

University of Nevada  
Reno

**VFIRE: Virtual Fire in Realistic Environments**  
**A Framework for Wildfire Visualization**  
**in Immersive Environments**

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

by

Michael A. Penick

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

May, 2007



University of Nevada, Reno  
Statewide • Worldwide

THE GRADUATE SCHOOL

We recommend that the thesis  
prepared under our supervision by

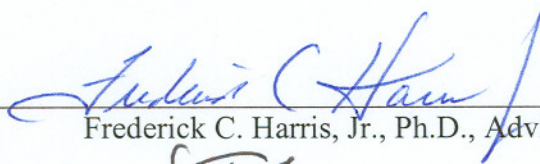
MICHAEL ALAN PENICK

Entitled

VFIRE: Virtual Fire In Realistic Environments  
A Framework For Wildfire Visualization in Immersive Environments

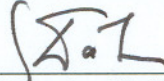
be accepted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE



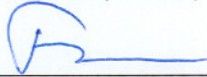
---

Frederick C. Harris, Jr., Ph.D., Advisor



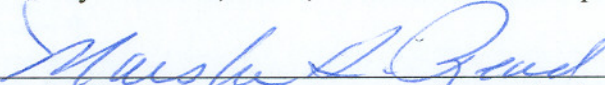
---

Sergiu Dascalu, Ph.D., Committee Member



---

Timothy J. Brown, Ph.D., Graduate School Representative



---

Marsha H. Read, Ph. D., Associate Dean, Graduate School

May, 2007

© by Michael Alan Penick 2007  
All Rights Reserved

## Dedication

To Danielle, you made this possible. Your inspiration has given me the strength to do great things.



## Abstract

Wildfire spread model output is used to make important and oftentimes expensive decisions. This thesis presents “VFire - Virtual Fire in Realistic Environment” an application and more importantly a framework for visualizing wildfire simulations in immersive environments. VFire will allow its users to visualize wildfires from perspectives and positions, which are normally too dangerous. Recent developments in graphics and virtual reality technology allow us to achieve this goal. It has recently become possible to visualize wildfire simulations in more realistic and immersive ways than has ever been achieved. To this end, VFire is an immersive visualization application that aids in wildfire training and data analysis endeavors and also a framework with which future wildfire applications can be developed.

## Acknowledgments

The work shown has been sponsored by the Department of the Army, Army Research Office; the contents of the information does not necessarily reflect the position or the policy of the federal government, and no official endorsement should be inferred. This work is funded by the CAVE Project (ARO# N61339-04-C-0072) at the Desert Research Institute.

I truly stand on the shoulders of giants. This work would not have been possible without the guidance and support of such wonderful people. My committee has been fundamental to the development of this project but also to me as a person. To Dr. Timothy Brown, who trusted me to partake in this difficult and very worthwhile endeavor and encouraged me along the way. To Dr. Sergiu Dascalu, you taught me how to build better software and your careful guidance is much appreciated. To Dr. Frederick Harris Jr., you took my freshman interest and experience and made them grow and develop into something beyond my imagination. Your direction and support as my graduate thesis advisor have been fundamental. To you, I extend my deepest and sincerest gratitude. I'd like to thank my co-researchers who have been of great help and inspiration. Roger Hoang, Joseph Mahsman, and Steve Koepnick, you have taken the project to the next level. William Sherman your vast knowledge and guidance have helped me tremendously. I am lucky to be in the company of such great individuals. Your influence has helped and formed me to do great things. Thank you.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>3</b>
2.1 FARSITE and Wildfire . . . . .	3
2.2 Remotely Sensed Data and Geospatial information . . . . .	4
2.3 Virtual Reality . . . . .	6
2.3.1 Depth Cues . . . . .	6
2.3.2 Stereoscopic Displays . . . . .	7
2.3.3 Head Tracking and Input . . . . .	10
2.3.4 VR Hardware: Render and Update Loops . . . . .	10
2.3.5 Virtual Reality Toolkits . . . . .	11
2.4 Scene graphs . . . . .	14
2.5 Fire and Smoke Visualization . . . . .	15
2.6 Terrain Visualization . . . . .	16
2.7 Vegetation Visualization . . . . .	22
2.7.1 Placement and Vegetation types . . . . .	22
2.7.2 Rendering Vegetation . . . . .	23
<b>3 Related Work</b>	<b>25</b>
<b>4 Complexities of Wildfire Visualization</b>	<b>28</b>
<b>5 VFire Software Specification and Design</b>	<b>30</b>
5.1 Framework Goals . . . . .	31
5.2 Hardware Support . . . . .	32
5.3 Requirements . . . . .	33
5.4 Use Cases . . . . .	34
5.5 Modeling and Design . . . . .	34

5.5.1	Main Class Hierarchy . . . . .	36
5.5.2	Simulation Graph . . . . .	39
5.5.3	Utility Class Functionality . . . . .	42
<b>6</b>	<b>VFire Prototype</b>	<b>44</b>
6.1	FreeVR and OpenSceneGraph Integration . . . . .	44
6.2	Terrain Implementation . . . . .	47
6.3	Vegetation Implementation . . . . .	48
6.4	Wildfire Implementation . . . . .	51
6.5	Visualization Hardware . . . . .	54
6.6	Kyle Canyon Scenario . . . . .	54
<b>7</b>	<b>Conclusions and Future Work</b>	<b>56</b>
7.1	Conclusion . . . . .	56
7.2	Future Work . . . . .	57
	<b>Bibliography</b>	<b>60</b>

# List of Figures

2.1	User moves their head to view other side of object [43]. . . . .	7
2.2	A Head Mounted Display (HMD) [43] . . . . .	8
2.3	A multi-screen projection system. . . . .	9
2.4	Intersense head tracking (a) and wand (b) input system. [21]. . . . .	10
2.5	The architecture of a shared memory visualization system. . . . .	12
2.6	The architecture of a cluster-based visualization system. . . . .	13
2.7	Particle fire using millions of particles [12]. . . . .	17
2.8	Sprite-based fire using volumetric textures [29] . . . . .	17
2.9	Terrain simplified using the ROAM algorithm [9]. . . . .	19
2.10	Terrain simplified using Geometrical Mipmapping [59]. . . . .	20
2.11	Terrain simplified using Chunked LOD [49]. . . . .	20
2.12	“Skirts” hide cracks between adjacent tiles. . . . .	21
2.13	“ribbons” hide cracks between adjacent tiles. . . . .	22
2.14	Vegetation placed according to FIA data [54]. . . . .	23
3.1	2D FARSITE visualization output for Kyle Canyon, Nevada scenario. . . . .	26
5.1	Use cases for the VFire prototype application. . . . .	35
5.2	The main system structure of the VFire system. . . . .	36
5.3	Spatial divisions minimize the number of polygon-point tests. . . . .	40
5.4	Vegetation placed in cells of grid data (e.g. FARSITE outputs). . . . .	41
5.5	An example of a simulation graph used in a typical visualization. . . . .	42
5.6	Utility classes for the VFire system. . . . .	43
6.1	700 MB, 1-meter satellite image terrain visualization in VFire system. . . . .	49
6.2	150,000 billboard trees visualized in VFire system. . . . .	52
6.3	Sprite-based fire used in VFire system. . . . .	53
6.4	Wildfire spreads across the terrain engulfing vegetation. . . . .	55

# List of Tables

2.1	Visualizable FARSITE outputs . . . . .	4
2.2	The ESRI Ascii(Grid) format. . . . .	5
5.1	Functional requirements for the VFire prototype application. . . . .	33
5.2	Non-functional requirements for the VFire framework. . . . .	34

# Chapter 1

## Introduction

Wildfires are very unpredictable. It is difficult to determine exactly where and when a wildfire will happen next, and it is even more difficult to determine how a wildfire will spread with absolute precision. It is the unpredictability of wildfires that make them so dangerous. It is this reason that so much time and money is spent researching wildfire behavior. A large focus of this research effort is spent on the development of computational models of wildfire. However, visualization of these models' outputs has been quite limited. Even less research has been undertaken on immersive visualization. There are many advantages to modeling the spread of wildfires.

Spread models can be used to develop plans to fight fire, initiate more predictable prescribed burning and also predict the risk involved if a wildfire occurred in a certain area. Determining how much risk there is to an area's inhabitants and their property can be used to spend money appropriately and devise a proactive plan for evacuation and fire fighting. Kyle Canyon in Southern Nevada is good example of a high danger wildfire zone. In the event of a wildfire, it would be very difficult, if not impossible, to evacuate its citizens and would most likely end in a high fatality rate. Many management decisions rely on the results of wildfire spread model simulations. It is important that these models, to some degree, accurately predict the spread.

Validating these models is difficult without the wildfire actually happening and comparing the results. Visualization of the wildfire model output in an immersive environment can be used to validate its output against environmental factors such as terrain slope, fuel moisture, wind vectors and weather conditions. It can also be

used to compare model outputs against video footage or a visual recreation of the scene from collected data. This is only a single but important reason for visualizing wildfire model output. Visualization of these model outputs can be used to better train firefighters and fire management and aid in burning more predictable prescribed fires.

Burning a wildfire for the purpose of training is dangerous and costly. Virtual reality technology makes it possible to recreate wildfire scenarios with more realistic results than previously possible. Recreating a wildfire or using model output can be used to train fire managers and firefighters and aid in the development of plans and precautions. With the development of a real-time wildfire simulation, it would be possible to run through several virtual scenarios very quickly. This could be used to better determine what precautions to take while burning a prescribed fire.

The remainder of this thesis is structured as follows: Chapter 2 is a discussion of material relevant to understanding the wildfire and immersive environments, Chapter 3 describes work related to VFire, Chapter 4 presents an overview of the VFire framework, applications, purpose and goals and Chapter 5 covers the requirements specification and software modeling design. The implementation of a prototype application is covered in Chapter 6, and conclusions and future work are presented in Chapter 7.



# Chapter 2

## Background

### 2.1 FARSITE and Wildfire

FARSITE is a well-established fire behavior and growth simulator developed by the USDA Forest Service. It is used by fire analysts from most federal and state fire management agencies [50]. Its importance and widespread use among fire professionals was a critical factor for choosing to visualize its simulation output. FARSITE incorporates existing models of surface fire, crown fire, point-source fire acceleration, spotting and fuel moisture to calculate the spread of fire on a landscape [13]. These various models are used to propagate vectors of fire parameter polygons. Intervals of these expanding polygons are interpolated to generate raster data that describe fire behavior. Table 2.1 presents the file extensions and a descriptions of FARSITE outputs used for visualization. The raster data outputs are stored in ESRI ascii files, of which six of the outputs are of importance to visually constructing the wildfire scenario. The FARSITE simulator uses data that describes the topography, weather and fuel as inputs.

The behavior of wildfire is controlled by a variety of factors of which topography, weather and fuel are of the greatest influence. A critical factor to understanding wildfire growth is the shape a wildfire burns under uniform conditions. The FARSITE model uses an ellipse as the basis for the shape of wildfire growth, and factors such as wind and terrain slope cause the variations. The vector based approach used in the FARSITE model overcomes problems associated with cellular and fractal based

Output Type	Ext.	Units	Description
Time of Arrival	.toa	hours	The time the wildfire reaches a location (cell) on the landscape.
Fireline Intensity	.fli	kilowatt per meter	The amount of heat produced by the fire front directly related to heat transfer.
Flamelength	.flm	meter	Length of flames when fire is at full rate of spread.
Rate of Spread	.ros	meters per min.	The fastest, constant speed reached by the fire front.
Spread Direction	.sdr	0-359 degrees az.	The direction of fire spread.
Crown-No Crown	.cfr	1=surface, 2=passive, 3=active	States whether there was a crown fire at a given cell and the type of crown fire.

Table 2.1: Visualizable FARSITE outputs

methods. A description and comparison of other wildfire growth shapes and methods can be found in [13]. Los Alamos has applied the FARSITE model in a real-time simulation of wildfire [56], which has many applications for training and data analysis purposes.

## 2.2 Remotely Sensed Data and Geospatial information

Reconstructing real world landscapes and wildfire scenarios requires the use of remotely sensed data and geographic information systems (GIS). Several different types of information are useful for accurately visualizing a realistic landscape not only including digital elevation models (DEMs), fuel load data and satellite images, but also physical features of the landscape such as roads and buildings. Data (e.g. elevation and images) are usually saved in raster data files such as ESRI Ascii(Grid), GeoTiff or Binary Terrain formats. The geospatial information available in the ESRI Grid format is shown in Table 2.2. Vector information can be used to describe features

such as roads and other boundaries in shape file formats. Geospatial information is often included in these data formats and can be used to place the information in physical space on the Earth. Consideration of geospatial information associated with remotely sensed data is crucial to its accurate visualization.

<b>Header Entry</b>	<b>Description</b>
NCOLS	The number of columns.
NROWS	The number of rows.
XLLCORNER	The position western edge of the data.
YLLCORNER	The position southern edge of the data.
CELLSIZE	The resolution of the cells of the data.
NODATA_VALUE	Denotes that there is data missing at a cell.

Table 2.2: The ESRI Ascii(Grid) format.

Before this data can be visualized, the coordinate systems and method of projection must be considered. Coordinate systems associated with the Earth generally use latitude, longitude and possibly elevation; however, coordinates may correspond to different points on the Earth's surface depending on the model used. The simplest model would assume the earth is a perfect sphere, but this is generally not adequate. More complicated models use reference datums such as the World Geodetic System of 1984 (WGS84) which have more accurate measurements of the Earth. Projection is the process of converting data measured on the Earth, a three-dimensional surface, and mapping it to a two-dimensional surface such as a map or raster data file. Different types of projection methods are suitable for different uses and different types of data. To visualize DEM data the method of projection is necessary to reverse this process so that it can be visualized in three dimensions.

The majority of the data for this project is described using raster data formats. The Geospatial Data Abstraction Library (GDAL) [14] is a software package used to open and process these formats. This software package has tools for resizing and

cropping raster data, and also can be used to convert the data between different coordinate systems and projections. This is especially important for terrain algorithms and graphics hardware that put size constraints on the sizes of the elevation data and textures.

## 2.3 Virtual Reality

Virtual Reality (VR) as it relates to this project is the use of hardware and software technologies to allow users to view and interact with computer-simulated environments. The goal of VR is to mentally immerse a user within these environments through different types of sensory feedback. Sight (visual) is the type of sensory feedback most important to this project. Other types of sensory feedback are aural (sound), touch (haptic) and smell (olfactory). The broad definition of VR includes the visualization of a 3D world on a desktop computer because this can and often creates a sense of immersion. However, the specific definition used in this project involves visualization on stereoscopic displays and interaction captured by tracking systems, because these technologies offer many advantages over desktop systems. The source of these advantages is through the support of depth cues not achievable on desktop systems. This is a very brief overview of a very broad topic, more information about VR, depth cues, immersive displays and input devices can be found in [43].

### 2.3.1 Depth Cues

Depth cues are the important information a viewer uses to discern distances between objects in a scene. The more depth cues a system can support, the more potential for immersion is possible. Monoscopic, stereoscopic and motion depth cues are important to VR systems. Monoscopic depth cues are achievable on both immersive and desktop systems. These can be seen in a single static image of a scene, and can be drawn from image features such as size, shading and occlusions. Stereoscopic image depth cues are the differences determined between the images obtained by each eye (left and right images). Motion depth cues can be seen when a viewer changes the

relative position between them and an object. Figure 2.1 gives an example of the use of motion depth cues in an immersive system through the use of a head tracking system that updates the system to the position of the viewer's head. The viewer can gauge the distance of an object based on how fast an object passes by when they change perspective. Closer objects appear to move faster than farther ones. A 3D desktop environment is only able to provide monoscopic depth cues, but an immersive system with stereoscopic displays and head tracking can simulate all depth cues discussed. The addition of these depth cues allows for an experience closer to reality, making such an environment suitable for training applications. Improved data analysis and model validation also becomes possible because of the addition of extra depth information allowing for better accuracy and throughput.

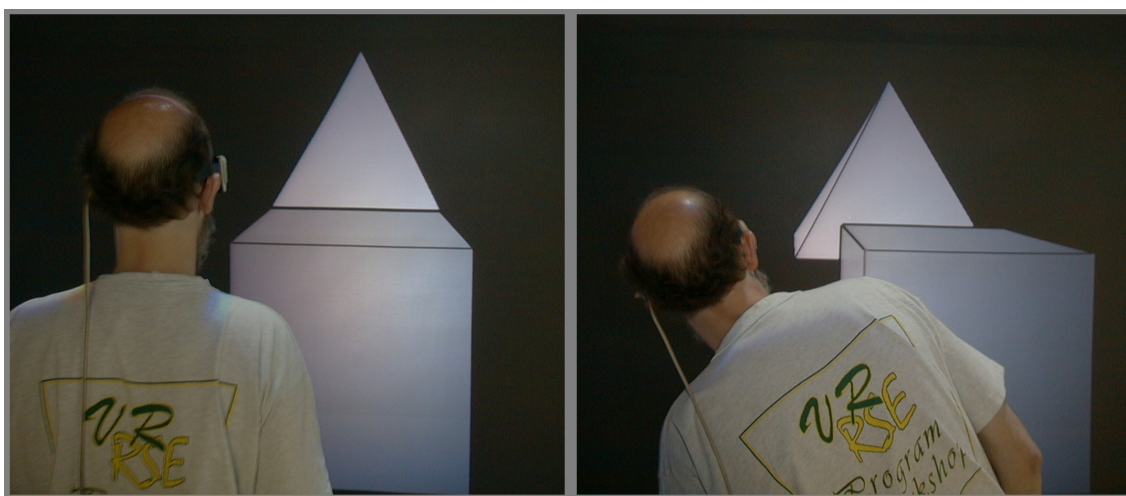


Figure 2.1: User moves their head to view other side of object [43].

### 2.3.2 Stereoscopic Displays

Rendering and displaying the viewpoint of each eye on a stereoscopic display allows the viewer to view stereoscopic depth cues. The two main types of displays for stereoscopic visualization includes head mounted displays (HMD) (Figure 2.2) and projected displays. Head mounted displays use two small, lightweight screens attached to the users head occluding the outside world. The occlusion of the outside world can



Figure 2.2: A Head Mounted Display (HMD) [43]

result in a diminished sense of immersion because the viewers cannot view themselves within the world. HMDs also suffer from latency issues where the rendering is not able to remain responsive as the user changes the orientation of their head. Multi-screen projection-based systems (Figure 2.3) solve these problems, but they are much more expensive.

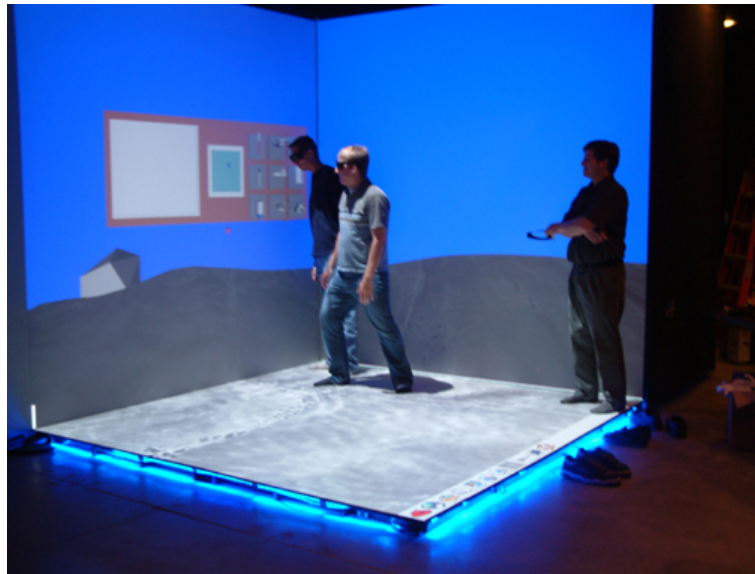


Figure 2.3: A multi-screen projection system.

Achieving stereo on projection-based displays can be done in several ways. Stereo can be achieved passively through the use of two different projectors, polarized filters and viewing glasses. On passive display systems, the left and right eye perspective are rendered on separate projectors with each having different polarized filters corresponding to the left and right eyes of the polarized viewing glasses. Another method, called active stereo, uses a single high-speed projector and shutter viewing glasses. The left and the right eye perspectives are delivered using temporal multiplexing, that is, each eye is rendered separately at different times, and the shutter glasses only allow the eye to view the scene.

Other crucial factors related to immersiveness are field of view and field of regard. Field of view (FOV) is the angular width that a display can cover. HMDs in general have less FOV than multi-screen projected displays with three screens or more. Pro-

jected displays can cover the overlap between the FOV of each eye resulting in better stereo. Field of regard (FOR) is the amount of the surrounding scene a system can display around the viewer. HMDs are capable of displaying 100% of a viewer's FOR, and more expensive 6-wall projection-based systems can also display 100% of viewer FOR.

### 2.3.3 Head Tracking and Input

Different types of display systems provide the ability to show stereoscopic depth cues which is important for objects closer to the user. Tracking the viewer's head position and more importantly orientation, provide the viewer with the motion-based depth cues. Orientation is particularly important for HMDs because this controls what is being rendered on the displays. Orientation is less important for a projection-based system because the system is constantly displayed from all directions. Position is used to render the scene from the viewer's point of view allowing them to inspect objects by simply changing the position of their head exactly as they would in a real world environment while wearing a head tracking device like the one shown in Figure 2.4a. The wand shown in Figure 2.4b is a common type of input device with virtual reality systems.

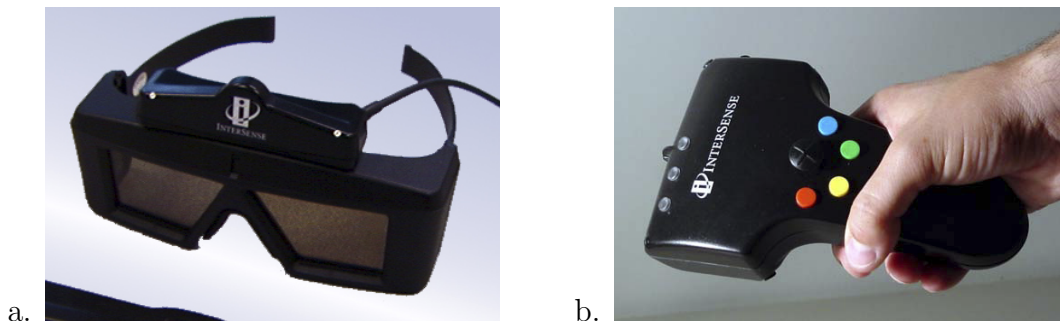


Figure 2.4: Intersense head tracking (a) and wand (b) input system. [21].

### 2.3.4 VR Hardware: Render and Update Loops

Multi-screen display systems require specialized hardware unlike HMDs, which can use hardware similar to desktop PCs. Multi-screen display systems require multi-



ple graphics pipes to keep real-time frame rates. The goal is to keep performance independent of the number of screens a system contains. A 6-wall system should have comparable performance of a 4-wall system if driven by similar hardware. Two configurations exist for achieving this goal. Shared memory systems with multiple graphics pipes and more recently cluster-based systems.

Shared memory systems support a single large memory image across all processors. Figure 2.5 shows a typical layout of a shared memory visualization system. On these systems a process is used to render each screen independently and one to many processes are used to update the simulation. Rendering each screen in parallel offers performance, which is independent of the number of screens in the system. Performance is further increased by allowing the simulation and rendering to run in parallel. The idea is to run as much of the update and rendering computation in parallel as possible, but because they are accessing the same data, the update writing and the rendering reading must be locked.

Cluster-based solutions run the simulation and rendering code on a node for each screen in parallel, and a head node keeps the simulations in sync as shown in Figure 2.6. It is also possible on these systems to run the rendering and simulation code in parallel for increased performance if multiple processors are available. The shared memory solution has the disadvantage of requiring expensive specialized hardware to support multiple graphics cards. High performance available commodity hardware can be used in cluster-based systems with a fast interconnect.

### **2.3.5 Virtual Reality Toolkits**

VR systems contain specialized input and tracking hardware. They also contain specialized screen and computational hardware configurations making writing software for these systems a monumental task. The hardware configurations for these systems can vary drastically between different organizations requiring software to be changed for each of these systems. VR toolkits attempt to abstract these differences so that applications can be written once and run on all of these systems. The large

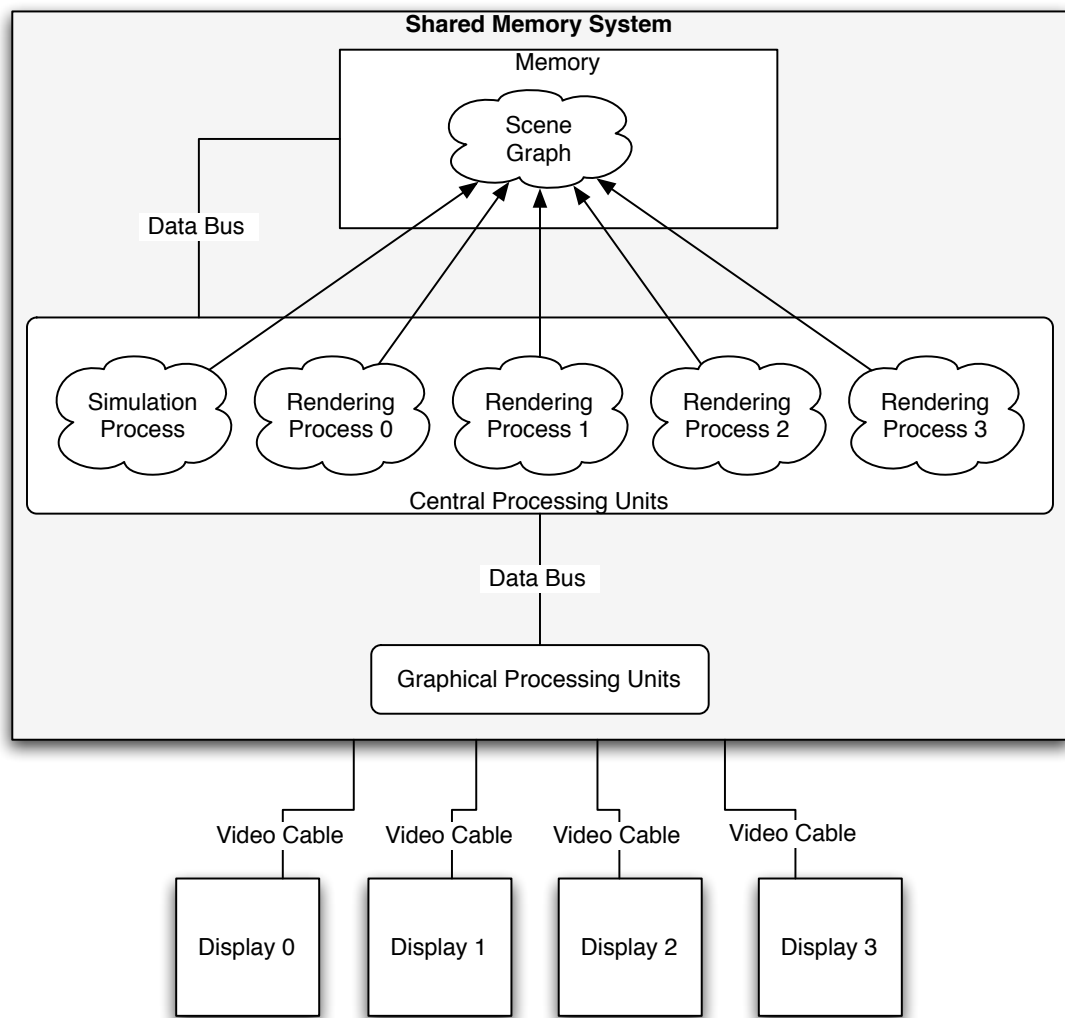


Figure 2.5: The architecture of a shared memory visualization system.

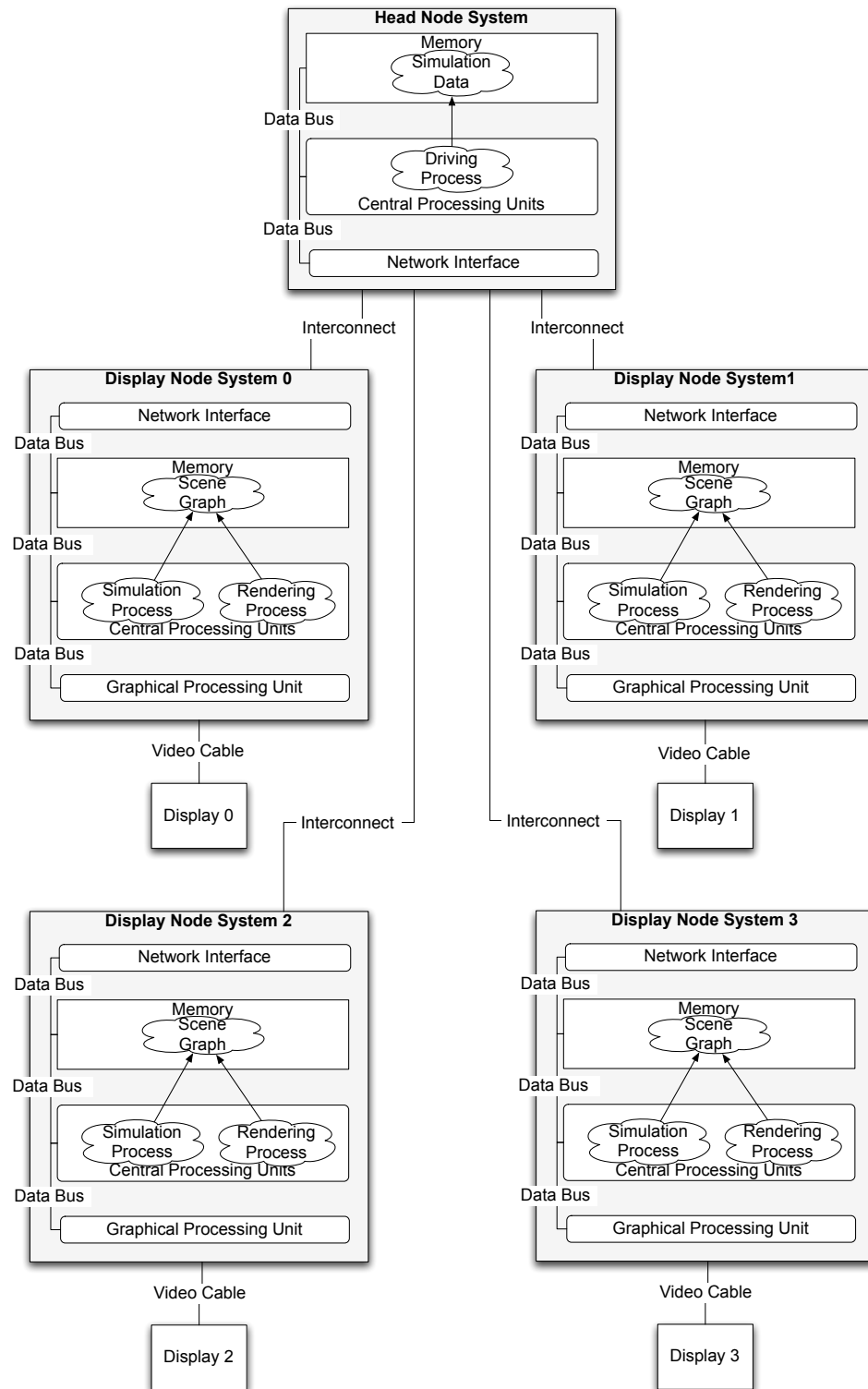


Figure 2.6: The architecture of a cluster-based visualization system.

difference between VR toolkits is the type of input hardware and computational configurations they support. William R. Sherman’s FreeVR [42] supports a variety of input and tracking systems such as common desktop game pads all the way to high end tracking systems such as InterSenses IS-900<sup>TM</sup>, but only supports shared memory systems. FreeVR is a free and open source alternative to VRCO’s CAVELib [52], one of the original VR toolkits. VRUI [25] and VR Juggler [20] supports similar tracking hardware and also support cluster-based systems.

## 2.4 Scene graphs

A scene graph is a high level data structure abstraction around lower level graphics libraries for the purpose of organizing and optimizing rendering. The definition of scene graph has also become ubiquitous with describing software packages that use the data structure. These software packages often offer more than just optimized rendering, but also include support for loading scene content. Scene graphs offer many benefits over traditional low level graphics libraries such as rapid development and scene optimization.

Scene graphs use many optimizations to speed up rendering. Many of these offer different types of culling which quickly determine visibility of geometry and reduce the amount of triangles which need to be rendered. Frustum culling removes geometry outside of the viewers point of view. Occlusion culling removes geometry which is not visible because it is behind other geometry in the scene. Small feature culling removes geometry smaller than a particular amount of screen space in pixels. Scene graphs also reduce the amount of expensive state changes such as changing shaders, textures and other rendering states. They also often implement lazy state updating, which only updates states that are not already set. Optimization traversals run at initialization time can optimize a scene graph data structure and the geometry contained within. These optimizations include organizing the scene graph into an octtree for optimized frustum culling or organizing triangle based geometry into optimized triangle strips.

High level abstractions around often tedious tasks such as texturing and content

loading make scene graphs excellent tools for rapid development of applications. Even when it is necessary to write code in the lower level graphics libraries, scene graphs provide tools which make this easier and often results in faster code.

Although there are many scene graphs, few are open source and suited to the development of virtual reality applications. Support for multi-pipe rendering is a fundamental feature necessary for rendering on a shared memory system. This support includes management of OpenGL objects (Texture object, Vertex buffer object, etc.) and a data protection mechanism. Data protection is often domain specific and a generalized approach cannot achieve equivalent performance. Both OpenSG [33] and OpenSceneGraph [32] are two scene graphs that meet this criteria. OpenSG implements a system which allows the scene graph to be transparently shared across multiple machines in a cluster or several processes on a single machine (details about the implementation of this system can be found in [37] and [40]). OpenSceneGraph does not have the ability to share the scene graph data structure, and must be protected using an external locking mechanism or the rendering and simulation update must be done sequentially. Both of these scene graphs share similar structure and function to SGI's Performer [38].

## 2.5 Fire and Smoke Visualization

Creating realistic, real-time fire and smoke is a very difficult problem. Massive amounts of research hours have been spent realistically rendering these phenomenon. Although there are several different methods, very few lend themselves to real-time rendering, and even fewer are computationally efficient enough for the scale required for rendering entire wildfires. In general, these algorithms fall into two categories: volumetric and particle based systems.

Volumetric based fire and smoke use fast fluid systems and properties of combustion to change the color and opacity of voxels. These methods have proved to render the most physically accurate and realistic results; however, volumetric methods are computationally expensive. Current research has shown that volumetric rendering

can be used to render fire and smoke efficiently enough for real-time, and its performance scalability looks promising [15]. Volumetric methods also have the added advantage of being implicitly efficient when calculating collisions with objects within the environment. They do not have the clipping artifacts associated with planar billboards collision with three-dimensional objects.

Particle based systems use particles to represent localized behavior within a fire. Reeves is the first fundamental work with representing natural phenomenon using particle systems [36]. An example of fire behavior visualized with a large numbers of particles can be seen in Figure 2.7. Representing localized fire behavior with smaller single point particles is often inefficient and it is difficult to accurately represent fire and smoke in real-time as in Figure 2.8.

A better and often more accurate alternative is to represent this localized behavior with animated sprites [58]. These sprites can be produced via offline, non-real-time rendering using an accurate, physically based method. Sprites are then mapped on to viewer faced two-dimensional billboards. Using animated sprites allows us to represent fire accurately using a small number of particles. It also becomes much more feasible to control these sections of fire behavior using a realistic, physically based model in real-time. A real-time fire simulation then could control behavior through varying the color and stage of animation of a particle local to the region. Particle-based systems historically suffer from hard edged artifacts in the event of collision with three-dimensional meshes. Current research has discovered a solution to this problem; however, it remains too inefficient to represent large particle systems associated with large wildfires [47].

## 2.6 Terrain Visualization

Wildfires can spread over large areas of land, large enough that they may not fit into memory and may be too large to view in real-time. To portray a scene accurately and increase the immersiveness of an application, higher detailed terrain height maps and textures are necessary. Terrain data are commonly depicted as height map data

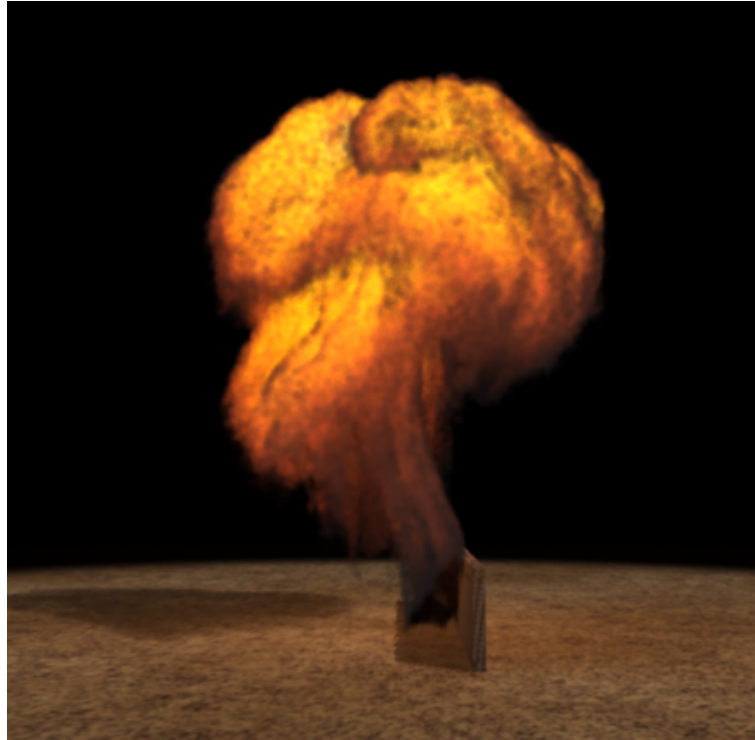


Figure 2.7: Particle fire using millions of particles [12].



Figure 2.8: Sprite-based fire using volumetric textures [29]

in a variety of raster data formats including image formats and digital elevation models; however, height map data are usually not directly rendered. This data is first tessellated before it is rendered and large data height maps result in meshes with millions of triangles. Rendering these higher resolution datasets is simply not possible using brute-force algorithms.

Several algorithms have been developed to increase the resolution and the visible detail of terrain. Instead of rendering the terrain at full detail with millions of triangles as with the brute-force methods, these algorithms optimize the visualization of these datasets by removing unnecessary detail. In general, there are two ways of completing this task. First, detail can be removed by optimizing the height field by removing unnecessary detail independent of the users position [1, 16]. Flatter, less “rough” areas can be represented with fewer triangles. This detail would not be noticeable regardless of the distance the user views that particular location on the terrain. View-dependent methods reduce the amount of detail based on the view and location of the viewer’s position. The idea behind these methods uses the assumption that the viewer is less likely to notice detailed features of the landscape as distance increases. Higher detail is used to represent the terrain closer to the user, and data are progressively simplified as the distance increases [19]. The combination of these two methods allows computers to visualize larger and more detailed datasets.

Initial work with algorithms using the second method focused on optimizing the view-dependent mesh on a per vertex level. The goal was to represent the terrain with an optimal mesh with a minimal amount of detail as possible without visual degradation. The process of optimizing the mesh per frame, known as continuous level of detail (CLOD), is a processor intensive task. An example of terrain using the Real-time Optimally-Adapting Meshes (ROAM) CLOD algorithm is shown in Figure 2.9. This algorithm tightly tessellates the mesh on a per vertex level. These were good methods for the less powerful graphical processing units (GPU) of the time. As GPUs grew to maturity and their power increased many orders of magnitude, central processing time became the bottleneck and these algorithms became less ideal. Algo-



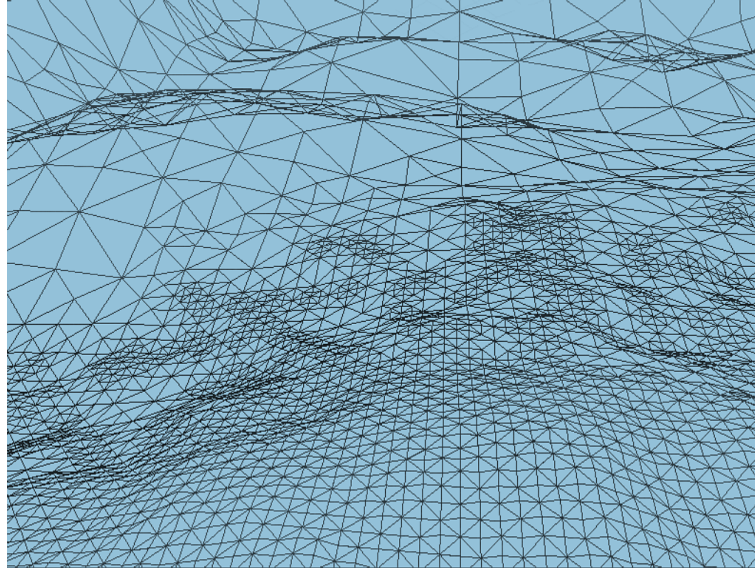


Figure 2.9: Terrain simplified using the ROAM algorithm [9].

rithms started to simplify the terrain using groups of vertices rather than per vertex computations increasing terrain performance and details to new levels. Figure 2.10 shows the GeoMipmapping algorithm which simplifies the terrain using patches rather than per vertex. To further increase performance, offline tessellation of terrain groups has removed the CPU as a bottleneck completely. Figure 2.11 shows the Chunked LOD algorithm which is a cross between the first two algorithms. Patches of vertices are simplified offline using a per vertex algorithm.

View-dependent algorithm’s progressive reduction of detail can lead to a consequence called “popping”. This is when the user is able to view the changes in the level of detail as the viewer moves around through the landscape. This problem has been amplified with the development of GPU-friendly algorithms, which treat the terrain as fewer and larger sections. A simple solution for decreasing this visual error is to extend the distance which higher detailed data is displayed. However, this increase in detail also means less performance. Another solution is called geomorphing, which linearly interpolates between the height values of a higher level of detail to the lower detail representation. In this way, popping can be minimized without having to increase the displayed level of detail. Geomorphing processing moved more

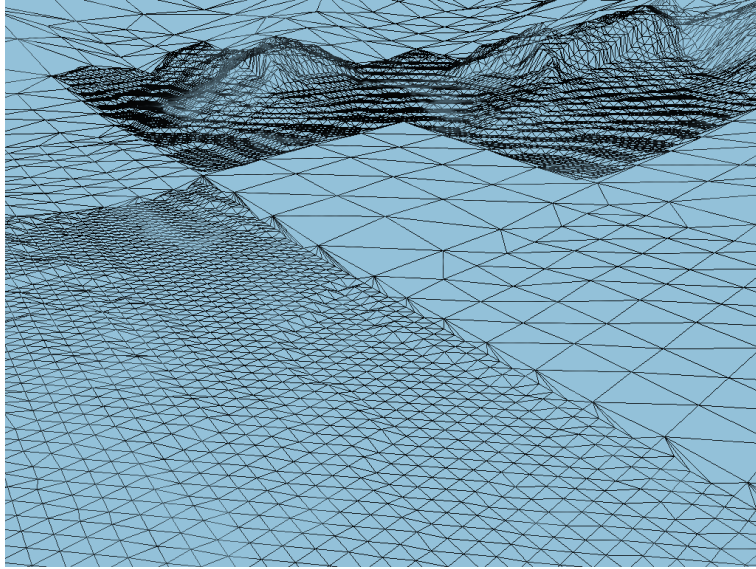


Figure 2.10: Terrain simplified using Geometrical Mipmapping [59].

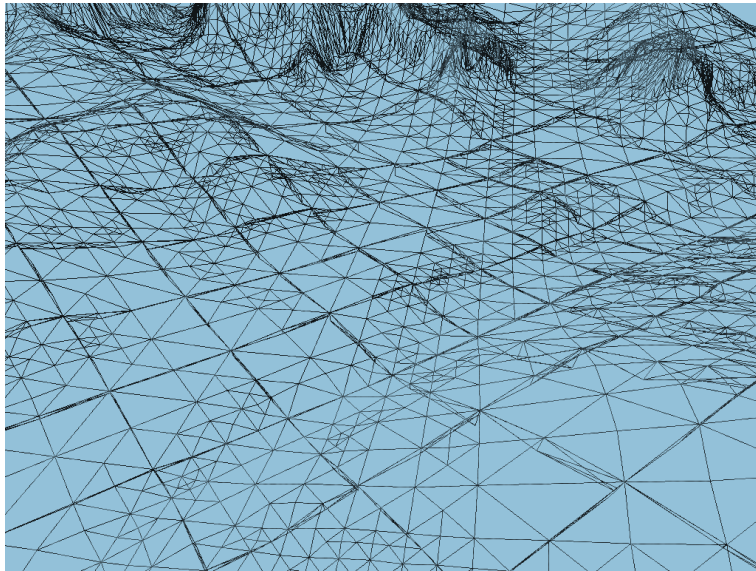


Figure 2.11: Terrain simplified using Chunked LOD [49].

work back on the CPU to process the smooth transitions between levels of detail [62]. With the advent of shaders, this processing can be moved to the GPU through the implementation of a vertex shader.

Another consequence of group oriented algorithms is the appearance of cracks between the different groups. These cracks appear when different groups are displaying different levels of detail. One solution is to fill the cracks with a “skirt”. A “skirt” is a wall of polygons attached to the edges, below each patch of terrain as shown in Figure 2.12. The advantage of a “skirt” is that it only has to be calculated once regardless of the level of detail of its neighbors. This means it can be calculated offline when the terrain is being tessellated. The other solution is to fill the cracks with a “ribbon”. A “ribbon” directly fills in the gap between adjacent tiles with a group of triangles, shown in Figure 2.13. The outside layer of triangles in higher detailed patches are adapted to lower detailed neighboring patches. This results in a perfectly smooth mesh; however, this is at the cost of extra computation. “Ribbons” must be recalculated when a patch or its neighbors change to a different level of detail.

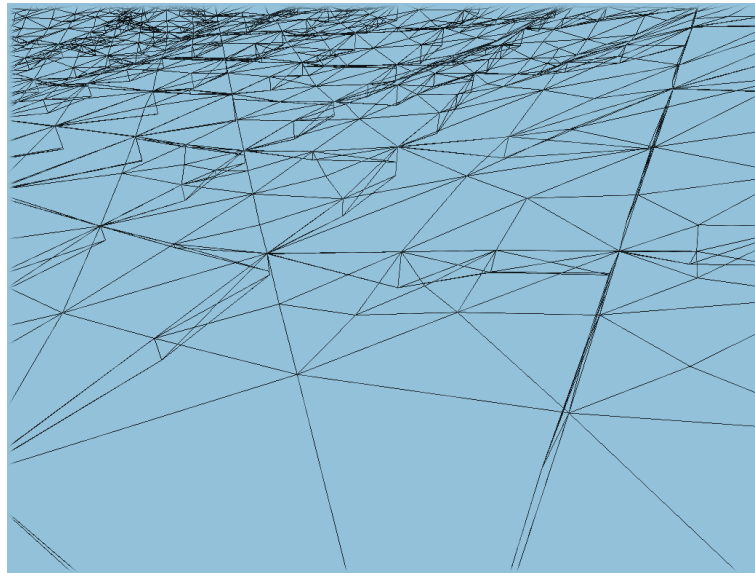


Figure 2.12: “Skirts” hide cracks between adjacent tiles.

Multi-screen systems bring a whole new set of problems to terrain rendering. View-dependent terrain algorithms depend on the status of the perspective matrix to

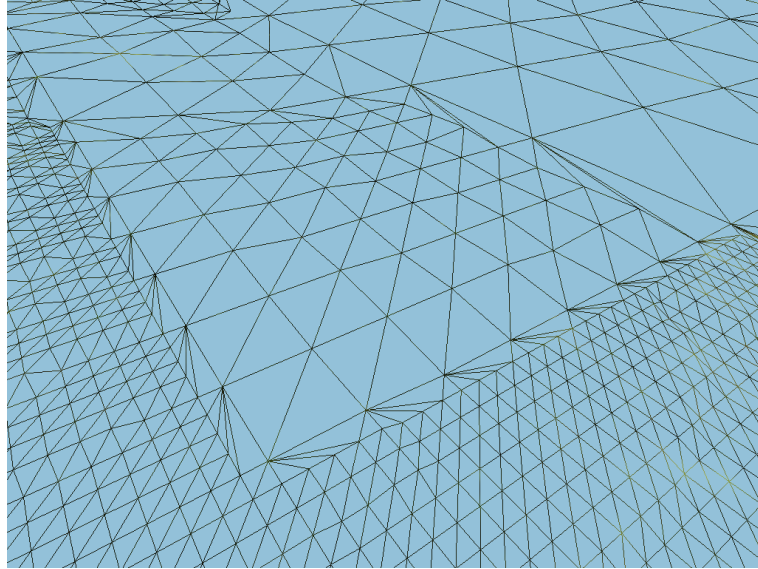


Figure 2.13: “ribbons” hide cracks between adjacent tiles.

simplify the terrain. The current status of the perspective matrix is not known until a screen is rendered. In a virtual reality system it must be calculated per screen and per eye. Having a single copy of the terrain is not feasible because each screen must protect the terrain data as it is modified, resulting in decreased performance. Each screen must wait for its turn to modify the terrain. A possible solution is to have a local copy of data for each screen to render the mesh; however, this may lead to visual inconsistencies between multiple screens. This may be further amplified by terrain level-of-detail algorithms, which assumes a symmetrical frustum, because a skewed frustum is used to correctly project a scene according to the viewer’s head position and the offset of their eyes.

## 2.7 Vegetation Visualization

### 2.7.1 Placement and Vegetation types

A crucial step to visualizing realistic landscapes is correct placement and sizes of different types of vegetation native to an area. Experts often have an intimate knowledge of the locations they are visualizing, and correct placement of the trees allows for more accurate visualization as well as increasing intellectual immersion. If vegetation

is placed incorrectly, it will be hard to prove the validity of the rest of the visualization to the expert. Vegetation types, sizes and placement can be determined from publicly available information and remotely sensed data. Forest Inventory and Analysis (FIA) data are often used to reconstruct forest landscapes [54, 55, 61]. An example visualization of a forest area constructed from FIA data is shown in Figure 2.14. FIA data are publicly available and free for areas within the United States (US). Fuel load data [41] used as inputs to FARSITE, along with an expert’s knowledge of vegetation types can be used to correctly reconstruct the visual forest ecology. Another possibility is to use satellite images directly to determine vegetation densities [6, 22]. Possibly more information such as type and approximate size can be determined from higher resolution data.

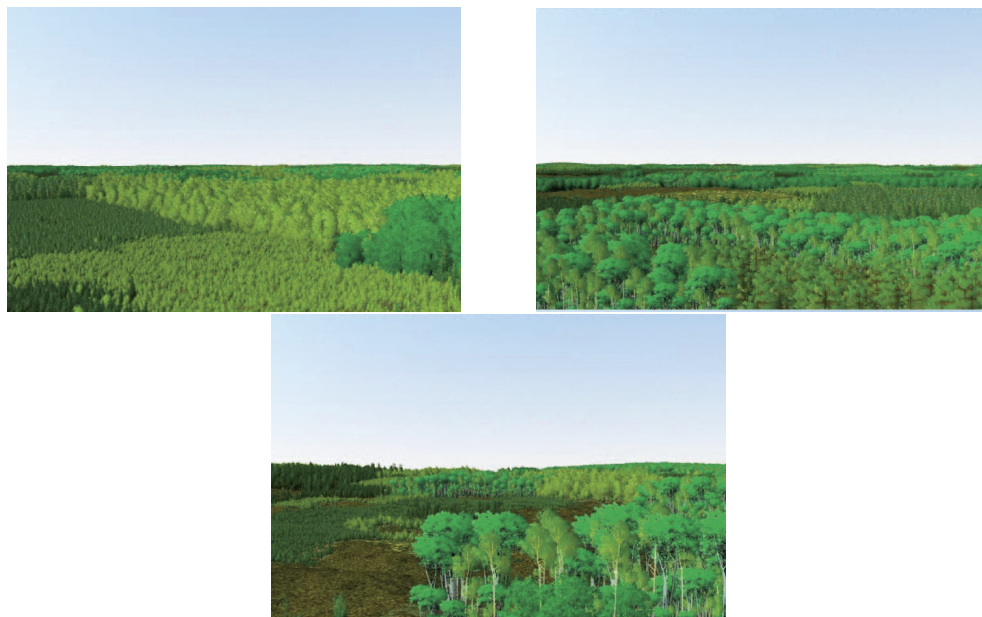


Figure 2.14: Vegetation placed according to FIA data [54].

### 2.7.2 Rendering Vegetation

Rendering a large, dense forest of trees in real-time is a difficult problem. This problem is intimately related to the rendering of each individual tree because it is necessary to reduce the rendered detail as its distance is increased from the viewer.



Hardware is not currently powerful enough to render an entire forest of trees at full detail.

Billboards and impostors are used to render trees far enough for the viewer while still maintaining visual integrity. Billboards are a common method of rendering vegetation in real-time applications, because this allows for the rendering of vast, dense forests. However, up close billboards do not have sufficient detail to maintain immersiveness, especially when displayed on a stereoscopic display. Borse and McAllister describe a method of efficiently rendering vegetation using billboards that also shows suitable visual results in stereo environments [4]. Distant vegetation can be represented by impostors, which are billboards generated on-the-fly, allowing for views from all perspectives while representing trees with minimal polygons [23].

Other methods use volumetric vegetation to render large, dense forests [8] or use ray-casting and vegetation cover data to give the appearance of trees [27]. These methods give good results for viewing forests from an aggregate perspective, but are not visually adequate for up-close views. Even when a user is adjacent to vegetation, it is not feasible to render the leaves and details of vegetation using polygons. These can be rendered using high-detail billboarded sprites, which give good visual results as well as being efficient [60]. Rendering individual trees is also related to an overall data structure for rendering large numbers of vegetation such as a quad tree, which can be used for fast visibility testing as well as resource management.

Obtaining realistic models for visualizing vegetation is usually done in two ways: images and algorithmically. The most accurate results can generally be obtained through reconstruction from images either through the use of algorithms [35] or an artistic recreation. Other methods use computational models to reconstruct the geometry of different types of vegetation [57].

# Chapter 3

## Related Work

A large amount of work has been the focus of developing computational wildfire models for the purpose of training, planning and analysis. However, immediate efforts to visualize this data have been quite limited. These visualizations are traditionally two dimensional or use graph output. The FARSITE application has a two-dimensional view (Figure 3.1) and also a limited three-dimensional view. These views are traditionally overhead and used to view the entire landscape. This is good for data analysis, but somewhat limited for training and planning where a higher sense of immersion is more beneficial. Higher information throughput is also possible for the purposes of model validation and data analysis because of the increase in depth cues and the extra third dimension. Virtual environments have proven useful especially for firefighting training applications.

Much work exists for visualizing fire for the purpose of training and analysis. However, little of this work has been done to visualize fire and wildfire in an immersive medium. A good amount of work has been spent visualizing computational models for in-building fires [5, 17]. The application of virtual environments and realistic spread models for application in firefighting training scenarios can be seen in [24] and [48]; however, the models and fire visualization are not applicable to outdoor, large-scale fires.

Los Alamos developed a tool for the visualization of wildfire data; however, both their model and visualization ran slower than real-time [2, 28]. This work explores the applicability of visualization of wildfire to training, and describes the graphical

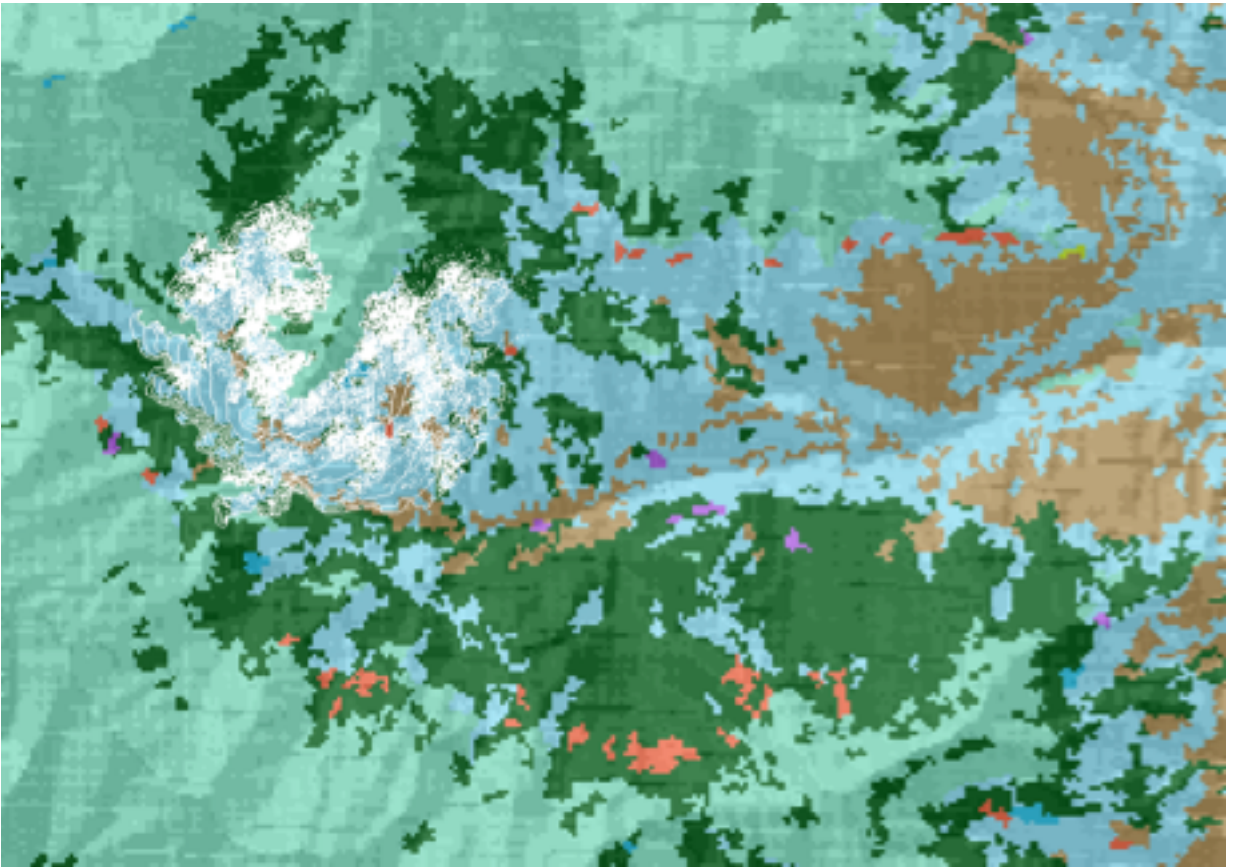


Figure 3.1: 2D FARSITE visualization output for Kyle Canyon, Nevada scenario.



elements necessary for visually reconstructing wildfire scenarios. Similar work uses GIS data to reconstruct forest landscapes and wildfire scenarios, but does not explore the immersive aspects [61]. They chose to implement their own elliptical wildfire spread model based on the Huygens principle of wave propagation, and they localized fire behavior using the Rothermel's model. They are able to achieve real-time frame rates on a desktop system using a custom forest level-of-detail system and wildfire spread model. Their paper did not address the validity of their wildfire spread model, and did not address the application or complexities of their application in a virtual environment.

The complexities of virtual reality, wildfire visualization, and scenario reconstruction are many. Previous work fails to encapsulate these complexities. This project also addresses the software engineering side of building this type of application. It is important to consider the functionality the system must support to building a complete and reusable framework.

# Chapter 4

## Complexities of Wildfire Visualization

VFire: Virtual Fire in Realistic Environments is a project with the goal of visualizing actual and simulated wildfire data of real physical locations using immersive systems and technologies. Managing scene and data complexity is a large overall theme of the VFire framework. The initial work exploring the requirements and the software methodologies behind the framework are presented in [18]. Managing this complexity will allow for the development of wildfire visualization applications, which visualize different wildfire models on different types of immersive hardware. The complexity and large datasets involved in the visualization of wildfire is considerable [34]. These complexities are compounded by the use of immersive systems with their specialized hardware and exotic input devices. [44] describes these complexities and how they were solved. This work has been integrated into the VFire framework.

Frequently, wildfire can cover large landscapes many times beyond the current computational and memory capacity of current visualization hardware. Each graphical element such as terrain, vegetation and fire is associated with a large dataset, either describing the landscape or driving the simulation. Each one of these elements has its own challenges and performance bottlenecks. The DEM and satellite images describe the terrain. The fuel load data is used to construct the vegetation environment. The wildfire is spread according to FARSITE outputs. The project goal is to achieve real-time visualization of wildfire that covers vast landscapes, and each

element must be managed to maintain visual fidelity and run in real-time.

## Chapter 5

# VFire Software Specification and Design

Chapter 2 introduced the complexities and the application of wildfire visualization. Managing computational and data complexity is a worthwhile problem, but it is only useful if it is presented to the developer in an accessible and flexible interface. The framework should provide an interface that requires less time and learning curve than developing a similar application from scratch. A rigorous understanding of the goals, requirements and functionality of the system is necessary for building a complete and easy to use framework.

This chapter presents a formal understanding of the goals, requirements and use cases. The requirements represent the basic features of a prototype application and the framework, and also the technologies needed to support them. The use cases describe the basic functionality of the system. More detailed descriptions of the overall system and subsystems are presented later in this chapter. The software engineering practices and principles used to develop VFire can be further understood in [3], [10] and [45]. It is the aim of this chapter to provide you an understanding of the structure and methodology behind the VFire framework, and how this supports the development of a prototype application.

## 5.1 Framework Goals

First, the framework has the ability to visualize the different graphical elements of a wildfire scenario. It must allow for different implementations of these elements to plug into the system because there are many different ways to visualize these elements. For instance, there are many different implementations and algorithms used to visualize fire and smoke, including volumetric and particle based systems. Each of these methods has their strengths and weaknesses, and future methods will improve upon these. Therefore, the goal is to accommodate these different and future implementations.

Time constraints and differences in expertise require many developers to work on the project in parallel. Another underlying goal is to allow for developers to build the different subsystems in parallel and add incremental improvements without affecting the entire application. Modular design and specification will prevent difficulty when integrating these different components developed by different individuals or groups. VFire is a large and complex piece of software that will require many people to work on several different components over many iterations of the project.

The previous goal also encapsulates code reuse because this object-based system maintains modular code, and can be reused to build different wildfire applications. This is important for building wildfire applications with different purposes and hardware requirements from previously implemented components.

A main idea behind the framework is to keep the data structures and the logic used to update these data structures together in a modular component. These modular components are linked together into different hierarchal structures to represent different relationships with one another. An example of this goal is to facilitate the development of different modules for the many ways of rendering fire or smoke and allow them to be easily added to a wildfire application. This type of implementation change of a single part of an application should require only very little or no changes to the rest of the application.

## 5.2 Hardware Support

Virtual reality hardware and toolkits are constantly evolving in drastic ways. Unlike PCs, virtual reality hardware changes are fundamental and architectural. Currently, two types of architectures are used in VR shared memory and cluster based systems. Drastic software changes are needed to take advantage of these hardware changes. The framework should be setup in such a way that minimizes these changes but also takes advantage of the underlying hardware.

These modular components make a clear distinction between the display updating and syncing of the data. This is important to the types of systems that require or might benefit from this type of distinction. This separation is not commonly seen in computer graphics; it is not uncommon to see data structures modified inside the display thread. Multi-screen VR systems require this separation because they have multiple display threads. To keep congruency across multiple displays, data cannot be modified in the display threads. This type of separation can also lead to increased performance on systems with multiple processors through coarse-grain multiprocessing; that is, updating and display logic can be run in parallel. It is often possible to write update logic so that it can run in parallel with display code only locking during data synchronization. A goal of this framework is to enforce this policy that distinguishes data display from data update. The idea is to have generic components for everything that makes up the system. Each of the components implement an interface that allows that component to interact with the rest of the system. This distinction also facilitates shared memory and cluster based systems.

To achieve high performance, the amount of display and simulation computation run in parallel should be maximized. Abstraction of dynamic graphical elements into several steps is necessary to achieve this goal. The processing of each element is broken into three stages: 1) rendering the code which displays the element; 2) updating the simulation code; and 3) synchronizing and maintaining congruency between the first steps. The first two steps are run in parallel modifying an independent copy of the

data and the last step synchronizes the two copies. This uses the assumption that the simulation code in step 2 is non-trivial and would require more time than syncing the data in step 3. OpenSG provides a similar, more generic mechanism; however, a generic solution is not always possible and a higher frame rate is better with some specialization.

## 5.3 Requirements

Before the rest of the system can be described, a basic understanding of the functionality and features need to be outlined (Table 5.1). This is the crux for the design of the rest of the system. These features were conceived through domain knowledge and input from the scientists that will use the system.

F01	V-Fire shall display a visual indicator of the wildfire spread.
F02	V-Fire shall display the terrain features of the landscape.
F03	V-Fire shall display the vegetation ecology of the landscape.
F04	V-Fire shall display fire and smoke controlled by simulation data.
F05	V-Fire shall allow the user to start the simulation.
F06	V-Fire shall allow the user to freeze the simulation.
F07	V-Fire shall allow the user to restart the simulation.
F08	V-Fire shall allow the user to change the simulation time scale.
F09	V-Fire shall allow the user to navigate the landscape.
F10	V-Fire shall allow the user to load different landscapes.
F11	V-Fire shall allow the user to load different wildfire scenarios.
F12	V-Fire shall display the current progress of the simulation.
F13	V-Fire shall display the current time scale setting.
F14	V-Fire shall display the file name of the current landscape scenario.
F15	V-Fire shall display the file name of the current wildfire scenario.
F16	V-Fire shall display the current location landscape.
F17	V-Fire shall display the current orientation using a 3D compass.
F18	V-Fire shall set the users location to the point of ignition.

Table 5.1: Functional requirements for the VFire prototype application.

Non-functional requirements (Table 5.2) show the supporting technologies, software features and most importantly outline the major features supported by the framework. This largely controls the interface and features presented to the devel-

N01	V-Fire shall use the OpenSceneGraph (OSG) library and C++.
N02	V-Fire shall implement modular components for visual elements.
N03	V-Fire shall support varying implementations for visual elements.
N04	V-Fire shall allow visual elements to be used with all simulation types.
N05	V-Fire shall maintain independence from windowing/VR toolkit.
N06	V-Fire shall maintain high temporal fidelity.
N07	V-Fire shall support different types of wildfire simulations.
N08	V-Fire shall separate rendering, updating and syncing.
N09	V-Fire shall place vegetation using fuel load data.
N10	V-Fire shall use digital elevation data to describe the terrain.
N11	V-Fire shall use a satellite image for texturing the terrain.
N12	V-Fire shall implement the Chunked LOD algorithm for terrain.
N13	V-Fire shall use the OpenThreads library for data protection.
N14	V-Fire shall abstract OSG for pure OpenGL programming.
N15	V-Fire shall support both real-time and data driven simulations.
N16	V-Fire shall implement particle-based fire and smoke.
N17	V-Fire shall use FreeVR library for prototype application.

Table 5.2: Non-functional requirements for the VFire framework.

oper.

## 5.4 Use Cases

The functionality provided to the end user is shown through use cases (Figure 5.1). These are developed using the functional requirements. Only functional requirements representing major features visible to the user are represented as use cases. There are two major actors that influence the VFire system. The user provides input and visualizes the simulation output. Time controls the update of the simulation, and animates user navigation and tools as a direct result of user input.

## 5.5 Modeling and Design

This section describes the structure and design features of the main VFire Visualization System. This includes the main system used to build visualization trees and the utility functionality used to interface the visualization system.



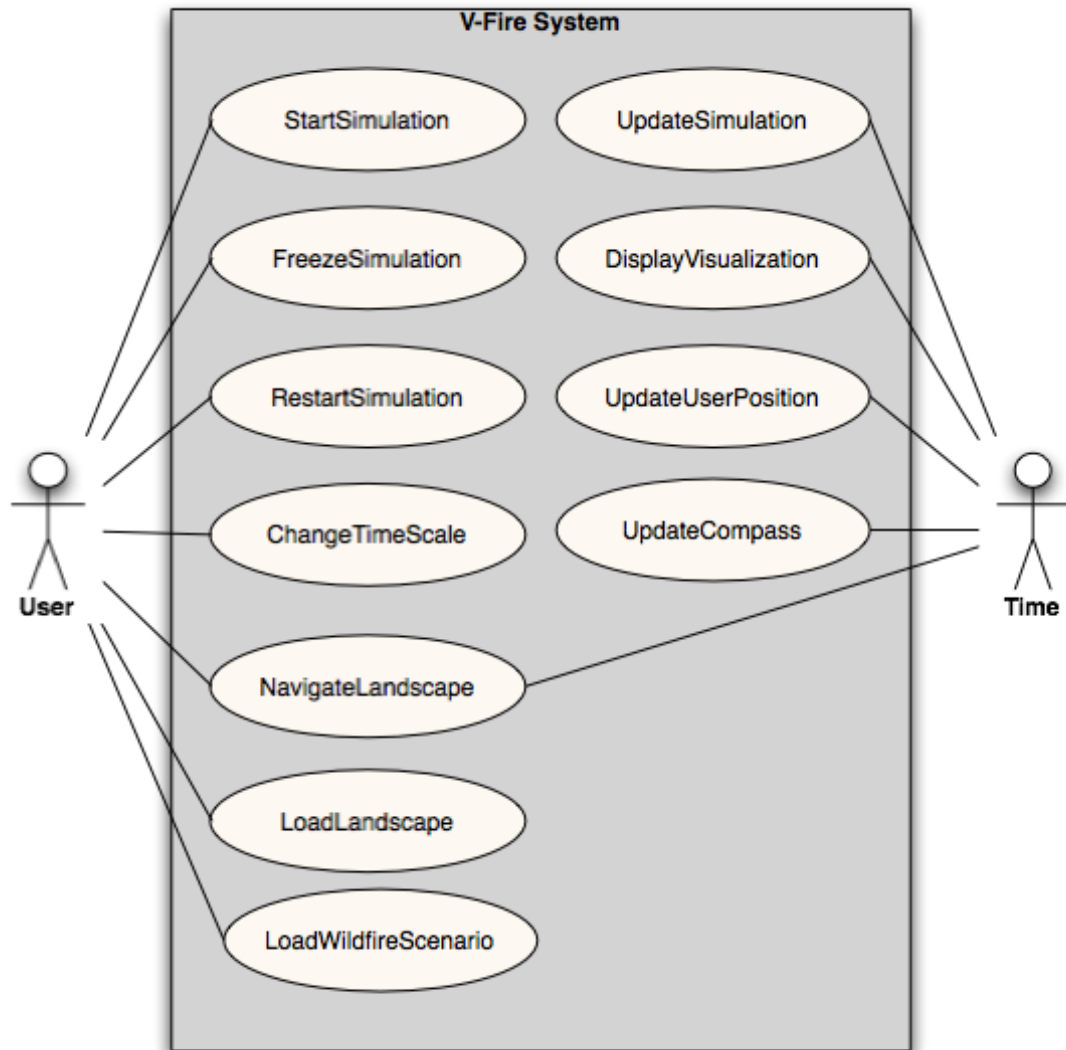


Figure 5.1: Use cases for the VFire prototype application.

### 5.5.1 Main Class Hierarchy

Figure 5.2 shows the inheritance hierarchy of the visualization system. This main system of classes is used to build visualization trees which describe a wildfire scenario. It's aim is to be flexible and extensible for further development and refinement of wildfire applications.

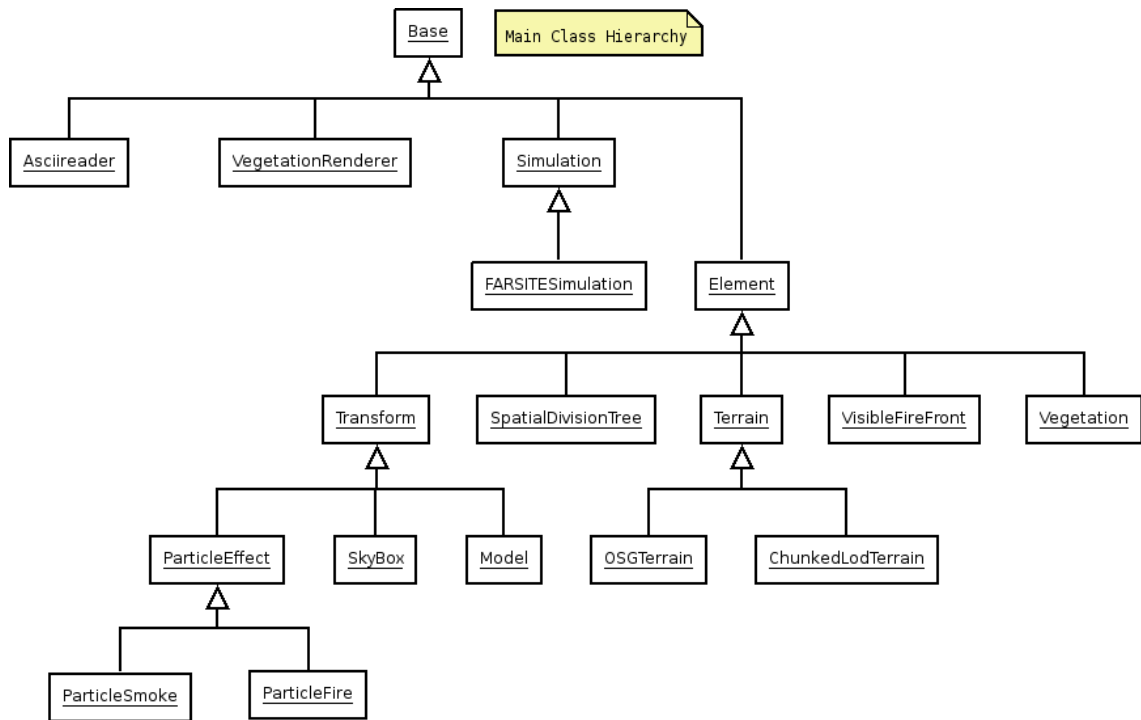


Figure 5.2: The main system structure of the VFire system.

#### Base Class

All memory managed objects are derived from this class. This class is responsible for keeping reference counts of the object, and it is deleted when no longer in use. This is important with data that are shared across multiple objects because these data should only be freed when the last object using it is also freed. Other management features such as the type name and a unique identifier are stored in this layer. These data can be used for runtime type determination and resource management.

#### Asciireader Class

This class is used to read and manage data read from ESRI Grid files. More

specifically this is used to read FARSITE outputs and DEM data. Currently this reads all data from the file into memory because the data are stored in an ASCII format, and cannot be randomly accessed. For datasets larger than the available amount of memory, the ASCII data would need to be converted to a binary format so that it could be paged in and out of memory.

### **Element Class**

All objects that are visualized in a scene are derived from this class. This not only includes objects that are part of the wildfire scenario such as terrain and vegetation, but also graphical user interface objects. Graphical elements derived from this class display the scene. These are equivalent to nodes in a scene graph, but differ in several ways. An element itself can encapsulate an entire scene graph, not just a single node. It also provides the mechanism to update its scene graph data to reflect the current state of the simulation.

This mechanism is used to update the sub-scene-graph contained within the element. Data and the logic used to update the scene graph can be separated from the graph itself. This is important because the scene graph cannot be changed while it is being rendered. Because an element has a local copy of data, the rendering thread can display the scene graph while the scene is being updated. The scene graph and the local copy of the data are synchronized at the end of each frame.

Because each display element is a modular thread-safe object, they can be easily be added to different simulation types and configured to develop different applications. The Element Class provides an interface and does not display or update anything. It is the job of the derived class to provide the structure, interface and implementation of more specific behavior.

### **Terrain Class**

The Terrain Class provides a generic interface for the implementation of different terrain rendering algorithms. The interface enforces functionality that allows other objects to obtain elevation and geospatial information. Elevation information is used to place objects correctly on top of the terrain and used for collision. Collision

detection is used to keep dynamic objects such as the user from falling through the terrain. Geospatial information is used to align other objects in the scene such as vegetation.

Two different implementations of the Terrain Class are currently implemented in the VFire system. The first implementation encapsulates the rendering of OSG terrain databases produced by the “osgdem” utility. The current implementation uses the Chunked LOD algorithm. The terrain interface allows different implementations to be substituted without having to modify the rest of the system.

### **Vegetation Class**

This class does not provide an interface, but rather provides the functionality to load vegetation databases produced by the “makevegqt” utility. The database files are structured so that vegetation data can be paged into memory. The VegetationRenderer Class is used to provide the implementation to render vegetation. The vegetation renderer is provided the locations, size, and types from the Vegetation Class. It is also responsible for optimizing rendering by only providing information for visible vegetation. Visibility testing is accelerated by the quadtree organization of the vegetation database.

### **ParticleEffect Class**

The ParticleEffect Class provides a wrapper around the OSG particle system. The particle system implementation used by OSG modifies data in rendering the loop. This class separates the particle update code from the rendering implementation. The creation of a custom particle system in OSG is a lengthy and complicated process. This process is simplified through this interface, and custom particle effects can be created by inheriting from this class. Particle-based fire and smoke used in the prototype application implement the functionality of the ParticleEffect Class.

### **Simulation Class**

The Simulation Class is the root node of the simulation tree and is the boiler plate interface between the simulation visualization and the user interface. This class supplies the functionality outlined by the functional requirements. Elements are

added to the simulation to visualize its state. Every element added to the simulation is provided access to it. It acts as a centralized database for the state of the simulation. The defined interface is important because it must support different types of wildfire models without having to require changes to the graphical elements.

A purely generic simulation interface is ideal for flexibility but can lead to compromises in performance. Understanding the output provided from the different types of wildfire models is needed to achieve both flexibility and performance. A specific assumption is made about the types of data outputted from these models. Data are either grid based or vector based. This directly corresponds to the type of file formats these could be stored in Section 2.2. Understanding of this data is important to efficiently determine when visual objects such as vegetation and buildings are ignited. With grid-based data, pointers to objects can be efficiently placed in corresponding simulation data cells within the same location. Vector-based data can be used to determine which objects are on fire using point-polygon tests. This could be a slow process using a brute-force method of whether objects are within the fire front. The `SpatialDivisionTree` class optimizes visual objects into a spatial hierarchy accelerating this process (Figure 5.3). FARSITE simulation output is visualized using the `FARSITESimulation` implementation of the `Simulation Class`. FARSITE uses grid-based data. Flammable objects are added to the internal representation of the FARSITE grid data within the implementation of this class when attached to the simulation as shown in Figure 5.4.

### 5.5.2 Simulation Graph

The visualization system described in Section 5.5.1 is used to build simulation graphs. This system describes the scenes visual object as well as the the current state of the simulation (Figure 5.5). All of the connected graphical elements attached to the graph have access to the simulation (always the root node). These elements then update their visual appearance to the current state of the simulation. In this way, elements are modular, and can be attached to different types of simulations (e.g., data driven

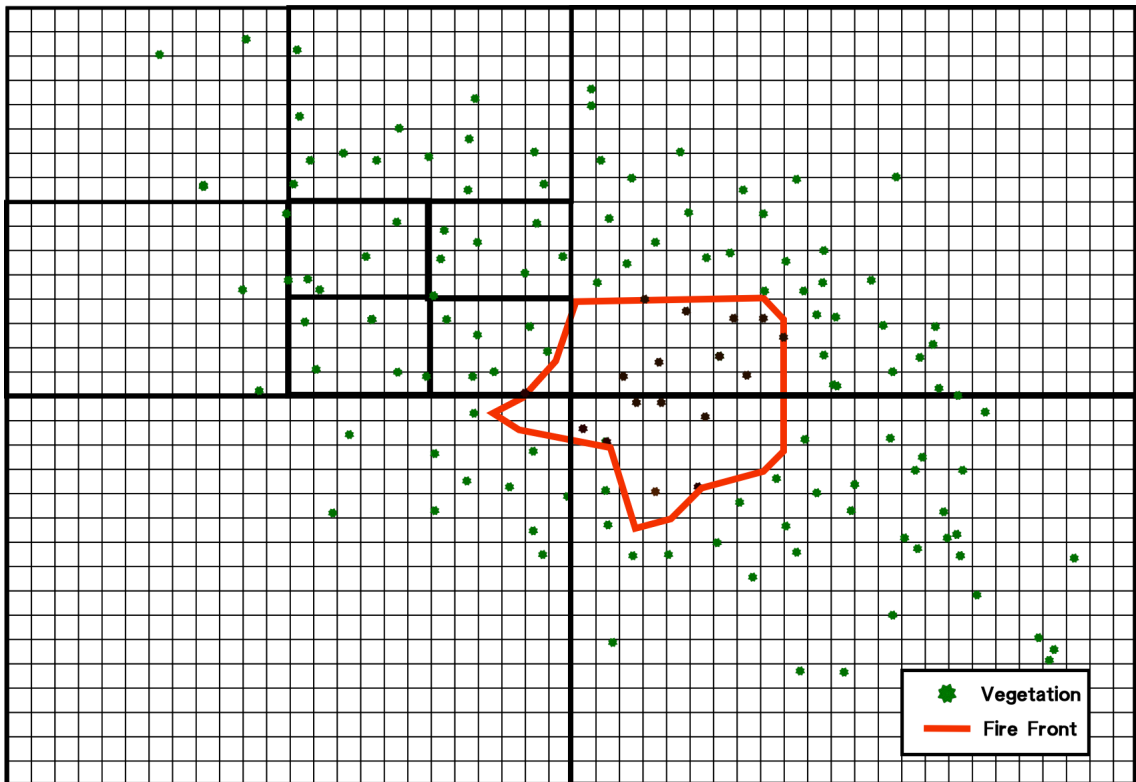


Figure 5.3: Spatial divisions minimize the number of polygon-point tests.

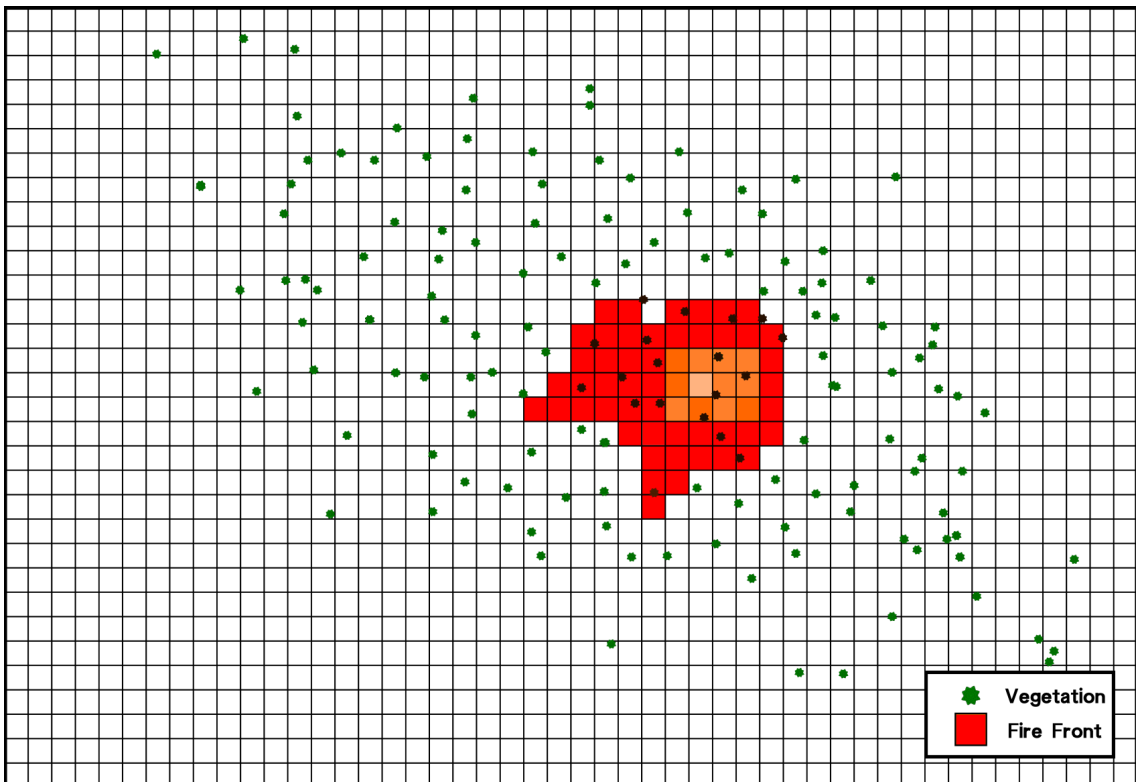


Figure 5.4: Vegetation placed in cells of grid data (e.g. FARSITE outputs).

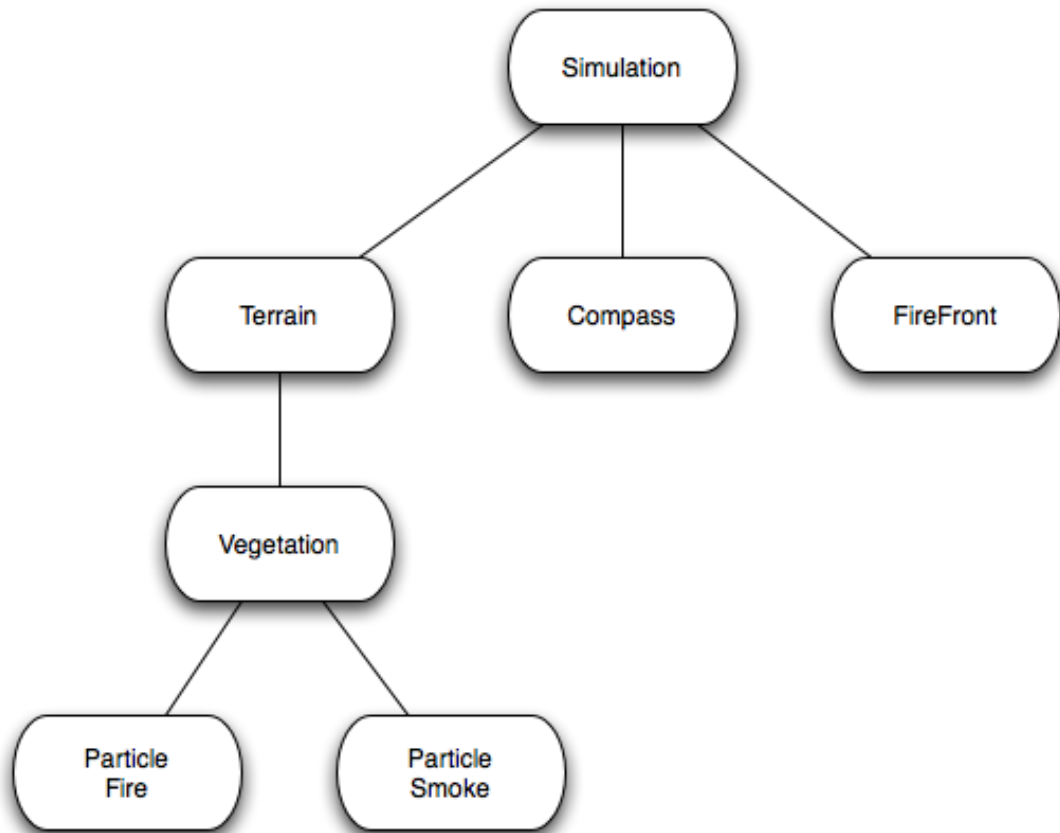


Figure 5.5: An example of a simulation graph used in a typical visualization.

or real-time).

### 5.5.3 Utility Class Functionality

The previous section described the main functionality for visualization of wildfire. The utility classes covered in this section are important for interfacing the system to different toolkits and data file types. Figure 5.6 describes the aggregation relationships between the main visualization system and supporting utility classes.

#### PassiveView Class

The PassiveView Class provides the interface between OSG and the VR or windowing toolkit (6.1). This class inherits from the `osg::SceneView` class responsible for setting up the OpenGL pipeline for rendering a scene graph. The PassiveView re-



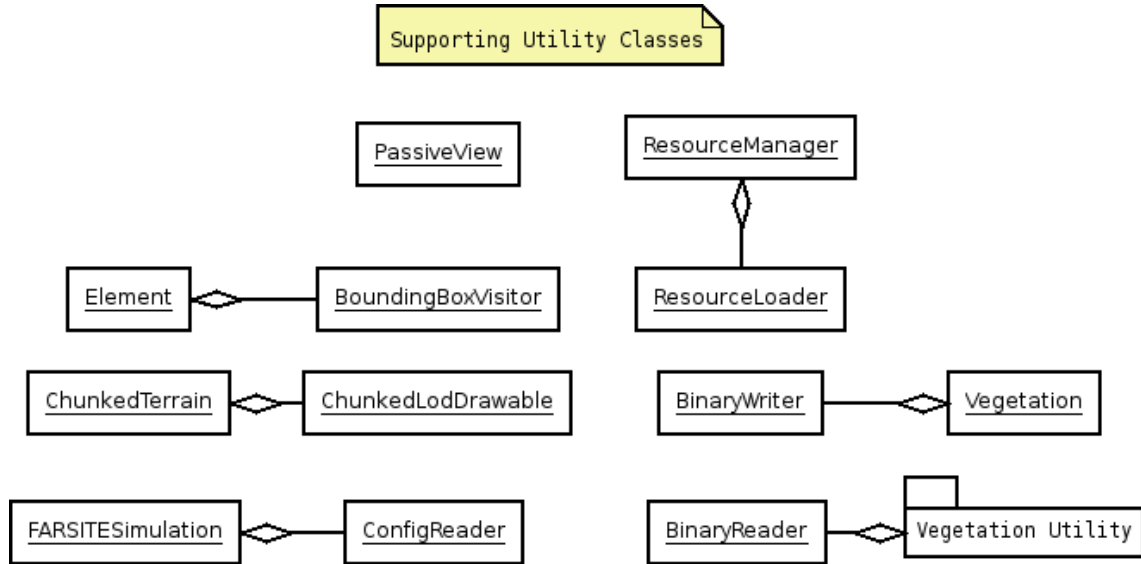


Figure 5.6: Utility classes for the VFire system.

moves buffer clearing, viewport adjustment and perspective matrix calculations from `osg::SceneView`.

### ConfigReader Class

The `ConfigReader` Class is used to read simulation configuration files. Configuration files provide the file location of landscape and scenario data used to construct a wildfire visualization. These files are also used to store simulation settings and user preferences.

### ResourceManager Class

The resource manager manages textures, models and simulation data. Constructing a wildfire simulation often results in reusing resources such as textures and models. Once in memory, these resources can be used by several different objects simultaneously reducing memory requirements and improving performance. The `ResourceManager` Class loads resources from file when necessary, and share a copy when already in memory. It is also responsible for removing the resource from memory when not in use anymore. The `ResourceLoader` Class provides an interface through which different types of resources are loaded. These resource loader implementations are used by the `ResourceManager` to load different types of resources.

# Chapter 6

## VFire Prototype

The first test and milestone of the VFire framework is the development of a prototype application used to visualize FARSITE outputs. The first step for building this framework and prototype application is the integration of OpenSceneGraph (OSG) and FreeVR. The VFire framework is built on top of OSG, and is heavily reliant on its features and optimizations. FreeVR provides the interface to the current immersive hardware setup. It should be noted that the VFire framework can be used with other VR toolkits, and porting requires minimal effort. The next phase includes the development of each graphical element used to visually reconstruct the landscape and wildfire scenario. Each graphical element of the scene strives to minimize its impact on the performance of the visualization. The goal was to section off as much work to the graphical processing unit as possible, and leave the CPU time available for visualization to other elements and simulation code. The following systems manage and minimize their use of system resources using level-of-detail algorithms specific to their data and visualization domain. The management of resources also allows the system to view landscapes that are larger than the available amount of system resources.

### 6.1 FreeVR and OpenSceneGraph Integration

VFire is built using open-source libraries including FreeVR and OSG libraries. The FreeVR virtual reality integration library is a cross-display VR library with built-

in interfaces for many input and output devices. It allows programmers to develop on a standard desktop machine, with inputs and display windows that simulate a projection or head-based immersive system. The OpenSceneGraph library is used to help with scene rendering. OSG allows 3D objects to be hierarchically organized within the environment, and also provides a system that optimizes the rendering through the use of various culling and sorting techniques.

FreeVR works well with lower level graphical rendering libraries including OpenGL. However, when interfacing a VR integration library with high level rendering libraries, there are many issues that need to be addressed. Four major issues are: 1) dealing with the perspective matrices, 2) memory allocation, 3) multi-processing, and 4) windowing and input device interfacing. A software interface between FreeVR and the SGI Performer scene-graph library already existed; therefore, the implementation of a similar FreeVR-OSG interface would be possible. Performer itself was avoided due to its closed-nature and expected lack of future support. OpenSG is open source, but did not support performance critical graphics hardware features.

OSG maintains control over the perspective matrices as with most scene graphs in order to handle culling and optimizations. FreeVR, as with most VR integration libraries, needs to set the perspective matrices for each screen such that it is current for the ever-changing position of the user. The first solution to address this conflict resulted in an implementation of a version of the FreeVR library that integrates OSG calls within the section that calculates perspective matrices. After making the perspective calculations, it adapts the matrix for the OSG coordinate system and copies it into the OSG context. Another implementation resulted in the development of a “passive” viewport derived from `osg::SceneView`, a wrapper around OSG’s rendering system. This implementation removes buffer clearing (color and depth), perspective matrix calculations and viewport control from OSG, and updates its internal state according to FreeVR. This improves on the first method through better access and locking of the scene graph, and does not require a special implementation of FreeVR.

The second major entanglement between FreeVR and OSG is memory allocation.

In order to render to multiple screens in independent processes or threads, along with a separate process for simulating the world's dynamics, FreeVR has memory allocation routines that provide access to data shared between all the processes. Because the internal memory allocation within OSG is opaque to the programmer, a version of FreeVR that uses Pthreads is used. The use of Pthreads causes all allocated memory to reside in a common shared space. Another option would be to overload C++ memory operations to use FreeVR's shared memory system; however, this would require modifying the entire OSG system, and make using the most current version a difficult task. There were also no performance benefits from using shared memory and full-weight processes.

OSG is somewhat based on the efforts of the Performer library. Specifically, it uses multiple traversals to handle different aspects of the rendering process to allow for multiprocessing and other efficiencies to be implemented. The three traversals implemented in OSG are the update (simulation) traversal, the cull traversal and the draw traversal. Ideally, traversals would operate in parallel as opposed to sequentially, allowing for course-grain parallelism. Performer double buffers the scene graph to allow this pipelining to work without corrupting the scene-graph. The update traversal modifies its copy of the scene graph while the rendering traversal renders the other copy. The rendering traversals are read-only operations allowing multiple viewports to rendering in parallel.

Two major differences between the OSG implementation and Performer are 1) OSG does not double-buffer the scene-graph or have an implicit data protection mechanism, requiring the update traversal to avoid making changes to the scene graph while a cull traversal is in progress, and 2) because many people contribute new node types to the open-source OSG, there is no strict enforcement of the rule preventing scene-graph modifications taking place outside the update traversal. Neither of these issues is typically a concern for desktop applications running on a single CPU system, but for multi-screen immersive systems, they are problematic.

To address these implementation issues, the scene-graph must avoid changing

shared data or maintain its own copy of the data when the multiple-renderings are taking place. FreeVR provides a semaphore-based locking/barrier system that was used to exclude writes to the scene-graph data during culling. Furthermore, specific node-types (e.g., the particle system node) used the culling traversal to make additional modifications to the scene-graph. The update code in these node-types has been separated and locked. The end result is a system that works satisfactorily, but the addition of each barrier results in lower frame rendering rates.

The fourth issue that needed to be addressed to make OSG work with FreeVR was the handling of window and input operations. In addition to perspective calculations, windowing and input interfaces are the core of what FreeVR provides. Therefore, little of the features OSG provides in these areas are required or used. Fortunately, the OpenSceneGraph system has separated these functionalities into the companion OpenProducer library. Due to this separation, the job of interfacing with FreeVR was made easier since FreeVR was allowed to handle this task without the need of going through the extra layer.

## 6.2 Terrain Implementation

In the first implementation of the terrain, the Geometrical Mipmapping [59] algorithm was used. This enabled the system to visualize terrain datasets much larger than brute-force methods; however, it required that the entire dataset be loaded into memory. This algorithm would result in a complex memory management system. Chunked LOD, although reducing CPU usage over previous methods, still could be improved. Thatcher Ulrich's Chunked LOD algorithm [49] offers several advantages including higher triangle throughput, low CPU usage and implicit memory management. This method, unlike Geometrical Mipmapping, requires offline tessellation of the elevation data. Chunked LOD uses two quad tree structured files at runtime. The first contains the geometrical description of the terrain (chunk file, .chu), and the other contains data used to texture the terrain (texture quad tree file, .tqt). The quad tree structure represents higher detail further down the tree, the highest level

of detail resides in the leaves of this tree. This structure allows terrains larger than the available memory (Figure 6.1) to be visualized because lower detail data near the top of the tree requires less memory space.

The Chunked LOD implementation is a port of Thatcher Ulirch's algorithm to work in OSG, originally created by Vladimir Vukicevic [53]. Several modifications were made to increase performance, portability and adapted to work on a VR system. Geometry for terrain tiles is tessellated offline, and does not change on a per frame basis, making it a great candidate to be placed in high performance video memory. Geometry was placed on the graphics hardware using OpenGL Vertex Buffer Objects [31, 46] with the `GL_STATIC_DRAW` flag. Portability was increased by porting the Nvidia Cg [30] geomorphing shader to OpenGL Shading Language (GLSL) [39] prompted by the unavailability of an Nvidia Cg implementation on the available SGI Prism visualization system. Chunked LOD is a view-dependent algorithm that requires data changes while rendering. This is a problem on a multi-screen system because the perspective matrices are different for each screen. This problem was overcome by providing each rendering process with its own copy of the terrain. Visual inconsistencies are not noticeable because Chunked LOD does not require a symmetrical frustum, and uses section-based refinement of the terrain mesh. The first copy of the terrain is used for collisions. The geometrical mipmapping algorithm also required separate data per screen because of the view-dependent refinement of the mesh. This is simplified through an OSG mechanism which provides a per context data structure for the management of per screen data.

### 6.3 Vegetation Implementation

The vegetation system is responsible for placement, rendering and updating the burned status of vegetation. Currently, there are two different placement systems, each having its advantages and disadvantages.

The first method places vegetation at runtime using the fuel type data, and uses a ray casting method to determine its placement on the terrain. To maintain real-time

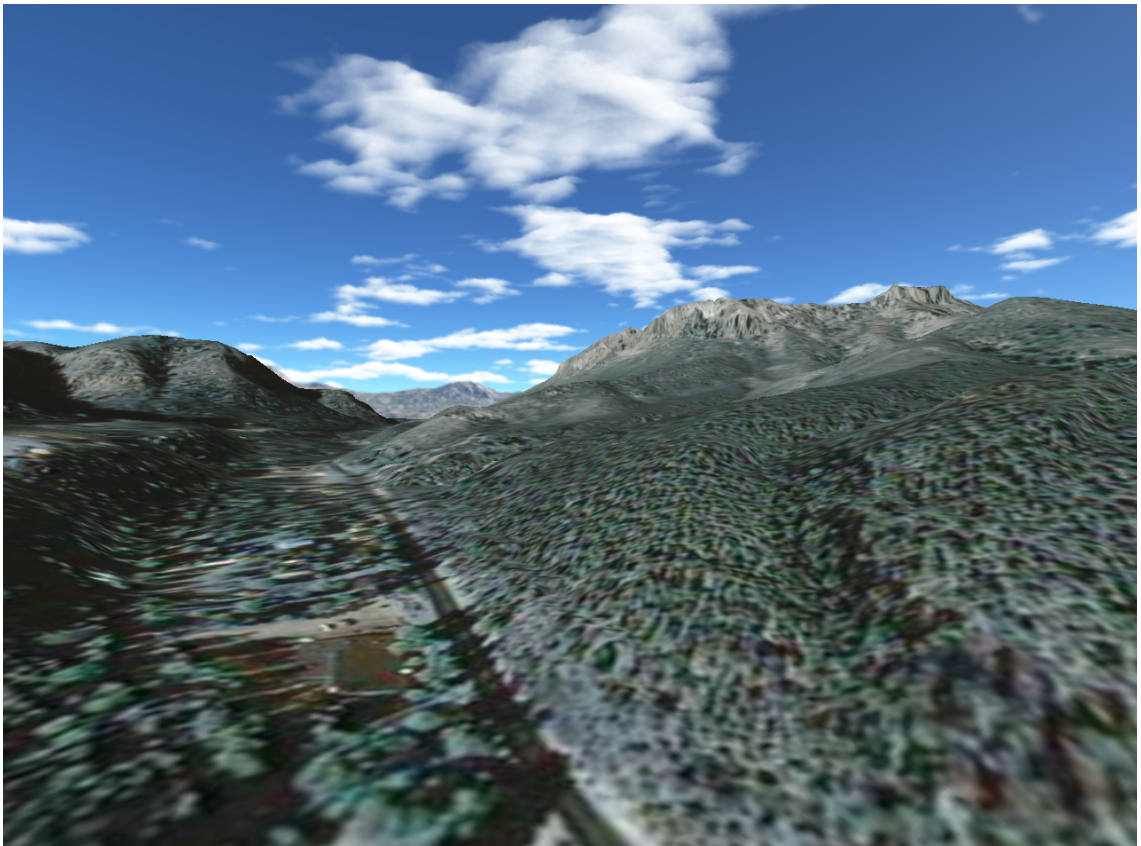


Figure 6.1: 700 MB, 1-meter satellite image terrain visualization in VFire system.

frame rates, the trees at sufficient distances are not rendered, which is a disadvantage of having visual artifacts as the vegetation appears and disappears. This could be minimized by fading vegetation from complete transparency to being fully opaque. This method is useful for rendering smaller areas because all the vegetation can be loaded in memory at once. This simplifies the algorithms to determine whether vegetation is under combustion. Ray casting against terrain at runtime leads to vegetation that are not always correctly placed on the terrain. To correct these visual inconsistencies, vegetation must be placed according to the highest level-of-detail on the terrain. This is not feasible at runtime because the Chunked LOD algorithm does not force the loading of higher level-of-detail tiles in order to maintain real-time frame rates. The highest level-of-detail terrain data can be requested, but the loading thread cannot be guaranteed to return the data promptly.

The second method also uses the fuel type data to place vegetation, but places them offline and uses the chunk file to place them correctly on the terrain. Because the utility is run offline, it can collide with the highest level-of-detail data. The vegetation utility uses the fuel load data input from FARSITE, and an expert's knowledge of the location to determine what types of vegetation are native to a location and their positions. The vegetation position, type and other values are placed in a file organized as a position quad tree. This has several advantages. Each level of the quad tree can represent different levels of detail of an area, meaning that each level further down the tree contains more trees than the previous levels. The leaves of the tree contain the highest number of trees. This method allows for the possibility of visualizing much larger forest; however, it comes at the cost of easily finding burnable locations. A formal heuristic was not developed to determine the transitions between levels of detail. The current implementation only displays trees in the leaves of the quad-tree at the highest level of detail, and distant trees are simply not displayed. This method reduces the overall amount of trees that can be used to represent a forest; however, it does not suffer from the visual artifacts of vegetation appearing from nowhere. The quad-tree organization of the file allows vegetation to be paged in from file if the



current number of trees is too large for the memory resources of the hardware.

It is possible for scenarios to have hundreds of thousands of pieces of vegetation (Figure 6.2). Rendering this much vegetation is a non-trivial problem. Vegetation is grouped into GPU-friendly sized batches and rendered together. The size of batches can vary widely based on the type of hardware a system has. Rendering too many groups, even if they contain very few polygons, will slow the frame rate considerably. This increases the amount of vegetation able to be rendered, but has the limit of only being able to process vegetation in groups. The vegetation is also organized into a quad tree data structure to speed up culling. Pixel error is used to determine when trees should not be displayed anymore. In general this is when the size of the rendered vegetation is smaller than a few pixels of screen space. Processing vegetation individually allows the system to change the appearance of vegetation as it burned, which results in slow rendering speeds. A compromise is to use instancing [7] to draw and process vegetation nearer to the viewer and process distant vegetation using groups.

## 6.4 Wildfire Implementation

Visualization of the wildfire is driven directly from the FARSITE simulation data using particle-based fire and smoke. Sprite-based particle systems can accurately and realistically represent fire with very few particles, and makes it possible to render fires at the scale of wildfire. This is the reason VFire implements particle-based fire and smoke as shown in Figure 6.3. The visual complexity of particle systems can be reduced progressively as the distance increases from the user; however, the physical properties of all particles must be calculated. Such a system has not currently been implemented, but is necessary to reduce the number of rendered particles. This is often a bottleneck because the amount of sprites necessary to render a wildfire quickly reaches the maximum fillrate capacity of the GPU.

Currently, only the time of arrival data is being used; a cell is ignited when the simulation time is equal or greater than the time for the cell. The method used before

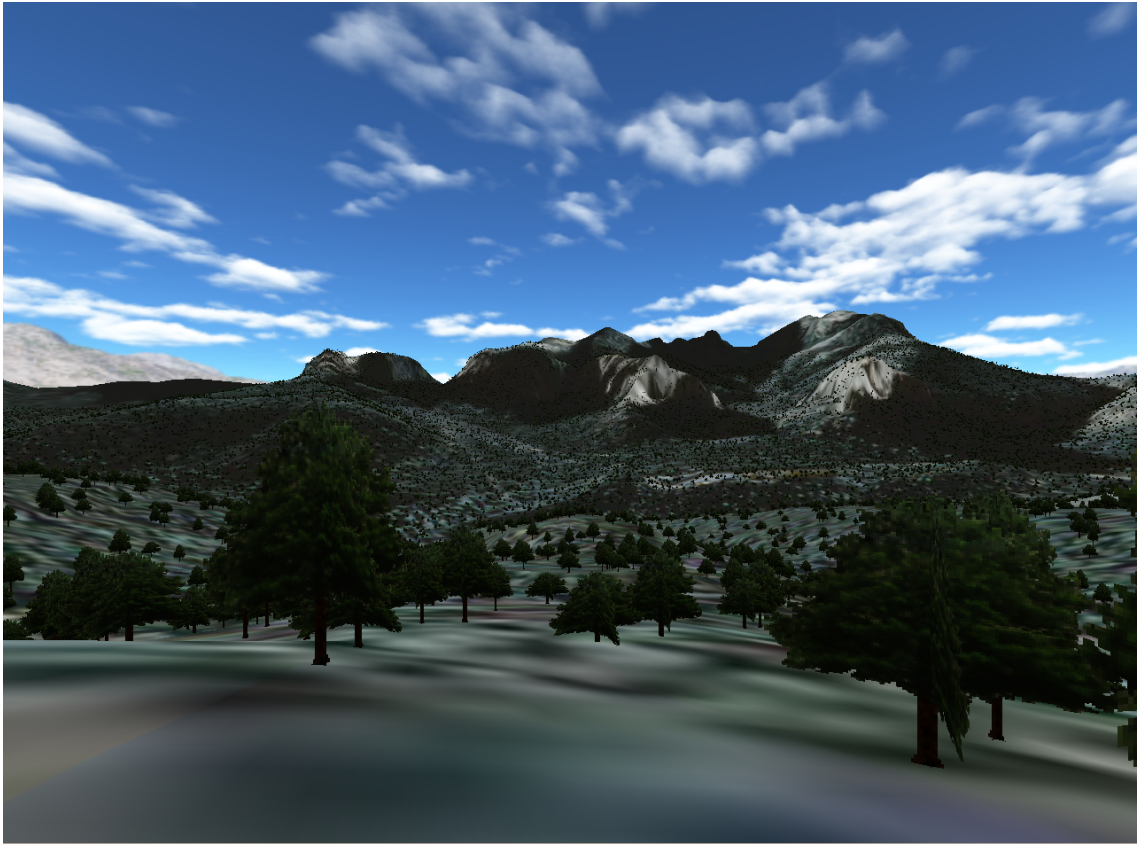


Figure 6.2: 150,000 billboard trees visualized in VFire system.

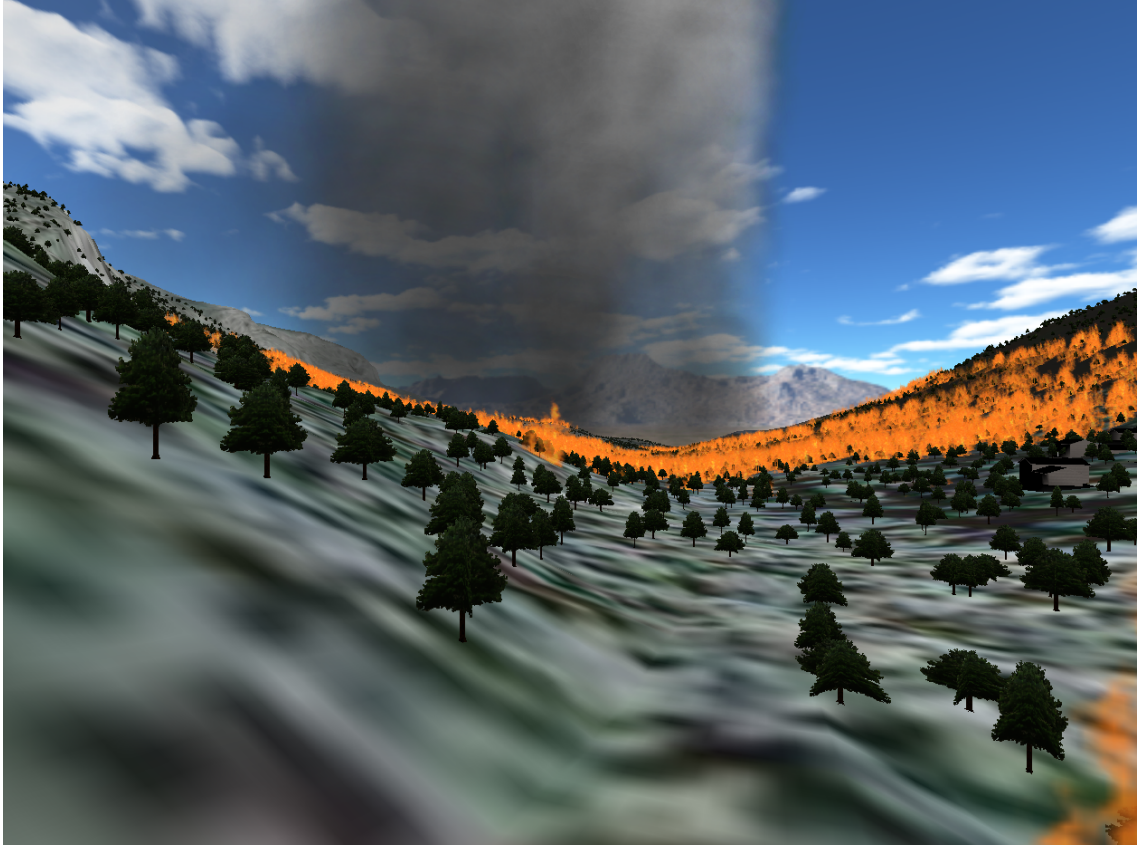


Figure 6.3: Sprite-based fire used in VFire system.

spread a convex hull around the cells under combustion at a given time. However, this had several limitations. It did not precisely describe the detailed surface of the fire front, and could not properly show spot fires. Time of arrival is the most crucial because it allows the visualization of the wildfire spread. Other outputs are crucial to increasing the accuracy of the visualization, and present all available information from the simulation. Spread direction and rate of spread can be used to estimate the spread of fire across vegetation within the space of the cell without increasing the resolution of the FARSITE simulation. Fireline intensity, flame length and crown-no crown data can be used to control the visual appearance of fire, and provide further data to the physics calculations for individual fires.

## 6.5 Visualization Hardware

VFire currently runs on our immersive visualization hardware that includes both a four-screen CAVE<sup>TM</sup>-like Fakespace FLEX<sup>TM</sup> [11] display and a single-screen Visbox-P1<sup>TM</sup> [51]. The FLEX<sup>TM</sup> display is driven by either a SGI Prism<sup>TM</sup> or quad AMD Opterons<sup>TM</sup> using four outputs of an NVIDIA Quadro® FX 4500 X2, both with four active-stereo capable graphics channels. The Visbox display is driven by a dual Opteron<sup>TM</sup> using both outputs of an nVidia GeForce 6800GT card to drive two projectors producing a passive stereo output.

Our FLEX uses an Intersense IS-900<sup>TM</sup> with wireless head and wand units for position [21]. The Visbox-P1 uses two forms of tracking. The first form is an Ascension Flock of Birds® which is used to track a standard multi-input gamepad controller as part of the hand interface. A proprietary infrared video system wirelessly tracks the user's head position using image processing.

## 6.6 Kyle Canyon Scenario

The prototype system was developed using terrain and fire simulation data of a wild-fire scenario in Kyle Canyon, Nevada (Figure 6.4). The scenario is constructed from remote sensed data including digital elevation model data, a satellite image, fuel load data and FARSITE outputs. Kyle Canyon, Nevada is constructed from a 981x728 cell DEM file with 10 meter spacing between sampling values. A 1-meter resolution satellite photograph is overlaid on this nearly 10km x 10km region. The FARSITE simulation operates on a grid of 287x203 cells that are 30 meters by 30 meters. This cell size corresponds to the fire load (i.e. vegetation) data that is provided as an input to both FARSITE and the VFire visualization systems. Provided adequate data for a location, the application can be easily extended to other locations and scenarios.

A typical wildfire will cover less than 100 acres (0.4km<sup>2</sup>), easily covered by our Kyle Canyon data. The Kyle Canyon scenario covers a 10km by 10km test range, approximately 25 million acres, which is a good experimental size demonstrating the

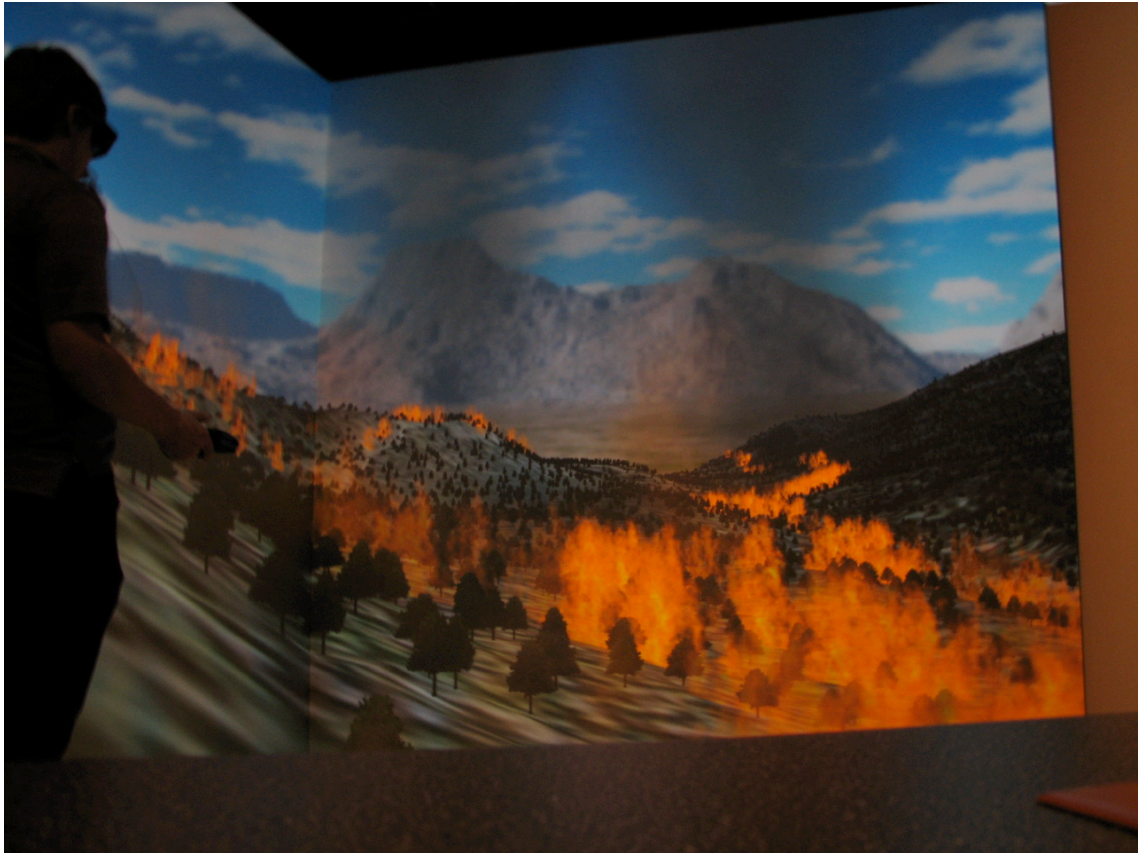


Figure 6.4: Wildfire spreads across the terrain engulfing vegetation.

scalability of the implementation. The terrain and vegetation system can easily cover scenarios of this size; however, the wildfire system is still the limiting factor.

# Chapter 7

## Conclusions and Future Work

### 7.1 Conclusion

Wildfire can spread over large amounts of area producing equally large datasets. This thesis has introduced methods that can manage and visualize these datasets interactively. Abstraction of the rendering and simulation computation results in modular code that allows for the integration and optimization of different implementations for rendering elements of a scene. They also take advantage of different hardware configurations used to drive virtual reality systems.

VFire adds considerable effort to previous related endeavors. Other works are more specialized and do not focus on determining interfaces to allow for code reuse and different wildfire simulation types. The VFire project allows for reconstruction of realistic landscapes from remote sensing data as with [61] and [28]. However, the VFire project encapsulates this similar functionality into a framework so that the research in this project can be expanded and reused with other similar projects and support different computational wildfire models. It also addresses the complexities of virtual reality environments to allow wildfire visualization to gain from the benefits of these systems.

Visualization of computational wildfire models has many applications. A prototype visualization system has been implemented using the VFire framework based on requirements and solution decisions described in the previous chapters. Initial response from state wildfire agencies has been positive. The result of which is the

further development and refinement of the VFire framework supporting the requirements of these agencies.

## 7.2 Future Work

Developing an immersive and realistic wildfire visualization is a non-trivial task that encapsulates many difficult problems across many disciplines in computer science and wildfire analysis, from algorithmic optimizations to computer graphics and computer vision. This section is for the future developers of the VFire project. The framework has been successfully used to develop an introductory wildfire visualization tool. A main goal of this project was to develop a framework for the future advancement of the project.

### Real-time Wildfire Simulations

The central focus of future work is improving the computational wildfire model and its visualization. Integration of more FARSITE model outputs would be a logical stepping stone and would significantly improve the visualization. FARSITE simulations require several hours making it impractical for adjusting simulation parameters during visualization sessions. Integration of a real-time wildfire simulation would be advantageous for using VFire for training and burning more predictable prescribed fires. This could include the development of a scaleable, parallelized version of the FARSITE model, or integration of the Los Alamos work using the FARSITE model [56]. Users could then adjust or add ignition points, change weather parameters, and change fuel data during a simulation and receive immediate feedback. Integration of such a system would ultimately result in a more comprehensive user interface.

### Interface Improvements

The integration of a real-time wildfire model would significantly change the requirements of the project. Refinement of the requirements should be a formal process with significant input from wildfire agencies. Currently, no graphical interface has been developed because the requirements are somewhat limited and the focus has



been on the development of supporting technologies. VRUI [25] is a VR toolkit that also provides a comprehensive toolkit for developing user interfaces for use with immersive applications. There are significant advantages for using immersive systems to visualize wildfire computational models; however, a formal study has not been developed to support a quantitative or qualitative measure of these advantages. A usability study would be a fundamental element for testing the requirements, but also discovering new and beneficial functionality.

### **Weather effects**

Visualizing all aspects which affect wildfire is crucial to accurate and realistic visualization. Weather is a large factor when determining how a wildfire will burn. Visualizing this behavior would undoubtedly increase the immersiveness of the application. Realistic visualization of this phenomena would be beneficial for training purposes. Less realistic visualization such as vector fields for wind or other graphical indicators would be useful for data analysis and model validation.

### **Vegetation Placement**

Accurately recreating the vegetation ecology is essential for wildfire visualization because fuel (vegetation) is a crucial element in determining how a wildfire will burn. Remote sensed data, more specifically satellite images are used to determine land cover and landscape ecology. These studies are usually very coarse scale classifying the landscape by  $1\text{km}^2$  patches. As GIS and remote sensing technologies advance, higher resolution data are becoming more common and available including, 0.5m resolution satellite images. This advancement with the integration of fuzzy systems and computer vision algorithms could be used for accurate placement and type determination of vegetation. These data could also be used to reconstruct urban environments by correctly reconstructing and placing buildings. Another solution for placing vegetation could use highly available Forest Inventory Analysis data [55]. Urban areas could also be constructed using a visual editor and artist recreation of buildings. An automated method holds many advantages and would be useful for larger areas.



### **Fire and smoke model**

Visualization of fire and smoke is the current limiting factor for the performance and size of the wildfire visualization. Improving the scalability of the fire and smoke visualization is necessary for the integration of more computationally expensive physically based simulation of individual fires. Very little work has been spent on scaleable rendering of realistic real-time fire and smoke with applications large enough for the scale required for wildfires. Current work only considers single or small scale fires over small objects. The complexity of realistically visualizing a single fire can quickly use the entirety of a computer system's resources. The creation of a specialized LOD system will further optimize the fire rendering system to support larger wildfires.

### **Burning vegetation and buildings**

FARSITE data controls the spread of the fire front across the entirety of the landscape but does not control the propagation of fire on individual pieces of vegetation or buildings. Currently, fires are assumed to be crown fires and smaller more precise behavior is not considered. Computationally this is an expensive problem even for a single small object [26, 63]. A method of fire propagation scalable enough for wildfire would be need to be developed.

# Bibliography

- [1] Pankaj K. Agarwal and Pavan K. Desikan. An efficient algorithm for terrain simplification. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 139–147, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.
- [2] James Ahrens, Patrick McCormick, James Bossert, Jon Reisner, and Judith Winterkamp. Wildfire visualization (case study). In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 451–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.
- [3] Jim Arlow and Ila Neustadt. *UML and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley, 2002.
- [4] Jitendra A. Borse and David F. McAllister. Real-time image-based rendering for stereo views of vegetation. In *Proceedings Electronic Imaging '02*, pages 292–299, 2002.
- [5] Richard Bukowski and Carlo Sequin. Interactive simulation of fire in virtual building environments. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 35–44, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
- [6] Mario Caetano and Teresa Santos. Updating land cover maps with satellite images. In *Geoscience and Remote Sensing Symposium, 2001. IGARSS '01. IEEE 2001 International*, pages 979–981. IEEE Press, July 2001.
- [7] Nvidia Corporation. Gls1 pseudo-instancing. Technical report, Nvidia, Oct 2004.
- [8] Philippe Decaudin and Fabrice Neyret. Rendering forest scenes in real-time. In *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, pages 93–102, June 2004.
- [9] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain: real-time optimally adapting meshes. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 81–88, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.
- [10] Albert Endres and Dieter Rombach. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories*. Pearson Addison-Wesley, 2003.

- [11] Fakespace Systems Inc. Fakespace Systems Inc. CAVE®. <http://www.fakespace.com/flexReflex.htm> (Accessed April 01, 2007).
- [12] Bryan E. Feldman, James F. O'Brien, and Okan Arikan. Animating suspended particle explosions. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 708–715, New York, NY, USA, 2003. ACM Press.
- [13] Mark A. Finney. Farsite: Fire area simulator-model development and evaluation. In *Res. Pap. RMRS-RP-4*. U.S. Department of Agriculture, Forest Service, 1998.
- [14] Open Source Geospatial Foundation. Geospatial data abstraction library. <http://www.gdal.org/> (Accessed on March, 29 2007).
- [15] Alfred R. Fuller, Hari Krishnan, Karim Mahrous, Bernd Hamann, and Kenneth I. Joy. Real-time procedural volumetric fire. In *Proceedings of Symposium on Interactive 3D Graphics and Games (i3D) '07*, New York, NY, USA, 2007. ACM Press.
- [16] M. Garland and P. Heckbert. Fast triangular approximation of terrains and height fields. In *SIGGRAPH '97 Course Notes*.
- [17] Jayesh Govindarajan, Matthew Ward, and Jonathan Barnett. Visualizing simulated room fires (case study). In *VIS '99: Proceedings of the conference on Visualization '99*, pages 475–478, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.
- [18] Frederick C. Harris, Michael A. Penick, Grant M. Kelly, Juan C. Quiroz, Sergiu M. Dascalu, and Brian T. Westphal. V-fire: Virtual fire in realistic environments. In *The 2005 International Conference on Software Engineering Research and Practice (SERP '05)*, pages 73–79, 2005.
- [19] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, New York, NY, USA, 1996. ACM Press.
- [20] Infiniscape. VRJuggler. <http://www.vrjuggler.org/> (Accessed March 29, 2007).
- [21] Intersense. IS-900 Wireless Tracking Modules. <http://www.intersense.com/products/prec/is900/wireless.htm> (Accessed April 01, 2007).
- [22] Michael F. Jasinski. Estimation of subpixel vegetation density of natural regions using satellite multispectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 34(3):804–813, May 1996.
- [23] Morgan Johansson. Gremm: a real-time graphics engine for forest rendering. Master's thesis, Umea University, Department of Computing Science, February 2003.
- [24] Tazama U. St. Julien and Chris D. Shaw. Firefighter command training virtual environment. In *TAPIA '03: Proceedings of the 2003 conference on Diversity in computing*, pages 30–33, New York, NY, USA, 2003. ACM Press.

- [25] Oliver Kreylos. VRUI. <http://graphics.cs.ucdavis.edu/~okreylos/ResDev/Vrui/index.html> (Accessed March 29, 2007).
- [26] Haeyoung Lee, Laehyun Kim, Mark Meyer, and Mathieu Desbrun. Meshes on fire. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 75–84, New York, NY, USA, 2001. Springer-Verlag New York, Inc.
- [27] Stephan Mantler and Stefan Jeschke. Interactive landscape visualization using gpu ray casting. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 117–126, New York, NY, USA, 2006. ACM Press.
- [28] Patrick S. McCormick and James P. Ahrens. Visualization of wildfire simulations. *IEEE Comput. Graph. Appl.*, 18(2):17–19, 1998.
- [29] H. Nguyen. *GPU Gems*. Addison-Wesley, 1st edition, 2004.
- [30] Nvidia Corporation. Cg. [http://developer.nvidia.com/page/cg\\_main.html](http://developer.nvidia.com/page/cg_main.html) (Accessed April 01, 2007).
- [31] Nvidia Corporation. Using Vertex Buffer Objects. [http://developer.nvidia.com/object/using\\_VBOs.html](http://developer.nvidia.com/object/using_VBOs.html) (Accessed April 01, 2007).
- [32] Open Source Community. OpenSceneGraph. <http://www.openscenegraph.org/> (Accessed March 8, 2007).
- [33] OpenSG Community. OpenSG. <http://opensg.vrsourc.org/> (Accessed March 20, 2007).
- [34] Michael A. Penick, Roger Hoang, Frederick C. Harris, Sergiu M. Dascalu, Timothy J. Brown, and William R. Sherman. Managing data and computational complexity for immersive wildfire visualization. In *The International Conference High Performance Computing and Simulation (HPCS'07)*, 2007.
- [35] Alex Reche-Martinez, Ignacio Martin, and George Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Trans. Graph.*, 23(3):720–727, 2004.
- [36] William T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 359–375, New York, NY, USA, 1983. ACM Press.
- [37] Dirk Reiners, Gerrit Vo, and Johannes Behr. Opensg: Basic concepts. In 1. OpenSG Symposium OpenSG 2002, 2002.
- [38] John Rohlf and James Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994. ACM Press.

- [39] Randi J. Rost. *OpenGL Shading Language*. Pearson Education Inc., 2 edition, 2006.
- [40] Marcus Roth, Gerrit Voß, and Dirk Reiners. Multi-threading and clustering for scene graph systems. *Computers & Graphics*, 28(1):63–66, February 2004.
- [41] Joe H. Scott and Robert E. Burgan. Standard fire behavior fuel models: A comprehensive set for use with rothermel’s surface fire spread model. Technical report, USDA Forest Service, Rocky Mountain Research Station, June 2005.
- [42] William R. Sherman. Freevr. <http://www.freevr.org/> (Accessed March 8, 2007).
- [43] William R. Sherman and Alan B. Craig. *Understanding Virtual Reality*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2003.
- [44] William R. Sherman, Michael A. Penick, Simon Su, Timothy J. Brown, and Jr. Frederick C. Harris. Vrfire: an immersive visualization experience for wildfire spread analysis. In *Proceedings of IEEE VR 2007*, 2007.
- [45] Ian Sommerville. *Software Engineering*. Addison-Wesley, 7 edition, 2004.
- [46] Standard Performance Evaluation Corporation. Vertex Buffer Objects (VBOs). [http://www.spec.org/gpc/opc.static/vbo\\_whitepaper.html](http://www.spec.org/gpc/opc.static/vbo_whitepaper.html) (Accessed April 01, 2007).
- [47] Lszlo Szirmay-Kalos Gabor Szijarto Tamas Umenhoffer. Spherical billboards and their application to rendering explosions. In *Proceedings of the 2006 Conference on Graphics Interface*, pages 57–64. ACM Press, 2006.
- [48] David L. Tate, Linda Sibert, and Tony King. Virtual reality: Using virtual environments to train firefighters. *IEEE Computer Graphics and Applications*, 17(6):23–29, /1997.
- [49] Thatcher Ulrich. Chunked lod:rendering massive terrains using chunked level of detail control. In *SIGGRAPH ’02: Proceedings of the 24th annual conference on Computer graphics and interactive techniques Notes*. ACM Press, 2002.
- [50] USDA Forest Service. FARSITE. <http://www.fire.org/index.php?option=content&task=category&sectionid=2&id=8&Itemid=27/> (Accessed March 20, 2007).
- [51] Visbox Inc. Welcome to Visbox, Inc. <http://www.visbox.com/> (Accessed April 01, 2007).
- [52] VRCO. CAVELib. <http://www.vrco.com/CAVELib/OverviewCAVELib.html> (Accessed April 9, 2007).
- [53] Vladimir Vukicevic. osgChunkedLod. <http://www.vlad1.com/vladimir/projects/osg/> (Accessed April 01, 2007).
- [54] Xianli Wang, Bo Song, Jiquan Chen, Thomas R. Crow, and Jacob J. LaCroix. Challenges in visualizing forests and landscapes. *Journal of Forest*, 104:316–319, September 2006.

- [55] Xianli Wang, Bo Song, Jiquan Chen, Daolan Zheng, and Thomas R. Crow. Visualizing forest landscapes using public data sources. *Landscape and Urban Planning*, 75:111–124, February 2006.
- [56] M. Diana Webb and Randy G. Balice. A real-time wildfire model for l. *International Journal Technology Transfer and Commercialisation*, 3(2):226–242, 2004.
- [57] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In *SIG-GRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1995. ACM Press.
- [58] Xiaoming Wei, Wei Li, Klaus Mueller, and Arie Kaufman. Simulating fire with texture splats. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 227–235, Washington, DC, USA, 2002. IEEE Computer Society.
- [59] Willem H. de Boer. Fast Terrain Rendering Using Geometrical MipMapping. [http://www.flipcode.com/articles/article\\_geomipmaps.shtml](http://www.flipcode.com/articles/article_geomipmaps.shtml) (Accessed March 20, 2007), 2000.
- [60] Qizhi Yu, Chongcheng Chen, Zhigeng Pan, and Tianhe Chi. Interactive 3-d visualization of real forests. In *Proceedings of 13th International Conference on Artificial Reality and Telexistence*, pages 263–269, 2003.
- [61] Qizhi Yu, Chongcheng Chen, Zhigeng Pan, and Jianwei Li. A gis-based forest visual simulation system. In *ICIG '04: Proceedings of the Third International Conference on Image and Graphics (ICIG'04)*, pages 410–413, Washington, DC, USA, 2004. IEEE Computer Society.
- [62] Christopher Zach. Integration of geomorphing into level of detail management for realtime rendering. In *SCCG '02: Proceedings of the 18th spring conference on Computer graphics*, pages 115–122, New York, NY, USA, 2002. ACM Press.
- [63] Ye Zhao, Xiaoming Wei, Zhe Fan, Arie Kaufman, and Hong Qin. Voxels on fire. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 36, Washington, DC, USA, 2003. IEEE Computer Society.