Computers & Graphics 34 (2010) 655–664

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Graphics for Serious Games

VFire: Immersive wildfire simulation and visualization

Roger V. Hoang [a,b], Matthew R. Sgambati [a,b], Timothy J. Brown [b], Daniel S. Coming [b], Frederick C. Harris Jr.[a,b,*]

[a] Department of Computer Science and Engineering, University of Nevada, Reno, Reno, NV 89557, United States
[b] Center for Advanced Visualization, Computation and Modeling, Desert Research Institute, 2215 Raggio Parkway, Reno, NV 89512, United States

ARTICLE INFO

Keywords:
Virtual reality
Computer-based training
Collaborative virtual environment
Earth and atmospheric sciences

ABSTRACT

The destruction caused by wildfires has led to the development of various models that try to predict the effects of this phenomenon. However, as the computational complexity of these models increases, their utility for real-time applications diminishes. Fortunately, the burgeoning processing power of the graphics processing unit can not only mitigate these concerns but also allow for high-fidelity visualization. We present VFire, an immersive wildfire simulation and visualization system. Users are placed in a virtual environment generated from real-world data regarding topology and vegetation. There they can simulate wildfires in real-time under various conditions. They can then experiment with various suppression techniques, such as fire breaks and water drops. The simulation is performed on the graphics card, which then provides visualization of the results. The system is intended to train fire chiefs in planning containment efforts and to educate firefighters, policymakers, and the general public about wildfire behavior and the effects of preventative measures.

1. Introduction

Every year, wildfires destroy millions of acres of land and cost millions if not billions of dollars to control. From 2000 to 2002, over 18 million acres were burned, over 2000 structures were destroyed, and over 3.4 billion dollars were expended just for suppression efforts. Beyond the immediate damage caused by wildfires, there are also lingering effects that are not only environmental, but social and economical as well [1].

To better understand wildfires, scientists have developed methods for modeling wildfire behavior. These models vary in terms of the types of spreading behaviors simulated and the simulation domain used [2,3]. Some models simulate the spread of fire across surface fuels while others simulate fire propagating into the canopies and possibly moving from tree top to tree top without interacting with the surface fuels. Still others simulate embers being lofted through the air and igniting spot fires potentially vast distances away from the source. These models take into account a variety of factors including wind, weather conditions, fuel types, and slope.

Visualization of wildfire behavior can provide a number of benefits. First of all this allows scientists to verify the accuracy of these models by comparing the results of an actual fire with the output of a simulated version. Once the model is validated, it can then be used to predict not only the behavior of an existing fire, but also the consequences of preventative measures, such as vegetation thinning and prescribed burns.

Displaying these predictions in a visually informative manner allows for community planners and/or city officials to better educate the public on existing fire hazards. Furthermore, enabling interactive manipulation of the simulation along with the visualization allows for training of fire bosses with respect to resource allocation and fire behavior. While experimentation would be dangerous and costly to perform in a real-life situation, these risks can be mitigated by simulating untested approaches first.

Virtual reality (VR) technology allows users to immerse themselves in their data. Stereoscopy provides depth information that is usually lost with standard desktop displays. This depth information enables the creation of 3D user interfaces, which can provide a more intuitive form of interaction. Additionally, combining depth information with high-fidelity graphics can allow an observer to better compare a simulated fire to a historic fire.

VFire is a wildfire simulation and visualization tool built for an immersive virtual environment. Users are able to load data about a geographical region and then experiment on this region by starting fires, manipulating fuels, and altering weather conditions. Fire simulation is performed on the graphics processing unit (GPU) to enable real-time simulation and visualization.

The remainder of this paper is structured as follows. Section 2 provides background information, while Section 3 gives an overview of the system. Section 4 outlines how the fire simulation is performed

* Corresponding author at: Department of Computer Science and Engineering, University of Nevada, Reno, Reno, NV 89557, United States.
E-mail address: Fred.Harris@cse.unr.edu (F.C. Harris Jr.).

0097-8493/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2010.09.014

on the GPU and Section 5 describes the methods used to visualize the simulation. Section 6 discusses the interaction of the user with the system while Section 7 details the current state of the system. Finally, Section 8 offers closing thoughts and avenues for future research.

2. Background and related work

This section details work related to and referenced by VFire. Particularly, a brief history of fire models is presented followed by a review of software projects that apply these models. Virtual reality and its previous applications to visualization are then discussed.

2.1. Fire models

Research into parameterizing various aspects of wildfires has been ongoing for decades. This section highlights components of this research that are relevant to this work. For a more complete treatment of fire models, the reader is referred to Pastor et al. [2].

Both Pastor et al. [2] and Perry [3] outline three types of fire models: theoretical, empirical, and semi-empirical. Theoretical models are models that rely solely on physical principles. The advantage of such reliance is that these models are based on known properties. On the other hand, the utility of these models in practice is questionable due to the difficulty of obtaining the appropriate inputs. On the other end of the spectrum are empirical models, which are generally statistics that can be used to predict fire behavior under certain conditions. Beyond this set of conditions, purely empirical models have had little success. Between these two extremes lies semi-empirical or semi-physical models that rely on some theoretical principles which are adjusted with some experimental data. Because of their reliance on experimental data, some calibration may be needed in order to apply these models to different conditions.

Most work done in determining the shape of a fire as it spreads has concluded that under homogeneous conditions, fire will spread in roughly an elliptical shape of some sort. Green et al. [4] compared the effectiveness of a simple ellipse, a double ellipse, an ovoid, and a rectangle and found that while any of those shapes could adequately fit the contours of various observed fires, the ellipse and double ellipse were found to fit best under homogeneous conditions. As wind speed increased, the length to width ratio of the ellipse would likewise increase [5].

Spread of a fire tends to be modeled using Huygens's Principle, which assumes that every point along a fire perimeter will ignite another fire that grows elliptically according to the environmental characteristics of that point. At the end of a time step, the new fire perimeter is given by the outline of all of these new ellipses [6].

Pastor et al. [2] further classifies models by the aspect of wildfire that a model is attempting to parameterize. This classification divides fire models into surface fires, crown fires, and spotting.

Surface fires are characterized by the burning of fuels that are less than 2 m in height. While numerous models for surface fire spread have been developed, few have been applied to real-world applications. The most successful of these models is a semi-empirical model developed by Rothermel in 1972 [2]. Rothermel [7] found that the forward rate of spread for a wildfire was approximated by the equation

$$R = \frac{I_R \xi (1 + \phi_w + \phi_s)}{\rho_b \chi Q_{ig}}$$

where I_R is the reaction intensity, ξ is the propagating flux under zero wind and zero slope conditions, ϕ_w is a coefficient resulting from wind, and ϕ_s is a coefficient resulting from slope. ρ_b is the bulk fuel density, χ is the effective heating number, and Q_{ig} is the amount of energy per unit mass required to ignite the fuel.

The previous equation approximates spread in one dimension with both the direction of maximum wind and the direction of maximum slope oriented along this axis. Rothermel [8] presents a method for finding a two-dimensional spread vector if the slope and wind directions are not aligned. The first component of the maximum spread rate is found using only the slope in the direction of maximum slope and no wind; likewise, the second component of the maximum spread rate is found using only the wind speed and direction and no slope. These two vectors are then added together to yield a vector describing the maximum spread rate.

Crown fires are characterized by the spread of fire into the crowns of trees. Modeling of such a phenomenon attempts to determine how a surface fire transitions to a crown fire and how such a crown fire would modify the spread of fire [2].

Van Wagner [9] classifies crown fires into three categories and details conditions required for each type. The conditions are tied to the surface fireline intensity and the surface fire spread rate. A passive crown fire characterized by the torching of trees occurs when the fireline intensity crosses a threshold intensity required for the crowns to be ignited, but the surface fire spread rate is less than the spread rate required for an active crown fire. Should the surface rate cross this threshold value, the fire is then considered an active crown fire. Such a fire spreads through both the surface fuels and crown fuels simultaneously. The surface fire aids in igniting the crowns while the crown fire increases the heat radiated to the surface fuels in front of the fire. If the horizontal heat flux required to ignite the crowns can be supplied completely by the burning crowns, then the fire is considered to be independently crowning, spreading without being linked to the surface fire below.

While Van Wagner outlined various crown fire types and the conditions for each, the quantitative effects of these fires had to be measured and calibrated for particular situations. Rothermel [8] provides one such set of values. The average spread rate of a crown fire was found to be roughly 3.34 times the spread rate computed for fuel model 10 (timber litter and understory) with the wind reduced by a factor of 0.4. Despite its purely empirical nature and relatively high standard deviations, these values have been applied to other situations [2].

Spotting occurs when burning embers are carried into the air and land somewhere in the landscape, possibly igniting it. This presents a slew of problems ranging from limiting the efficacy of fire barriers to altering the shape, size, and progression of a fire [2]. As an example, spotting was observed as far away as 10 km from the fire front [10].

Major models in this category have been primarily theoretical in nature and have focused on determining the maximum spotting distance. Albini proposed a set of models to determine the maximum spotting distance from torching trees, fuel piles, and surface fires [2]. In the torching tree case, embers are lofted vertically to some maximum height and then fall horizontally with the wind field [11]. While Albini's models computed the movement of cylindrical particles that ignored wind parameters as they were being lofted upwards, more recent research has been focused on other shapes such as spherical particles being lofted [12] and propagated [13] and disk-shaped particles moving through a three-dimensional plume that accounts for wind during the lofting stage [14].

2.2. Fire simulators

As computer technology advanced, fire models were incorporated into various software applications. In the one-dimensional case,

several applications were developed that could compute various aspects of wildfires under some set of conditions [15,16]. With the increase in availability of spatial landscape data, several simulators have been developed that account for both spatial and temporal variations in landscape characteristics. Such simulators can be divided into two classes: vector-based and raster-based.

Vector-based approaches more strictly follow the idea of elliptical wave propagation; that is, the fire shape after some time step can be found by generating ellipses along the previous fire shape and determining the new outline. Simulation is done on a continuous space. While greater in accuracy when compared to raster-based approaches, their complexity and time requirements are also greater [2]. Compared to the number of raster-based ones, few simulators use this approach including SiroFire [17] and Prometheus [18], and of them, the most commonly known is FARSITE.

Utilized globally [2], FARSITE [11] is a vector-based wildfire simulator. It incorporates a number of fire models, including Rothermel's surface spread model and Van Wagner's crown fire model. Spotting is implemented via Albini's spotting model for torching trees with an adjustable percentage reduction in the number of brands that actually ignite new fires. Fire acceleration is also accounted for. Acceleration addresses changes such as a fire heating nearby fuels and increasing the potential for spread as well as to prevent instantaneous jumps in spread rate when the perimeter enters an area with a different fuel type. More recent work on FARSITE includes the integration of a post-frontal combustion model [19]. The simulator has been used in a number of fuel treatment assessment applications both by itself [20] and in conjunction with FlamMap [21].

Raster-based approaches, or cellular approaches, propagate fire through some set of rules across a uniform grid. While faster and simpler to implement, they lack precision when compared to vector-based approaches [2]. Depending on the number of paths that a fire can travel across, distortion is possible [11].

HFire [22] is a cellular fire spread simulator designed to compute the spread of surface fire in chaparral fuels. It allows fire to spread from cell to cell in eight directions, four orthogonal and four diagonal. A fire ellipse is computed using Rothermel's spread rate equation and then spread rates in each of these spread directions are derived from the result. The simulator assumes that fire spreads at the maximum rate (no acceleration is accounted for) and mitigates the effects of distortion due to the limited spread directions by using an adaptive time step and finite fractional distances. The adaptive time step determines the minimum time required for a fire to spread from at least one burning cell to another burning cell. Using such a mechanism allows the simulation to quickly simulate large time steps when a fire moves slowly and vice versa. In conjunction with an adaptive time step, finite fractional distances allow a fire to spread some partial distance to other cells within a time step. Should a fire spread farther than the distance separating two cells, the extra portion of that spread is contributed to the partial distance burning out of the newly burning cell.

While FARSITE computes the spread characteristics of a fire over time, FlamMap [23] assumes that the entire landscape is ignited and computes the spread characteristics for each cell of the terrain. It also provides functionality to compute the minimum time required for a fire to spread from an ignition point to any other point on the landscape given constant temporal conditions [24]. In essence, every node in the grid is connected to each other. The cost in time of traversing any connection is computed accounting for changes in fuels and the length of the connection across each fuel type. Given this information, a shortest path algorithm is executed to determine the minimum travel time to every node in the simulation space. Optimizations such as stopping the search if no travel times are updated within a certain range can be used to speed up this algorithm.

While vector-based approaches may be more precise, they can also be rather time-consuming. While Stratton [21] computes the spread of fire and a few other characteristics using FARSITE, another set of characteristics were computed with FlamMap instead since a FARSITE simulation could run for hours while FlamMap output was almost instant. On a set of trial runs, HFire was found to run roughly 92 times faster than FARSITE running with several of FARSITE's capabilities disabled to more correctly compare the outputs [22].

2.3. Virtual reality and visualization

Virtual reality is a medium through which users are immersed in artificial environments. With respect to this work, this immersion is primarily achieved through visual feedback. This is accomplished through the use of depth cues and specialized hardware. There are three main types of depth cues: monoscopic, stereoscopic, and motion. Monoscopic cues provide information from only one eye, while stereoscopic cues provide information from two eyes and motion provides information from moving objects or a moving eye.

To achieve stereoscopic vision there needs to be two images that view the same point but from different perspectives. There are two ways to achieve stereoscopic vision: passive and active stereo. Passive stereo uses two projectors, one for each image that pass the image through a unique polarized filter, and the users then wear glasses that match the polarization of the projector filters. Active stereo uses a single projector that cycles back and forth between the left and right images. The users wear shutter glasses that are in sync with the cycling of the projector.

VR has been shown to be effective for crises training by Sniezek et al. [25]. Systems have been developed to train Civil Support Teams to handle radiological disasters [26] and to educate oilrig workers to combat fires [27]. VR is also used to visualize many scientific data, such as arterial flow data [28] and LIDAR [29]. This visualization is important because it allows researchers to view and interact with their data in ways not possible through a standard desktop system.

Within the realm of visualizing fire data, much work has been done both to present data in a meaningful way and to improve the visual quality. Ahrens et al. [30] and McCormick and Ahrens [31] discuss the generation of non-interactive videos from wildfire simulation data using volume rendering for fire and smoke. Govindarajan et al. [32] explores various methods for visualizing room fire simulations, including the superposition of multiple simulation results into a single image, while Rushmeier et al. [33] present a technique for visualizing pool fires. Pegoraro and Parker [34] apply fire physics to render realistic fire. Bukowski and Séquin [35] integrate an architectural walkthrough program with a fire simulator to visualize building fires. This work on VFire we are presenting is based off of early wildfire visualization work detailed by Harris et al. [36], Sherman et al. [37], Penick et al. [38], and Hoang et al. [39].

3. System overview

In this section, we describe the system upon which VFire is built, beginning with the hardware platform, followed by the hardware abstraction library, and ending with the types of data used. The composition of the VFire application itself is then discussed in detail in the three subsequent sections.

3.1. Virtual environments

There are two environments that our system runs on. The first system is a four-sided CAVE™ with one projector per display, each having a resolution of 1048×1048. The entire system is driven by a Symmetric Multi-Processor machine running Ubuntu 7.10 with four Intel Xeon E7320 2.13 GHz processors, 48 GB DDR2 RAM, and an Nvidia QuadroPlex Model IV. The system is shown in Fig. 1.

The other system, shown in Fig. 2, is a six-sided CAVE-like display with each wall being composed of two 1920×1080 resolution projectors partially overlapped to form a 1920×1920 display. The rendering of the 12 projectors is done by 12 machine nodes each running Ubuntu 9.10. Each node contains two Intel Xeon W5590 3.33 GHz processors, 24 GB DDR3 RAM, and two Nvidia Quadro FX 5800s.

Both systems track the positions and orientations of the user's head and a 6-DOF wand input device with an InterSense IS-900 tracking system. In addition to tracking information, the wand provides the user with six input buttons and a joystick. The tracker and wand are shown in Fig. 3.

Fig. 1. A 4-sided CAVE™.

Fig. 2. VFire running in a 6-sided CAVE-like environment.

Fig. 3. (Left) A 6-DOF wand device. (Right) A 6-DOF tracking device mounted on stereoscopic glasses.

3.2. Hydra

VFire was implemented using Hydra [40], a library for developing applications in a virtual reality environment. It abstracts away details about devices and instead provides the application programmer with a set of generic inputs. Along with input handling it also automatically computes off-axis view frustums for all the screens in the environment. For clustered systems, co-simulation is performed across all nodes in lock-step, forcing synchronization between all nodes. A head node replicates all inputs as well as time deltas to all nodes. The time delta synchronization is necessary to ensure that non-linear systems do not experience drift over time.

3.3. Data

There are three main types of data in VFire: topological, fuel, and tree placement. The topological data provides VFire with the elevations of the geographical area of interest. This data was acquired using a variety of techniques, including LIDAR, and is stored as a digital elevation model (DEM). The heightfield terrain and texture applied to it are generated from this data. The fuel data provides VFire with the necessary vegetation characteristics, such as densities, types, BTUs/ton, etc. and can be obtained through LANDFIRE [41]. Scientists gathered this data on foot using several instruments and stored it as raster data, which is later converted to texture data and used by the GPU simulation. The tree placement provides VFire with more accurate positions for the vegetation and was determined by using the method presented by Brown [42], which takes the satellite image provided by the topological data and applies some image processing to generate the tree positions, as can be seen in Figs. 4 and 5. The system then uses these positions to place the trees and buildings in the visualization.

4. GPU fire simulation

The core of VFire is its wildfire simulator. To achieve interactive feedback, the simulator was implemented on the GPU using OpenGL and GLSL. The result of this decision is that the simulation output resides within graphics memory, ready to be visualized. At the time of its development, more direct GPU programming mechanisms such as CUDA were unavailable. Future efforts will explore their use to improve efficiency. This section details the various components of the simulator and how they were mapped onto the GPU. More details are given by Hoang [43].

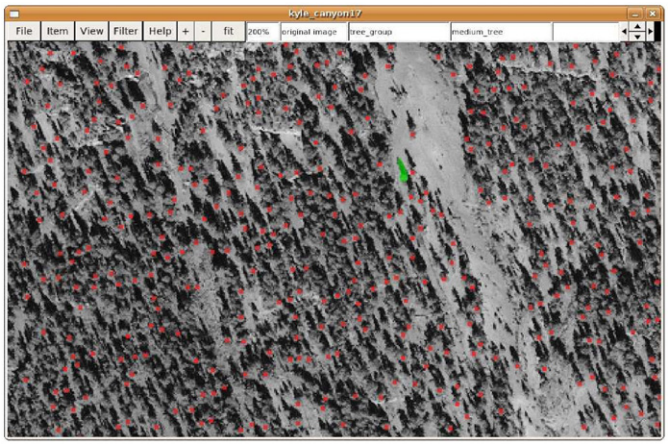

Fig. 4. Possible trees identified using [42].

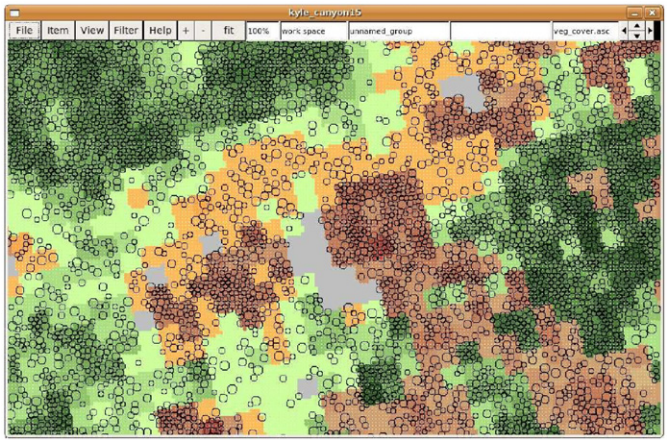

Fig. 5. Random tree placements that were made according to the vegetation data.

4.1. Fire spread

For basic fire spread, the simulator is based on Rothermel's fire spread equations, which are based on Huygens's principle. Thus, fire spread is modeled as an ellipse expanding at some maximum rate R_{max} with some orientation ϕ and eccentricity ε based on wind and slope. Given this information, the spread rate in any arbitrary direction Θ can be computed using

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

as described by Morais [22]. Given this spread rate r, the time required for a fire to travel from one point to another point separated by a distance d is given by

$$t = d/r$$

Because the desired output is a set of rasters that can be visualized and analyzed, the simulation is modeled as a two-dimensional regular rectangular lattice in a fashion similar to that of HFire [22]. Essentially, the center of each cell is connected to the centers of each of its eight neighboring cells by a straight line of fuel. Fire can spread from a burning cell to any unburnt cell by burning the entire distance separating the cell centers. In the case of multiple lines burning towards the same center, the first line to completely burn is used to determine the time of arrival and other fire characteristics. The time of arrival logic can be summarized by Algorithm 1. It may be executed repeatedly until the resulting times of arrival no longer change. To account for changing spread conditions, times of arrival can only be computed up to the point when these conditions are altered; any arrivals after this point are simply discarded.

Algorithm 1. Time of arrival computation.

1: **for all** Surrounding Cells c **do**
2: t=timeOfArrival(c) + spreadTime(c, *thisCell*)
3: timeOfArrival(*thisCell*)= min(timeOfArrival(*thisCell*), t)
4: **end for**

To achieve this logic on the GPU, the times of arrival are double-buffered with a pair of 32-bit floating point textures. All times are initialized to the very end of the simulation. A fragment shader is executed over each cell of the simulation space. This shader executes a single iteration of Algorithm 1 and uses the resulting value as the fragment color as well as the fragment depth. During execution, one texture is used as the times of arrivals accessed while the other texture is used to collect the outputted values. After each iteration, the values are copied from the write texture to the read texture. Fragments that would write a time that is later than the one currently stored for that cell would be discarded. To compute times of arrival up to a certain point in time, the acceptable time range is set to the simulation's current time step window (earlier fragments need not be written again) and the fragment shader is executed repeatedly until no more fragments are written.

While some arrival times may exceed the threshold time and thus would not be recorded during that simulation step, that does not necessarily imply that nothing occurred between each pair of links. In fact, if that were the case, using a time step that is too short would result in no fire propagation at all since the arrival times would always exceed the time window. To prevent this problem, fractional burning is utilized. That is, while a fire may not be able to travel the entire distance between two links, it has at least burned through some of that distance; as a result, that distance is reduced by the product of the spread rate and the elapsed time.

$$d = d - r\Delta t$$

The result of this effect is that the distance will eventually be reduced to the point where the fire will spread across the link within the simulated time window. It also necessitates the modification of Line 2 of Algorithm 1 to

$$t = \max(\text{timeOfArrival}(c), \textit{windowStartTime}) + \text{spreadTime}(c, \textit{thisCell})$$

to account for the time spent partially burning the distance.

On the GPU, implementation of partial burning is straightforward. The remaining distance from the center of each cell to each of its neighboring cells is stored in two textures, with the orthogonal distances in one four-component floating point texture and the diagonal distances in another. At the end of a simulation step, a fragment shader is executed with the write targets set to these distance textures. The fragment shader fetches from a set of textures the spread rates in each direction, multiplies them by the elapsed time, and outputs the result as the fragment colors. Subtractive blending results in the distances already stored in the write targets being reduced by the output fragment values.

4.2. Surface fire

The basic spread characteristics of a wildfire are computed using Rothermel's model. Spatial information such as fuel model

and terrain characteristics are stored in two-dimensional textures. Various fuel model properties such as packing ratio and fuel bed density are stored in one-dimensional textures that are accessed using the integer fuel model used for each cell. Computation of the spread characteristics is performed by executing a fragment shader over the simulation space with the write target set to a four-component floating point texture. The outputted fragments contain the maximum spread rate of fire including the contributions of slope and wind, the eccentricity of the ellipse, the direction of maximum spread, and an intensity modifier value I_m

$$I_m = \frac{12.6 I_R}{60.0\sigma}$$

where I_R is the reaction intensity and σ is the ratio of the fuel bed's surface area to volume. The modifier, I_m, is used later in computing crowning.

4.3. Fire acceleration

A newly ignited fire does not immediately begin to spread at its maximum rate. Instead, it accelerates towards its maximum rate over time. Fire acceleration in the GPU simulator uses a modified model of that used by FARSITE [11]. Given the maximum rate R_{max}, the spread rate at time t is given by

$$R(t) \approx R_{max}(1.0 - e^{-a_a t})$$

where a_a is an acceleration constant. From this equation, the time Δt_{max} required to achieve the maximum spread rate given the current spread rate R is given by

$$\Delta t_{max} = \frac{1.0 - \frac{R}{R_{max}}}{a_a}$$

During the fire acceleration phase, the spread rate for every burning cell is increased by

$$dR = \frac{dt}{\Delta t_{max}}(R_{max} - R_{current})$$

A fire started from an ignition will have an initial spread rate of zero and will steadily increase to its maximum rate. As the fire spreads from cell to cell, the cell inherits the current spread characteristics of the cell that ignited it. If the spread rates exceed the maximum spread rates of the newly burning cell, they are clamped instantaneously to their maximums. If, however, they are slower than the cell's maximum rates, they are accelerated by the difference dt given by

$$dt = \text{timeOfArrival}(thisCell) - \max(baseTime, \text{timeOfArrival}(ignitingCell))$$

where *baseTime* is the last time that the burn distances were updated.

To implement fire acceleration, the maximum spread rate in each of the eight spread directions is computed and stored in a set of two textures, one for the orthogonal directions and another for the diagonal directions. A separate pair of textures is used to maintain the current spread rates for each cell and is initialized at the beginning of the simulation to be all zeroes. At the end of every simulation step, all burning cells have their spread rates accelerated as previously discussed.

Propagation based on burn times and times of arrival is still used, although it was modified to account for changing spread rates as a result of acceleration and rate inheritance. Spread rates are double-buffered like the time of arrival textures, and a copy from write texture to read texture for each is done at the end of the iteration. The propagation shader was modified to output the inherited and accelerated rates from the igniting cell.

Three additional textures were added. The first two store integer time stamps that denote the last time a cell was updated. If any cell surrounding the current cell were updated in the last round, the current cell is checked again to see if a lower time of arrival is possible. These textures are swapped in terms of reading and writing at the end of each iteration; no compositing is necessary since only cells that changed in the previous iteration are of concern. At the beginning of the propagation phase, all stamps are cleared to zero to force all cells to at least check once with new spread times. Cells that are updated write the next time stamp out to the write buffer. The third texture contains the texture coordinate of the cell that ignited the current cell. If the data in that cell becomes invalid, that is, if the spread properties of that cell have changed and the time stamp is equal to the previous iteration's time, the current cell's time of arrival is invalidated and the original time of arrival used at the beginning of the simulation step is used to proceed.

4.4. Crown fire

Whether a fire will spread into and through the crowns of trees depends on a number of factors. As with FARSITE's implementation [11], active crown fires have a different maximum spread rate than surface and passive crown fires approximated by

$$R_{max_{crown}} = 3.34 R_{10}$$

where R_{10} is the surface spread rate for fuel model 10 with a wind reduction factor of 0.4. The actual maximum spread rate of a crown fire depends on the crown fraction burned, given by the equation

$$CFB = 1 - e^{-a_c(R - R_0)}$$

where

$$a_c = \frac{-\ln(0.1)}{0.9(RAC - R_0)}$$

and

$$R_0 = I_0 \frac{R}{I_m}$$

I_0 is the threshold intensity for a crown fire to occur and is given by

$$I_0 = (0.010 CBH(460 + 25.9M))^{3/2}$$

where *CBH* is the crown base height and *M* is the foliar moisture content. *RAC* is the threshold spread rate at which a crown fire is promoted from passive to active. Given the crown bulk density *CBD*, this threshold is given by

$$RAC = 3.0/CBD$$

Crowning is modeled in the simulation as an increase in maximum spread rate in the case of an active crown fire. Canopy height is stored in a texture, and crowning is only considered if the canopy height is greater than zero. *RAC* is computed with a shader and stored in a texture. Modification of the *CBD* texture results in the recomputation of the *RAC* texture. I_0 is also computed with a shader and is only updated if the *CBH* texture is modified. Before accelerating spread rates, the current spread rates are used to test whether the fire is now actively crowning, and if so, the maximum spread rate is adjusted accordingly. This computation is done on a per direction basis.

4.5. Simulation progression

At the beginning of the simulation, all directional spread rates are initialized to zero. Times of arrival are cleared to a user-defined end time. Burn distance textures are set to the complete distance to each neighbor.

Algorithm 2 outlines the flow of the simulation. Because spread properties can change due to alterations caused by events, simulation progresses in substeps between events.

Algorithm 2. Simulation flow.

```
1:  endTime = currentTime
2:  startTime = lastUpdateTime
3:  while startTime != endTime do
4:    stepTime = min(endTime, nextEvent.time)
5:    Update corrupted data()
6:    Propagate fire(startTime, stepTime)
7:    Burn distances(startTime, stepTime)
8:    Accelerate(startTime, stepTime)
9:    Trigger next event()
10:   startTime = stepTime
11: end while
12: lastUpdateTime = endTime
```

5. Visualization

Simulation programs generally concentrate on the simulation and less on the presentation of the simulation. Our system concentrates both on the simulation and the presentation. Providing enhanced visualizations can possibly help users to better interpret and understand what is happening at any point in time during the simulation. We concluded that a good visualization of our simulation would be one that has detailed terrain, vegetation, buildings, fire, and smoke.

5.1. Terrain

Slope is an important factor of fire spread. Visualizing topology is therefore necessary to provide better comprehension of how the fire spreads in the environment. To achieve this visualization the terrain was implemented as a 2.5D terrain, which uses the height field data discussed in Section 3.3. The satellite image is overlaid on the terrain; however, the resolution of this image is usually inadequate for the size of the terrain. When rendered to scale, a single pixel of the image correlates to roughly a 1×1 m patch of terrain. Thus, the terrain near the user tends to appear blurry. To ameliorate this, we modulate a repeating texture onto the satellite image to add more detail. In addition to the satellite image, we also overlay several data layers from the simulation, the opacities of which can be modified as discussed in Section 6.3.

5.2. Objects

One of the key components to increase the visual fidelity of the visualization is the vegetation, which was handled by SpeedTree [44], a commercial tree rendering package. SpeedTree has an adjustable threshold distance that is set by the user. Based on this threshold, SpeedTree renders trees within the threshold with full geometry models, while trees outside the threshold are rendered as billboards. Another threshold value is used to transition from the full geometry models to the billboards, which eliminates popping. Since SpeedTree was developed for video games, which are typically single screened environments, modifications were necessary to allow it to be used in a virtual environment, which can contain multiple rendering contexts and displays. The main modification was creating context buffers, which contained graphic assets for each rendering context.

Another key component for having increased visual fidelity when simulating a geographical area is rendering the buildings of that area. The locations used to place the buildings is gathered using the technique described by Brown [42]. The buildings are composed of a cube with a gable roof, which are drawn from a single point passed to a geometry shader, which then creates the remainder of the structure. The geometry shader accounts for the length and width of the building; the current version does not, however, account for its orientation, though this is scheduled for the next release.

5.3. Fire effects

Visualization of fire behavior is achieved through the use of a particle system on the GPU. The particle system is composed of two types of objects: emitters and particles. Once the simulation has been updated to the current simulation time, an emitter is created at every newly ignited cell. To accomplish this, a vertex is sent to the geometry shader for each cell in the simulation space. If the cell is determined to have ignited within the previous update window, an emitter is streamed out to a vertex buffer. Immediately following this, all previously existing emitters are streamed through another geometry shader, updating their state. If an emitter survives the update step, it is streamed into the same vertex buffer as the newly generated emitters, resulting in a final list of all active emitters.

Particles are generated and updated in a similar fashion. Particles are generated by streaming emitters through a geometry shader that emits particles based on the state of the emitters. Existing particles are then streamed through an update shader, and surviving particles are appended to the same buffer as new particles. A part of this updating process is to apply environmental effects to each particle, such as wind. Particles are rendered as billboards spherically aligned to the user's head position. While the particle images are simply artistically generated currently, future work will focus on altering flame and smoke color based on the burning fuel generating the particles. Rendering of these particles presents difficulties due to different blending behaviors exhibited by the two types of particles: fire and smoke. While fire particles use additive blending, smoke particles use back-to-front blending, where the resulting pixel color is given by $color_{src}\alpha_{src} + color_{dest}(1-\alpha_{src})$. This back-to-front blending of smoke requires that fire particles be rendered simultaneously from back-to-front in order for the smoke to correctly obscure the fire. Due to the large number of interspersed particles being rendered, it would be impractical to switch blending types each time the particle type is changed; instead, blending is controlled within the fragment shader. To accomplish this, we set the blending function to add the new fragment color to the old fragment color multiplied by the new fragment alpha ($color_{dest} = color_{src} + color_{dest} * \alpha_{src}$). All particles are then sorted and rendered from back to front. When a fire particle is rendered, the particle texture color is outputted as the output color while the output alpha is set to 1.0, resulting in additive blending. However, when a smoke particle is rendered, the output fragment color is premultiplied by the α while the outputted alpha value is set to $1-\alpha$, yielding correct obscuring behavior for smoke particles.

In addition to particles, fire spread is also visualized by illuminating the terrain and disintegrating trees and buildings as the fire spreads through an area. To illuminate the terrain, the

time-of-arrival texture computed during simulation is used to determine which areas are on fire based on the current simulation time. Those cells that fall within a time window around the simulation time render a bright fragment into a light texture while those cells that do not simply render a black fragment. The light texture is then blurred and additively blended onto the terrain when it is rendered. To destroy objects such as vegetation, a burn progression value is computed per vertex based on how long the cell beneath the vertex has been on fire. In the fragment shader, the interpolated progression value is used to transition the texture color to an orange glow before darkening. It is also used as a threshold value for discarding fragments. The alpha channel for each object's textures is modulated by low-frequency noise. If the modulated alpha is lower than the threshold value, the fragment is discarded; otherwise, the fragment is rendered at full opacity. Using these techniques, objects appear to char and disintegrate over time.

To better control the brightness of the fire and lighting effects, high dynamic range lighting (HDR) techniques are employed. Regardless of all other lighting in the scene, fire and any illumination effects caused by it are rendered at a constant intensity. The color intensity of the screen is then tone mapped based on the average luminance of the entire image as described by Reinhard et al. [45]. Since the average luminance for each screen in a virtual environment can be different based on what is actually rendered onto the screen, the average of all the averages in the virtual environment is used for tone mapping. As a result of using a constant fire brightness with HDR, the visual effect of the fire is more subdued during daytime scenes with greater environmental illumination while more pronounced during nighttime scenes.

6. Interaction

Upon starting the application, the user is presented with the virtual world rendered to physical scale. The user is able to navigate by simply pointing the wand in the direction he wishes to go and pushing up on the *y*-axis of the joystick. The speed at which the user travels is dependent upon the distance between the user and the terrain directly below him. At greater distances the user travels more quickly, while at lower distances, movements are much slower, allowing for fine navigation around objects such as trees and structures. Based on the virtual environment, rotation may be mapped to the *x*-axis of the joystick. For example, in a four-sided CAVE^{TM} environment without a back wall, rotation is necessary for the user to see objects behind him. On the other hand, rotation is not necessary in a fully enclosed environment such as a six-sided CAVE-like display as the user can physically turn to see objects out of view.

6.1. Construct

In order to interact with the world, the user must first select a tool to use. To do so, a button can be pressed to enter an alternate world called the "construct." By entering the construct, the current simulation is paused and a number of tools appear around the user. To differentiate those objects within the construct and those within the simulation world, the former is rendered over the latter, which is muted by converting the colors to grayscale and blending it with white, as seen in Fig. 6. Navigation is disabled, and the construct objects can simply be selected by pointing at them with the wand, which highlights the object with the user's color. Pressing the trigger on the wand results in the highlighted tool being selected, returning the user from the construct to the simulation world.

Fig. 6. VFire's "construct" allows users to select tools to manipulate the simulation.

6.2. Tools

There are four tools currently available in VFire: an ignition tool, a fire break tool, a moisture modification tool, and a wind modification tool. The *ignition tool*, represented by a matchstick, allows the user to start fires. After selecting the tool, the user can point the wand to an area he wishes to ignite. To aid the user in selecting the point of ignition, a line is drawn from the wand out in the direction it is pointing. Additionally, a matchstick is rendered at the ignition point. By pressing the trigger, a fire is started at the ignition point. If the trigger is held, the user can outline a path of ignition points. For visibility purposes, the matchstick, along with all tools that require the user to point at a location on the terrain, is scaled based on the distance between the user and the terrain point.

The *fire break tool*, represented by a bulldozer, allows the user to render parts of the terrain unburnable. Interaction with the tool is identical to the interaction with the ignition tool. To render the selected area unburnable, the fuel model of the selected area is replaced with the fuel model for an unburnable type. To prevent fire from moving diagonally through cells due to aliased rasterization of break lines, the thickness of each line was set to twice the size of a single cell.

The *moisture modification tool*, represented by a water drop, allows the user to increase the amount of moisture in a cell. Again, the user manipulates the tool as he would the ignition tool. Moisture is added to the current moisture content in the simulation layers by rasterizing the added moisture onto the original moisture data using additive blending.

The *wind modification tool*, represented by a tornado, allows the user to alter the direction and magnitude of the wind. To do so, the user presses the trigger to create an initial anchor point. By holding the trigger down and moving the wand around, the user creates a vector from the anchor point to the current position of the wand. The direction of this vector determines the wind azimuth while the length determines the wind speed. While performing this action, a line is drawn from the anchor point to the current wand position; the azimuth and windspeed are also rendered along this line.

6.3. Data visualization

To allow the user to see different aspects of the current simulation state, a layer visualization interface is provided. This interface displays a 3×3 grid composed of eight data layers surrounding a composite image of all the layers overlapped on top

Fig. 7. The opacities of various data layers can be modified by the user.

of each other, as shown in Fig. 7. The opacity of each layer can be adjusted by pointing to the desired layer and pressing left or right on the joystick. The composite image is used as the texture rendered onto the terrain.

7. Current status

The current version of VFire runs in the six-sided CAVE-like display described in Section 3.1 at full resolution. As a test dataset, a 10 km by 10 km area around Kyle Canyon, NV, was selected. The simulation runs at 10 m resolution initially at 200 times real-time, although this parameter is adjustable at run-time. At higher simulation speeds, performance drops as fire is able to propagate across more cells in a single update step. With the initial settings, the system runs at approximately 30 frames per second in stereo and can drop to about 15 FPS depending on what is visible to the user. For example, if the user flies directly into smoke, the system becomes pixel-bound as every individual particle consumes more screen space.

The system has been demonstrated to both the general public and fire experts with positive responses. We plan to work with fire officials to generate scenarios that can be used to train fire managers as well as investigate the effectiveness of the system. To this end, incorporation of a scoring metric is being researched, and current work is focused on recording the simulation and user decisions in order to evaluate performance afterwards.

8. Conclusions and future work

We have presented VFire, an immersive wildfire simulation and visualization system. By harnessing the processing capabilities of the GPU, we created an interactive wildfire simulator, the output of which is immediately visualized for the user. We developed various graphical techniques to create a visualization that leveraged both image quality and utility. Using virtual reality, we provided users with unique interaction methods while also immersing them within the data. This process has been tied in with cyclic reviews and demonstrations to various fire officials in preparation the user studies.

There are a number of ways we intend to extend this work. First, we plan to improve the fire model to incorporate other wildfire phenomena such as spotting. We also plan to introduce an atmospheric model that is affected by and affects the fire simulation. To facilitate more realistic training for fire chiefs, associating and limiting costs in terms of human resources, time, and money to various suppression tactics could be employed. Another area for improvement is the visualization, where methods will be explored to intuitively display more information regarding the situation and to improve the visual fidelity. Likewise, given the unique input capabilities of virtual reality systems, alternative interaction methods will be examined. The efficacy of VFire as a training tool must also be validated through a user study.

Acknowledgements

This work is funded by the U.S. Army's RDECOM-STTC under Contract no. N61339-04-C-0072 at the Desert Research Institute.

References

[1] Morton DC, Roessing ME, Camp AE, Tyrrell ML. Assessing the environmental, social, and economic impact of wildfire. Technical Report, The Global Institute of Sustainable Forestry; 2003.
[2] Pastor E, Zrate L, Planas E, Arnaldos J. Mathematical models and calculation systems for the study of wildland fire behaviour. Progress in Energy and Combustion Science 2003;29(2):139–53, doi:10.1016/S0360-1285(03)00017-0. URL 〈http://www.sciencedirect.com/science/article/B6V3W-4899V97-1/2/79f2053cdee67b01a3f5f22ca7e677ae〉.
[3] Perry G. Current approaches to modelling the spread of wildland fire: a review. Progress in Physical Geography 1998;22(2):222–45. URL 〈http://0-search.ebscohost.com.innopac.library.unr.edu/login.aspx?direct=true&db=aph&AN=4089240&site=ehost-live〉.
[4] Green DG, Gill AM, Noble IR. Fire shapes and the adequacy of fire-spread models. Ecological Modelling 1983;20(1):33–45, doi:10.1016/0304-3800(83)90030-3. URL 〈http://www.sciencedirect.com/science/article/B6VBS-48YNTMB-FG/2/9a7171795b8e9f1a0a124a11d112682e〉.
[5] Anderson HE. Predicting wind-driven wild land fire size and shape. Technical Report, US Forest Service; 1983.
[6] Richards G. An elliptical growth model of forest fire fronts and its numerical solution. International Journal for Numerical Methods in Engineering 1990;30(6):1163–79.
[7] Rothermel RC. A mathematical model for predicting fire spread in wildland fuels. Technical Report, U.S. Forest Service; 1972.
[8] Rothermel RC. Predicting behavior and size of crown fires in the northern rocky mountains. Technical Report, U.S. Forest Service; 1991.
[9] Wagner CV. Conditions for the start and spread of crown fire. Canadian Journal of Forestry Research 1977;7(1):23–40.
[10] Billing P. Otways fire no 22 – 1982/83. Technical Report, Victoria Department of Sustainability and Environment; 1983.
[11] Finney M. FARSITE: fire area simulator-model. Development and evaluation. Technical Report, USDA Forest Service; 1998.
[12] Woycheese JP, Pagni PJ. Brand lofting above large-scale fires. In: International conference on fire research and engineering proceedings, 1998. p. 137–50.
[13] Woycheese JP, Pagni PJ, Liepmann D. Brand propagation from large-scale fires. Journal of Fire Protection Engineering 1999;10(2):32–44. arXiv:http://jfe.sagepub.com/cgi/reprint/10/2/32.pdf, doi:10.1177/104239159901000203. URL 〈http://jfe.sagepub.com/cgi/content/abstract/10/2/32〉.
[14] Sardoy N, Consalvi J-L, Porterie B, Fernandez-Pello AC. Modeling transport and combustion of firebrands from burning trees. Combustion and Flame 2007;150(3):151–69, doi:10.1016/j.combustflame.2007.04.008. URL 〈http://www.sciencedirect.com/science/article/B6V2B-4NWN3BN-2/2/125b9b8e2ba99cf7036f8c5d7e9df667〉.
[15] Andrews PL. BEHAVE: fire behavior prediction and fuel modeling system—burn subsystem, part 1. Technical Report, U.S. Forest Service; 1986.
[16] Andrews P. BehavePlus fire modeling system: past, present, and future. In: Proceedings of 7th symposium on fire and forest meteorology, 2007. p. 22–6.
[17] Coleman J, Sullivan A. A real-time computer application for the prediction of fire spread across the Australian landscape. Simulation 1996;67(4):230.
[18] Canadian Interagency Forest Fire Centre, PROMETHEUS, 〈http://www.firegrowthmodel.com/index.cfm〉, accessed August 6, 2010 (May 2008).
[19] Finney M. Spatial modeling of post-frontal fire behavior. Technical Report, Rocky Mountain Research Station; 1999.
[20] Van Wagtendonk J. Use of a deterministic fire growth model to test fuel treatments. In: Sierra Nevada ecosystem project, final report to congress, vol. 2, Citeseer, 1996. p. 1155–66.
[21] Stratton RD. Assessing the effectiveness of landscape fuel treatments on fire growth and behavior. Journal of Forestry 2004;102(7):32–40.
[22] Morais ME. Comparing spatially explicit models of fire spread through chaparral fuels: a new algorithm based upon the Rothermel fire spread equation. Master's thesis, University of California, Santa Barbara; 2001.
[23] Finney MA. An overview of FlamMap fire modeling capabilities. In: Fuels management—how to measure success: conference proceedings, 2006. p. 213–20.

[24] Finney M. Fire growth using minimum travel time methods. Canadian Journal of Forest Research 2002;32:1420–4. (5).
[25] Sniezek J, Wilkins D, Wadlington P. Advanced training for crisis decision making: simulation, critiquing, and immersive interfaces. In: Hawaii international conference on system sciences, vol. 3, 2001. p. 3042.
[26] Koepnick S, Norpchen D, Sherman WR, Coming DS. Immersive training for two-person radiological surveys. In: Proceedings of IEEE virtual reality, 2009, p. 171–4.
[27] Tate DL, Sibert L, King T. Using virtual environments to train firefighters. IEEE Computer Graphics and Applications 1997;17(6):23–9.
[28] Forsberg A, Laidlaw D, Van Dam A, Kirby R, Karniadakis G, Elion J. Immersive virtual reality for visualizing flow through an artery. In: Proceedings of the conference on visualization'00. IEEE Computer Society Press; 2000. p. 457–60.
[29] Kreylos O, Bawden G, Kellogg L. Immersive visualization and analysis of LiDAR data. In: Advances in visual computing. Lecture notes in computer science, vol. 5358. Springer; 2008. p. 846–55.
[30] Ahrens J, McCormick P, Bossert J, Reisner J, Winterkamp J. Case study: wildfire visualization. In: Visualization '97. Proceedings, 1997, p. 451–4. doi:10.1109/VISUAL.1997.663919.
[31] McCormick P, Ahrens J. Visualization of wildfire simulations. IEEE Computer Graphics and Applications 1998:17–9.
[32] Govindarajan J, Ward M, Barnett J. Visualizing simulated room fires. In: IEEE visualization 1999, Citeseer, 1999, p. 475–8.
[33] Rushmeier H, Hamins A, Choi M. Volume rendering of pool fire data. IEEE Computer Graphics and Applications 1995;15(4):62–7.
[34] Pegoraro V, Parker S. Physically-based realistic fire rendering. In: Proceedings of the Eurographics workshop on natural phenomena, 2006. p. 51–9.
[35] Bukowski R, Séquin C. Interactive simulation of fire in virtual building environments. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques. ACM Press, Addison-Wesley Publishing Co.; 1997. p. 35–44.
[36] Harris Jr F, Penick M, Kelly G, Quiroz J, Dascalu S, Westphal B. V-FIRE: virtual fire in realistic environments. In: The 4th international workshop on system/software architectures in proceedings of the 2005 international conference on software engineering research and practice, Citeseer, 2005. p. 73–9.
[37] Sherman W, Penick M, Su S, Brown T, Harris F. Vrfire: an immersive visualization experience for wildfire spread analysis. In: IEEE virtual reality conference, 2007. VR'07, 2007, p. 243–6.
[38] Penick M, Hoang R, Harris Jr F, Dascalu S, Brown T, Sherman W, et al. Managing data and computational complexity for immersive wildfire visualization. In: Proceedings of high performance computing systems (HPCS'07), Prague, Czech.
[39] Hoang RV, Mahsman JD, Brown DT, Penick M, Harris Jr FC, Brown TJ. VFire: virtual fire in realistic environments. In: Proceedings of IEEE virtual reality conference 2008 (VR 2008), Reno, Nevada.
[40] Hoang R, Hydra, 〈http://www.cse.unr.edu/hpcvis/hydra/〉, accessed August 6, 2010.
[41] LANDFIRE, Landfire, 〈http://www.landfire.gov〉, accessed May 31, 2010.
[42] Brown DT, Hoang RV, Sgambati MR, Harris Jr FC. An application for tree detection using satellite imagery and vegetation data. In: Proceedings of the ISCA 18th international conference on software engineering and data engineering (SEDE'09), 2009.
[43] Hoang R. Wildfire simulation on the GPU. Master's thesis, University of Nevada, Reno; 2008.
[44] Interactive Data Visualization, Inc., Speedtree, 〈http://www.speedtree.com〉, accessed May 30, 2010.
[45] Reinhard E, Stark M, Shirley P, Ferwerda J. Photographic tone reproduction for digital images. ACM Transactions on Graphics 2002;21(3):267–76.