V-FIRE: Virtual Fire in Realistic Environments

**Frederick C. Harris Jr., Michael A. Penick, Grant M. Kelly,
Juan C. Quiroz, Sergiu M. Dascalu, Brian T. Westphal**

Department of Computer Science and Engineering
University of Nevada, Reno
Reno, NV 89557, USA
{fredh, penick, gkelly, quiroz, dascalus, westphal}@cse.unr.edu

Abstract. *V-FIRE is a 3D fire simulation and visualization software tool that allows users to harness and observe fire evolution and fire-related processes in a controlled virtual environment. V-FIRE is an example of a complex project developed by a small team using a rapid development process and an adapted form of the UML notation. This paper presents the details of the process followed, including requirements specification, software architecture, medium and low-level design, and user interface design. Both the process and the main elements of the architectural solution can be reused in similar 3D simulation projects. A discussion of development challenges and pointers to future work are also provided.*

Keywords virtual environment, fire simulation, requirements specification, software design, prototype user interface.

1. Introduction

Fire is a phenomenon that humans have studied for ages. Yet its dynamics and properties are not fully understood [1]. Thus, fire modeling is a task that has not been achieved to great precision. Most importantly, the only way scientists have been able to study wildfires is by collecting data from actual fire disasters, which this poses many problems. Firstly, a wildfire is a dangerous phenomenon. Due to the fact that fires create their own weather systems, being anywhere near a fire is unsafe. Furthermore, when a wildfire breaks out, it causes a large amount of damage, both to the environment and the community. Thus, the idea of having a fire burning freely just for observation purposes is preposterous. Consequently, the demand for tools which can aid in the studying of fires has increased due to technological advances in computer graphics and computer modeling. The increasing number of firefighter casualties and the staggering costs of damages due to wildfires [2, 3, 4] show the need for a tool and remedy for this malady.

Computer modeling of fires is an effective alternative for scientists to experiment and study the patterns of wildfires. Mathematical models of fires exist, but they do not correctly reflect the true behavior of fire. Fire is such a dynamic system, that a true model representation of it is hard to create. There are many factors that need to be taken into consideration. Most models rely on simplifications and assumptions in order to make the system solvable [1, 5, 6, 7]. Furthermore, the results of such models are tables of data, possibly graphs. The visualization

of the model is even more difficult [8, 9]. The behavior of a wildfire can change drastically from a slight variation of the atmospheric pressure or of the speed and direction of the wind.

We propose through V-FIRE a visual approach to fire research – a software application developed following an adapted version of the traditional waterfall software process. As detailed in [10], we call this process "streamlined" because it consists of the most necessary (essential) development phases and activities. With the exception of the final phase, in which a complete product is delivered (that is, all its items are mandatory – to satisfy the top level project requirements), all other phases produce an essential set of artifacts (deliverables), which consists of a subset of mandatory items and a subset of sample non-mandatory items.

In terms of artifacts created during the development process, the decision we made was to require completeness for those items deemed essential (functional and non-functional requirements, system architecture, and implementation of first level priority requirements). Elements such as second level priority requirements were only partially completed, thus allowing for the construction of an operational prototype in the rather short timeframe of an academic semester (four months).

The remainder of this paper is organized as follows: Section 2 presents background information on the project, Section 3 shows details of the system's requirements, Section 4 describes V-FIRE's use case diagram and use cases, Section 5 focuses on the high-level design of the system, Section 6 provides information on V-FIRE's detailed design, Section 7 describes elements of the system's implementation and shows project results, Section 8 discusses ideas and directions for future work, and Section 9 presents our conclusions.

2. Background

V-FIRE is a 3D fire simulation and visualization tool that is intended to allow users to harness and observe a fire within a controlled environment. The system has been designed to model a wildfire as realistically as possible with the use of marketable graphics, an efficient physics model, and a mathematically based spreading algorithm. In addition, users will also be able to visualize the interaction of fire with other objects such as smoke, vegetation, and buildings. Moreover, as an empirical tool, V-FIRE is also intended to provide the user with the ability of multiple view points for the main camera, such as and aerial view and/or full immersion [11].

The long term goal of V-FIRE is to create real-time, marketable-quality graphics for fire visualization in a Cave Automatic Virtual Environment (CAVE). A CAVE provides a full-immersion experience for its users [12]. Thus, the integration of V-FIRE with a CAVE would create a full 3D simulation in which users would be able to physically interact with a wildfire environment.

Wildfires are complex and dynamic phenomena. As such, the fire simulation with computer graphics poses a great challenge. A fire has a chaotic nature. Consequently, the modeling of fire with computer graphics must introduce a random factor into the algorithms used. Also, the transition from 2D graphics to 3D graphics is a challenging endeavor that needs to be researched further.

3. Requirements Specification

Throughout the tools' development, the UML notation [13] [14] was used to create software models and followed software engineering techniques and guidelines from [15] and [16] (with certain adaptations, as detailed in [10], [17] and [18]). The functional and non-functional requirements for the V-FIRE system are provided in the following two subsections. The requirements define the "reference points" for the remainder of the system's development. The success of a software project is often determined by the completion of requirements. A crucial time saving step is to have a firm understanding of the customers definition of the requirements before any type of design begins.

3.1 Functional Requirements

Functional requirements specify the basic functions that the system must deliver to the user [15] [16] [17]. In V-FIRE, this includes details of the rendering of fire and smoke on the screen. A complete list of functional requirements with priority levels (indicated in square brackets) is given in Table I using the simple and efficient notation proposed in [15].

Table I V-FIRE Functional Requirements

R01	[1]	V-FIRE shall display 3D fire in the simulation window.
R02	[1]	V-FIRE shall display 3D smoke in the simulation window.
R03	[1]	V-FIRE shall display 3D vegetation in the simulation window.
R04	[1]	V-FIRE shall display 3D buildings in the simulation window.
R05	[1]	V-FIRE shall display interaction between fire, smoke, vegetation, and buildings.
R06	[1]	V-FIRE shall allow the user to start the simulation.
R07	[1]	V-FIRE shall allow the user to stop the simulation at any time.
R08	[1]	V-FIRE shall allow the user to change the vegetation density of the terrain.
R09	[1]	V-FIRE shall allow the user to change the number of buildings on the terrain.
R10	[1]	V-FIRE shall allow the user to change the point of view.
R11	[1]	V-FIRE shall allow the user to view instructions on using the system.
R12	[2]	V-FIRE shall support various point of view presets, including but not limited to: three-quarter view, birds-eye, and ground-level.
R13	[2]	V-FIRE shall support a flying camera.
R14	[2]	V-FIRE shall allow the user to load terrain maps.
R15	[2]	V-FIRE shall allow the user to save terrain maps.
R16	[2]	V-FIRE shall allow the user to initiate a fire by clicking on the map.

R17	[2]	V-FIRE shall allow the user to pause the simulation.
R18	[2]	V-FIRE shall allow the user to fast-forward the simulation.
R19	[2]	V-FIRE shall allow the user to rewind the simulation.
R20	[3]	V-FIRE shall simulate a fire based on a set of input data.

Table I V-FIRE Functional Requirements [continued from the previous page]

Given the complex nature of the project, and the elements of software engineering risks that cannot be ignored (technical and organizational), we enforced throughout the project a prioritized treatment of requirements, on three levels. Level one translates to "must be implemented", level two to "might be implemented", and level three to "would like to implement, but most likely will not have time for it". However, the requirements not implemented will be considered for the next release of the system, planned for Spring 2006.

3.2 Non-Functional Requirements

Non-functional requirements specify the overall qualities of the system concerning its functionality, such as robustness, portability, extensibility, and efficiency [16]. In V-FIRE, this includes qualities such as making the system a cross-platform application, and the choice of graphics and windowing environment libraries. Table II contains the main non-functional requirements of the V-FIRE software tool.

Table II V-FIRE Non-Functional Requirements

T01	V-FIRE shall render in real-time.
T02	V-FIRE shall be a cross-platform application.
T03	V-FIRE shall be implemented with Qt, OpenGL, and OpenSG.
T04	V-FIRE shall be multi-threaded environment compatible.
T05	V-FIRE shall have marketable-quality graphics.
T06	V-FIRE shall maintain a simple user interface.
T07	V-FIRE shall use particle based fire and smoke.
T08	V-FIRE shall use dynamic "combustible" models.

4 Use Case Modeling

To gain further insight into V-FIRE's functionality the system's behavior has been "broken up" into use cases and scenarios [15], [18], [19]. The diagram shown in Figure 1 outlines the controls that allow the user to interact with the system, as well as the larger scale functionality of the backend. As shown in Subsection 4.2, a direct mapping between the system's use cases and its functional requirements was also established. Mapping requirements to use cases facilitates the creation of an architecture that supports the required functionality of the software. Often the user is the only or main actor considered when developing software. However, it is fundamental to consider time in the development of a real-time application.

The use cases which control the graphical output, DisplayMap and Update Simulation, were the most time consuming to develop because of their complexity.

Use cases show a subset of all possible interactions between the user and the V-FIRE system. All of the interaction between the end user and V-FIRE is through the user interface. Thus, the research done on possible use cases allowed for an insight into necessary features of the user interface needed for the system. Furthermore, it created the distinction between necessary requirements to provide the minimal functionality and those requirements which were add-ons and not essential to the system. The diagram shown in Figure 1 outlines the high level use cases, with the end user and time as the main actors. By concentrating on the most important use cases, the development team obtained a thorough understanding of the vital functionality to be provided by the system. The strong focus on specific use cases allows for a controlled development of software. The use cases, on which the end user is the main actor, consist of the minimal set of controls needed by the user to run and control a fire simulation through the user interface. The large scale functionality of the backend, the graphical output of the system to the user, is shown on the use cases which interact with the time actor. As shown in Subsection 4.2, a direct mapping between the system's use cases and its functional requirements was also established.

4.1 Use Case Diagram

The use case diagram shown in Figure 1 outlines the system's functionality and the roles actors play in the V-FIRE system. In order to further clarify the functionality, detailed descriptions of each use case are presented next.

UC01 *StartSimulation* The user selects the start location of the fire. The user then pushes the start button to run the simulation at the specified start location.

UC02 *StopSimulation* The user stops a running simulation of the fire by pushing the stop button. The simulation is paused and can be restarted from the current time frame.

UC03 *FastFowardSimulation* The user fast forwards the simulation of the fire by pushing the fast forward button. The simulation can be returned to regular speed by pushing the play button. The user can also stop the simulation while fast forwarding.

UC04 *RewindSimulation* The user rewinds the simulation of the fire by pushing the rewind button. The simulation can be returned regular speed by pushing the play button. The user can also stop the simulation while rewinding.

UC05 *ChangePointOfView* The user can specify the point of view of the camera by selecting one of the presets from the "View" menu. The user has the option for a free flying camera controlled by the keyboard and mouse.

UC06 *OpenMap* The user can load a preexisting map by selecting the "Open Map" option from the "File" menu. The user selects the map to load by selecting from a file selection dialog.

UC07 *SaveMap* The user can save a loaded map by selecting the "SaveMap" option from the "File" menu. The user can also select to rename the map by selecting the "Save As ..." option from the "File" menu.

UC08 *EditMap* The user can edit the currently loaded map by selecting the "Edit Map" option from the "Edit" menu. The editing changes are saved by pushing the "OK" button.

UC09 *ViewHelp* The user can view instructions on how to use the system. The instructions can be accessed by selecting the "Tutorial" option from the "Help" menu.

UC10 *DisplayMap* A map is loaded by default when the user runs V-FIRE. A default number of trees and buildings are included on the default map.

UC11 *UpdateSimulation* V-FIRE displays to the user the interaction of fire, smoke, vegetation, and buildings. The texture of the buildings and vegetation are updated according to damage done by fire. Smoke plumes are displayed proportionately to the fire size.

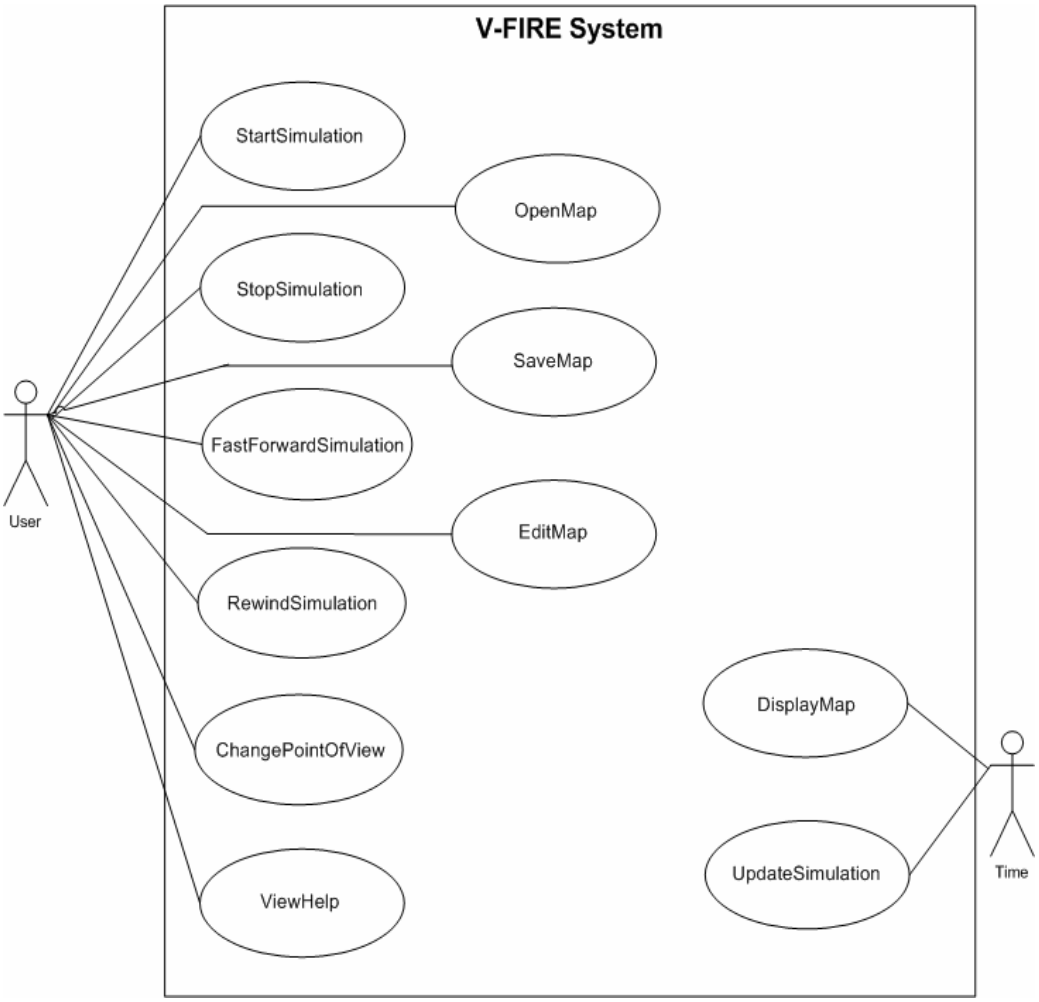

Figure 1 V-FIRE Use Case Diagram

4.2 Requirements Traceability Matrix

A requirements traceability matrix was used to map requirements to use cases [15]. The matrix specified the validity of the use cases by determining if a use case did entail a specified requirement. In turn, if a use case showed desired functionality for the system, and no requirements mapped to it, then it was found that necessary requirements were missing. The tracing of requirements and use cases also helped prioritize the importance of the systems' requirements. The larger the number of requirements mapped to a use case, the higher the importance and need to implement the functionality shown by the use case. The requirements traceability matrix for V-FIRE is shown in Figure 2.

		Use Case										
		UC01	UC02	UC03	UC04	UC05	UC06	UC07	UC08	UC09	UC10	UC11
Requirement	R01											X
	R02											X
	R03										X	
	R04										X	
	R05											X
	R06	X										
	R07		X									
	R08								X			
	R09								X			
	R10					X						
	R11									X		
	R12					X						
	R13					X						
	R14						X					
	R15							X				
	R16	X										
	R17		X									
	R18			X								
	R19				X							

Figure 2 The Requirements Traceability Matrix for V-FIRE

5 Architectural Design

The architecture of V-FIRE was developed with unit testing and extensibility as the driving foundations [13, 15, 16]. The system was broken down into independent units which could be developed concurrently, and integrated with minimal effort. As such, the architecture is comprised of piece-wise units with strict communication between unit components. The module which integrates the entire backend of the system is the Simulation subsystem. The decision to use the Qt library as the windowing environment was made during the design of the architecture. Qt, which supports object oriented design, was able to facilitate the separation of the system into two main components, the graphical user interface with event handling, and the backend for computing the graphical data.

The architecture of V-FIRE is structured into six subsystems as shown in Figure 3: GUI, Simulation, Terrain, Model, Material, and Fire and Smoke. The communication and display of the graphical information to the user is handled by the coupling of the GUI with the Simulation subsystems. The GUI subsystem is one of the most important modules of V-FIRE. As the main goal of V-FIRE is to run graphical simulations which are controlled by the user, the development of an intuitive interface with which the user could interact with ease was a strong focus of the development process. Furthermore, the visual information computed in the backend of the system must also be portrayed to the user in a way which is visually appealing and which the user can understand without having to read a manual.

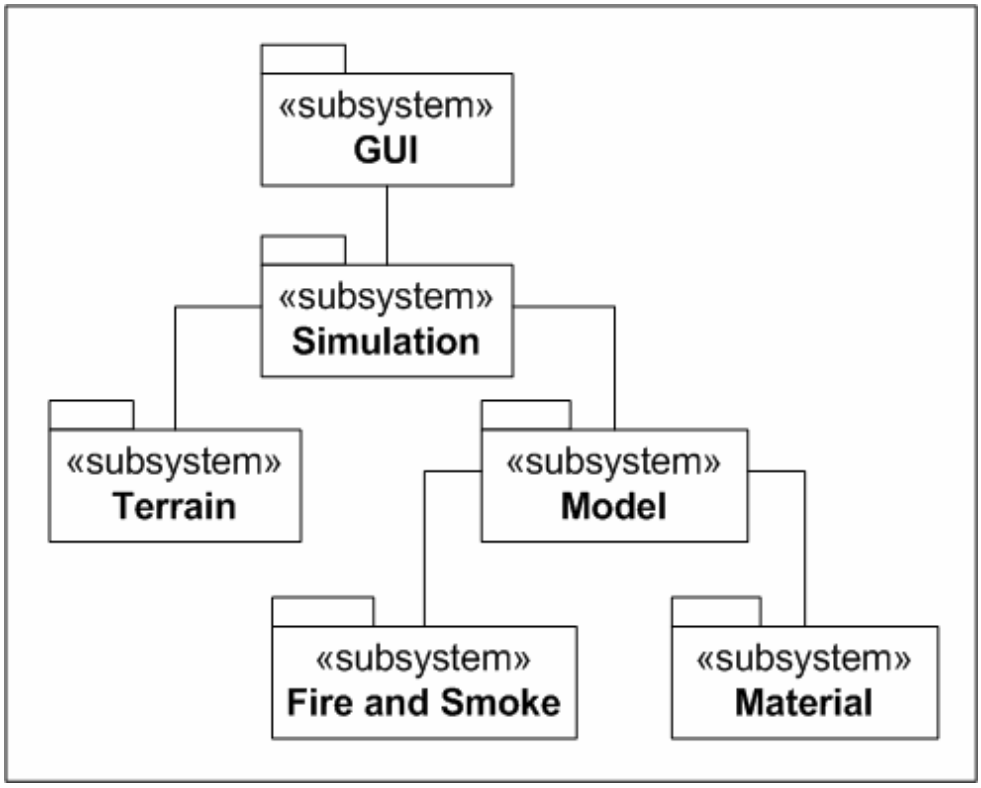

Figure 3 V-FIRE System Architecture

The Simulation subsystem integrates the entire backend of the system. The clear distinction between an area, such as a landscape, on which the simulation runs, and the entities, which are placed on it, allowed for concurrent development of the Terrain and the Model subsystems. Furthermore, this segregation of allows the system to be highly flexible and modifiable. The inherent properties and attributes of the Terrain subsystem can be modified without affecting the structure of the Model subsystem. This is a common feature of object oriented design, the ability to reuse code and modify independent entities without corrupting data.

One of the major issues when designing the architecture of V-FIRE was creating a flexible system, which was robust enough to sustain modifications and extensibility. The need for such a system was required in order to run simulations adapted to various scientific communities. As such, the subsystems were designed and implemented as generically as possible. This was largely accomplished during the implementation by storing large amounts of data and configurations in files. The models used on the first release of V-FIRE were pine trees with a small set of geometry. However, the architecture developed allows for the system to be modified with any kind of objects. This flexibility is extremely powerful, for the simulation of a forest landscape can be easily changed into the simulation of an urban area. However, the complexity of having different models burn with object dependent attributes was an issue of the architecture design, and later in the implementation phase, which were not solved due to the need to develop a system under extreme time constraints.

Descriptions of V-FIRE's architecture subsystems are as follows:

GUI The *GUI subsystem* contains the classes that interface with Qt, the window environment library, and OpenSG, a scene graph used for the optimization of graphical rendering. This subsystem also controls the user interaction with the system and feeds this information into the computations in the backend.

Simulation The *Simulation subsystem* contains the backend that controls the simulation. Through a generic interface, a programmer can control the simulation with little knowledge of lower level subsystems' implementations. This subsystem is also responsible for the placement and overall density of models on the graphical terrain.

Terrain The *Terrain subsystem* contains classes that describe the topographic features of the visible terrain and provide methods to load a terrain map from a file.

Model The *Model subsystem* contains classes that describe the visible state of 3D models used in the simulation. Such models include vegetation and inhabitable structures. This subsystem is responsible for the loading of models from files and maintaining a model's visible state throughout its life of combustion.

Fire and Smoke The *Fire and Smoke subsystem* contains classes that describe the visible fire and smoke used in the simulation. The state of this subsystem is controlled by logic in the simulation subsystem.

Material The *Material subsystem* contains classes that describe the properties and states of burnable materials in the simulation.

The concurrent development of the subsystems allowed for the rapid development of robust unit structures. The clear modularization of the architecture allowed for efficient unit testing, and reduced the amount of time spent on the integration of the independent modules. The system was then conglomerated with a bottom-up approach. A stripped down GUI subsystem, with limited functionality, was used in order to have a plain window on which to test the rest of the architecture units. Development then continued with the Fire and Smoke and Material subsystems, for the creation of graphic fire and smoke was the most important requirement of V-FIRE. The process was then continued with the implementation of the Terrain subsystem and its integration, through the Simulation subsystem, to the Model Subsystem. Finally, the GUI subsystem was integrated with the Simulation subsystem, and the full functionality of the GUI subsystem was completed.

6 Detailed Design

V-FIRE was designed using an object-oriented approach. The transition from the high level specification of the software architecture to the detailed design phase was simplified by the clear definition of the architecture modules. A clear correlation between subsystems and the objects that implement them was enforced in order to maintain the planned system structure. The organization of V-FIRE into program units and a sample activity chart are given, respectively, in Subsections 6.1 and 6.2.

6.1 Class Diagram

A class diagram of V-FIRE, showing the modularization of the system into object classes is presented in Figure 4. To be noted above all is the close resemblance of the system architecture in Figure 3 to the class diagram. The intuitive nature of the architecture subsystems made the creation of abstract objects a simple task. The object oriented approach was used for the benefits of data abstraction and data encapsulation. The modularization of the system also facilitates the expansion of the system into further applications. The class design also provides a generic interface through the simulation class to the lower level classes. Thus, little knowledge is needed of how low level classes such as Emitter and ParticleGroup work. The diagram also includes details of operations, attributes, relationships, multiplicity constraints, and visibility for each class. Complete class and method descriptions can be found upon request via [20]. The backend fire simulation is done through the Emitter and the ParticleGroup class. Particle groups define a fire and the overall behavior of the particles is controlled by the corresponding emitter.

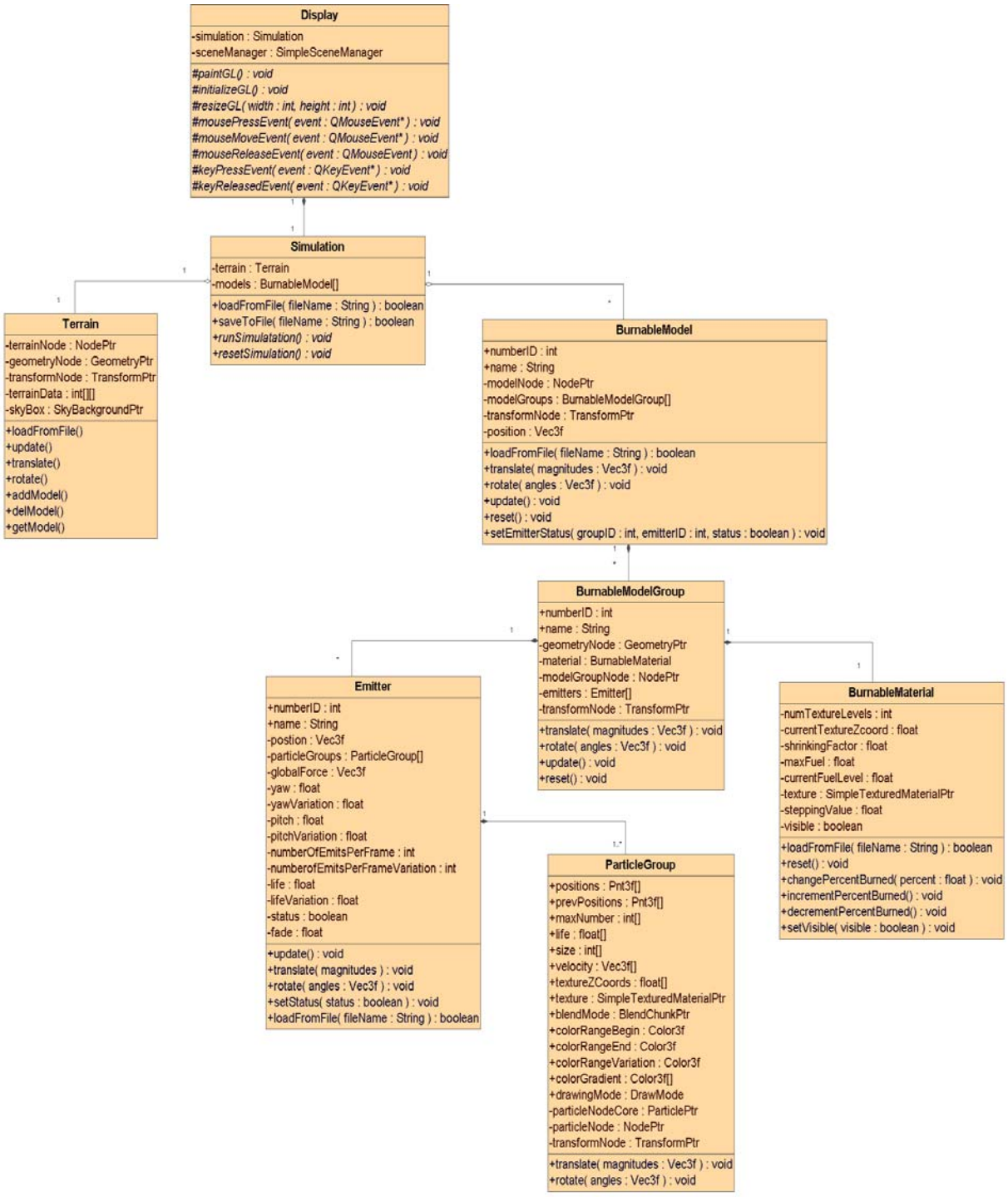

Figure 4 V-FIRE Class Diagram

6.2 System Activity Chart

In order to thoroughly cover the design of V-FIRE, various diagrams were created as part of its software model, including activity charts, state charts, and flow charts [13, 15, 16]. The

majority of the interaction between the system and the user occurs before the simulation is started. This is quite logical, for the system is set up so that a simulation of a wildfire is run based on the parameters that the user is free to modify through the interface. A low level view of the activity process that occurs during the setup and preparation of the system for the simulation is presented in Figure 5. The activity chart starts by showing the process of opening a simulation file or loading a default map. The following branches of activities show options which are presented to the user before the simulation is started. These include editing the current terrain, changing the point of view of the camera, a feature needed in order to get a sense of a full immersive 3D environment. The challenge of creating activity charts was that V-FIRE is an abstract application. The majority of the time the user is viewing the graphical output computed on the backend based and influenced by user customizations.

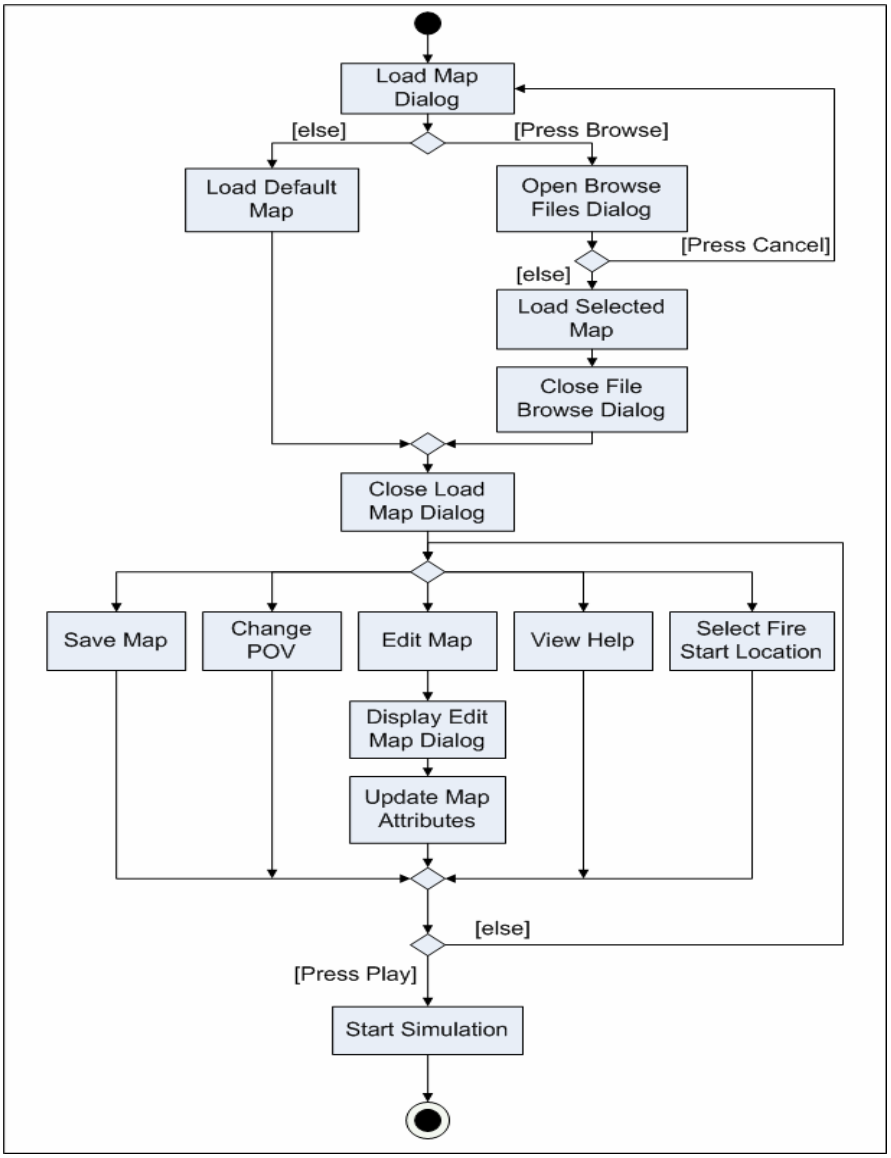

Figure 5 Sample Activity Chart of V-FIRE
[shows the operations of the system before a fire simulation is started]

V-FIRE is a wildfire simulation tool. As such, the most important requirement is the graphical rendering of realistic fire. This entailed creating graphics which looked and behaved like fire. Yet, the behavior of fire is an intuitive notion most people have, that is, they know that something is fire when they see it. Yet the careful description of fire's behavior comes to a major halt on all areas of research. Chaotic phenomena are extremely complex to imitate, especially imitating with an efficient algorithm in polynomial time. As such, the design of an object oriented approximate solution to the behavior of fire's chaotic behavior was a feat of tremendous work and complexity. It was initially intended to create an application which would run on a CAVE environment, that is, V-FIRE was to create fully volumetric fire simulations. Under the time constraints, and the need to develop a robust application rapidly in a short period of time, we were lead to consider a solution which gave the appearance of volumetric fire, which was rendered as 2D objects. The technique is called billboarding, which gives the illusion of an object being volume by always having the object face in the direction of the camera. The decision to use a particle system to represent fire was reached after comparing it to other rendering methods such as voxels. The particles used had properties such as velocity, location, and life span. A particle system has the advantage of giving a large number of small objects common attributes, which when altered slightly and taken as a conglomerate give the illusion of a random effect such as fire exhibits. The particle system together with billboarding gives each the particle a 3D view when rendered to the user.

The challenge posed next was that of controlling the overall behavior of the particles in order to simulate a wildfire. For this a large number of particles was needed, in addition for the need of the ability to control the fire in the backend. Minimal control functionality described in the use cases was necessary in order to have a useful system. The ability to start fire in a specified location, stopping and restarting the fire required a transcendental control over the desired random behavior of the particles. In order to solve this problem we used the notion of an emitter of particles, which would emit fire particles at a specified location on the terrain. The emitter would have properties to control the fire, that is, to control the flow of particles at fixed locations. The emitters were then placed on the trees. So that the lighting of a tree on fire was done by turning on its particles emitter. Figure 6 shows the activity chart for the updating of the attributes of a single particle. Each particle has a lifetime. After the lifetime expires, its position is reset and it is hidden until it is emitted once again by the emitter. The process shows how the emitter updates the position based on the particles velocity, and the velocity in turn is affected by a global force, including gravity and wind.

7 V-FIRE in Action

As a result of efficient design and optimization V-FIRE is able to render a forest on several hundred trees under combustion in real-time on a mainstream personal computer with a high performance video card. However, V-FIRE's architecture allows for scalable performance on multiprocessor machines and machines with several video cards. To achieve the desired performance and realism, fire and smoke are implemented using particle systems and volumetric textures.

The simulation component of V-FIRE's architecture allows for generic access to the rest of the system. This framework was designed with an emphasis on extensibility and flexibility allowing for creation of more realistic fire-spreading models. For the purposes of demonstration a simple fire-spreading model was created to drive the simulation.

The graphical user interface and interaction was designed to facilitate viewing and control over the simulation. To create a point of origin for a wildfire a user must only select an object to ignite. From this point the simulation would take over and spread the fire. The user is able to control their viewing with the mouse. Many wildfire applications allow the user to on have only a distant view over the simulation. A main advantage of V-FIRE's approach is the ability to view the simulation from any angle or distance. It is even possible to view simulations from within a fire. This makes it an ideal for training and fire prevention as well as visualization [2, 6, 21].

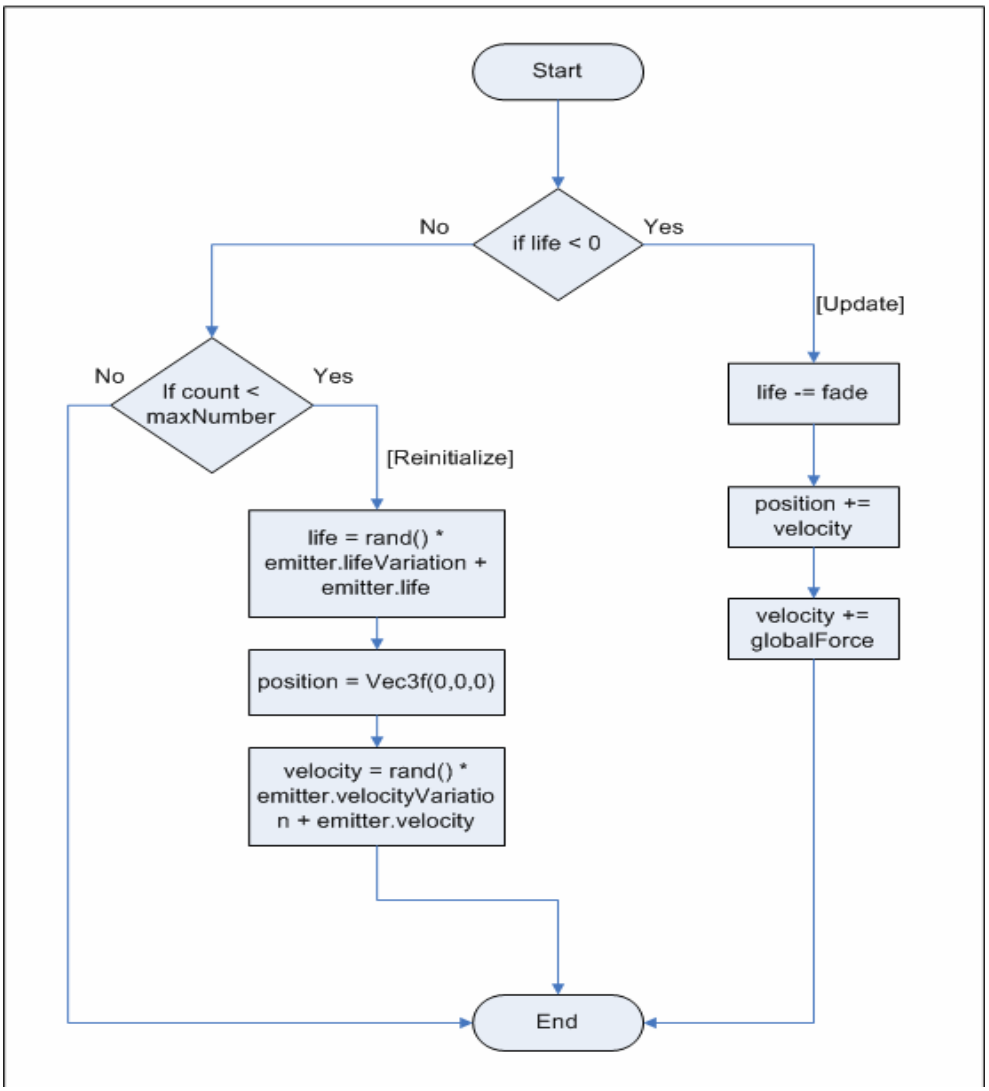

Figure 6 Sample Activity Chart of V-VIRE
[updating of the attributes of a single fire particle by a fire emitter]

8 Future Work

The specification, design, and implementation phases for the first release of the V-FIRE project have been completed. We have laid the groundwork for the visual components of the project, which are functional in the initial prototype. From the implementation work done so far, it became evident to us that the framework created is solid for driving the work ahead.

There are a number of areas of work, however, that are outside the scope of the first phase of this project. V-FIRE is designed to support models and visualizations of fire. The models currently being used, such as those described in [2, 5, 6, 22, 23, 24] do not take all the complexities of fire-environment interaction into account. More advanced models will be created and implemented for future releases of V-FIRE. As processing capabilities increase, more detailed models will be able to run in real-time. Parallel processing techniques can also be used to increase the amount of complexity that can be handled.

As more advanced graphics hardware and rendering techniques and tools become available, they will be incorporated into the system. Tools such as programmable pixel shaders [25], which require the latest graphics hardware, can be used to create more realistic lighting and shadow effects generated by fire and smoke. New graphics engines, such as the Unreal 3 Engine [26], should also be considered as possible platforms for future work.

Lastly, as V-FIRE is used in research environments and integrated with other tools, it is likely that requests for different types of system interactions will be made. A comprehensive physics engine could be added to aid in creating realism within the environment. This would be especially important in applications such as firefighter or evacuation training software.

Future work on V-FIRE demands focus on three main areas: advanced user control of the simulation, advanced graphics techniques, and a thorough physics and environmental model. The rapid software development of the V-FIRE system forced certain soft (second-level priority) requirements to be left until a second release. The requirements implemented on the first release provide a powerful interface which provides the user with an array of options to control the system. However, advanced control features that would be useful and important if V-FIRE were to be used as a professional scientific tool would include controlling the speed of the simulation, including rewinding and forwarding of a running simulation.

V-FIRE provides realistic looking fire, but there is still more research to be done in order to improve the results obtained. Further detail could be added to the current fire model in order to make the system further scientifically accurate. Shadowing needs to be added to the trees, or any other object models, added to the system. Level of detail also needs to be further improved. Currently, trees are replaced by lower polygon models when the camera is farther away from the terrain. This has the advantage of improving the rendering frames per second.

Lastly, a realistic physics model must be added to account for the fluid dynamics of smoke and fire. The current behavior of the particles is random, which gives rather satisfactory results. Instead, the particles must have a fluid model which follows the Navier-Stokes

equations or the Euler 2D equations for compressible and uncompressible flow of fluids. However, in order to reproduce a forest fire from data taken from an actual fire the conditions on the system must reflect those of the real world. The system must also react in the same way the environment would react in a wildfire. The system must also react accordingly to the application of environmental factors such as wind, elevation, terrain topology, and air and terrain humidity.

Finally, a challenge which was encountered over many times was the refactoring of code in order to improve the running time of the application. A lot of time was spent on writing efficient code which would increase, and not hinder, the performance of V-FIRE. Yet, improvements to the running time of the system can always be made.

9 Conclusions

V-FIRE is an application designed to help researchers visualize models of fire in realistic environments. The system provides a safe context for learning how wildfires, accidental fires, and arson affect objects in the real world. Although this system will eventually be used for fully immersive 3D modeling and event re-creation, its current implementation with single monitor support provides the structure on top of which the software can continue to evolve.

To offer some insight into the "look and feel" of the environment, Figure 7 shows a screenshot of the V-FIRE application's main window and Figure 8 presents a configuration editor for environment parameters. The system was developed under a tight schedule and a short time period.

The specification and design processes undertaken by the project team have helped to create a flexible architecture that can be expanded without retooling the core elements of the system. Specifically, each subsystem is encapsulated to support more advanced fire modeling and visualizations in future work. Having a flexible and well-encapsulated architecture for this type of research is essential, as technical advances that can be applied to fire modeling and visualization are frequent.

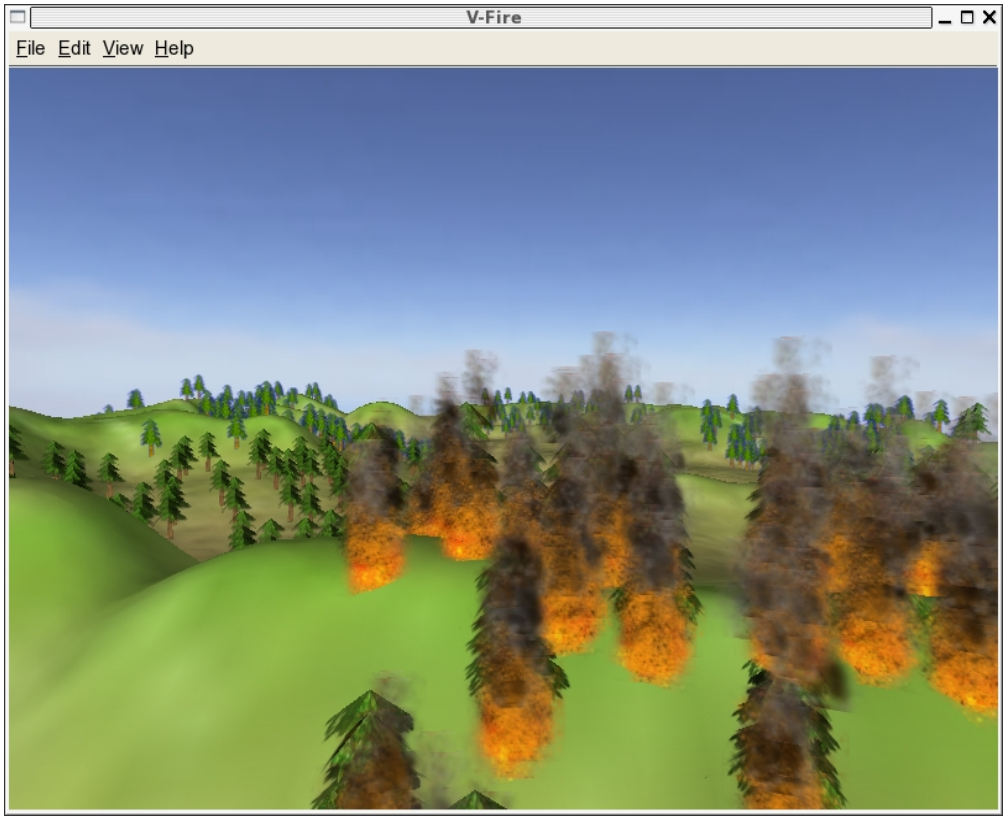

Figure 7 V-FIRE: Main Interface

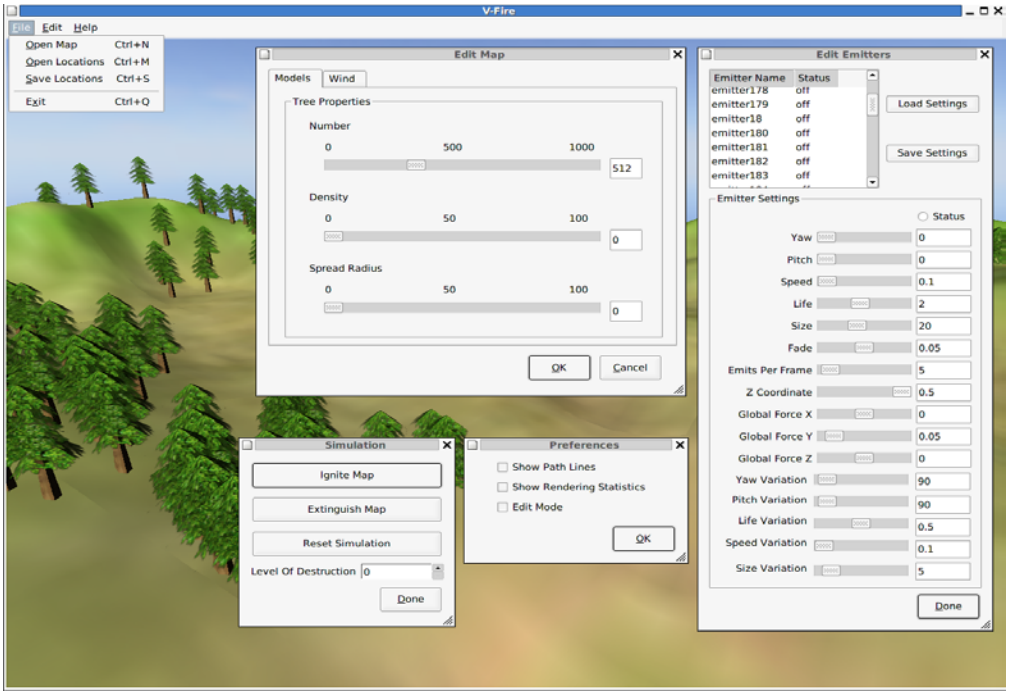

Figure 8 V-FIRE: Settings and Options

References

[1] D. Drysdale, *An Introduction to Fire Dynamics,* Wiley & Sons, 2001.

[2] P.S. McCormick and J.P. Anrens. Visualization of Wildfire Simulations. *IEEE Computer Graphics and Applications*, vol. 18, no. 2, 1998, pp. 17-19.

[3] S. Takeuchi and S. Yamada. Monitoring of Forest Fire Damage by Using JERS-1 InSar. *Proceedings of the 2002 IEEE Geosience and Remote Sensing Symposium*, vol. 6, pp. 3290-3292.

[4] K. Satoh, S. Weiguo, and K.T. Yang, A Study of Forest Fire Danger Prediction System in Japan, *Proceedings of the 15th International Workshop on Database and Expert Systems Applications*, 2004, pp. 598-602.

[5] D.Q. Nguyen, R. Fedwik, and H.W. Jansen. Physically Based Modeling and Animation of Fire. *Proceedings of the 29th ACM Intl. Conference on Computer Graphics and Interactive Techniques*, 2002, pp. 721-728.

[6] Z. Melek and J. Keyser. Interactive Simulation of Fire, *Proceeding of the 10th Pacific Conference on Computer Graphics and Applications, 2002*. October 9-11 2002, pp. 431-432.

[7] X. Wei, W. Li, K. Mueller, and A. Kaufman. Simulating Fire with Texture Splats. *Proceedings of the 2002 ACM Conference on Visualization,* pp. 227-235.

[8] Q. Yu, C. Chen, Z. Pan, and J. Li. A GIS-Based Forest Visual Simulation System. *Proceedings of the International Conference on Image and Graphics (ICIG 2004)*, pp. 410-413, Third 2004.

[9] Q. Zhu, T. Rong, R. Sun, and Y. Shuai. A Study of Fractal Simulation of Fire Forest Spread. *Proceedings of the IEEE Symposium on Geoscience and Remote Sensing*, 2001, vol. 2, pp. 801-803.

[10] S. Dascalu, Y. Varol, F.C. Harris, Jr., and B.T. Westphal. Computer Science Capstone Course Senior Projects: From Project Idea to Prototype Implementation, *Proceedings of the IEEE Frontiers in Education 2005 (FIE 2005),* October 2005 (to appear).

[11] F. Randima, *GPU Gems: Programming Techniques, Tips, and Trips for Real-Time Graphics.* Addison-Wesley, 2004.

[12] H. Creagh. CAVE Automatic Virtual Environments. *Proceedings of the 2003 IEEE Conerence. on Electrical Insulation and Electrical Manufacturing & Coil W. Technology*, pp. 499-504.

[13] Booch, G., Rumbaugh, I. , and Jacobson, I. *The Unified Modeling Language: User Guide*, Addison Wesley, 1999.

[14] OMG's *UML Resource Page*, accessed September 19, 2005 at http://www.omg.org/uml

[15] J. Arlow, I. Neustadt. *UML and the Unified Process: Practical Object-Oriented Analysis & Design*, Addison-Wesley, 2002.

[16] I. Sommerville. Software Engineering, 7th Edition, Addison-Wesley, 2005.

[17] S. Dascalu, P. Hitchcock, N. Debnath, and A. Klempau. From Graphical Representations to Formal Specifications and Return: Translation Algorithms in the Harmony Environment. *Proceedings of the IEEE Conference on Information Reuse and Integration (IRI-2004)*, 2004, pp. 215-221.

[18] S. Dascalu, and P. Hitchcock. An Approach to Integrating Semi-formal and Formal Notations in Software Specification. *Proceedings of SAC 2002, the ACM Symposium on Applied Computing*, 2002, pp. 1014-1020.

[19] J. Preece, Y. Rogers, H. Sharp. *Interaction Design: Beyond Human-Computer Interaction,* Wiley & Sons, 2002.

[20] V-FIRE Project, accessed September 19, 2005 at http://www.cse.unr.edu/~gkelly/v-fire

[21] J. Ahrens, P. McCormick, J. Bossert, J. Reisner, Winterkamp. Case Study: Wildfire Visualization. *Proceedings of IEEE Visualization'97*, pp. 451-454.

[22] A. Muzy, E. Innocenti, A. Aiello, J.-F. Santucci, and G. Wainer. Cell-DEVS Quantization Techniques in a Fire Spreading Application. *Proceedings of the Winter Simulation Conference, 2002*, 2002, vol. I, pp. 542-549.

[23] T.C. Henderson, P.A. McMurtry, P.J. Smith, G.A. Voth, C.A. Wight, D.W. Pershing. Simulating Accidental Fire and Explosions. *Computing in Science & Engineering*, vol. 2, issue 2, March-April 2000, pp. 64-76.

[24] E. Innocenti, A. Muzy, A. Aiello, J.-F. Santucci, and D.R.C. Hill. Active-DEVS: A Computational Model for the Simulation of Forest Fire Propagation. *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, 2004, pp. 1857-1863.

[25] nVIDIA website, *Pixel Shaders*, accessed September 18, 2005 at http://www.nvidia.com /object/ feature_pixelshader.html

[26] *Epic Games*, accessed Sept. 18, 2005 at www.epicgames.com/ UnrealEngineNews.html