A passive crown fire will burn in the tops of the trees but is not hot enough to contribute to the overall spread of the

fire. Fire acceleration is the phenomena that accounts for the dynamic speed of a fire. A forest fire will not automatically spread at the maximum rate for which it has the potential. Depending on the properties previously mentioned, such as the moisture within certain types of foliage, the acceleration can build gradually (due to high moisture content) or quickly (due to low moisture content). Spotting is the phenomena of fire embers being blown forward ahead of the fire to ignite new fires in zones isolated from the main body of the fire. The implementation of these four main aspects that are natural occurrences in all fires corresponds to a state of the art fire simulator. However, constructing such a simulator that is real-time and accurate has to calculate an enormous amount of data; therefore GPGPU computing is a necessity.

Utilizing the GPU as a general purpose computing device is now ubiquitous among software applications, particularly on problems requiring a large amounts of data processing. The GPU is ideally suited to high volume data processing applications through its ability to processes millions of inputs simultaneously and in appropriate conditions asynchronously. In comparison, a CPU may process only one or a few at a time [35] even if its core clock speed on threads is significantly higher, the sheer number of single instruction, multiple data processors or SIMD contained in a GPU can outperform for certain tasks. This thesis builds upon the work done by Roger Hoang creator of vFire [22] and Jessica Smith creator of vFireLib [39].

There are many wildfire simulation and visualization tools. However, most of them do not offer the tool as a service, which means the performance depends on the user's machine. Our tool uses server-client architecture and offers the simulation as a service. Therefore, as long as the user's device supports our client application (a browser window or a Unity application), they can construct wildfire simulations and visualize the results. Because we use the GPU to do most of the calculations, a fast simulation is guaranteed. All the calculations are done on the server, so there is no need for the user's device to have a GPU.

The remainder of this thesis is structured as follows. Chapter 2 provides background and a review of the literature. Chapter 3 focuses on software engineering,

examining functional and non-functional requirements. Chapter 4 details the simulator's implementation. Chapter 5 provides test cases and instructs how the simulator is run. Chapter 6 presents conclusions and future work.

# Chapter 2

## Background and Related Work

In 1972 Richard C. Rothermel published “A Mathematical Model for Predicting Fire Spread in Wildland fuels” as a research paper with the United States Department of Agriculture Forest Service [32]. This is the cornerstone for nearly all fire simulators to date. In 1976 Albini published this paper [1] that elaborated upon the 11 fuel models Rothermel had created and added two of his own fuel models in the mix. These fuel models are considered the original 13 and are still in use today when dealing with and creating a wildfire simulator. The rest of this chapter will elaborate on the fire science equations used to predict wildfire behavior, where to find such datasets needed for an accurate simulation, an overview of past and present fire simulators, and will give insight on GPU computing and data interpretation.

### 2.1 Fire Science

Creating an accurate real-time wildfire simulator has been researched heavily for decades [5, 9, 18, 32]. However it has only been achieved fairly recently; within the last 30-40 years, a few have been developed. Most of all the developed fire simulators all follow certain fire science principles. For this thesis we followed Rothermel’s fire spread representation of an elliptical pattern [2, 12]. An elliptical fire burn shape seems to be a reliable phenomenon in nature as represented by [2, 12].

### 2.1.1 Taxonomy of Fire Models

The fire model this thesis incorporates is considered semi-empirical. The reason for this is that it is a mixture of theoretical and empirical fire modeling. Theoretical is used for research only and includes physical principals; this model alone is not sufficient enough to base an entirely accurate fire simulator upon. On the other hand, the empirical model is done in tightly controlled labs lacking any physical principle and purely statistical. Empirical models being so constrained does not support the variability of nature and cannot be used alone. Therefore, combining the two will give our fire model the best advantages of both and provide an accurate and adaptable fire simulator [39]. The decision to make this fire simulator semi-empirical provides enormous value when simulating actual events in real-time. This is necessary if fire suppression efforts are to trust the simulator in its accuracy to deploy resources effectively. Using this semi-empirical approach allows for the fire simulation to be as accurate as possible to a natural phenomenon that can be mathematically modeled [32]. It also provides the possibility to validate the fire simulators accuracy by modeling historical fires if the datasets exist or can be procured.

### 2.1.2 Fire Shape

In order to build a wildfire simulator, wildland specific data must be integrated. Rothermel developed the first eleven fuel models that are still used to this day [32]. The method by which these eleven fuel models were created is the basis for the development of all modern fuel models. The fuel model contains information on properties of the wildland in a particular region. The topography is broken up into cells which creates a certain granularity aspect; known also as resolution. Each cell contains properties which are in comparison with the appropriate fuel models, moisture levels, wind speed and direction at the given moment in time or season. The cell's resolution can be fine grain which adds more cells to the computation time and load resulting in a more accurate fire spread. An example of cell size is such that a cell can be a  $30 \times 30$  meter grid containing mostly shrub fuel type models, therefore that cell will

offer the fuel and moisture of a 30x30 meter dense region of shrubbery to be burned and calculated as such. Opposed to a fine grain simulation where that same 30x30 meter area can be split into 9 cells comprised of 10x10 meter grids and some will be classified as shrub fuel type models but possibly due to the higher resolution a cell or two may be described as grass fuel type models which will offer a more accurate simulation. The finer grain may not even provide a difference in fuel, but because of increased detail fire spread accuracy and timing will be increased. This simulator is confined to the resolution of the data; therefore the accuracy of both the fire model computation and real-world prediction is limited to the available datasets granularity. Most terrain, fuel, wind, and moisture data is default to 30x30 meter cells and can be found here [42]. LANDFIRE is a program established by the United States Department of the Interior and maintained by the United States Geological Survey or USGS department. More detail on how to extract the data needed for the wildfire simulator will be detailed in the fuel model section of this chapter.

Almost all of the previous forest fire simulators implement Rothermel's fire spread equations [32]. This thesis works off the same fire science to implement all three burn methods. The burn method for cells catching fire and checking their neighbors conditions differ, but the calculations are all the same in terms based on Rothermel's Equation 2.1 providing a rate of spread for fire, which is based on fuel models, terrain, weather, and suppression efforts.

$$R = \frac{(I_p)_o(1 + \phi_w + \phi_s)}{\rho_b \varepsilon Q_{ig}} \quad (2.1)$$

Where

$(R)$  = is the rate of spread.

$(I_p)_o$  = is the no-wind propagating flux.

$\phi_w$  = is the additional propagating flux introduced by wind.

$\phi_s$  = is the additional propagating flux introduced by slope.

$\rho_b$  and  $\varepsilon$  = is the effective bulk density.

$Q_{ig}$  = is the heat of pre-ignition.

The values can be found in the cells describing fuel model or can be calculated depending on the simulation's conditions. Certain values such as  $\rho_b$ ,  $\varepsilon$ , and  $Q_{ig}$  describe certain variables that do compel more detail.  $\rho_b$  and  $\varepsilon$  are considered to represent Effective Bulk Density. Effective Bulk Density is the amount of fuel per unit volume of the fuel bed raised to ignition ahead of the advancing fire.  $Q_{ig}$  represent the heat of pre-ignition; defined as the heat required to bring a unit weight of fuel to ignition.

Using the rate of spread equation provided, the basic output of the fire simulator is a time of arrival map or known as TOA for short. This map is the time that the fire will have spread to the cell in its respective location. Once the fire has arrived to a cell, use of one of the three burn methods presented in this thesis are implemented to determine where the fire will spread, and how fast it will spread. Essentially each cell has a finite fuel determined by the fuel model that is represented in the cell. Once this fuel is burned all the way through the cell no longer contributes to the fire line intensity or spread and has either been suppressed or the fire has moved forward. There are other factors that affect the TOA map and those variables are explained in the subsequent sections. These sections are labeled manipulative, because the user can superimpose changes to these variables in order to create a more accurate and precise simulation in context of when and where a fire may be engaged.

### **Wind Manipulation**

The ability to change wind in the middle of an ongoing simulation is a very important feature to consider. Firefighting suppression efforts can change with weather changes in an instant. Wildfires can even create their own wind effects which can have an overall effect on the fires behavior [25]. There is also the possibility that abrupt wind changes can create dangerous and even deadly conditions for firefighters and civilians. Wind can affect the spread of a fire in many ways other than just increasing its speed. Wind speed can affect fire phenomenon such as whether a crown fire is passive or active, an active crown fire can induce an extreme change to the spread

of a wildfire. Wind is also a variable in spotting effects when embers are released from active crown fires that can set fires ahead of the initial, main fire line. Wind in this thesis was implemented on a 2D scale that can be toggled on or off for testing and research purposes. Toggled on, the simulator will include an X and Y vector containing the wind speed in the corresponding direction in terms of mile per hour. These vectors are used in Rothermel's equation which can increase or decrease the spread of the fire. There is an X and Y wind speed value for each cell along with the fuel model data. To manipulate the wind data a user can change the file directly, which is slow and inconvenient, or use the web-service provided by this thesis to simply click and drag a box around the corresponding area on a map of the terrain. The user selected cells can then be changed accordingly. This will be sent to the simulator and elicit an immediate effect upon the spread of the fire. More detail on the web-service will be provided in later chapters.

### **Vegetation Manipulation**

Change in vegetation is another important feature a fire simulator should possess in order to see how fire suppression efforts can change the spread of a wildfire. examples of such changes in vegetation due to suppression efforts can include controlled burns, bull-dozing, and or cut lines. Controlled burns are essentially small fires started by firefighters ahead of the main fire line in an attempt to burn fuel ahead of the oncoming flames to starve it out and render the spread to peter out. When bull-dozing firefighters will use large machinery to sweep away fuel from the path of a wildfire. Cut lines do exactly what bull-dozing does, but it is completed by pure man power, picks, shovels, and axes. These are just a few examples of how vegetation can change and therefore the simulator presented in this work allows the user to change the vegetation at any time the same way wind is manipulated. The only difference is that the fuel model number will be changed to rock for all three of the above examples. Rock is the best representation of fuel for these situations essential due to the fact that rock as a fuel is unburnable. If the bulldozer, control burn, and or

cut line is enacted, then essentially there is no fuel for the wildfire to consume in that area and would burnout. Significant vegetation change can happen naturally due to seasonal conditions. If the user wanted to change the simulation to this degree the manipulation option offered by this thesis's web-service may be inefficient at completing this task. LANDFIRE's resources do offer seasonal vegetation in TIFF file format which the simulator application can read based on latitude and longitude.

### **Moisture Manipulation**

Much like the necessity for wind and vegetation manipulation, moisture changes are just as important. Moisture changes can happen due to natural and artificial phenomena. Natural moisture change is due to curing of fuels which is when humidity, sunlight, and temperature can dry out various fuels and can add to the rate of fire spread [18, 21]. Another natural moisture change is precipitation, if rain were to occur during a wildfire, depending on where the rain is being dropped, can reduce the threat and rate of spread. Artificial changes to moisture include water drop by helicopter or plane in suppression efforts. These changes in moisture are abrupt, unlike precipitation which is gradual. Moisture changes the values held within a fuel model, and as mentioned in Rothermel's equation, these values are what directly affect the rate of spread. Therefore changing moisture is pertinent to the fire simulator presented. The user can, once again, change it the same way wind and vegetation are changed with simply a click and drag on the web-service. How moisture changes the rate of spread will be discussed in later chapters; for now it is simply enough to understand how and why moisture change is a significant implementation in an accurate fire simulator.

### **2.1.3 Surface Fire**

Now that the various elements that can affect the spread of a fire have been discussed the different approaches to calculating the propagation of a fire is necessary. The propagation method also determines the method used to calculate the dynamic

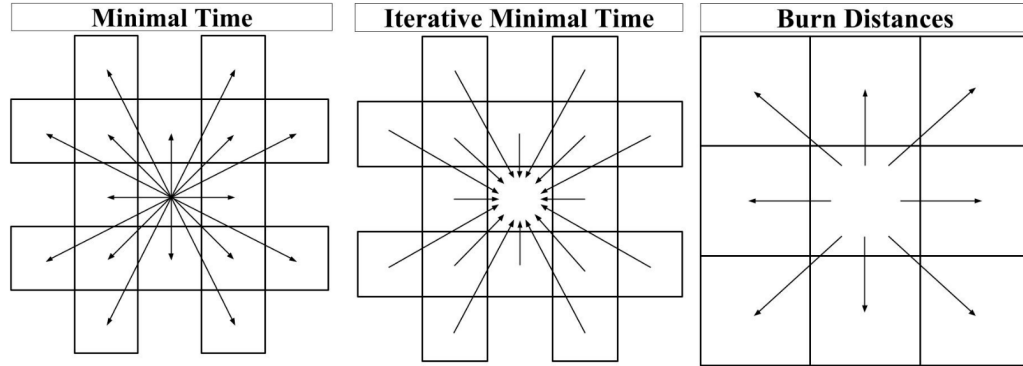


Figure 2.1: Fire Propagation Methods [22]

timestep of the simulation. This timestep is used to iterate through the simulation and is equivalent to approximately 100 seconds time in the real world scenario of a wildfire event. This thesis present three ways to calculate the time of arrival map. These are illustrated in Figure 2.1. All three burn propagation methods utilize a dynamic time step for race conditions that arose from attempting static time steps; this will be looked at later for clarification.

### **Burn Distances**

This propagation method is intuitive in the sense of how fire spreads and burns throughout the sim. The simulation is broken into cells which represent a 30x30 meter grid of a certain fuel model. The use of these cells calculate the fire spread using the time it takes for the fire to burn completely across the cell to the next. The distance is dynamic in the sense that it is appropriated to the cell's type of fuel model and the cell size read from file. The fire burns through this distance and checks which of the cell's neighbors burn from the center cell outwards; this is represented in Figure 2.1. The Burn Distance method is based on previous work done at the University of Reno Nevada by Roger Hoang's vFire [22].

### **Minimal Time**

This propagation method is determining the smallest or shortest time of arrival value a cell with have. This is achieved on every time step a cell checks if it is on fire. If

it is on fire then nothing is calculated, this saves computation time ensuring there is no redundancy. If the cell is not on fire it will check its neighbors to see if they are on fire. The cell then compares the current time with the time of arrival in its neighboring cells to see if they are smaller, the fire arrives faster, and will store the faster time within it. Essentially the cell that is not lit is checking its neighbors for the fastest time of arrival being computed based of Rothermel's fire spread equation. The method on how the cell checks can be seen in Figure 2.1 as well. In previous work done by Sousa, dos Reis, and Pereira this method was the most time intensive of their three methods [41].

### **Iterative Minimal Time**

Propagating cellular fire with this method was first done by the same Sousa, dos Reis, and Pereira paper mentioned in Minimal Time. Each cell calculates its own time of arrival ignition value every time step. However, the cell does not stop doing this until its ignite time and the difference of one of its neighbors ignite times surpasses a certain threshold. Figure 2.1 shows how the cell accesses its neighbors ignite times. The threshold is found through experimentation, and produced the fastest simulation computation speed in the work of Sousa, dos Reis, and Pereira.

#### **2.1.4 Crown Fire**

According to [21], Forest fire simulators resort to using a theoretical concept to understand and calculate the natural phenomenon of crown fires. Crown fires are when a fire spreads from the ground and trunk area of a tree and spread upwards to the higher branches and foliage. The theoretical concept is known as Crown Fraction Burned or CFB for short. This value is implemented as a threshold and determines if the crown fire is of active or passive in behavior. A passive crown fire is when a surface fire is capable of igniting a single tree or a group of tree crowns, but wind speed is insufficient in strength to allow propagation to other trees in the vicinity. This does not contribute to the overall spread of the fire as a whole. On the other side

of spectrum is the CFB threshold is surpassed the crown fire is classified as active in status. An active crown fire can greatly increase the surface fires overall intensity and spread rate. This also suggests that the wind speed is aggressive enough to spread the fire not only through grounded means but can travel treetop to treetop. This thesis bases the calculations for this type of fire effect on the previous fire simulator vFire [22], which is heavily reliant on the sequential fire simulator FARSITE [18], which worked off publication by Van Wagner [45, 46] and Rothermel [34].

### 2.1.5 Spotting

Spotting fire behavior is a very dynamic and complex phenomenon that requires many factors to implement accurately; and even then it is at best probabilistic in nature. Spotting occurs when an active crown fire emerges and is intense enough to create upward pressure lofting fire embers from within to be carried by the wind and possibly ignite fires ahead of the main fire. There are three main parts to evaluate and mimic spotting fire [21], the first is to take into consideration the starting position of the ember or embers, the size of them, and quantity. The second factor is calculating the distance the ember(s) can travel downwind and land. The third and final is where probability takes place in determining under various conditions is the ember(s) will ignite or not at their new location. A visual example of spotting and the three factors can be seen in Figure 2.2.

Sousa, dos Reis, and Pereira also used the GPU to improve their running times and ported `fireLib` to the GPU [41], and were the first to use the parallel programming language CUDA [26]. They implemented three kernels in which they explored three different propagation types. This paper based two of the spread methods (Minimal Time and Iterative Minimal Time) on the work done by Sousa, dos Reis, and Pereira. Their work will be covered in more detail further in the Fire Propagation Models section of the paper. The third propagation method implemented in this paper was based on work found in vFire [22, 23]. Investigations into using GPU computation for optimizing fire simulation have been explored by several researchers,

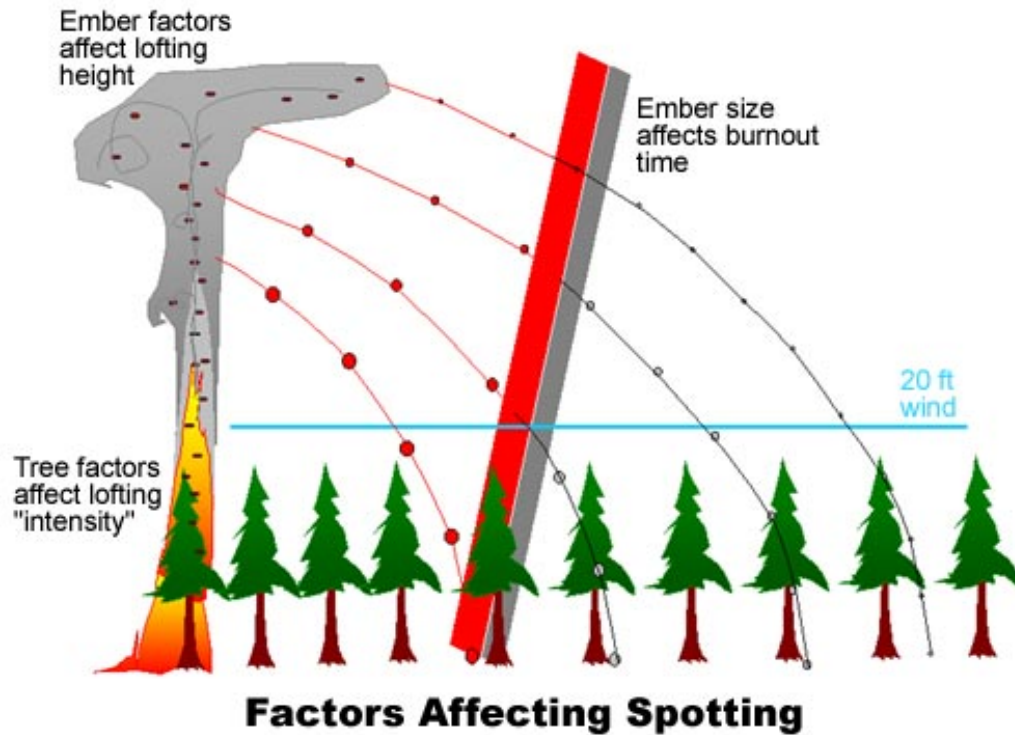


Figure 2.2: Spotting Fire Phenomenon [21]

including a paper by Arcaa, Ghisub, and Trunfioc [6]. Their implementation used GPU computation to optimize fuel treatments across a landscape. Their focus was not on using the GPU to calculate base propagation, rather the influence of fire breaks. The work by Baranovskiy explores the usage of GPU computing to enhance the performance of theoretical-based propagation models [8]. While the work is useful for exploring the usage of theoretical models, this work implements a semi-empirical approach, and therefore the direct results are not comparable.

## 2.2 Fire Simulators Past and Present

Using Rothermel's fire spread model published in 1972 [32] many fire simulators were developed in order to aid the fight against wildfires and for purely research experimentation.

Since Rothermel's paper was published in 1972, several fire spread simulators have been developed. Every major forest fire simulator has used his spread equations as the basis for their simulation. While there have been new models developed for the spread of fire [29, 37], the existing forest fire simulators use Rothermel's because it is an approximation of the spread of fire accurate and simple enough to be calculated in the time allowed for simulations. The first major fire spread simulator was developed in 1986 called BEHAVE [5]. BEHAVE had two main functions to the application. The first function allowed users to load in fuel models from Rothermel's paper, but also to develop and save new fuel models. The simulator then had the ability to integrate the newly developed fuel models in its simulations. The second function of the application would run a simulation and burn prediction on the desired fuel model. The output of this simulator appeared in a table which represented the times of arrival for each cell in the simulation. There was no visualization method available for this simulator. The simulation was meant to be used as a training tool rather than a real-time tool to be used to fight wildfires.

### **2.2.1 BEHAVE**

In 1986, BEHAVE [5] was developed more as a training tool rather than an actual real-time application to assist personnel on the ground. BEHAVE is able to use Rothermel's fuel models as well as create and store new ones. Once a fuel model has been loaded or created, the system calculates the burn time and behavior of the fire. The output is simply a table of the estimated times of arrival for a cell to be on fire. A cell is simply a portion of an area, such as a 10x10 meter grid, that contains the properties of the surrounding area. The properties can be, for instance in a forest, a 10x10 grid of coniferous trees and the corresponding fuel model. BEHAVE is still being used to this day and is a very resourceful tool for fire researchers to develop new and up to date fuel models.

### 2.2.2 fireLib

Following the development of BEHAVE, in 1996 fireLib used BEHAVE as the foundation to create a revitalized fire simulator in the C programming language. FireLib runs faster than BEHAVE and outputs the burn time of arrivals in an array format. Considering BEHAVE was a single application where only an instance of a fire was computed; fireLib allows the user to modify and create new fire propagation techniques and includes a time parameter in order to simulate fire in a dynamic time scale. In addition, fireLib is an open source library rather than an application.

### 2.2.3 FARSITE

FARSITE was created in 2004 [18]. It has been supported and upgraded since and is to this day in use. It is arguably the most advanced and accurate forest fire simulator developed. However, it is completely sequential and not very fast making real-time results not possible. It offers more fire effect variables and phenomenon beyond the ability to calculate fire spread, crown fires, surface fires, fire acceleration, and spotting. This thesis used much of the fire spread implementation from a forest fire simulator called vFire [22, 23]. vFire was based on hFire, and both are cellular based spread models. They run faster than FARSITE, but do not have the same level of precision [30]. vFire implements dynamic time stepping to burn distances between cells to determine an accurate time of arrival for the fire spread. The important feature that vFire accomplished was porting the computation of the fire spread to the GPU using OpenGL shaders [38]. Because the computation was ported to the GPU, it accomplished a very high speedup over its sequential implementation. vFire provided the outline for the data processing portion of this work as the code was available to this project.

### 2.2.4 vFire

vFire builds upon all of the above; it is also the fire simulator our fire library was built upon. vFire implements cellular fire spread models. It also boasts an increased

computation speedup over FARSITE, however it does not carry the same degree of accuracy. `vFire` is unique due to its implementation of the fire spread calculation through the use of the GPU rather than the CPU. `vFire` incorporates the visualization and fire spread calculations simultaneously through what was at the time of development the only programming language capable of communicating with the GPU, OpenGL Shader Language. Using GLSL `vFire` had taken advantage of the massively parallel instructional multi-cores within a GPU, this inherently made the computations conducted in parallel rather than sequential, speeding up the fire simulation capable of real-time results. `vFire` is unique due to its implementation of the fire spread calculation through the use of the GPU rather than the CPU. `vFire` incorporates the visualization and fire spread calculations simultaneously through what was at the time of development the only programming language capable of communicating with the GPU, OpenGL Shader Language. Using GLSL `vFire` had taken advantage of the massively parallel instructional multi-cores within a GPU, this inherently made the computations conducted in parallel rather than sequential, speeding up the fire simulation capable of real-time results.

### 2.2.5 `vFireLib`

`vFirelib` was developed to reproduce `vFire`'s simulation results. However, there are very specific functions that are implemented differently in order to create a more versatile and efficient fire simulator. As previously stated, `vFire` was written in GLSL, which inherently takes advantage of the GPU's parallel computational structure, but this also means the simulation and visualization are dependent on one another and cannot be run autonomously. This is the crux of differences between `vFire` and `vFirelib`. Our new library incorporates NVIDIA's Compute Unified Device Architecture known as CUDA, and we have completely separated the computation from the visualization. CUDA allows developers control of the hundreds and thousands of cores within a GPU or within many GPUs. This allows for a much greater number of instructions a CPU would be capable of processing in the same amount of time.

The speed advantage is now obvious to using CUDA, but there is the added advantage that we can now run the visualization component and simulation component independently of one another. This is useful in order to run many simulations with different factors that can affect a wildfire, such as differing fuel loads, weather conditions, and man-made firebreaks in suppression efforts; all of these can be simulated and visualized with dynamic time stepping. `vFirelib` is a library created to incorporate all the advantages mentioned above, it is written in C++/CUDA. The fire simulation is mapped to the GPU and not the CPU due to the enormous number of cells within each time step of the fire simulation. In every time step for each cell the fires spread, acceleration, crowning, and spotting all need to be calculated and when this is mapped to a GPU hundreds of thousands of cells can be calculated in parallel. `vFirelib` offers speed, efficiency, and versatility improvements over the previous work of `vFire`.

## 2.3 Fuel Models

Fire simulators need to be able to load and compute clearly defined fuel models. As stated above Rothermel created the first eleven original fuel models. These Eleven were expanded upon and even two more were added by Anderson in 1982 [4]. In June 2005 forty more models were added to these thirteen by Scott and Burgan [36]. This work is capable of utilizing all 53 fuel models. Fuel models are essential data collected upon certain shrubs, grasses, timber, trees and foliage in general. There are different burn rates, fuelbed depth, fire line maximum intensity values and more for each of these individual plants and debris. There are burnable and nonburnable elements, such as rocks, buildings, and water. This thesis simply lumps all unburnable models into a single one for simplicity sake. Among the fuel models there are many factors and classifications that can have a great effect on the spread of a fire. Moisture values within fuel models can be defined as low, moderate, and high. These moisture values correlate with the fuel models Heat per Unit Area value. Fuel models can be static or dynamic; static fuel models simply have their values read from a file and

are considered the default standard of how these fuels would react with fire under average weather and seasonal conditions. Static fuel models are inflexible due to this reason and therefore, this thesis incorporated dynamic models to be possible. Dynamic fuel models are when the simulator can take into consideration moisture changes due to season, time, or suppression efforts and then calculate the correct spread rate accordingly. Moisture among fuel models is divided into two categories, live herbaceous and dead fuel moisture. Dynamic fuel models have a proportional fuel load transfer when dealing with moisture change from live to dead or vice versa; this will be divulged into further in chapter 4. Fuel models are literally that which the fire spread rate, direction, and intensity calculations are based upon. There are many variables for each fuel model and can be seen in Figure 2.3.

## 2.4 GPGPU Computing and Architecture

This work is dealing with host and device programming techniques. Compute Unified Device Architecture or CUDA with C++ is used to manage memory and computations between the CPU and GPU. The CPU known as the host is used for file data input and output. The host is needed for file I/O and also helps with the preprocessing of certain data that cannot be done on a GPU for the moment. It is also necessary to utilize the Geospatial Data Abstraction Library known as GDAL to import the detailed terrain that is needed for an accurate fire simulation. The bulk of the computation for all the cells and the entirety of the fire simulation is done on the GPU or known as the device.

GPU's used to be strictly specialized fixed function pipeline programming. They were used discretely as graphics accelerators. This made using a GPU for general purpose computing almost impossible unless it was incorporated with a visualization library as well. An example of this is vFire described above. However, as stated having such a dependency on a visualization library meant that the visualization and computation of the problem could not be separated and limited its functionality. Nvidia released CUDA in 2006, this allowed the GPGPU computing experience today

Fuel model code	Fuel load (t/ac)					Fuel model type <sup>a</sup>	SAV ratio (1/ft) <sup>b</sup>			Fuel bed depth (ft)	Dead fuel extinction moisture (percent)	Heat content BTU/lb <sup>c</sup>
	1-hr	10-hr	100-hr	Live herb	Live woody		Dead 1-hr	Live herb	Live woody			
GR1	0.10	0.00	0.00	0.30	0.00	dynamic	2200	2000	9999	0.4	15	8000
GR2	0.10	0.00	0.00	1.00	0.00	dynamic	2000	1800	9999	1.0	15	8000
GR3	0.10	0.40	0.00	1.50	0.00	dynamic	1500	1300	9999	2.0	30	8000
GR4	0.25	0.00	0.00	1.90	0.00	dynamic	2000	1800	9999	2.0	15	8000
GR5	0.40	0.00	0.00	2.50	0.00	dynamic	1800	1600	9999	1.5	40	8000
GR6	0.10	0.00	0.00	3.40	0.00	dynamic	2200	2000	9999	1.5	40	9000
GR7	1.00	0.00	0.00	5.40	0.00	dynamic	2000	1800	9999	3.0	15	8000
GR8	0.50	1.00	0.00	7.30	0.00	dynamic	1500	1300	9999	4.0	30	8000
GR9	1.00	1.00	0.00	9.00	0.00	dynamic	1800	1600	9999	5.0	40	8000
GS1	0.20	0.00	0.00	0.50	0.65	dynamic	2000	1800	1800	0.9	15	8000
GS2	0.50	0.50	0.00	0.60	1.00	dynamic	2000	1800	1800	1.5	15	8000
GS3	0.30	0.25	0.00	1.45	1.25	dynamic	1800	1600	1600	1.8	40	8000
GS4	1.90	0.30	0.10	3.40	7.10	dynamic	1800	1600	1600	2.1	40	8000
SH1	0.25	0.25	0.00	0.15	1.30	dynamic	2000	1800	1600	1.0	15	8000
SH2	1.35	2.40	0.75	0.00	3.85	N/A	2000	9999	1600	1.0	15	8000
SH3	0.45	3.00	0.00	0.00	6.20	N/A	1600	9999	1400	2.4	40	8000
SH4	0.85	1.15	0.20	0.00	2.55	N/A	2000	1800	1600	3.0	30	8000
SH5	3.60	2.10	0.00	0.00	2.90	N/A	750	9999	1600	6.0	15	8000
SH6	2.90	1.45	0.00	0.00	1.40	N/A	750	9999	1600	2.0	30	8000
SH7	3.50	5.30	2.20	0.00	3.40	N/A	750	9999	1600	6.0	15	8000
SH8	2.05	3.40	0.85	0.00	4.35	N/A	750	9999	1600	3.0	40	8000
SH9	4.50	2.45	0.00	1.55	7.00	dynamic	750	1800	1500	4.4	40	8000
TU1	0.20	0.90	1.50	0.20	0.90	dynamic	2000	1800	1600	0.6	20	8000
TU2	0.95	1.80	1.25	0.00	0.20	N/A	2000	9999	1600	1.0	30	8000
TU3	1.10	0.15	0.25	0.65	1.10	dynamic	1800	1600	1400	1.3	30	8000
TU4	4.50	0.00	0.00	0.00	2.00	N/A	2300	9999	2000	0.5	12	8000
TU5	4.00	4.00	3.00	0.00	3.00	N/A	1500	9999	750	1.0	25	8000
TL1	1.00	2.20	3.60	0.00	0.00	N/A	2000	9999	9999	0.2	30	8000
TL2	1.40	2.30	2.20	0.00	0.00	N/A	2000	9999	9999	0.2	25	8000
TL3	0.50	2.20	2.80	0.00	0.00	N/A	2000	9999	9999	0.3	20	8000
TL4	0.50	1.50	4.20	0.00	0.00	N/A	2000	9999	9999	0.4	25	8000
TL5	1.15	2.50	4.40	0.00	0.00	N/A	2000	9999	1600	0.6	25	8000
TL6	2.40	1.20	1.20	0.00	0.00	N/A	2000	9999	9999	0.3	25	8000
TL7	0.30	1.40	8.10	0.00	0.00	N/A	2000	9999	9999	0.4	25	8000
TL8	5.80	1.40	1.10	0.00	0.00	N/A	1800	9999	9999	0.3	35	8000
TL9	6.65	3.30	4.15	0.00	0.00	N/A	1800	9999	1600	0.6	35	8000
SB1	1.50	3.00	11.00	0.00	0.00	N/A	2000	9999	9999	1.0	25	8000
SB2	4.50	4.25	4.00	0.00	0.00	N/A	2000	9999	9999	1.0	25	8000
SB3	5.50	2.75	3.00	0.00	0.00	N/A	2000	9999	9999	1.2	25	8000
SB4	5.25	3.50	5.25	0.00	0.00	N/A	2000	9999	9999	2.7	25	8000

<sup>a</sup> Fuel model type does not apply to fuel models without live herbaceous load.

<sup>b</sup> The value 9999 was assigned in cases where there is no load in a particular fuel class or category

<sup>c</sup> The same heat content value was applied to both live and dead fuel categories.

Figure 2.3: The Standard 40 Fuel Models [18]

where there is no need for a co-dependent visualization software. Using a GPU to calculate makes sense in the fact that a CPU has on average 4 to 8 cores that can process calculations at an amazing speed, but a GPU has thousands of processors that can operate simultaneously albeit a slower clock speed. The quantity of processors located in a GPU more than make up for the slower computation power and increase throughput possible for a large data driven problem. This is very well suited for a fire simulation when there are literally millions of cells all with varying data that need the same calculation and checks done on them every time step. Code is written in kernels and each kernel's code runs on each thread within a GPU. GPU architecture starts with symmetric multiprocessors (SM). Each SM has cores and software threads are run on the cores. The NVIDIA 1080 GTX has 28 SM(s) and each SM has 128 cores for a total of 3,584 cores each of which can compute at the same time. To get a visual of what this might look like please reference Figure 2.4 This architecture can be accessed through CUDA and begins at the grid level. Block level follows then thread; this can be seen in Figure 2.5. Each thread has its own ID, this ID is unique to the block it is in. Blocks usually contain 1024 threads and a grid contains many blocks. Instructions can be assigned to the thousands of threads across multiple blocks essentially computing data in parallel chunks at a time.

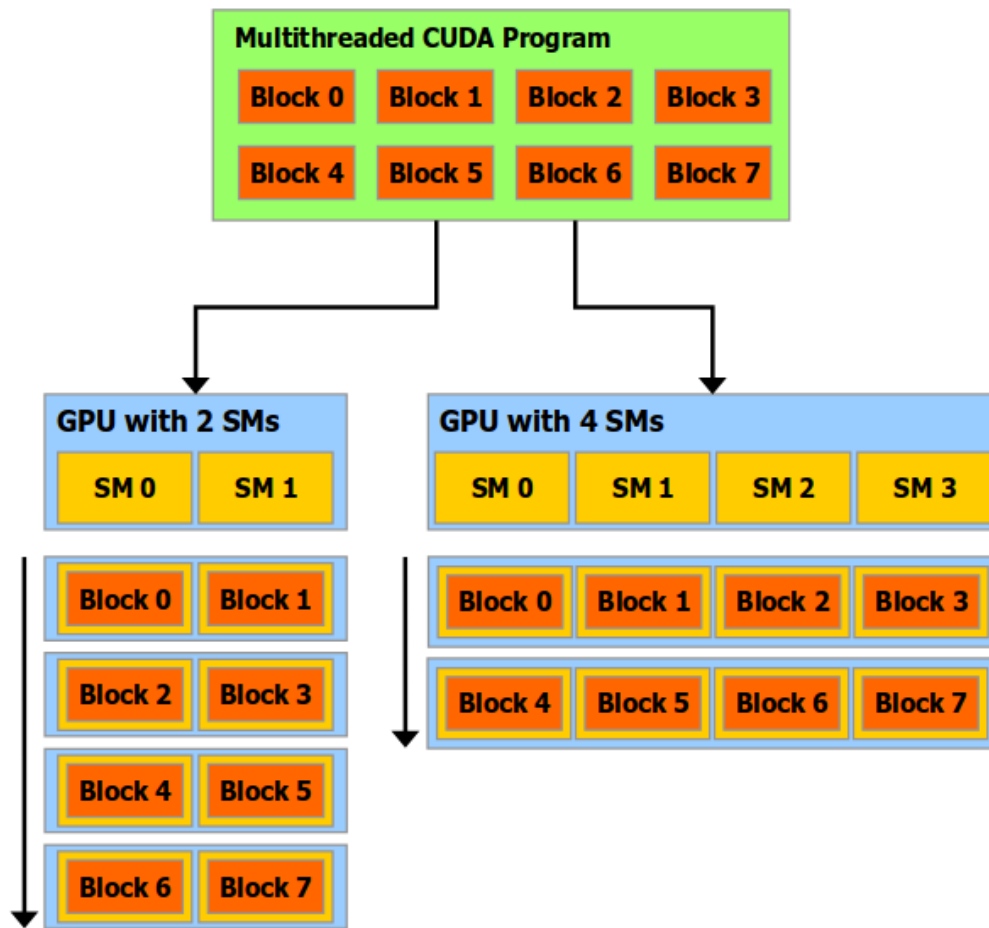


Figure 2.4: GPU SM CUDA Programming [35]

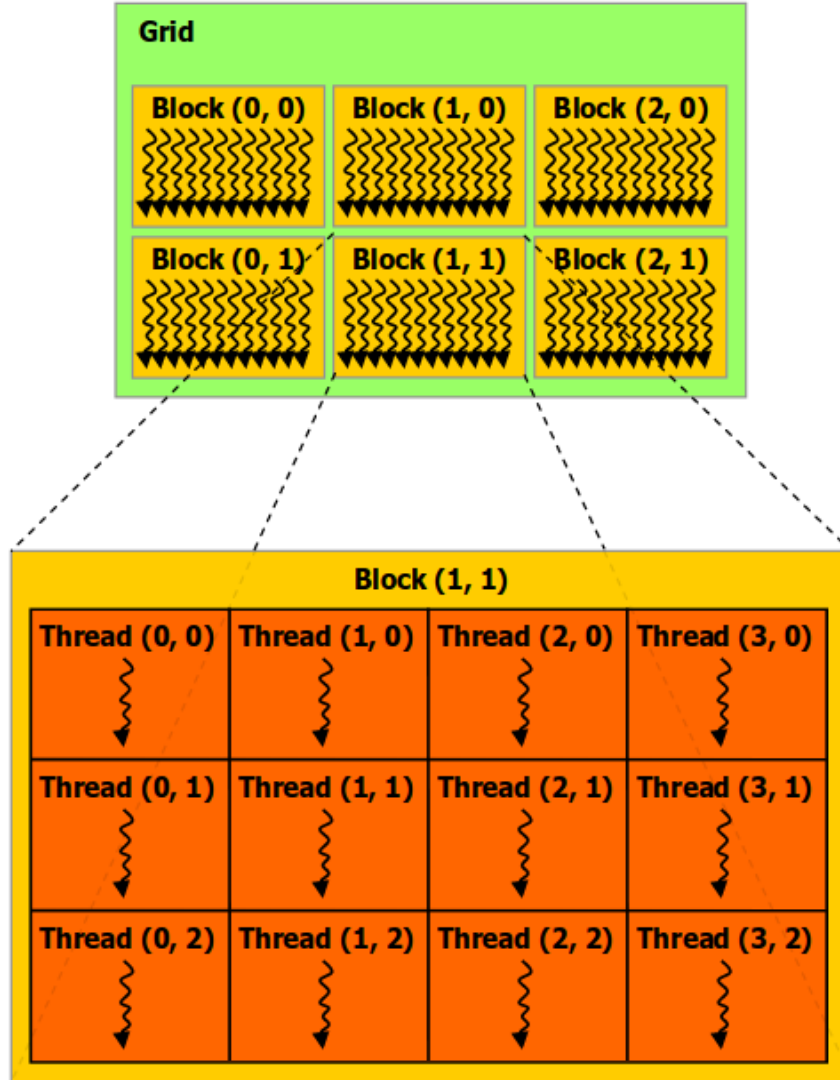


Figure 2.5: GPU Architecture [35]

## 2.5 GDAL

Geospatial data is extremely important to building a real-time accurate wildfire simulator. Terrain plays a huge role in where a fire can spread and affect the rate at which the fire spreads. GDAL is a Geospatial Data Abstraction Library used in this work to obtain terrain data such as slope and integrate it into the fire simulator. During the preprocessing phase, there are several data files which need to be read

and interpolated among the simulator dimensions. The fuel data and slope data are stored in files containing interpolated data such as size of cell, width, and height of the data grid. GDAL [19] is used to interpolate the data from the terrain and fuel files into the desired size of simulation. The library allows our library to easily read and manage TIFF formatted files; this was chosen in respect to the fact that LANDFIRE offers not only terrain data, but vegetation data as well in TIFF format which can be used in this work in order to simulate fires in various geological locations.

# Chapter 3

## Software Engineering

### 3.1 Overview

vFireLib.v2 is simply an optimized and functionally advanced library initiated and presented by Jessica Smith [39] which was based on the work of Roger V. Hoang [22]. There are clear differences between the two vFireLib library versions, vFireLib.v2 contains more functionality, better utilization of data, and is compatible with more than one platform not previously provided. However, there will be function that were not changed in name but were either optimized and or restructured in order to better the simulator. This version of the library's built in functions are still to utilize the fire data given, process the data, calculate the spread of the fire, implement certain fire effects such as spotting, moisture change, and many others and then provide visualization and data back to the user. At any point the user shall have the ability to impose changes to the fire simulation and create an infinite number of simulated fires with different constraints and effects in order to experiment and or fit the simulation to a desired context for increased accuracy.

### 3.2 Fire Simulation Requirements

In [40] technical requirements for a program are broken down into functional and non-functional requirements. The functional and non-functional requirements for vFireLib.v2 are presented in this section.

### 3.2.1 Functional Requirements

The functional requirements of the new advanced vFireLib.v2 can be found in Table 3.1. These functional requirements were designed in respect to provide a user using either the web service or unity game engine platforms the ability to modify and run fire simulations. These behaviors were a necessity to build an accurate and practical fire simulator.

Table 3.1: The functional requirements of vFireLib.v2

Number	Description
FR01	The library shall read in fuel model, moisture model, and terrain data.
FR02	The library shall read in canopy height, canopy bulk density, and canopy cover data.
FR03	The library shall define the resolution of simulation size.
FR04	The library shall initialize all data members for CPU and GPU chosen simulations.
FR05	The library shall calculate the maximum spread rate of every cell.
FR06	The library shall be able to process simulations using one of the three spread methods: BD, MT, and IMT.
FR07	The library shall be able to allow for moisture manipulation.
FR08	The library shall be able to allow for vegetation manipulation.
FR09	The library shall be able to allow for wind manipulation.
FR10	The library shall be able to allow for crowning fires to be toggled on and off.
FR11	The library shall be able to allow for spotting effects to be toggled on and off.
FR12	The library shall be able to provide feedback if data is missing or incorrect.
FR13	The library shall be able to pause the fire spread at a certain time tick.
FR14	The library shall be able to resume the fire spread at a certain time tick.
FR15	The library shall store all needed data for accurate pause and resume of fire spread.
FR16	The library shall allow for changes to the fire simulation through the web service.
FR15	The library shall allow for visualization of the fire spread through Unity Game Engine.

### 3.2.2 Non-functional Requirements

The non-functional requirements of the new advanced vFireLib.v2 can be found in Table 3.2. These requirements are based on the tasks and communication necessary for GPGPU computing. For example, the communication and data transfer between the host and device; as well as the preprocessing done by the host to allow for optimal device calculations and output.

Table 3.2: The Non-functional requirements of vFireLib.v2

Number	Description
NFR01	The library shall be written in C++ and CUDA.
NFR02	The library must use CUDA version 6.0 or higher.
NFR03	The librarys sequential implementation must be a direct reflection of its parallel implementation.
NFR04	To run the parallel version of the code, an NVIDIA GPU with CUDA Compute Capability 2.0 or higher is needed.
NFR05	This library must be used on the Linux platform.
NFR06	The library must use GDAL version 1.10.1.
NFR07	The library must be compiled with CMake Version 2.8 or higher.

### 3.3 Use Case Modeling

Sommerville also presents a standard software engineering diagram known as a Use Case Diagram [40]. It defines how different actors can interact with the system. The use case diagram for vFireLib.v2 is shown in Figure 3.1 and descriptions of these use cases follow.

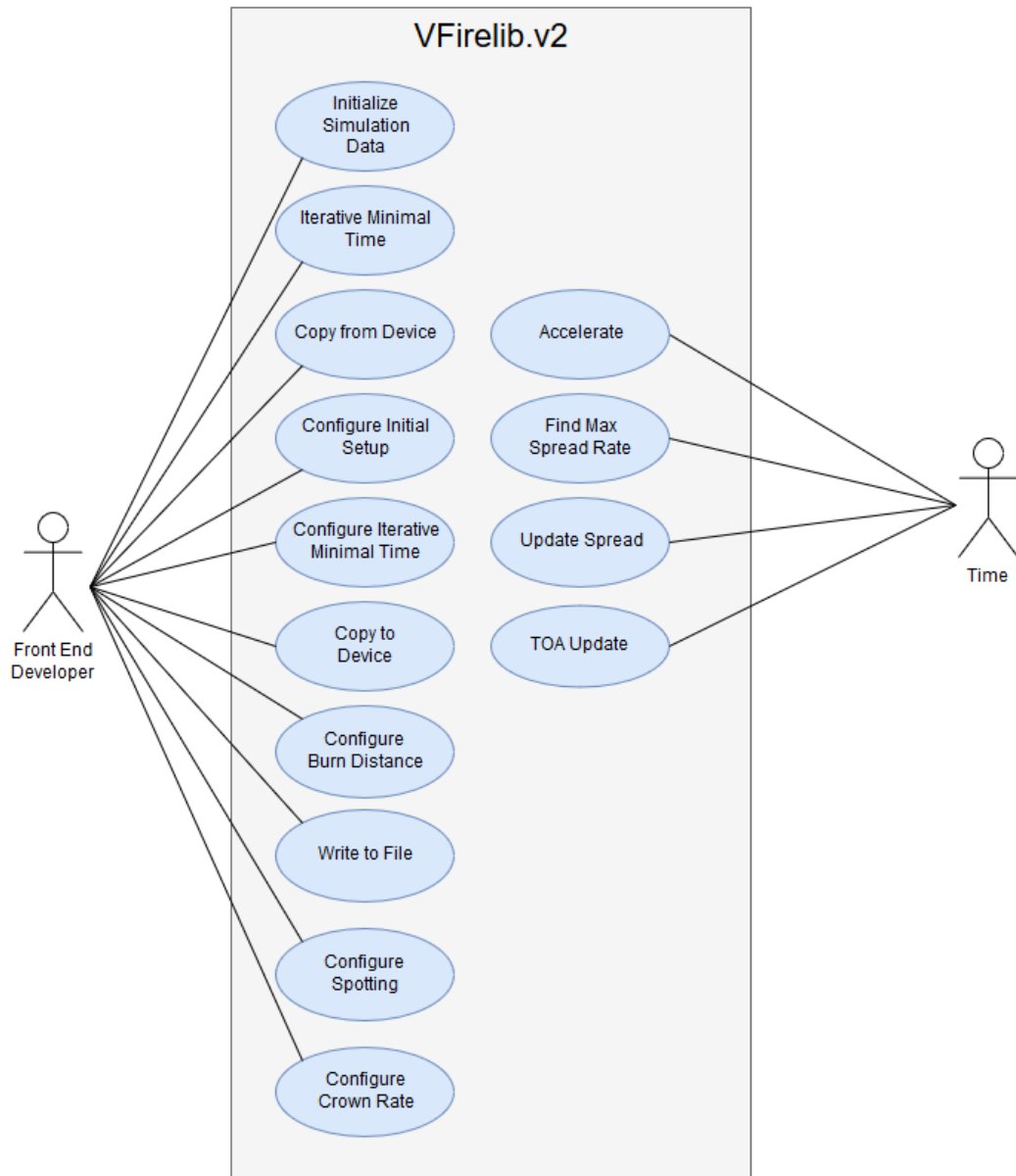


Figure 3.1: Use Case Diagram

**Initialize Simulation Data:** This is done entirely on the host serially. The data is read in from files necessary for the fire simulation. The data for terrain, moisture, wind, fuel, crown base heights, crown bulk density, and canopy height are all processed and stored in CUDA vector types [15]. This requires the host after reading in the data to then determine the granularity and correct resolution per cell data. Calculating for granularity and resolution based on cell size is done using GDAL [19]. The above

is all for a default fire simulation, if the fire simulation is paused and resumed by the user or calculated in tick intervals then additional data needs to be dealt with. This additional data includes the current rate of the fire spread, where the fire has burned and will burn to, as well as any modified data the user changed such as wind, vegetation, fuel, and moisture. This was done to accommodate for fire suppression effects that include water drops, changing weather, firebreaks such as bulldozing, controlled burns, and firefighter cut trenches clearing fuel for an advancing fire in order to contain it.

**Copy To Device:** All of the data for a default and paused or resumed fire simulation is then transferred from the host to the device. This configuration will transfer all necessary data to the device and will provide error feedback if essential data is missing and or corrupt.

**Configure Initial Setup:** This configuration was implemented on the host in the previous version of vFirelib, however, this was re-implemented on the device as the initial kernel for vFireLib.v2. It calculates the Rothermel values for each cell which include the max spread rate of the individual cell, ellipse eccentricity, spread direction, and the spread modifier. These are based on moisture, wind, slope, and fuel type characteristics.

**Configure Spread Update:** This kernel is calculating for every cell and its neighbor the maximum spread rate the fire can reach in a certain direction using all of the four above Rothermel values as well as the 16 angles the neighbors are located. This is done once per simulation, and will be called every time a pause resume occurs to account for any user imposed changes in moisture, fuel, wind, etc.

**Find Max Spread Rate:** This kernel is a max reduce to find the maximum possible spread rate among all the cells in the simulation. This is used to calculate a dynamic time step for the simulation which was needed in order to inhibit fire spreading to

the same cell causing a race condition and extra computation. Similar to the Initial Setup and Update Spread configurations this is done once for every simulation and or pause resume case.

**Configure Burn Distance:** In brief if this propagation method is chosen by the user this configuration implements the burn distance spread method in which the distance for the fire to travel within a cell is calculated and then checked to see if that distance has been reached igniting any of its neighboring cells not on fire. This is done every time tick in order for the wildland fire to be calculated accurately.

**Configure Minimal Time:** This propagation method configuration implements the work done by Sousa, Dos Reis, and Pereira’s paper [41]. At each time tick all cells are checked if they are on fire or not and whether they are a neighbor of a lit cell. If the cell is on fire the spread is calculated outwards. If the cell’s neighbors are already on fire they are ignored if they are not on fire the time of arrival for them will be calculated. However, the cells TOA can be overwritten if a faster TOA is calculated due to another cell that is closer and is lit; this is possible with the dynamic timestep.

**Configure Iterative Minimal Time:** The last choice a user can decide to implement for propagation methods is IMT. This configuration is similar to MT, but instead of checking if it’s neighbors are lit or not outwards its neighbors check inwards. Race conditions are avoided by using input and output vectors avoiding the use of atomic configurations. IMT also calculates the TOA of cells independently of others and uses convergence to determine when the correct TOA has been computed.

**Configure Crown Rate:** There are technically two implementations of this configuration. One is to include spotting if the user desires and the second one is to exclude spotting. This was done for experimental purposes, the configuration’s main utility is the same for both. This checks cells if a crown threshold has been reached.

If the crown is passive it does not add to the spread rate of the fire. If the threshold is surpassed then the resulting active crown will change the maximum spread rate of the cell, thus effecting the overall spread. This is where the one form of this configuration will then commence the initial calculations for spotting effects due to an active crown fire. The initial computation for spotting in this configuration is using an ember map to track which cells are active crown fires, and therefore release embers in the air. The embers travel, height, and fall speed are set here and marked on the ember map for the spotting configuration. The ember properties are updated every time tick.

**Configure Spotting:** This configuration checks the ember map. If the ember map indicates that an ember is there it checks if the ember has touched the ground. The ember is then decided to whether or not to set that cell in which it has landed on to ignite updating the cells time of arrival to the current time tick. Determining whether the ember ignites is based on probability and the amount of moisture that is in the fuel for that particular cell. If the cell fuel type contains more moisture then a fire is unlikely and vice versa.

**TOA Update:** This is used in support of the spotting configuration and will update all the time of arrivals that have been changed due to an ember prematurely setting it off.

**Accelerate:** After the current time tick This checks all cell spread rates and max spread rates to determine the new spread rate of the cells in the simulation.

**Copy From Device:** This provides all time of arrivals in an array format saved to a CSV file is the simulation was simply run from start to end. If the simulation was paused then this would write not only the TOA but the current rate of spread and all cell information at the tick paused. This is necessary to resume the simulation at the exact same spot with all pertinent information in order to simulate the fire spread with changes accurately.

**Write to File:** This configuration writes specific data to the appropriate file with correct granularity and resolution. These files are either then used for visualization and or resumption of the simulation with user changes.

# Chapter 4

## Implementation

### 4.1 Overview

Since the goal of this thesis was to illustrate the difference between a parallel and sequential fire simulator implementation two programs were developed and written. The sequential implementation was instructed on a CPU with no multi-threading, this could be a comparison for a future topic of research. The parallel instruction was written for GPGPU computing with CUDA and therefore requires an Nvidia graphics card. Unlike the former vFireLib library developed by Jessica Smith at the University of Reno Nevada, vFireLib.v2 separated all possible pre-processing computation from the HOST and ported it all to the DEVICE. The timings discussed in chapter 5 will reflect this change. The main reason this change was done is to cut down on the entire program compute time and to paint a better picture of the difference in throughput and speed between a sequential and parallel fire simulator. To note as well the pre-processing was not taken into consideration for total compute time but needs to occur every time a simulation is run. However, because in vFireLib.v1 the pre-processing was the exact same for both programs the compute time was dependent solely on the calculations for the fire propagation and therefore not considered. vFireLib.v2 needs to take the pre-processing into consideration due to the significant changes in the parallel implementation. The simulations outline is still consistent between the sequential and parallel implementations and can be seen below in Algorithm 4.1.

---

**Algorithm 4.1** Simulation Progression
 

---

```

1: InitializeSimulation();
2: while Simulation !Complete do
3:   RunSimulation();
4: end while
5: GenerateOutput();

```

---

This outline will be referred to throughout the rest of the chapter. The first half will be detailing and examining the sequential implementation of the fire simulator. The second half will go over the parallel implementation.

## 4.2 Fire Propagation Models and Their Sequential Implementations

The `InitializeSimulation()` for the sequential fire propagation involves quite a bit of calculations in order to setup the Rothermel values for every cell within the simulation. This is also where GDAL and file IO is handled. Since all data needed for the simulation comes from LANDFIRE TIFF files or user manipulated CSV files the values must be extracted and then placed in vectors. CUDA has developed vector types which can be used ubiquitously on HOST and DEVICE [15]. Once inside the CUDA vectors the HOST in this case can access the values for every cell such as fuel model, moisture content, wind, etc. to calculate the Roth data. If there is any change in the fire simulation all of the pre-processing must be done again. A wildfire is a dynamic entity that can change due to weather or man-made events and thus is the main reason why a sequential implementation is not suitable for a real-time fire simulator. In reality running a single simulation may be done in a reasonable amount of time but running multiple simulation to accommodate the constant changes that can occur in most wildfires can be problematic when making time critical decisions in the use of resources and positioning of personnel in or out of harms way.

Once the pre-processing is complete the actual fire propagation method takes place and is calculated to predict the rate of spread and direction. This is the Run-

Simulation() section of the simulator’s outline displayed in Algorithm 4.1 As mentioned previously in the thesis, there are three methods to achieving a fire simulation prediction. There have been a few comparisons among fire simulators and their different propagation methods [27, 28, 37]. To provide an encompassing fire simulator the user of the software will be able to choose which method of propagation they wish to incorporate. They are all based upon Rothermel’s equations and that is a constant among all three. The biggest difference in implementation between the methods is how each interacts with its neighboring cells. All three methods will produce results stored in files only accessible to a software developer interested. However, for non technical users the results can be seen through the web service visually. The results are stored in a cell by cell mapped value format, such as the time of arrivals, moisture content, current rate of spread, max rate of spread, and threshold values for crowning and spotting. Based on the work done by Sousa, dos Reis, and Pereira [41], Minimal time and Iterative Minimal times were mirrored in implementation. The third fire propagation method was a reproduction of vFire [23]. All three methods were studied and written in CUDA and C++ to increase the performance of them and to compare GPGPU computing throughput verse CPU throughput.

The rate of spread or propagation of the fire for this thesis was all completed in the previous work in vFireLib by [39]. In this work denoted as vFireLib.v2, the simulator was advanced greatly beyond a simple fire spread under no wind, and uniform terrain and fuel model conditions. vFireLib.v2 also implements advanced user manipulation to the fire simulations, as well as providing a web service [48] that communicates with the CUBIX [13] server and helps visualize the scientific data and fire spread on a given terrain. The fire propagation equation used for all three methods is Rothermel’s Equation 4.1.

$$r(\Theta) = R_{max} \frac{1.0 - \varepsilon}{1.0 - \varepsilon \cos(\phi - \Theta)} \quad (4.1)$$

Where

$R_{max}$  = is the maximum fire rate of spread.

$\varepsilon =$  is the eccentricity given through slope and wind.

$\phi =$  is the fire's orientation angle.

$\Theta =$  is the fire spreads direction.

$Q_{ig}$  is the heat of pre-ignition.

In the previous version of vFireLib  $R_{max}$ ,  $\varepsilon$ , and  $\phi$  were determined before fire propagation was calculated. This is still the case for the sequential version as there is no front end platform for the user to manipulate the fire simulators variables available in the parallel implementation. For the sequential version nothing was changed in terms of how the preprocessing and burn methods were calculated. One last mention for clarification is that  $\Theta$  is set as the direction angle value of neighboring cells.

### 4.2.1 Burn Distance Model

The burn distance model (BD) is the most intuitive in terms of how one would think fire burns and spread through nature. All the cells in a simulation have a set distance or area that the fire needs to burn through before igniting another neighboring cell. The set distance makes sense considering all the cell in the simulation will be determined by the resolution of the data files. All data in these files are extracted and then interpolated across the simulation to ensure accuracy. The distance the fire burns is constant, what changes is the speed or rate at which it travels. This is dependent on factors such as the fuel model contained within the cell, the moisture, wind, and terrain. The distance the fire has burned across the cell is tracked every timestep. The timestep for all three methods is dynamic, because if a static timestep is implemented race conditions will overwrite time of arrivals and produce incorrect results. The race condition occurs when the timestep is too large causing the fire from one cell to propagate to another cell before it should have. The dynamic timestep keeps this from happening guaranteeing that the timestep will accommodate whatever simulation dimensions arise. This timestep is found by simply calculating the max spread rate of every cell, find the greatest rate, and dividing it by the cell size; this creates a timestep that is never too large for a race condition to occur.

The cells in a BD simulation spread fire through distance burned. Once the distance across a cell has burned completely it will spread the fire to a neighboring cell. The distance the fire has traveled is tracked every timestep found by calculating the following Equation 4.2.

$$d = d - r\Delta t \quad (4.2)$$

Where

$d$  = is the distance left to burn.

$r$  = is the distance decreased each timestep through the rate of spread.

$\Delta t$  = is the timestep size to avoid overburn due to a static timestep.

The time of arrival for the fire to reach a cell is denoted exactly when the complete distance has been burned. The equation to calculate the TOA of the cell the fire has reached is found in Equation 4.3.

$$TOA = t_{now} + \frac{d_{over}}{r} \quad (4.3)$$

Where

$TOA$  = is the TOA.

$t_{now}$  = is the current timestep.

$d_{over}$  = is the distance threshold needed for the fire to spread.

$r$  = is the cell's calculated rate of spread.

The BD model approach is laid out step by step in the following Algorithm 4.2. The BD spread method can be seen previously in Chapter 2 shown in Figure 2.1.

---

**Algorithm 4.2** Burn Distances Algorithm
 

---

```

for  $cell = 0$  to  $numCells$  do
  // Check to see not on fire
  if  $ignTime[cell] == INF$  then
    Skip
  end if
  // Check Neighbors for fire propagation
  for  $n = 0$  to  $7$  do
    if  $ignTime[neighborCell] < INF$  then
      Skip
    end if
     $ROS =$  Compute ROS according to Equation 4.1
     $burnDistance(totDist[neighborCell], ROS, timeStep)$ 
    if distance is burnt then
       $ignTime[neighborCell] = timeNow$ 
    end if
  end for
end for

```

---

### 4.2.2 Minimal Time Model

The MT propagation model checks every cell per timestep to determine whether it has been ignited or not. If the cell has been ignited, then its neighbors must be looked at. The spread method from neighbor to neighbor checking can be seen in Figure 2.1. Ignited neighbors are ignored in order to not waste computation time. A cell in the current timestep can be checked to see if a earlier TOA is available. If the cell is not on fire the TOA for that cell is calculated using the Equation 4.1 mentioned above. This method starts and increments with a dynamically calculated timestep in order to avoid waiting through possible time steps in which no cell is ignited. The timestep is based on the occurrence of a new cell being ignited, which means a TOA is calculated for that cell. This new TOA is stored in a ‘timeNext’ value and is compared to the current ‘timeNext’ value. If the new value is less than the current TOA then we need to adjust our timestep for the smaller TOA to avoid the race condition. The MT propagation model’s logic is displayed step by step in the following Algorithm 4.3.

---

**Algorithm 4.3** Minimal Time Algorithm
 

---

```

1: for cell = 0 to numCells do
2:   if timeNext > ignTime[cell] AND ignTime[cell] > timeNow then
3:     timeNext = ignTime[cell]
4:   else if ignTime[cell] == timeNow then
5:     // Propagate Fire
6:     for n = 0 to 15 do
7:       //If neighbor is unburned
8:       if ignTime[neighborCell] > timeNow then
9:         ROS = Compute ROS according to Equation 4.1
10:        ignTimeNew = timeNow +  $L_n/ROS$ 
11:        if ignTimeNew < ignTime[neighborCell] then
12:          ignTime[neighborCell] = ignTimeNew
13:        end if
14:        if ignTimeNew < timeNext then
15:          timeNext = ignTimeNew
16:        end if
17:      end if
18:    end for
19:  end if
20: end for

```

---

### 4.2.3 Iterative Minimal Time Model

The IMT propagation model is used to calculate the time of arrival for a cell independent of neighboring cell data. Each cell determines its own TOA based purely on spread rates. The cell will calculate the correct TOA when a convergence threshold is reached in between the steps  $k$  and  $k + 1$ . The difference between the minimal time values for the two steps must be lower than the convergence threshold in order to be marked as complete and notify the cell it has the correct TOA. According to Sousa, dos Reis, and Pereira [41] the threshold value is found through trial and error experimentation. The Iterative Minimal Time approach can be seen in Algorithm 4.4 and the cell check method can be found in Figure 2.1.

---

**Algorithm 4.4** Iterative Minimal Time Algorithm
 

---

```

1: for  $cell = 0$  to  $numCells$  do
2:   // Check for simulation completion:
3:   if  $|ignTime[cell] - ignTimeNext[cell]| < thresh$  then
4:     //Mark as converged
5:   end if
6:   if  $ignTime[cell] > 0$  then
7:      $ignTimeMin = INF$ 
8:     //Propagate Fire
9:     for  $n = 0$  to 15 do
10:       $ROS =$  Compute ROS according to Equation 4.1
11:       $ignTimeNew = timeNow + L_n/ROS$ 
12:       $ignTimeMin = MIN(ignTimeNew, ignTimeMin)$ 
13:    end for
14:   end if
15: end for

```

---

### 4.3 Parallel Implementation

Unlike the sequential implementation the bulk of the pre-processing was moved from the CPU or HOST and ported to the GPU or DEVICE for computation speedup. The only pre-processing that does not take place in the parallel implementation is essentially the file I/O. File read in and out is not a capability in GPGPU at this time. As mentioned before CUDA is the programming language made available and developed by Nvidia [26]. This does not limit the access to running the fire simulator if one does not have an Nvidia capable device. The construction of the web-service platform allows any user with internet access can run a fire simulation. The web-service will be discussed later on but in short it communicates with the CUBIX [13] server to run the kernels on Nvidia capable GPU's. Kernels are instructions written for the GPU's Single Instruction Multiple Data processors (SIMD). These kernels execute the propagation logic on thousands of these processors, each containing thousands of threads that can execute the code with their own data. The data is the fire simulation cells divided optimally on the threads. Threads each get data to compute, because idle threads is a waste of possible computation power. The simulation run on the GPU is outlined in Algorithm 4.5 which displays how the kernels are launched.

---

**Algorithm 4.5** Simulation Composition
 

---

```

while Simulation !Complete do
  computeKernel<<< Blocks, Threads >>>(inputs)
  terminateKernel<<< Blocks, Threads >>>(inputs)
end while

```

---

Algorithm 4.5 displays the blocks and threads variables for the kernel calls. This is interactive before each simulation is run. They can be changed to optimize the speed of the simulation, however these values do change in the context of where the fire simulation is geographically located, the size of the terrain, total number of cells, cell size, and resolution or granularity of the fuel model data. This simply implies that optimizing these values for the given context is important and at the moment is only available to the UI developer and should be considered future work to make autonomous or at the very least interactive. Multiple kernels have to be implemented to allow for block-wise synchronization. Once a kernel is done blocks are synced and the data is preserved. The sequential version simply loops through to mimic this effect.

### 4.3.1 Minimal Time

Due to the sequential implementation logic being almost exact to the parallel implementation, as it should, the algorithm is not displayed. The minor changes are effectively due to GPGPU compute architecture. Kernels code is executed on every thread and there are thousands of threads active at the same time. This creates the balancing act of data synchronization. During the fire simulation every cell is being handled by a single thread. However, all the threads must access the TOA mapped values in order to read and write back. This sets up the issue of race conditions again in a parallel sense. If a thread writes back a time of arrival value before a thread that needed the read the previous value stored the fire simulation will become compromised and the accuracy will fail. Atomic operations which are unique to CUDA are the only way to read and write to a shared memory location safely and avoid the race condition [35]. In order to circumvent the overwriting of a smaller time of

arrival this thesis used the built in `AtomicMin()` function to guarantee that which ever thread at the current time is accessing that value location is locked. This means that no other thread can read or write to this location while it is being processed by another. This solves the race condition problem. There is a disadvantage to using atomic operations and it is the issue of speed. Since only a single thread can access this location other threads have to wait in order to ensure accuracy. At the moment there is no way around this but possibly can be optimized or avoided in the future due to developing software and or hardware.

Minimal Time's method of dynamic time stepping also posed its own issue. In GPGPU computation architecture threads are not designated in any order, therefore there is no way to determine when all threads are done processing their data. For example, thread ID 0 could be the thread to finish its computation last and thread ID 500 finishes first [35]. The solution as seen in Algorithm 4.5 is the termination kernel. Having a second kernel is more overhead, but it saves the program from read and writing data back and forth between the HOST and DEVICE which creates even more overhead. Once the first kernel is done this guarantees a block-wise synchronization which also implies that all threads are done processing. The second kernel then assigns the old `timeNext` variable to the new `timeNext` and sets the old to `INF`. This kernel is a single operation on a single thread. This is possibly the worst use of GPGPU computing, however it is the most optimal solution at the moment with the least amount of overhead and will have to be looked at in future work.

### 4.3.2 Iterative Minimal Time

The parallel IMT propagation method is once again very similar to the sequential and therefore will not have pseudocode presented. Also, since IMT is very similar to MT in cell neighbor check logic it has the same synchronization issue present in the MT method. Fortunately, for this method we do not need to essentially force an undesirable kernel with a single thread operation as in the MT method. Using two in and out vectors respectively is a simple solution with minimal if any downside.

The race condition is the same for this method where a TOA value is overwritten before another thread was able to access the previous information that it needed to be correctly processed. Using the two vectors we always have a old value stored and a place for the new value to go without overwriting the preexisting one. If it is unclear as to why this is not a solution for MT it is because of the data context. MT's issue was with its dynamic time stepping necessary for an accurate fire simulation and the fact that all the threads needed to be synchronized before the next loop could even begin. In IMT the data has to do with the old and newly calculated time of arrivals for each cell. This does not need to wait for the threads to be synchronized to keep processing the fire simulations cells and therefore a more efficient solution to allow asynchronous data access. Once the kernel is done, the next kernel will copy the old TOA map vector with the new TOA mapped vector and repeat every time step. This avoids the possibility of the read and write errors that will arise from using a single time of arrival vector.

### 4.3.3 Burn Distances

The burn distance propagation method is very similar to the sequential implementation and has the same race condition as the iterative minimal time propagation method. Pseudocode can be referred to Algorithm 4.2 and the same solution for IMT was used.

## 4.4 Other Features

This thesis presents an advanced vFireLib.v2. The advancements in functionality and user interaction include wind, moisture, and vegetation manipulation. Crowning fire calculations were optimized and the user of the fire simulator now has the option to toggle crown fires with spotting fire effects, or without. In the previous work presented by [39] crowning was incomplete without spotting and spotting computation was discussed but not implemented. vFireLib.v2 changes the pre-processing for the parallel implementation immensely compared to the previous vFireLib.v1 [39]. This

allowed the time step calculations to no longer be dependent on the HOST; this also includes: calculating the roth data, and finding the maximum spread rate for a given wildfire simulation. This was able to allow implementation for all the added features with insignificant increase to total run time. This will be discussed more so in the next chapter with the timings and results.

#### **4.4.1 Spotting**

Spotting fire effects was the next big change to the simulator. This calculates based off crown fires reaching a threshold intensity great enough to produce fire embers. These fire embers can be created by any cell in which the crown fire threshold has been reached. Once the cell sets off an ember it is tracked. The ember is affected by wind and gravity to determine where it lands in the simulation and whether or not it is within the bounds of the simulation. Once the ember has landed probability under certain moisture and fuel model variables play a role in the chances of the ember igniting a fire ahead of the main fire line. This happens regularly in severe wild land fires; the probability of ignition is demonstrated in Figure 4.1.

	DB Temp (°F)	1-hr Moisture Content (%)															
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
<b>0-10% shading</b>	110+	100	100	90	80	70	60	50	40	40	30	30	30	20	20	20	10
	100-109	100	90	80	70	60	60	50	40	40	30	30	20	20	20	10	10
	90-99	100	90	80	70	60	50	50	40	30	30	30	20	20	20	10	10
	80-89	100	90	80	70	60	50	40	40	30	30	20	20	20	10	10	10
	70-79	100	80	70	60	60	50	40	40	30	30	20	20	20	10	10	10
	60-69	90	80	70	60	50	50	40	30	30	30	20	20	20	10	10	10
	50-59	90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
	40-49	90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
	30-39	90	70	60	60	50	40	40	30	30	20	20	20	10	10	10	10
	<b>10-50% shading</b>	110+	100	100	80	70	60	60	50	40	40	30	30	20	20	20	10
100-109		100	90	80	70	60	50	50	40	40	30	30	20	20	20	10	10
90-99		100	90	80	70	60	50	40	40	30	30	30	20	20	20	10	10
80-89		100	90	80	70	60	50	40	40	30	30	20	20	20	10	10	10
70-79		100	80	70	60	50	50	40	40	30	30	20	20	20	10	10	10
60-69		90	80	70	60	50	50	40	30	30	20	20	20	20	10	10	10
50-59		90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
40-49		90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
30-39		80	70	60	50	50	40	30	30	20	20	20	10	10	10	10	10
<b>60-90% shading</b>		110+	100	90	80	70	60	50	50	40	40	30	30	20	20	20	10
	100-109	100	90	80	70	60	50	50	40	30	30	30	20	20	20	10	10
	90-99	100	80	80	70	60	50	40	40	30	30	20	20	20	10	10	10
	80-89	100	80	70	60	60	50	40	40	30	30	20	20	20	10	10	10
	70-79	90	80	70	60	50	50	40	30	30	30	20	20	20	10	10	10
	60-69	90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
	50-59	90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
	40-49	90	70	60	50	50	40	30	30	30	20	20	20	10	10	10	10
	30-39	80	70	60	50	50	40	30	30	20	20	20	10	10	10	10	10
	<b>100% shading</b>	110+	100	90	80	70	60	50	50	40	30	30	30	20	20	20	10
100-109		100	90	80	70	60	50	40	40	30	30	20	20	20	20	10	10
90-99		100	80	70	60	60	50	40	40	30	30	20	20	20	10	10	10
80-89		90	80	70	60	60	50	40	30	30	30	20	20	20	10	10	10
70-79		90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
60-69		90	80	70	60	50	40	40	30	30	20	20	20	10	10	10	10
50-59		90	70	60	60	50	40	40	30	30	20	20	20	10	10	10	10
40-49		80	70	60	50	50	40	30	30	20	20	20	10	10	10	10	10
30-39		80	70	60	50	40	40	30	30	20	20	20	10	10	10	10	10

Figure 4.1: Probability of Ignition [21]

#### 4.4.2 Pause and Resume

A number of advancements were made to allow user manipulation possible as well. This thesis provides a fire simulator where the user can change in real-time moisture,

wind, and vegetation. The user can then see the immediate effects of this change to the fire's spread rate, direction, and intensity. To allow for this to be possible this work had to make one more change to the program; a pause and resume function had to be implemented. This is intuitive for a simulator to be able to do, but the variables needed were extensive. To be able to pause the simulation the development of saving the state for every cell up to a certain timestep was crucial. This included cell values such as the current maximum rate of spread, current rate of spread, which cells are on fire, not on fire, where embers are in the simulation, what the crowning thresholds are currently, etc. To resume the simulation just as much data was needed. Resume needed to know what changes were made to wind, moisture, or vegetation from user manipulation. This was to mimic the possible changes in weather or to gauge how a suppression effort event would affect the overall spread of the fire. An example of these changes could be as simple as rain or an increase or decrease in wind, it could demonstrate the effect a bulldozed area of vegetation would retard the fire or even an aerial water drop. Upon resumption the simulator would need to recalculate the roth data as well, find the new maximum rate of spread, and redetermine the appropriate timestep; these reasons alone was enough to port these computations to the DEVICE/GPU instead of leaving them to the serial HOST/CPU. vFireLib.v2 provides the user with more functionality and there is still more to add for future work; these will be discussed in the last chapter.

### **4.4.3 Moisture, wind, and Vegetation Manipulation**

Manipulation of moisture, wind, and vegetation was one of the primary objectives to include within this fire simulator. Wind manipulation consisted of adding an x and y vector that holds the value for each individual cell in the simulation. This value can be changed through file editing or through our web-service interface which is described more in the next chapter. Vegetation manipulation can also be changed for each cell individually much the same way as wind. The biggest concern was change in moisture. Moisture can be affected by water drops in wildfire suppression efforts,

precipitation, and even certain fuel models have a dynamic moisture content [21]. A dynamic fuel models moisture content transfer from live herbaceous to dead fuel is represented by Figure 4.2.

<b>Herbaceous Moisture Content</b>	<b>Level of Curing (fuel load transferred)</b>	<b>Curing Classification</b>
<b>120% or more</b>	0/1 cured	Uncured
<b>98%</b>	¼ cured	Partially cured
<b>90%</b>	1/3 cured	Partially cured
<b>75%</b>	½ cured	Partially cured
<b>60%</b>	2/3 cured	Partially cured
<b>53%</b>	¾ cured	Partially cured
<b>30% or less</b>	1/1 cured	Fully cured

Figure 4.2: Dynamic Fuel Load Transfer Table [21]

# Chapter 5

## Results

### 5.1 Results

#### 5.1.1 Overview

All timings for the sequential and parallel version were run on the same desktop for consistency. The results include the pre-processing computation. For the sequential this changes nothing, except for the addition of necessary data for the included features. This simply implies that the pre-processing for the sequential is all the same as vFireLib.v1 but has to manage more data. This did not seem to cause too much of a delay in the timings. For the parallel implementation the pre-processing was dramatically changed. In vFireLib.v1 the Rothermel values and all the data necessary for their computation was done on the CPU. As well as calculating the maximum spread rates for the entire simulation, finding the maximum of the max spread rates which is used to calculate the dynamic time step needed for all three burn methods. This was all moved to the DEVICE in order to take advantage of GPGPU computing. The timings for all was done with Kyle Canyon in Las Vegas, Nevada. The terrain was extracted in TIFF format, the wind is set uniformly to no speed, moisture is default and respective of the fuel models found in the areas type of vegetation. All of this data can be found at LANDFIRE [42]. The results can be found in the following sections.

## Hardware

The hardware was an Intel i7-3770 with a base clock rate of 3.40 GHz and a maximum clock rate of 3.90 GHz, 4 cores with 2 threads per core, and a 8 MB Cache. The RAM available was approximated at a little over 20 GB and the Graphics Processing Unit was an Nvidia GeForce GTX 1080 TI. The GPU specs are 3,584 cores and 11 GB GDDR5X on card memory.

## Timings

The tests for the timings ranged on a cell by cell area starting as small as 64x64 to 906x642. The values in between were kept to the powers of two and were square. The final cell by cell area that is not a power of 2 is actually the size of Kyle Canyon and can be seen in Figure 5.1. This was done to show a real-time use of the fire simulation. These timings were done with spotting fire effects enabled, because in vFireLib.v1 it was mentioned that crowning fire effects did not significantly change the run time of the simulation. To show the significant change in run time spotting

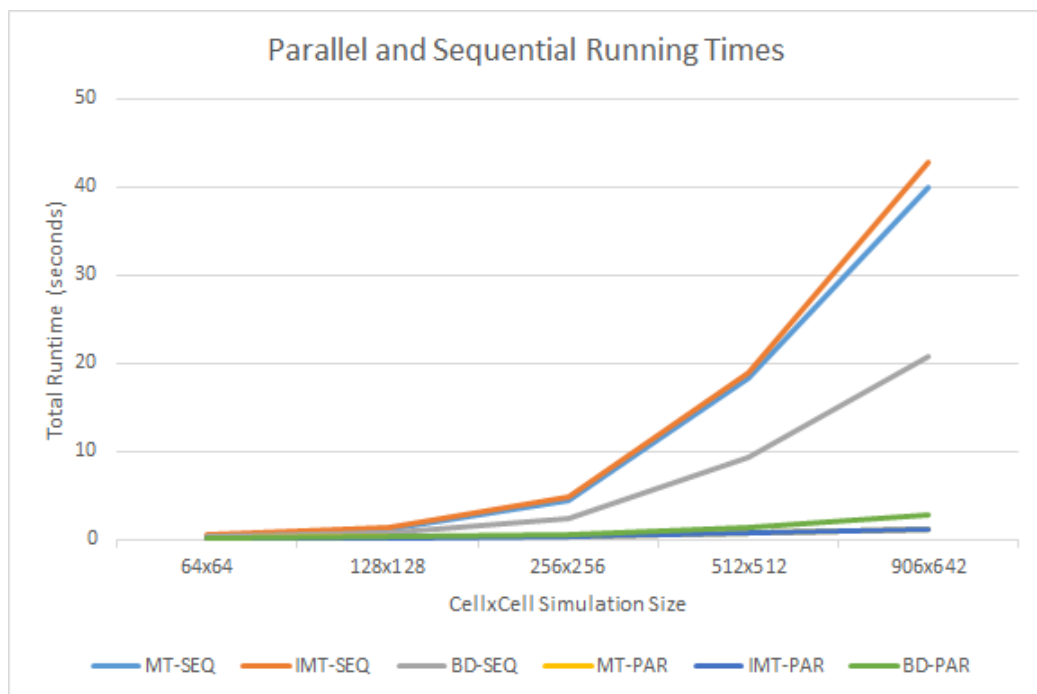


Figure 5.1: Timings in seconds for all the implementations with spotting.

imposed upon the simulation.

Table 5.1 shows the run times of both the parallel and sequential without spotting. Table 5.2 displays the run times of both sequential and parallel with spotting. The sequential run times with and without spotting did not change significantly. This is most likely due to the fact that the spotting effect computation isn't the issue in regards to complexity and computational burden. It more has to do with the fact that the sequential implementation does not need to pass large amounts of data back and forth through PCIE slots. GPGPU computing requires data to be passed from the HOST to the DEVICE, compute the simulation, then pass back all pertinent data back. The pass back from the GPU is usually less. Spotting, wind, and moisture manipulation added large amounts of data to be passed from the HOST to the DEVICE. This is likely the cause in the significant change in run time for the parallel version.

Table 5.1: Execution times without Spotting

cells	Sequential (seconds)			Parallel (seconds)		
	MT	IMT	BD	MT	IMT	BD
64	0.482	0.467	0.310	0.230	0.251	0.246
128	1.230	1.345	0.770	0.224	0.238	0.227
256	4.322	4.642	2.493	0.333	0.329	0.316
512	16.837	18.450	9.166	0.505	0.691	0.561
KC	36.561	40.750	20.078	0.917	1.029	0.925

Table 5.2: Execution times with Spotting

cells	Sequential (seconds)			Parallel (seconds)		
	MT	IMT	BD	MT	IMT	BD
64	0.405	0.506	0.322	0.213	0.236	0.228
128	1.216	1.289	0.744	0.271	0.256	0.295
256	4.350	4.843	2.503	0.371	0.429	0.521
512	18.245	18.891	9.406	0.688	0.825	1.424
KC	40.007	42.864	20.697	1.101	1.177	2.738

Speedup, which is the sequential run time divided by the parallel run times, can be seen in Figure 5.2. This is scaled in seconds and shows the vast difference between the parallel and sequential run times. The second speedup graph is on a logarithmic scale and is shown in Figure 5.3. In vFireLib.v1 the sequential usually beat out the parallel implementation in the smaller cell by cell ranges. However, due to hardware differences and advancements, this has changed.

Fire simulation on large terrains allow GPGPU computing to show that it has a high capability to compute such large data dependent problems. Regarding the Kyle Canyon timings, it is important to note that with or without spotting the entire fire can be simulated within seconds. The quick computation added feature of a pause and resume function makes the parallel implementation even more suited for real time results.

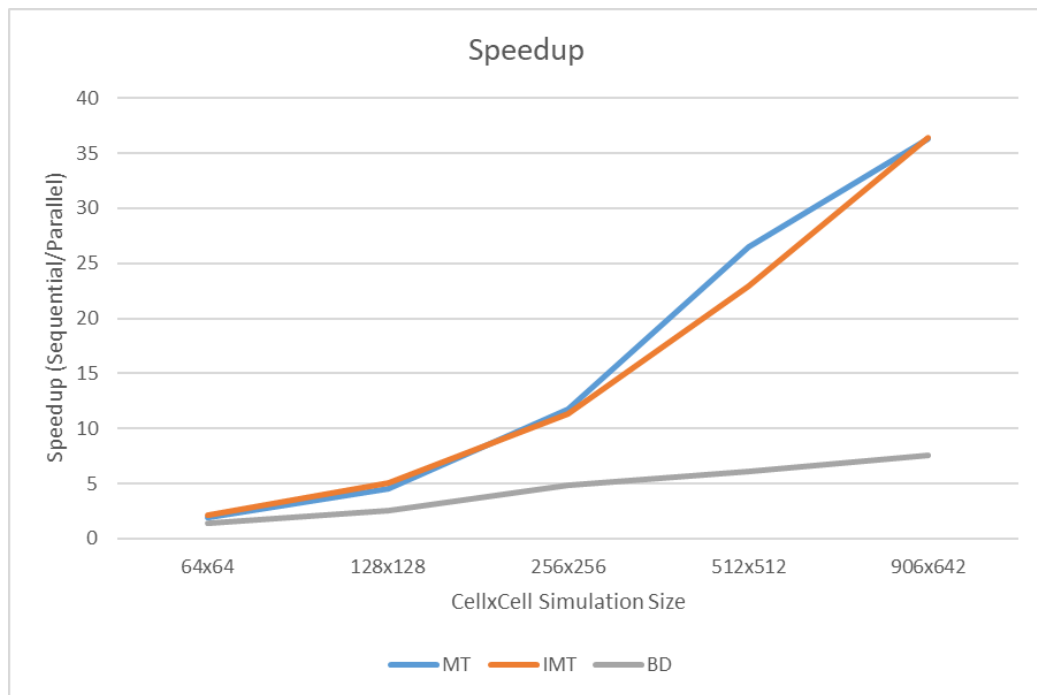


Figure 5.2: Speedup graph found by dividing CPU/GPU running times.

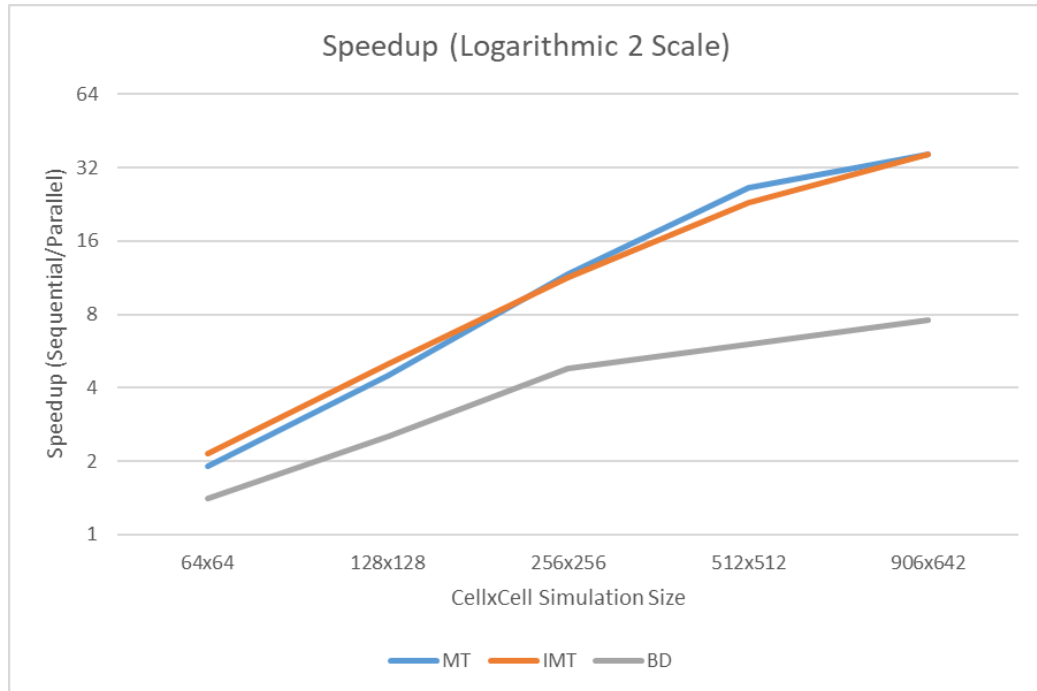


Figure 5.3:  $\text{Log}_2$  based graph of the speedup.

### 5.1.2 Web Service Architecture

To make it easier to use `vFireLib`, we have designed and implemented a wrapper for the library. Any other applications can submit fire simulation request and obtain the simulation outputs through RESTful apis. REST is short for representational state transfer and is an architectural style of application programming interfaces, which is defined in Roy Thomas Fielding PhD dissertation [22].

By using a RESTful api, the client side does not need to know details about the server side and the client side can finish operations with GET, POST, PUT, and DELETE requests. In this paper, we present two example systems (a web-based visualization application and a 3D unity visualization application) to demonstrate how to use the library through the RESTful api.

Our system architecture is displayed in Figure 5.4. We created a wrapper on top of `vFireLib` to make it available to other applications. This means all the simulations are done in `vFireLib` and the inputs and results are stored in the wrapper. Other

applications only need to get and post data using RESTful apis offered by the web service segment. Here are some RESTful api examples:

- **/upload\_scenario (POST):** The application can post a scenario (zip file) to the server as an input.
- **/upload\_files (POST):** The application can use this api to post four input files to the server. The files are fuel model, on-fire cells, x wind, and y wind.
- **/api/scenario\_zip (GET):** This api returns current simulation scenario as a zip file. It contains current fuel model, on-fire cells, and wind information.
- **/api/get\_final\_results (GET):** This api returns a CSV (comma separated values) file containing the simulation results.
- **/api/get\_log (GET):** This api returns the vFireLib simulation log, such as the execution time and fire spread method.
- **/api/get\_err\_log (GET):** This api returns the vFireLib simulation error log. The common errors are the input files are supported (wrong format) and some input values are invalid. If the simulation does not have any errors, this api will return an empty file.

By using this architecture, the client side only needs to process and visualize the result files and the simulation procedure performance is guaranteed by the hardware on the server.

## 5.2 Web-Based Visualization Application

### 5.2.1 Overview

This section introduces the web-based client of our system. The web-based client is very easy to use and convenient. The user can modify vegetation, wind, and choose different places to start a fire. There are three methods to input data into the server side for the simulation: (1) upload the four input files (fuel model, on fire cells, x-axis

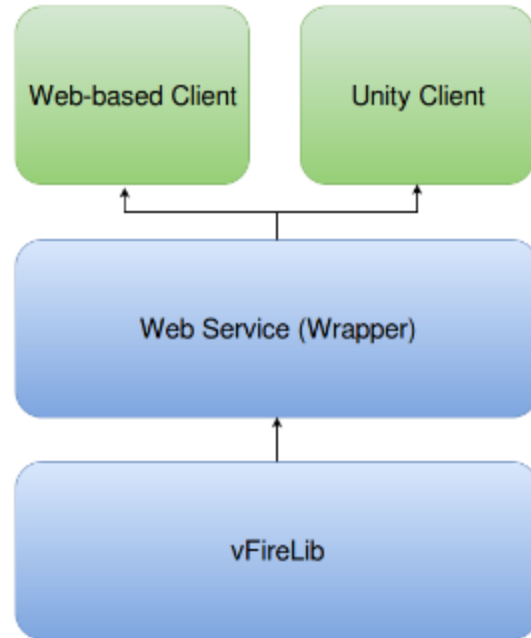


Figure 5.4: System Structure (Blue Rectangles–Server; Green Rectangles–Client)

wind file, y-axis wind file); (2) choose available datasets stored in the server; or (3) upload a scenario zip file. This component is built on our previous work [48].

Figure 5.5 shows a web-based client screenshot. The user can start the simulation by clicking the play icon. Then the system will display how the fire will spread with transparent red. The user is able to select different places to set a fire. All they need to do is to click and drag on the 2D grid map and release the mouse button (the chosen cells are marked with bright red), check the You want to choose on fire cells? checkbox, and click the play icon button. Then the client sends a request with the on-fire cells information to the server. The server reruns the simulation and returns the results back to the client. During the procedure, the play icon button is unclickable and shows loading..., which keeps the user from sending multiple requests to the server.

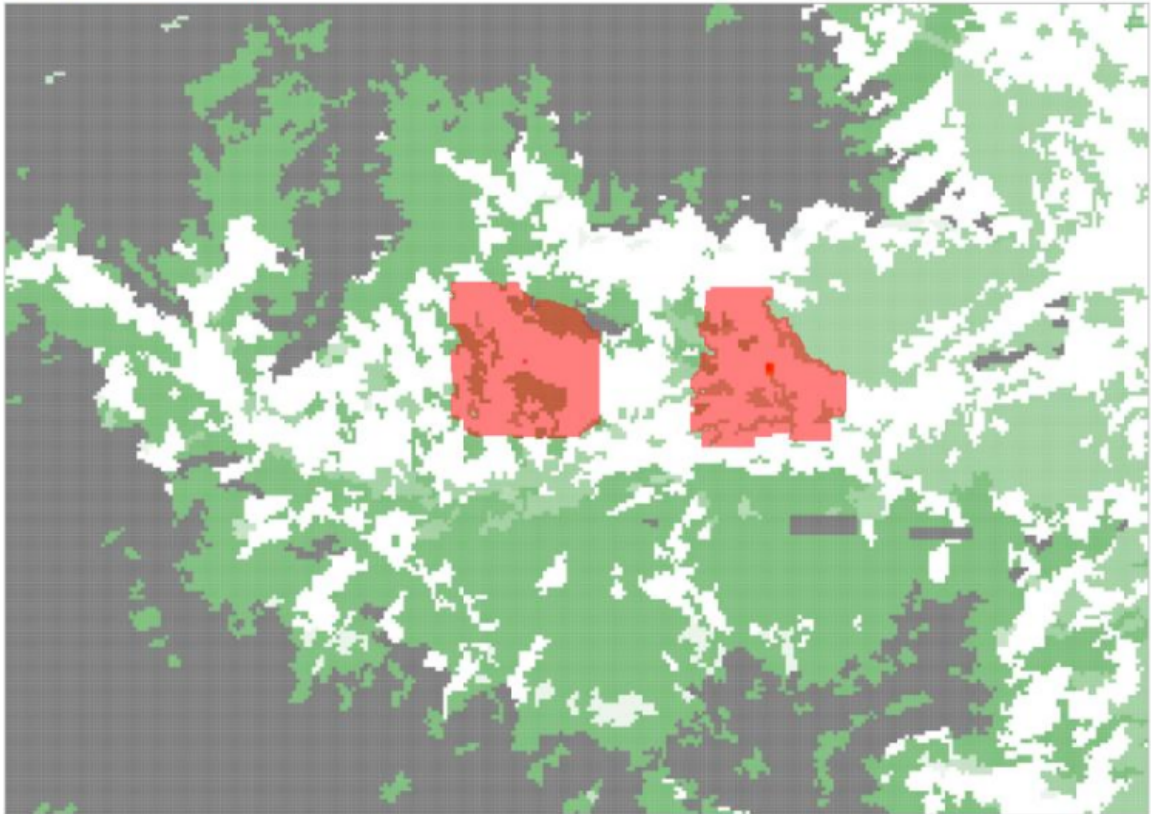
**vFireLib** supports eight vegetation types (more details can be found in Section 3). The user can modify vegetation coverage (fuel model) as Figure 5.6 displays. First, the user needs to choose a vegetation type from the radio buttons. Each radio button has a color square, which is used to represent specific type vegetation in

You can modify vegetation or wind

Vegetation Modification

Wind Modification

Display fire simulation with 2D map



You want to choose on fire cells?



Download Scenario

Figure 5.5: Web-based Client Screenshot, the red color means a cell is burning, the black color means a cell is unburnable, and other colors stand for different vegetation types. The user can start/stop the simulation and also choose cells to set on fire.



Figure 5.6: Vegetation Modification Bar. The user can change the vegetation on the 2D grid map by using the bar. When the cursor hovers over the button, detailed information will pop up (this can be seen in the white text on black background). The current version supports eight types of vegetation.

the 2D grid map. The number in the radio button is the vegetation index. When user's mouse cursor hovers over the radio button, the brief vegetation introduction will show up. Figure 5.7 shows an example where two fire barriers (bare ground - probably bulldozed) are set up in the middle of the 2D grid. The fire spread results show that the fire is blocked by the barriers and spreads faster on vegetation type 6 (Dormant brush, hardwood slash) than other vegetation types such as vegetation type 142 (Moderate Load Dry Climate Shrub). This part of work is inspired by the application introduced in [48].

Wind is another very important factor for the fire simulation. The web-based client presents the 2D wind with vectors as shown in Figure 5.8. The web-based client allows the user to modify the wind globally (all the cells) or choose different areas to change. Global wind modification is easy and simple. The user only needs to input x-axis wind value and y-axis wind value and click Confirm button. To change wind of a certain area, the user needs to choose the cells by clicking mouse left button, drag along a direction, and release the button. The chosen areas will be marked with yellow. After clicking Confirm button, the wind information of the chosen cells is replaced with the input x-axis wind and y-axis wind speeds. By repeating these steps, the user can modify different places with different wind vectors.

We know it may be hard for some users to collect data by themselves. Therefore, we have prepared some datasets on our server. The users can choose and study this data. After they modify the datasets at their will, the users can download the modified datasets as a scenario zip file. The users can also upload scenario zip files

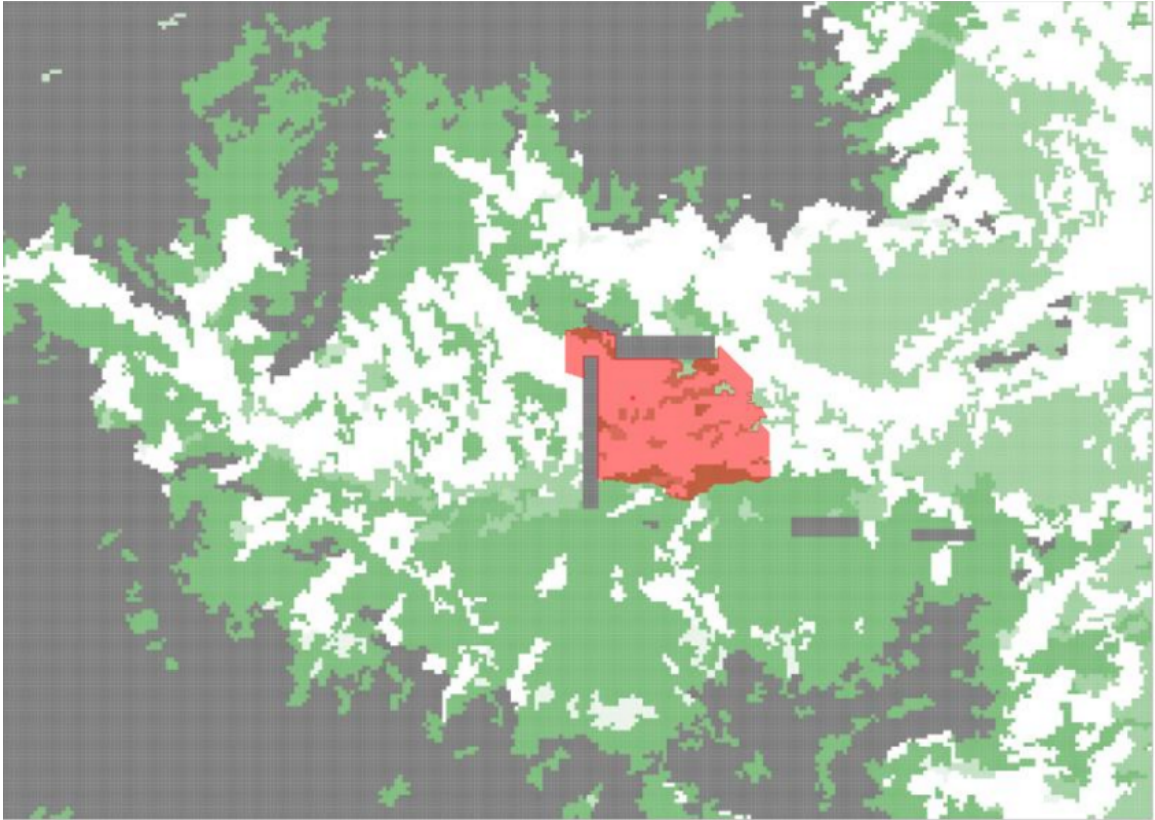


Figure 5.7: Vegetation Modification Map. This is an example that some cells (in the middle of the figure) are changed into unburnable. It is clear that these cells stop the fire spreading.

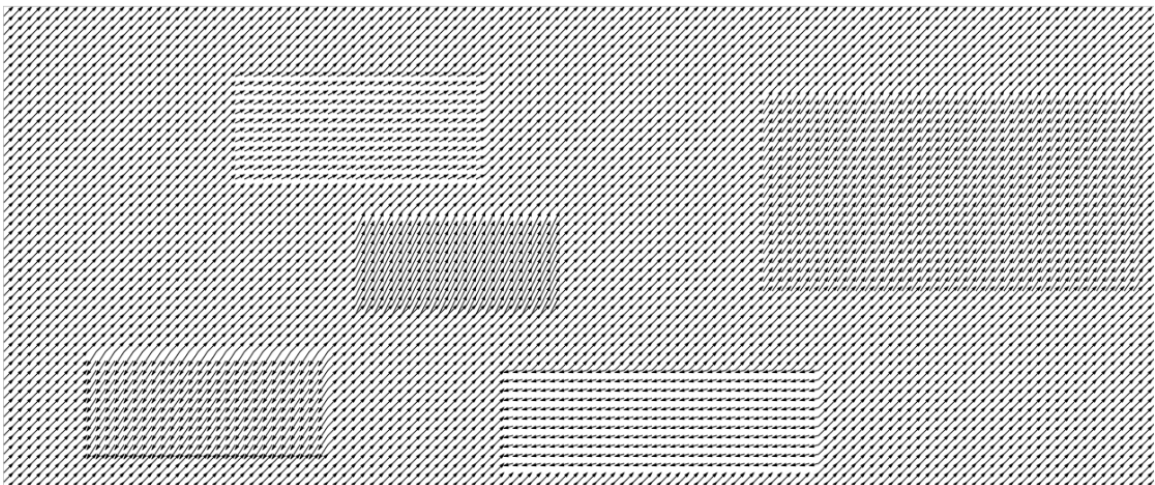


Figure 5.8: Wind Modification of Chosen Area. The user can change the 2D wind vectors in each cell. These vectors are represented by arrows. Longer arrows are used to represent more forceful wind and the arrow direction is the same as the cell wind direction.

to the server. In the future, we plan to allow users to automatically obtain data from some repositories, WindPower [47] and Global Index of Vegetation-Plot Database [7].

## **5.3 3D Visualization**

### **5.3.1 Overview**

The need to study the wildfire spread across terrain can be properly visualized by the web tool in a 2D view, but to be fully immersed in the data a 3D visualization gives a better understanding of the landscape. This is useful since fires can occur in difficult to get to areas and the fire tool is able to view the spread of the fire across the terrain in a virtual environment. The 3D visualization tool was built in the Unity Game Engine [44]. This visualization is built on top of an existing project that works with watershed science data. In the 3D world tools are available to the users to interact with the data displayed on the terrain.

The 3D visualization offers more features to the user than the web-based client. These features are all part of the need to make the fire spread more realistic and to give the user the perspective of a real world scene. Features such as being able to change the wind in a two dimensional format can be made to the fire simulation and then displayed in 3D to see the interaction between it and the fire's behavior.

### **5.3.2 Fire Integration**

The Unity application communicates with the fire web service to get the data that relates to when an area catches on fire. The application uses the Unity Networking library in order to communicate with the RESTful service. The returned data is parsed to pull out some meta-data on the fire and the values for each cell of the grid of terrain the fire spread was conducted on. The grid designed data has the value when the certain grid point catches on fire. This data must be translated into temporal data so it can be presented to the user in the application.

The fire component will generate each individual time of arrival which is then

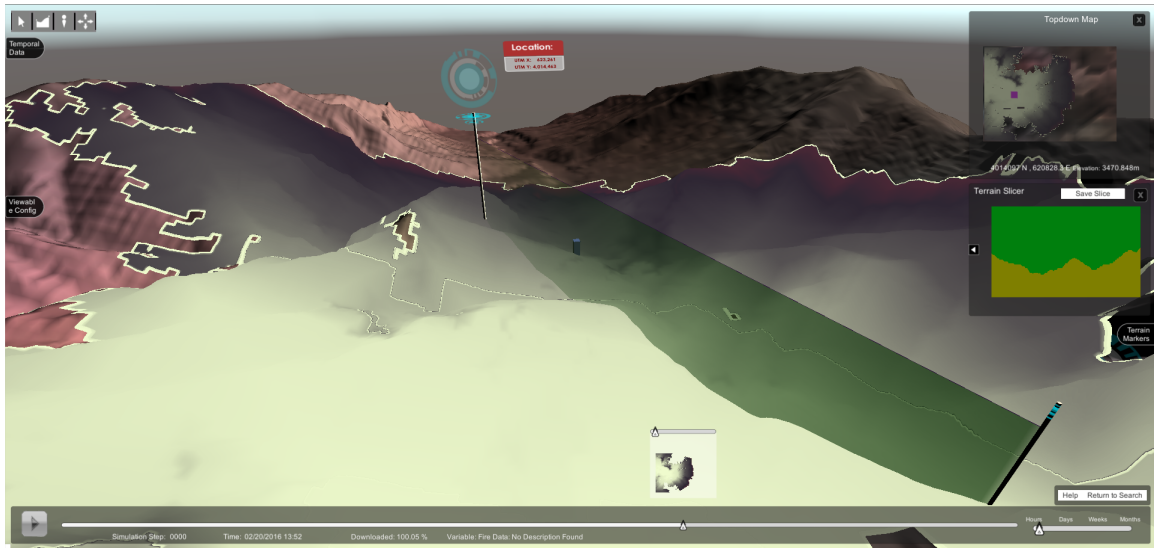


Figure 5.9: The Unity application where the white edges show where the fire edge is at this point in the simulation and the yellow to black coloring of the terrain shows the early to late time of arrival of the fire at that point.

viewed on the terrain. The terrain is generated from a digital elevation map (DEM). The DEM has geospatial data, therefore the fire data is spatially correct when viewed on the terrain. To texture the terrain in the virtual world the DEM geospatial data is used to pull satellite data from a free web service. The terrain, satellite texture, and the fire spread data can be seen running in the application in Figure 5.9.

## Features

The Unity application offers many features that were built to handle environment data. These tools include a time slider across the bottom that will allow you to move through temporal data, a configuration panel to adjust the presentation of data, a terrain slicer, and a data point graph. These tools were catered toward the initial data that was being researched which dealt specifically with snow pack modeling, but they work with the fire data as well.

The time slider bar is similar to video players, allowing the play/pause, speed of viewing, and scrubbing of the temporal data. The time slider is the controller of the data that is viewed on the 3D terrain. The time slider will show a preview of the

data in a small box that is the exact representation of the data placed on the terrain. In relation to the fire data, it is loaded into the data set and the time slider will play through the spread of the fire across the terrain.

The configuration panel is the main controls of the way the data is presented to the user and viewed on the terrain. There are preset color patterns that will represent the values associated to each point of the data. These colors can be adjusted to represent a greater range of values, which is useful when attempting to pinpoint a specific time frame the fire has started. Another feature of the panel allows for the data representation as a point or bi-linear interpolation. The point will color each cell of the grid to the proper color value, and the bi-linear will interpolate the colors of the cells around to give a more blended image. The panel also offers the option for the user to export the current image being viewed.

# Chapter 6

## Conclusions and Future Work

### 6.1 Conclusions

This thesis describes the implementation of a working fire simulator using the GPU enhanced library `vFireLib.v2`. Our fire simulator can calculate a fire spread in three different propagation methods known as Minimal Time, Iterative Minimal Time, and Burn Distances. The parallel implementations of these fire propagation methods when compared to the sequentially implemented algorithms produced significant speedups in every spread method. The results presented in this thesis show clearly that a forest fire simulator can be implemented using the GPU as the main processing workhorse. Using the GPU, a real-time simulator has now been created, and it is a vast improvement over what was the current state-of-the-art. The work presented in this thesis is a comprehensive forest fire library with the ability to calculate fire effects such as: active and passive crown fires, fire spotting occurrences, 2 single dimensional wind vectors that create a modifiable X and Y wind field and cell to cell vegetation manipulation, as well as moisture manipulation and the ability to simulate a wild land fire in user specified time ranges. `vFireLib.v2` is available as a service and can be run through a web application or through a Unity3D game application.

### 6.2 Future Work

For future work there are areas that can be improved upon to incorporate the GPU's strength in its ability to conduct asynchronous data transfers. The ability to transfer

partially complete data back to the visualization system would increase the versatility of the simulator. For example, we would be able to process calculations simultaneously while portions of the simulation are being viewed by the user or audience. However, the kernels will need to be adjusted to account for the asynchronous data transfers. Currently our `vFireLib.v2` does all or none with regards to data transfers, the first call after the pre-processing step to transfer data from the CPU to the GPU and the second call once the kernels have finished calculating the entire simulation back to the CPU from the GPU.

Other potential functionality is to expand the role of GDAL in the fire simulator to give the user the choice to choose any location through a given latitude and longitude value and be able to pull or create the appropriate terrain data files in order to run a customized fire simulation at any location in the real world. This would be accompanied by the development to pull the correct data for the corresponding vegetation and wind data for the terrain region chosen. The last addition to the fire simulator is to implement and produce fire and smoke effects within the unity application; as well as tree placement according to satellite imagery. These additional features would create an even more in depth and accurate fire simulator and provide a more immersive 3D experience for the Unity application.

# Bibliography

- [1] Frank A Albini. Estimating Wildfire Behavior and Effects. Technical report Gen. Tech. Rep. INT-30, USDA Forest Service, Ogden, UT: Intermountain Forest and Range Experiment Station, 1976. URL: <https://www.fs.usda.gov/treesearch/pubs/29574> (visited on 09/02/2018).
- [2] Martin E. Alexander and Miguel G. Cruz. The elliptical shape and size of wind-driven crown fires. *Fire Management Today*, 73(4):28–33, August 2014.
- [3] Ilkay Altintas, Block Jessica, Raymond de Callafon, Daniel Crawl, Charles Cowart, Amarnath Gupta, Mai Nguyen, Hans-Werner Braun, Jurgen Schulze, Gollner Michael, Amaud Trouve, and Larry Smarr. Towards an integrated cyberinfrastructure for scalable data-driven monitoring, dynamic prediction and resilience of wildfires. *Procedia Computer Science*, 51:1633–1642, 2015. DOI: <https://doi.org/10.1016/j.procs.2015.05.296>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050915011047> (visited on 09/02/2018).
- [4] Hal E. Anderson. Aids to Determining Fuel Models For Estimating Fire Behavior. Technical report Research Paper INT-RP-122, USDA Forest Service, Ogden, UT: Intermountain Forest and Range Experiment Station, 1982. URL: <https://www.fs.usda.gov/treesearch/pubs/6447> (visited on 09/02/2018).
- [5] Patricia L Andrews. BEHAVE: fire behavior prediction and fuel modeling system-BURN subsystem, Part 1. Technical report INT-194, USDA Forest Service, Intermountain Forest and Range Experiment Station, 1986. URL: <https://www.fs.usda.gov/treesearch/pubs/29612> (visited on 09/02/2018).
- [6] Bachisio Arca, Tiziano Ghisu, and Giuseppe A Trunfio. GPU-accelerated multi-objective optimization of fuel treatments for mitigating wildfire hazard. *Journal of Computational Science*, 11:258–268, 2015.
- [7] Fabio Attore, Flavia Landucci, Florian Jansen, Gabriela Lopez-Gonzalez, Jrgen Dengler, and Wieger Wamelink. Global index of vegetation-plot databases (givd). URL: <http://www.givd.info/> (visited on 02/13/2017).
- [8] NV Baranovskiy. Algorithms for parallelizing a mathematical model of forest fires on supercomputers and theoretical estimates for the efficiency of parallel programs. *Cybernetics and Systems Analysis*, 51(3):471–480, 2015.

- [9] Collin D Bevins. fireLib user manual and technical reference. Technical report, Systems for Environmental Management, October 1996. URL: <http://www.fire.org/downloads/fireLib/1.0.4/firelib.pdf> (visited on 09/02/2018).
- [10] Chase D Carthen, Thomas J Rushton, Nolan Burfield, Christine M Johnson, Aaron Hesson, Daniel Nielson, Bryan Worrell, Donna M Delparte, Tucker Chapman, W Joel Johansen, Roger Lew, Nicholas R Wood, Mathew Ziegler, John W Anderson, Sergiu M Dascalu, and Frederick C Harris Jr. Virtual watershed visualization for the wc-wave project. *International Journal of Computers and Their Applications*, 23:195–207, 2016.
- [11] Chase D Carthen, Thomas J Rushton, Christine M Johnson, Aaron Hesson, Daniel Nielson, Bryan Worrell, Donna M Delparte, W Joel Johansen, John W Anderson, Roger Lew, Nicholas R Wood, Mathew Ziegler, Sergiu M Dascalu, and Frederick C Harris Jr. Design of a virtual watershed client for the wc-wave project. In *Collaboration Technologies and Systems (CTS), 2015 International Conference on*, pages 90–96. IEEE, 2015.
- [12] EA Catchpole, Martin E. Alexander, and A.M. Gill. Elliptical-fire perimeter- and area-intensity distributions. *Canadian Journal of Forest Research*, 22(7):968–972, February 1992.
- [13] Cubix. Cubix xpander. URL: <http://www.cubix.com/xpander/> (visited on 02/13/2017).
- [14] James A. Cumming. Effectiveness of prescribed burning in reducing wildfire damage during periods of abnormally high fire danger. *Journal of Forestry*, 62:535–537, 8, August 1964. DOI: <https://doi.org/10.1093/jof/62.8.535>.
- [15] Developer Zone CUDA Toolkit Documentation. B.3. built-in vector types. 2018. URL: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#built-in-vector-types> (visited on 08/30/2016).
- [16] Roy Thomas Fielding. *Architectural styles and the design of network-based software architecture*. PhD thesis, University of California, Irvine, 2000.
- [17] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
- [18] Mark Arnold Finney. FARSITE: Fire area simulator: model development and evaluation. Technical report Res. Pap. RMRS-RP-4, U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station, 2004. URL: <https://www.fs.usda.gov/treesearch/pubs/4617> (visited on 09/02/2018).
- [19] GDAL Development Team. *GDAL - Geospatial Data Abstraction Library, Version x.x.x*. Open Source Geospatial Foundation. 2000. URL: <http://www.gdal.org> (visited on 02/13/2017).

- [20] Ross W Gorte and Kelsi Bracmort. Forest fire/wildfire protection. Technical report 7-5700, Congressional Research Service, CRS Report for Congress, March 2012. URL: <https://fas.org/sgp/crs/misc/RL30755.pdf> (visited on 09/02/2018).
- [21] National Wildfire Coordinating Group. Fire behavior field reference guide. URL: <http://www.fbfrg.org/crown-fire/crown-fire-behavior> (visited on 08/05/2018).
- [22] Roger V. Hoang. *Wildfire Simulation On the GPU*. Master's thesis, University of Nevada, Reno, December 2008.
- [23] Roger V. Hoang, Matthew R. Sgambati, Timothy J. Brown, Daniel S. Coming, and Frederick C. Harris Jr. Vfire: immersive wildfire simulation and visualization. *Computers & Graphics*, 34(6):655–664, 2010. ISSN: 0097-8493. DOI: <http://dx.doi.org/10.1016/j.cag.2010.09.014>.
- [24] Anna Klimaszewski-Patterson, Peter J. Weisberg, Scott A. Mensing, and Robert M. Scheller. Using paleolandscape modeling to investigate the impact of native americanset fires on pre-columbian forests in the southern sierra nevada, california, usa. *Annals of the American Association of Geographers*:1–20, 2018. DOI: <https://doi.org/10.1080/24694452.2018.1470922>. URL: <https://www.tandfonline.com/doi/full/10.1080/24694452.2018.1470922> (visited on 09/02/2018).
- [25] Neil P. Lareau and Craig B. Clements. The mean and turbulent properties of a wildfire convective plume. *Journal of American Meteorological Society*, 2017. DOI: <https://doi.org/10.1175/JAMC-D-16-0384.1>. URL: <https://journals.ametsoc.org/doi/10.1175/JAMC-D-16-0384.1>.
- [26] Nvidia. CUDA C programming guide, 2008. URL: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/> (visited on 02/13/2017).
- [27] Tonja Opperman, Jim Gould, Mark Finney, and Cordy Tymstra. Applying fire spread simulators in new zealand and australia: results from an international seminar. In *Fuels Management-How to Measure Success: Conference Proceedings RMRS-P-41*, pages 201–212, March 2006.
- [28] G. D. Papadopoulos and F. Pavlidou. A comparative review on wildfire simulators. *IEEE Systems Journal*, 5(2):233–243, June 2011. ISSN: 1932-8184. DOI: 10.1109/JSYST.2011.2125230. URL: <https://ieeexplore.ieee.org/document/5738359/> (visited on 09/02/2018).
- [29] E. Pastor, L. Zarate, E. Planas, and J. Arnaldos. Mathematical models and calculation systems for the study of wildland fire behaviour. *Progress in Energy and Combustion Science*, 29(2):139–153, 2003.

- [30] Seth H Peterson, Marco E Morais, Jean M Carlson, Philip E Dennison, Dar A Roberts, Max A Moritz, David R Weise, et al. Using HFire for spatial modeling of fire in shrublands. Technical report Res. Pap. PSW-RP-259, US Department of Agriculture, Forest Service, Pacific Southwest Research Station, 2009. URL: <https://www.fs.usda.gov/treearch/pubs/31914> (visited on 09/02/2018).
- [31] Caley Ramsay and David Shum. Ocean of fire destroys 2,400 structures but 85% of fort mcmurray still stands, May 2016. URL: <http://globalnews.ca/news/2688553/notley-in-fort-mcmurray-monday-to-survey-wildfire-damage/> (visited on 02/13/2017).
- [32] Richard C Rothermel. A mathematical model for predicting fire spread in wild-land fuels. Technical report Research Paper INT-RP-115, USDA Forest Service, Ogden, UT: Intermountain Forest and Range Experiment Station, 1972. URL: <https://www.fs.usda.gov/treearch/pubs/32533> (visited on 09/02/2018).
- [33] Richard C Rothermel. Effectiveness of prescribed burning in reducing wildfire damage during periods of abnormally high fire danger. *Journal of Forestry*, 62, 8, 1964.
- [34] Richard C Rothermel. How to predict the spread and intensity of forest and range fires. Technical report Research Paper INT-RP-143, USDA Forest Service, Ogden, UT: Intermountain Forest and Range Experiment Station, 1983. URL: <https://www.fs.usda.gov/treearch/pubs/24635> (visited on 09/02/2018).
- [35] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.
- [36] Joe H Scott and Robert E Burgan. Standard fire behavior fuel model: a comprehensive set for use with Rothermel’s surface fire spread model. Technical report Gen. Tech. Rep. RMRS-GTR-153, USDA Forest Service, Fort Collins, CO: Rocky Mountain Research Station, 2005. URL: <https://www.fs.usda.gov/treearch/pubs/9521> (visited on 09/02/2018).
- [37] O Séro-Guillaume, S Ramezani, J Margerit, and D Calogine. On large scale forest fires propagation models. *International Journal of Thermal Sciences*, 47(6):680–694, 2008.
- [38] Dave Shreiner. *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999. ISBN: 0201657651.
- [39] Jessica Smith. *vFireLib: A Forest Fire Simulation Library Implemented on the GPU*. Master’s thesis, University of Nevada, Reno, 2016.
- [40] Ian Sommerville. *Software Engineering*. Pearson, 10th edition, April 2015.

- [41] F.A. Sousa, R.J.N. dos Reis, and J.C.F. Pereira. Simulation of surface fire fronts using firelib and GPUs. *Environmental Modelling & Software*, 38:167–177, 2012. ISSN: 1364-8152. DOI: <http://dx.doi.org/10.1016/j.envsoft.2012.06.006>. URL: <http://www.sciencedirect.com/science/article/pii/S1364815212001867>.
- [42] United States Geological Survey. Landfire. 2018. URL: <https://www.landfire.gov/> (visited on 07/22/2018).
- [43] Ker Than. New firefighting technologies: drones, super shelters. 2013. URL: <https://news.nationalgeographic.com/news/2013/07/130702-yarnell-hill-wildfire-firefighting-technology-science/> (visited on 07/02/2013).
- [44] Unity Development Team. Unity - unity3d game engine, version 5.5, 2016. URL: <https://unity3d.com/> (visited on 02/13/2017).
- [45] CE Van Wagner. Conditions for the start and spread of crown fire. *Canadian Journal of Forest Research*, 7:23–24, 1, 1977. DOI: <https://doi.org/10.1139/x77-004>. URL: <http://www.nrcresearchpress.com/doi/abs/10.1139/x77-004?src=reccsys#.W4yLk0hKiCo> (visited on 09/02/2018).
- [46] CE Van Wagner. Prediction of crown fire behavior in two stands of jack pine. *Canadian Journal of Forest Research*, 23:442–449, 3, 1993. DOI: <https://doi.org/10.1139/x93-062>. URL: <http://www.nrcresearchpress.com/doi/10.1139/x93-062#.W4yL5-hKiCo> (visited on 09/02/2018).
- [47] WindPower. Wind energy database. URL: <http://www.thewindpower.net/index.php> (visited on 02/13/2017).
- [48] Rui Wu, Chao Chen, Sajjad Ahmad, John M. Volk, Cristina Luca, Jr. Frederick C Harris, and Sergiu M. Dascalu. A real-time web-based wildfire simulation system. In *Proceedings of the 2016 IEEE Industrial Electronics Conference (IECON 2016)*. IEEE, October 2016.