V-FIRE

Virtual Fire In Realistic Environments

Design

Grant Kelly, Juan Quiroz, Michael Penick

Advisors
Sergiu Dascalu, PhD., Brian Westphal, M.S.C.S., Frederick C. Harris, PhD.

University of Nevada, Reno
CS 426
3/20/2005

Table of Contents

Introduction

Computer modeling of fires is an effective tool for scientists to experiment, study, and forecast the patterns of wildfires. Mathematical models of fires exist, but none of them correctly reflect the true behavior of fire. Fire is such a dynamic and chaotic 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 make simplifications and assumptions in order to obtain results, that is, in order to make the system solvable. Furthermore, the results of such models are data, possibly graphs. The visualization of the model is even more difficult. The behavior of a wildfire can change drastically by a slight change in the atmospheric pressure or by a wind gust.

V-FIRE will be a 3D fire simulation and visualization tool. V-FIRE will allow users to harness and observe a fire within a controlled environment. The system will be designed to model a wildfire as realistic 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. Furthermore, as an empirical tool, V-FIRE will also provide the user with the ability of multiple view points for the main camera, such as aerial and full immersion.

The long term goal of V-FIRE is to create real-time, marketable-quality graphics for fire visualization in a CAVE. A CAVE, Cave Automatic Virtual Environment, provides a full-immersion experience for their inhabitants. 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.

Two months after the submission of the project proposal for V-FIRE, Team 01 has successfully completed the software requirements specification and the design phases. Nevertheless, details of the implementation phase have been under way. The particle system for the V-FIRE system is at the early states of development. Furthermore, the loading of tree models into the scene graph has successfully been achieved. Team 01 also continues to master the intricate details of OpenSG.

After careful deliberation, Team 01 has decided that it is not feasible at this time to implement the “Fast Forward” and “Rewind” features of a wildfire simulation discussed on the software requirements specification document. It is regrettable to make such a decision, but the features will not be included on the final product.

Through the design phase of V-FIRE, several refinements have been made to the initial planned structure of the system. First of all, Team 01 has decided to use emitters placed in objects instead of treating fire and objects as separate entities. As a consequence, the fire will emanate from the objects at a specified angle, velocity, and direction. This will not only polish the structure and implementation of V-FIRE, but it will also make an efficient use of the scene graph. A further refinement to V-FIRE is that a file type will be developed in order to load data into the scene graph. This will especially be used for the initialization of the objects in the scene. Data loaded from files will include textures, the placement of objects and emitters, and further details.

The remainder of this paper is structured into five sections: high level and medium level design, detailed design, user interface design, list of references, and contribution of team

members. The high level and medium level design section presents a system level and class diagram of the V-FIRE system, along with descriptions of the classes and their corresponding methods. The detailed design section presents the low level structure of the V-FIRE system through activity charts, flowcharts, and a sample scene graph for rendering objects. In the user interface design section, snapshots of the graphical user interface of V-FIRE with their corresponding descriptions are presented. The list of references section presents annotated references of four articles and a domain book, which were used for the design and specification phases of V-FIRE. Finally, the contribution of team members section describes the tasks completed by each member of Team 01.

High Level and Medium Level Design

System Level Diagram

The system architecture is composed of several subsystems, as shown in Figure 1. Each subsystem contains classes that are logically similar in functionality. Links between subsystems are provided by a single class in each subsystem.

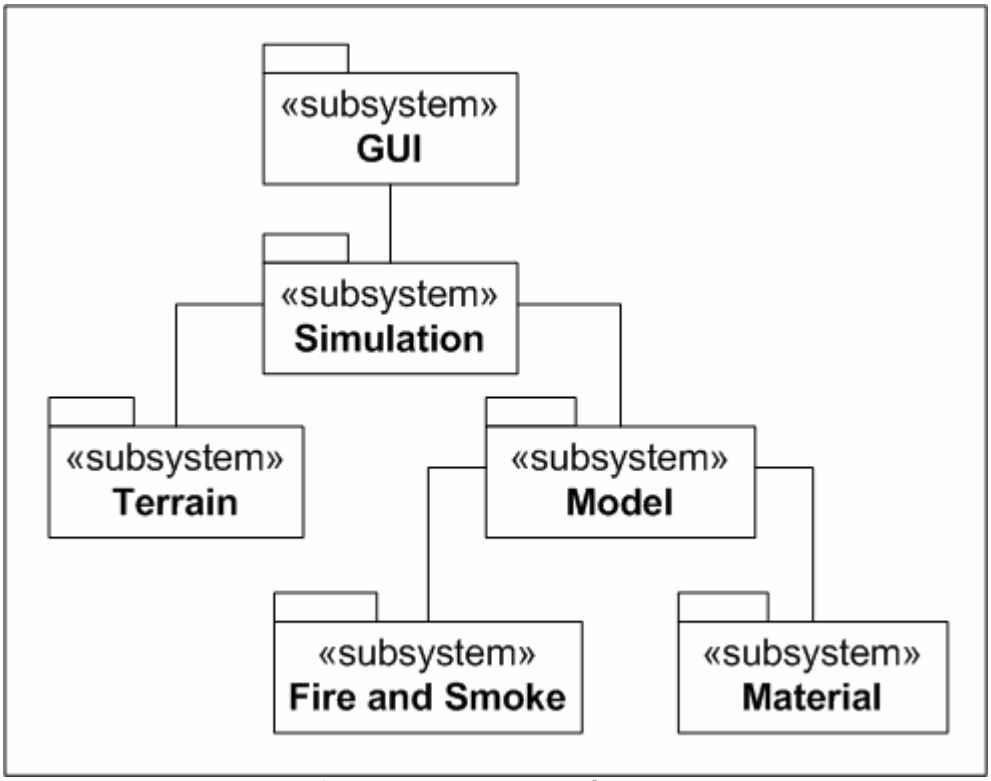

Fig. 1 V-FIRE System Architecture

Descriptions of subsystems are as follows:

GUI
The GUI subsystem contains the classes that interface with Qt, OpenSG, and control user interaction with the system. This subsystem also connects the viewable front-end and the simulation back-end.

Simulation
The Simulation subsystem contains the back-end that controls the simulation. Through a generic interface, a programmer can control the simulation with little knowledge of the other subsystems' implementations. This subsystem is also responsible for the placement and overall density of models on the 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.

Class Diagram

The class diagram shows the structure of V-FIRE in program units.

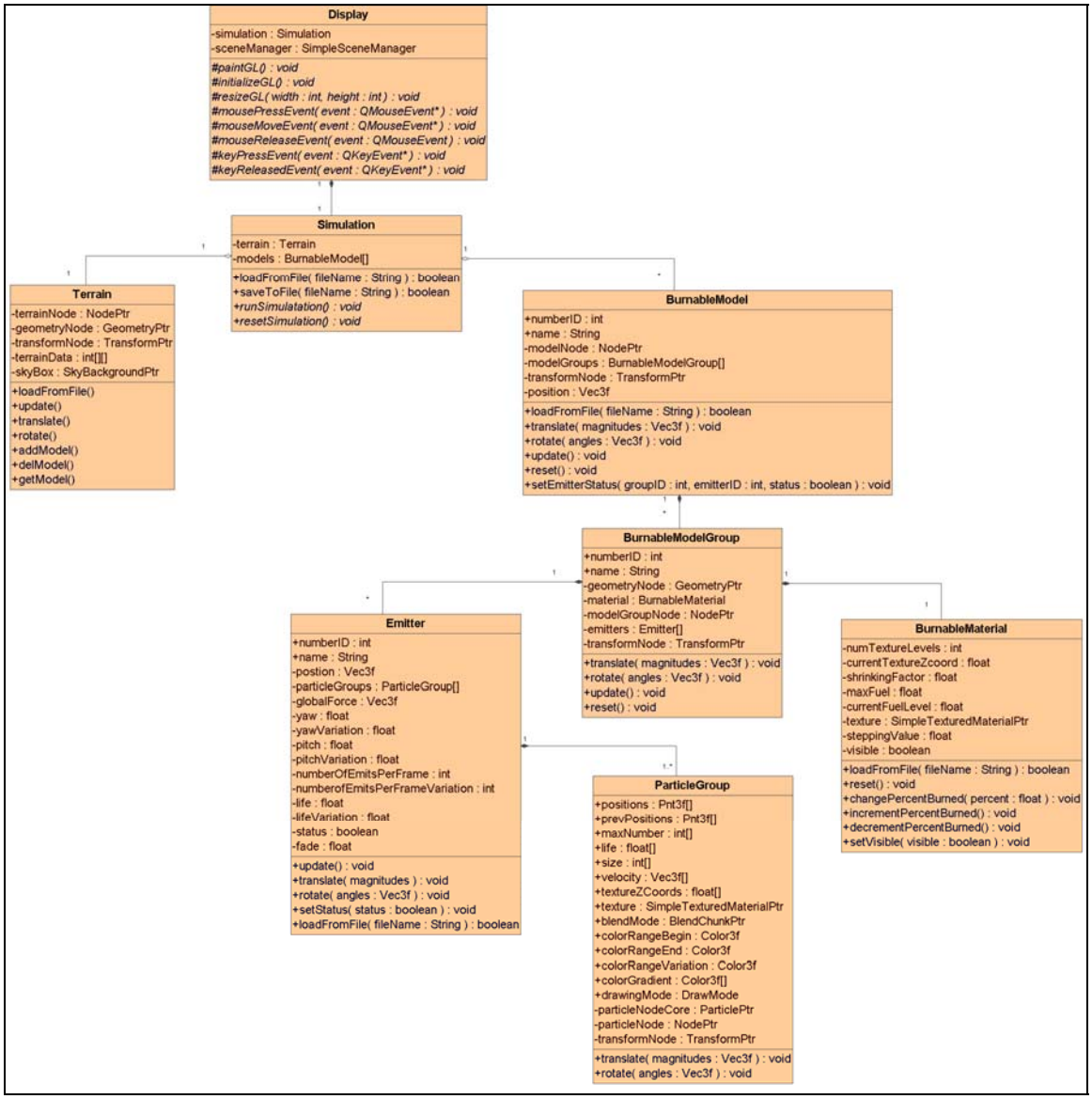

Fig. 2 V-FIRE Class Diagram

Class Method Descriptions

In this section, each class method is briefly described with respect to visibility, parameters, and return type.

ParticleGroup Class

The ParticleGroup class maintains the states and attributes of particles in a particle group. Groups of particles make up the visual elements, such as fire and smoke. Particle attributes include texture and color. On the other hand, states of particles include position, velocity, size, and life.

Class	ParticleGroup
Method	translate
Visibility	public
Return type	void
Parameters, types	magnitudes:Vec3f
Description	This method translates the particle group by setting its transformation node.

Class	ParticleGroup
Method	rotate
Visibility	public
Return type	void
Parameters, types	angles:Vec3f
Description	This method rotates the particle group by setting its transformation node.

Emitter Class

The emitter class initializes and animates particles contained in particle groups. This is done by modifying attributes, that is, their position, velocity, color, etc.

Class	Emitter
Method	translate
Visibility	public
Return type	void
Parameters, types	magnitudes:Vec3f
Description	This method translates the particle group by setting its transformation node.

Class	Emitter
Method	rotate
Visibility	public
Return type	void
Parameters, types	angles:Vec3f
Description	This method rotates the particle group by setting its transformation node.

Class	Emitter
Method	update
Visibility	public
Return type	void
Parameters, types	none
Description	This method animates the particles contained in the particle groups.

Class	Emitter
Method	setStatus
Visibility	public
Return type	void
Parameters, types	status:boolean
Description	This method is used to start and stop the animation of the particles.

Class	Emitter
Method	loadFromFile
Visibility	public
Return type	Boolean
Parameters, types	fileName:String
Description	This method loads emitter data from a file, for initialization purposes.

BurnableMaterial Class

The BurnableMaterial class maintains the appearance of a BurnableModelGroup instance based on material properties loaded from a file and particle interaction.

Class	BurnableMaterial
Method	loadFromFile
Visibility	public
Return type	Boolean
Parameters, types	fileName:String
Description	This method loads material data from a file.

Class	BurnableMaterial
Method	reset
Visibility	public
Return type	void
Parameters, types	none
Description	This method resets the material of an object to its unburned state.

Class	BurnableMaterial
Method	changePercentBurned
Visibility	public
Return type	void
Parameters, types	percent:float
Description	This method changes the level at which an object's material appears to be burned.

Class	BurnableMaterial
Method	incrementPercentBurned
Visibility	public
Return type	void
Parameters, types	none
Description	This method increments the level at which an object's material appears to be burned.

Class	BurnableMaterial
Method	decrementPercentBurned
Visibility	public
Return type	void
Parameters, types	none
Description	This method decrements the level at which an object's material appears to be burned.

Class	BurnableMaterial
Method	setVisible
Visibility	public
Return type	void
Parameters, types	visible:Boolean
Description	This method toggles a model group's visibility.

BurnableModelGroup Class

The BurnableModelGroup class contains the geometry, appearance, and the way things burn depending on the group type and placement of emitters. It also manages the location of the emitters and the angle at which particles are emitted.

Class	BurnableModelGroup
Method	reset
Visibility	public
Return type	void
Parameters, types	none
Description	This method resets a model group to its original unburned state.

Class	BurnableModelGroup
Method	update
Visibility	public
Return type	void
Parameters, types	none
Description	This method updates a model group's particle systems and materials states.

Class	BurnableModelGroup
Method	rotate
Visibility	public
Return type	void
Parameters, types	angles:Vec3f
Description	This method rotates the particle group by setting its transformation node.

Class	BurnableModelGroup
Method	translate
Visibility	public
Return type	void
Parameters, types	magnitudes:Vec3f
Description	This method translates the particle group by setting its transformation node.

BurnableModel Class

The BurnableModel class contains the logical division of the model into BurnableModelGroups. The division of groups is loaded from a file. It is also used to update the states of groups contained in a model.

Class	BurnableModel
Method	rotate
Visibility	public
Return type	void
Parameters, types	angles:Vec3f
Description	This method rotates the particle group by setting its transformation node.

Class	BurnableModel
Method	translate
Visibility	public
Return type	void
Parameters, types	magnitudes:Vec3f
Description	This method translates the particle group by setting its transformation node.

Class	BurnableModel
Method	loadFromFile
Visibility	public
Return type	void
Parameters, types	fileName:String
Description	This method loads geometric, group, emitter, and material data from a file.

Class	BurnableModel
Method	update
Visibility	public
Return type	void
Parameters, types	none
Description	This method updates the state of its model groups.

Class	BurnableModel
Method	reset
Visibility	public
Return type	void
Parameters, types	none
Description	This method resets the model to its unburned state.

Class	BurnableModel
Method	setEmitterStatus
Visibility	public
Return type	void
Parameters, types	groupID:int, emitterID:int, status:boolean
Description	This method toggles an emitter status by using its identification number and groups.

Terrain Class

The Terrain class maintains the map and where the sky, trees, buildings, and models in general are placed.

Class	Terrain
Method	translate
Visibility	public
Return type	void
Parameters, types	magnitudes:Vec3f
Description	This method translates the terrain by setting its transformation node.

Class	Terrain
Method	rotate
Visibility	public
Return type	void
Parameters, types	angles:Vec3f
Description	This method rotates the terrain by setting its transformation node.

Class	Terrain
Method	loadFromFile
Visibility	public
Return type	void
Parameters, types	fileName:String
Description	This method loads terrain information from file.

Class	Terrain
Method	addModel
Visibility	public
Return type	void
Parameters, types	x:int, y:int, numberID:int
Description	This method adds a model to the specified position on the terrain.

Class	Terrain
Method	deleteModel
Visibility	public
Return type	void
Parameters, types	x:int, y:int
Description	This method deletes a model from the specified position on the terrain.

Class	Terrain
Method	getModel
Visibility	public
Return type	int
Parameters, types	x:int, y:int
Description	This method returns the id of the model at the specified position on the terrain. If not found returns -1.

Display Class

The Display class is the interface between the simulation back-end and the graphical user interface. Input devices such as the keyboard and mouse can be used to control different aspects of the scene and simulation.

Class	Display
Method	paintGL
Visibility	protected
Return type	void
Parameters, types	none
Description	This method is called by Qt to render scene using OpenGL renderer.

Class	Display
Method	initializeGL
Visibility	protected
Return type	void
Parameters, types	none
Description	This method is called in by after the constructor to initialize OpenGL states.

Class	Display
Method	resizeGL
Visibility	protected
Return type	void
Parameters, types	width : int, height : int
Description	This method is called when a window is resized. Its function is to resize the scene according to the window size and aspect ratio.

Class	Display
Method	mousePressEvent
Visibility	protected
Return type	void
Parameters, types	event : QMouseEvent*
Description	This method responds to events where a mouse button is pressed and allows the scene to respond in an appropriate manner.

Class	Display
Method	mouseReleaseEvent
Visibility	protected
Return type	void
Parameters, types	event : QMouseEvent*
Description	This method responds to events where a mouse button is released and allows the scene to respond in an appropriate manner.

Class	Display
Method	mouseMoveEvent
Visibility	protected
Return type	void
Parameters, types	event : QMouseEvent*
Description	This method responds to events where the mouse is moved and allows the scene to respond in an appropriate manner.

Class	Display
Method	keyPressEvent
Visibility	protected
Return type	void
Parameters, types	event : QKeyEvent*
Description	This method responds to events where a keyboard button is pressed and allows the scene to respond in an appropriate manner.

Class	Display
Method	keyReleaseEvent
Visibility	protected
Return type	void
Parameters, types	event : QKeyEvent*
Description	This method responds to events where a keyboard button is released and allows the scene to respond in an appropriate manner.

Simulation Class

The Simulation class is responsible for placing models on the terrain and controlling the spread of fire. In essence it contains all the information that describes the scene. The abstract classes allow a general interface for driving the simulation.

Class	Simulation
Method	loadFromFile
Visibility	public
Return type	boolean
Parameters, types	fileName : String
Description	This method loads a terrain, models and simulation data from a file.

Class	Simulation
Method	saveToFile
Visibility	public
Return type	boolean
Parameters, types	fileName : String
Description	This method saves the current state of the terrain and model density to a file

Class	Simulation
Method	runSimulation
Visibility	public
Return type	void
Parameters, types	none
Description	This method is used to update the current state of the simulation. A model for animating and spreading fire is defined in the method.

Class	Simulation
Method	resetSimulation
Visibility	public
Return type	void
Parameters, types	none
Description	This method is used to initialize or reset a simulation to its original state before a simulation is run.

Detailed Design

This section contains diagrams depicting low-level details of pieces of the V-FIRE system.

Figure 3 depicts an activity chart before the simulation has started. Activity includes opening a map from a file, saving a map to a file, editing an existing map, changing the point of view, selecting a location to start a fire, and viewing the system help.

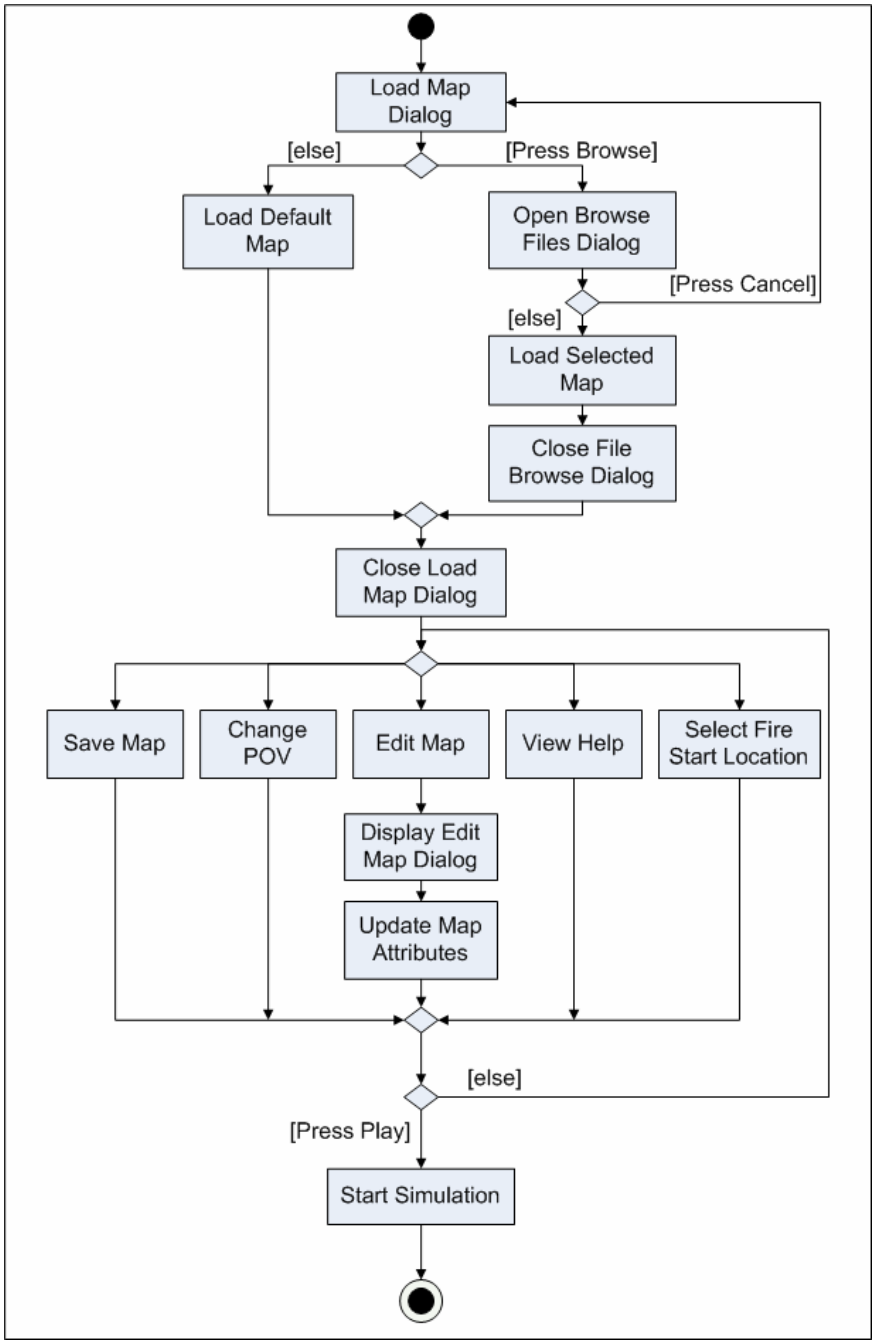

Fig. 3 V-FIRE Activity Chart

Figure 4 depicts the activity available after the simulation has been started.

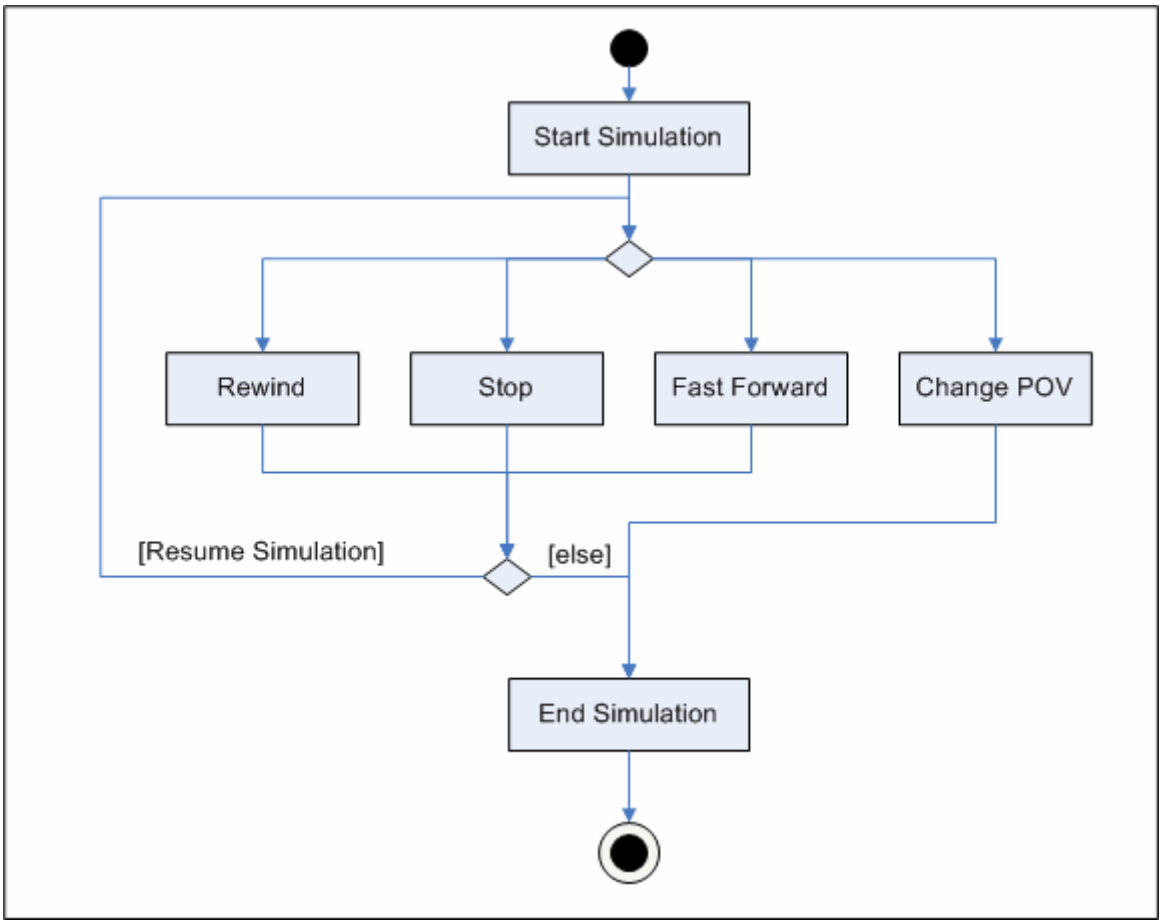

Fig. 4 V-FIRE Activity Chart During Simulation

Figure 5 is a flow chart showing the method BurnableMaterial::incrementPercentBurned() which adjusts a material's burn-level both visually and statistically.

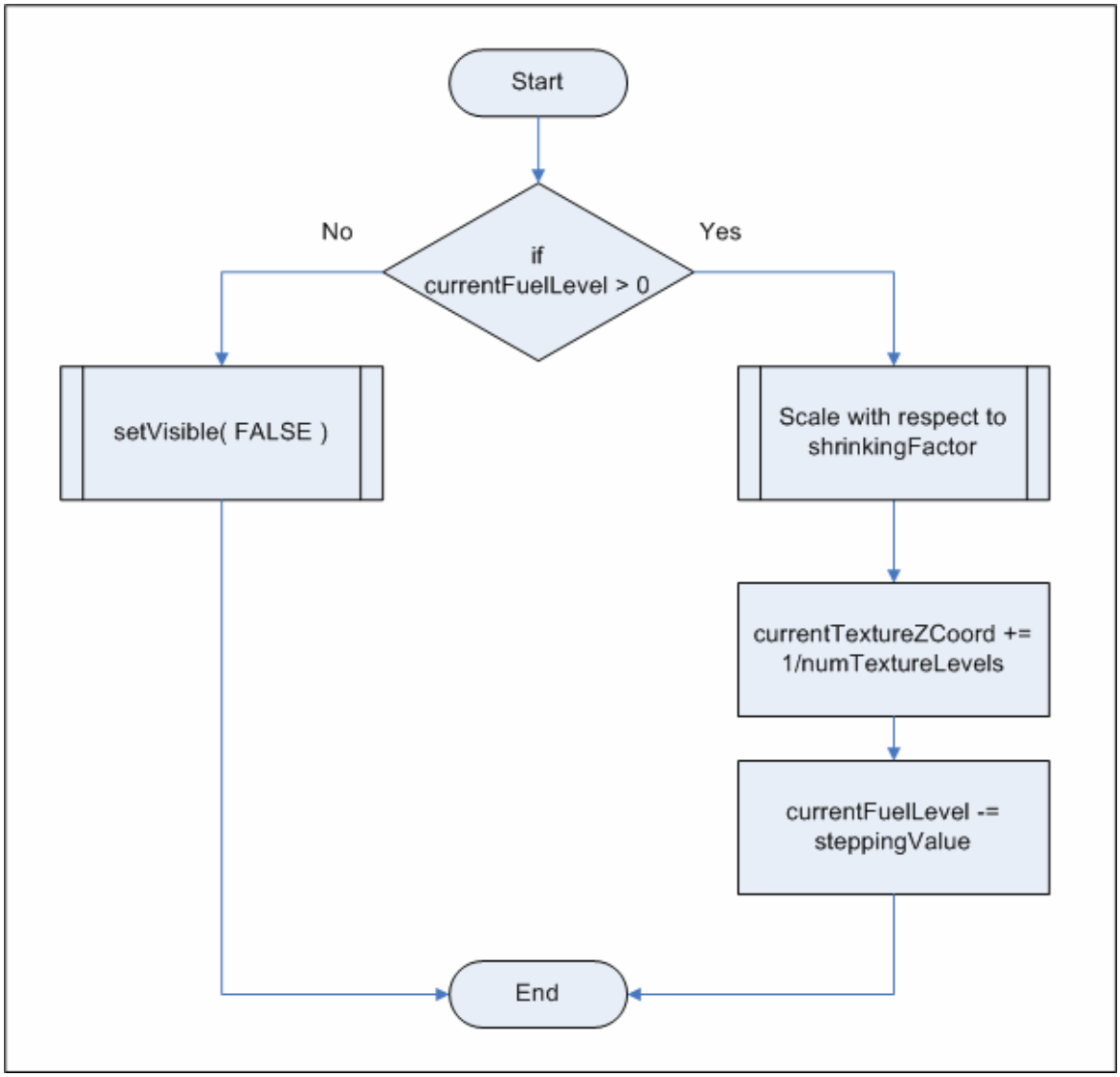

Fig. 5 The incrementPercentBurned() method from the BurnableMaterial class.

Figure 6 shows the details of the Emitter::update() method for a single particle.

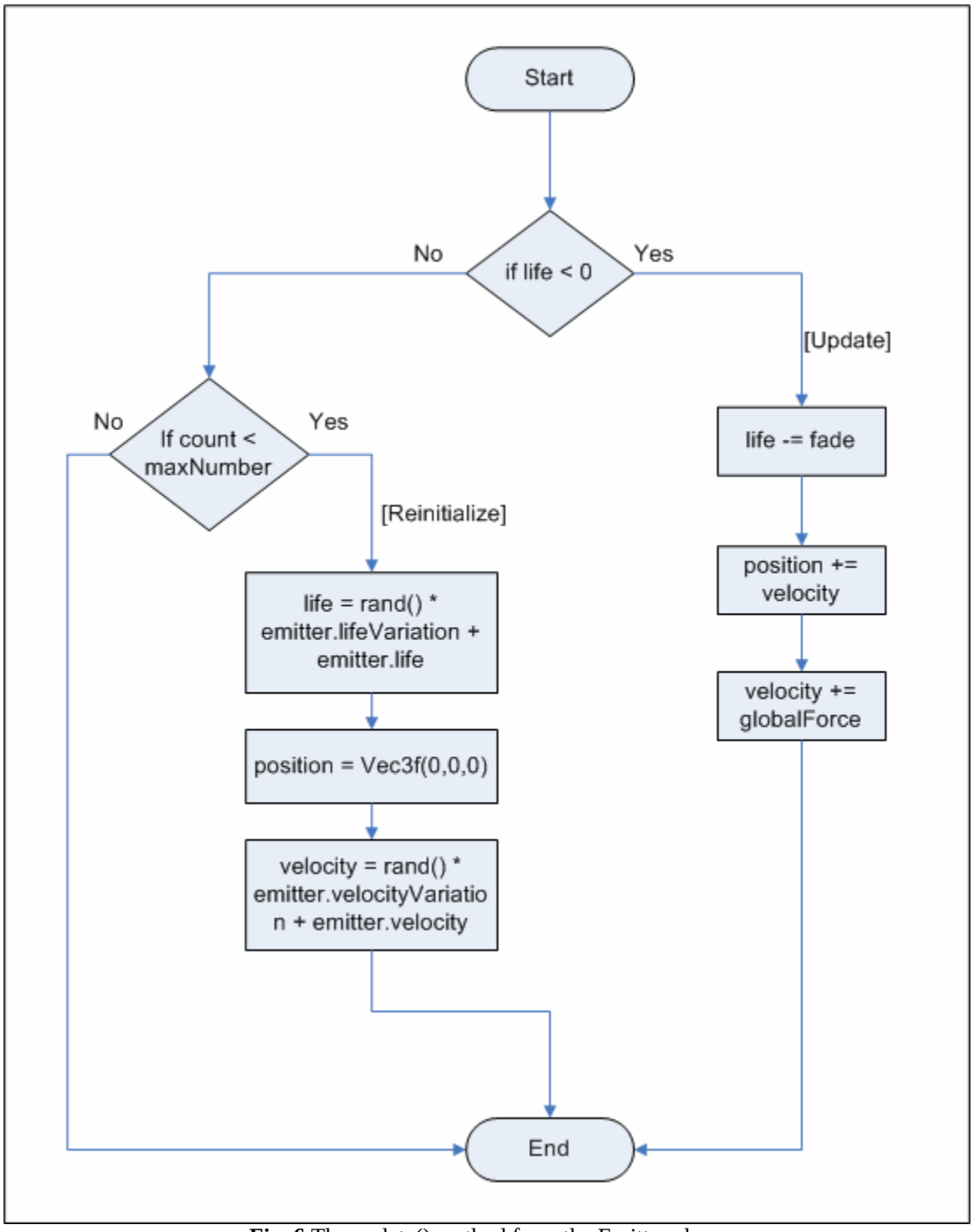

Fig. 6 The update() method from the Emitter class.

Figure 7 shows the details of the Terrain::addModel() method.

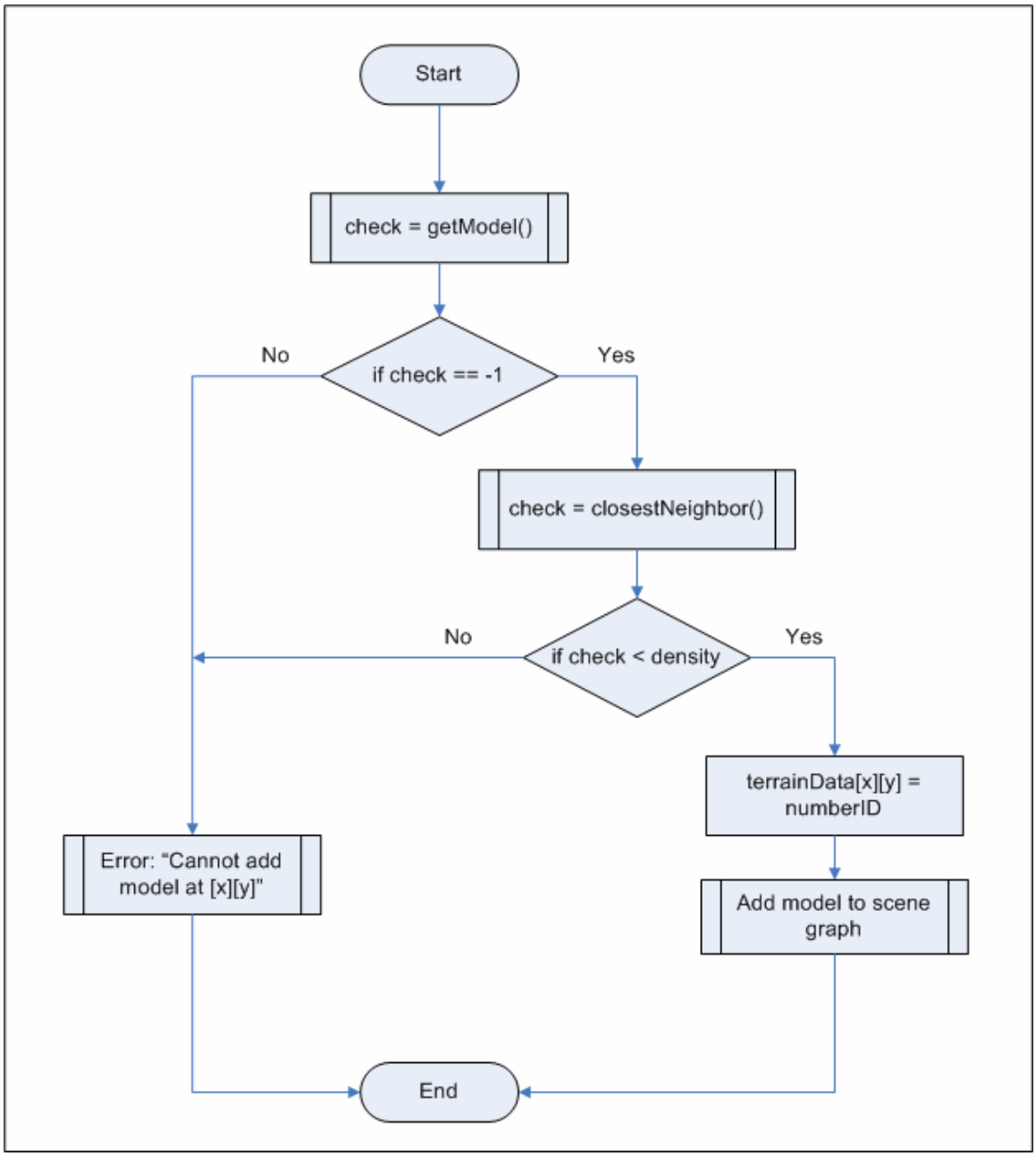

Fig. 7 The addModel() method from the Terrain class.

Figure 8 is a representation of scene data that would be rendered by OpenSG. The subsystems of V-FIRE manipulate this graph to fulfill their responsibilities.

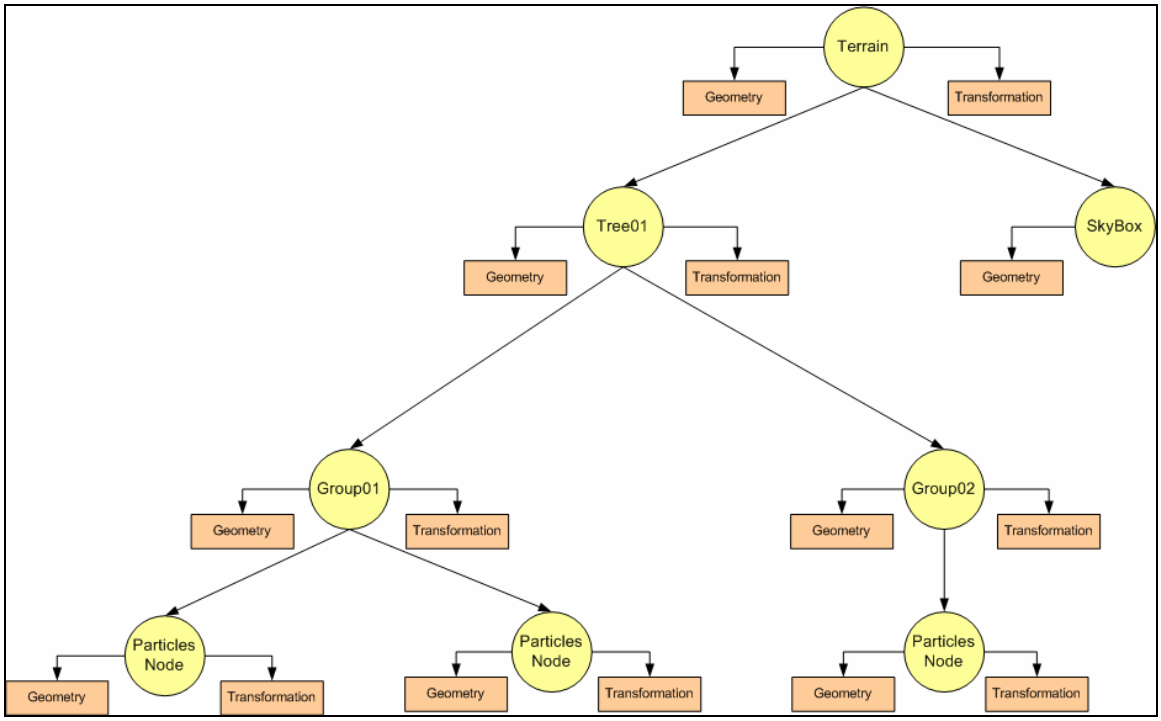

Fig. 8 An example scene graph.

User Interface Design

This section contains screen shots of a prototype for the V-FIRE user interface.

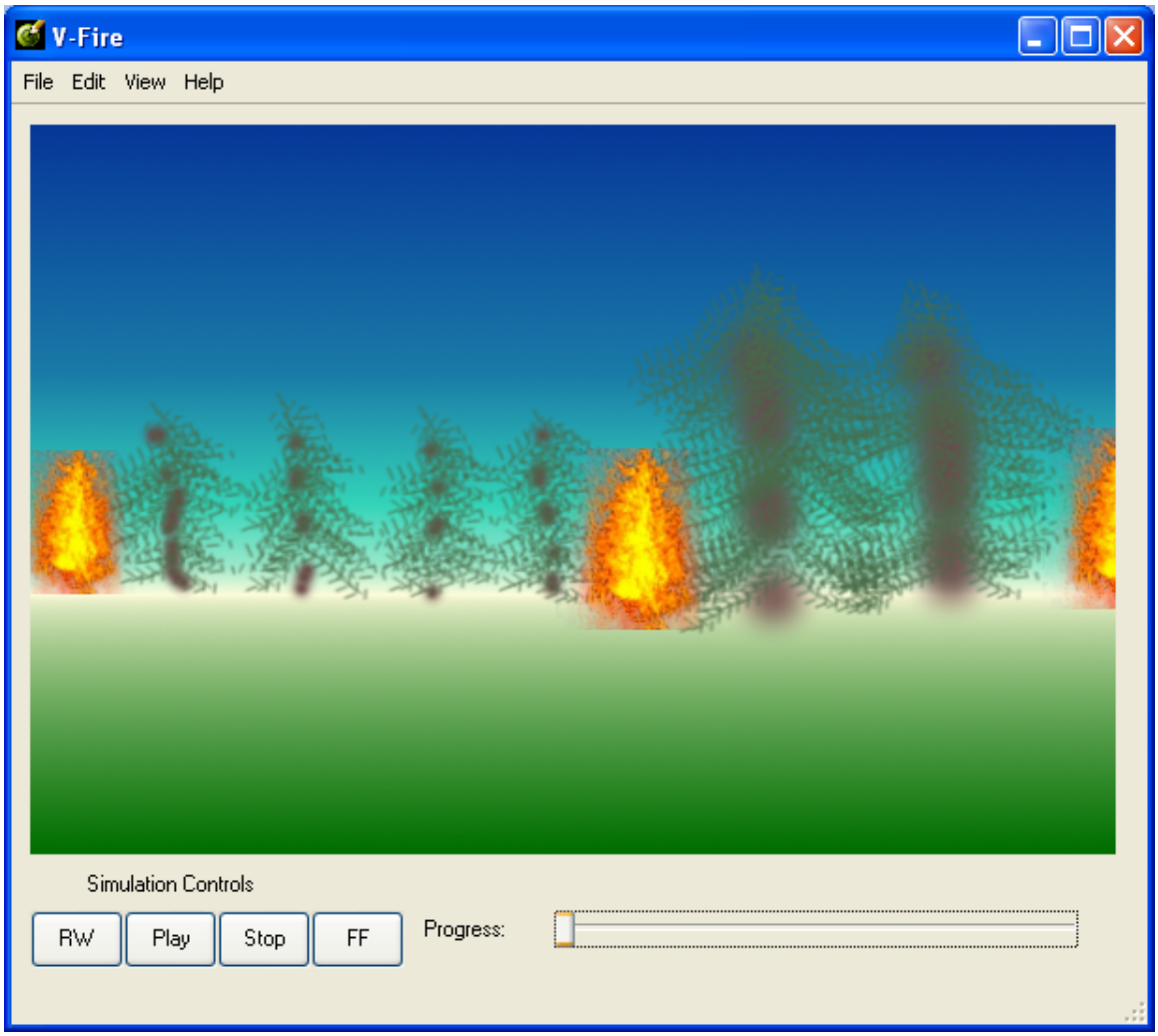

Fig. 9 A screen shot of the main simulation window. The user can visualize and control a simulation from the main simulation window.

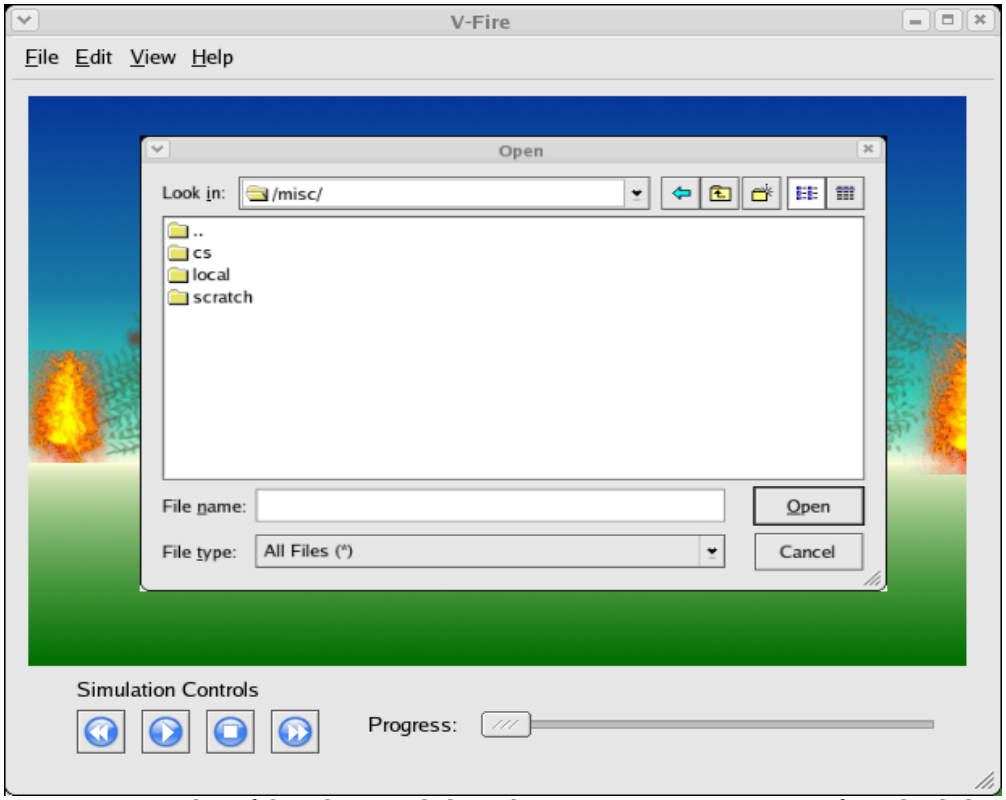

Fig. 10 A screen shot of the File Open dialog. The user can open terrain maps from the dialog.

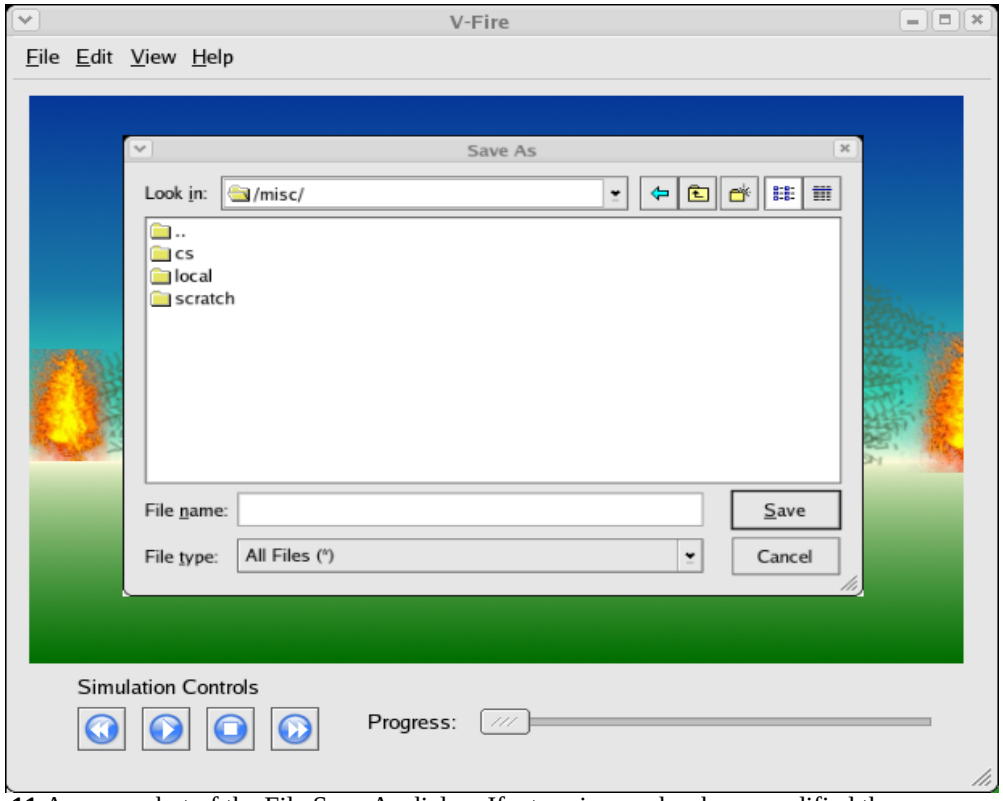

Fig. 11 A screen shot of the File Save As dialog. If a terrain map has been modified the user can save change through this dialog.

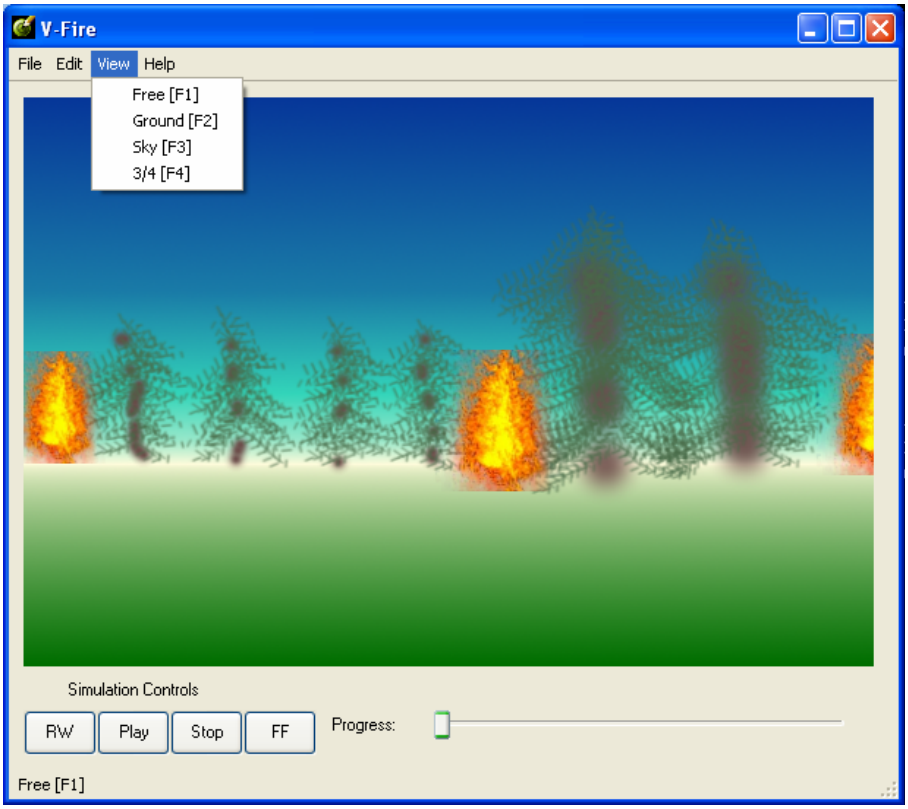

Fig. 12 A screen shot showing the View menu, providing different point of view presets.

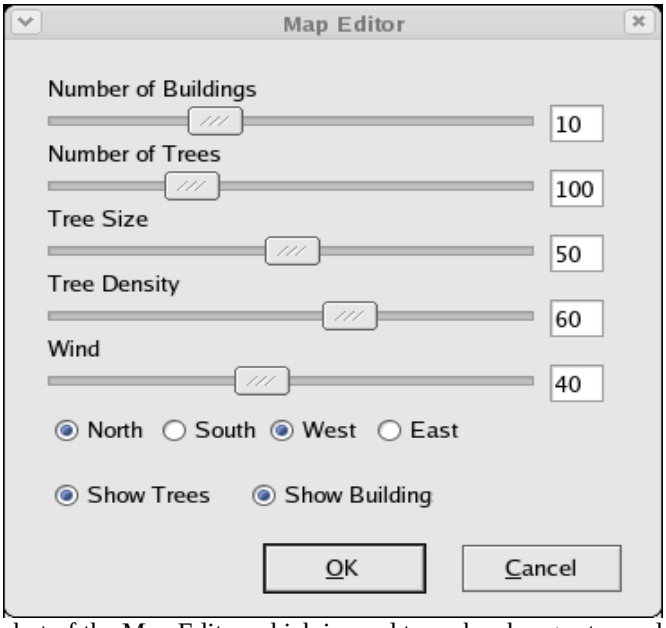

Fig. 13 A screen shot of the Map Editor which is used to make changes to models on the terrain.

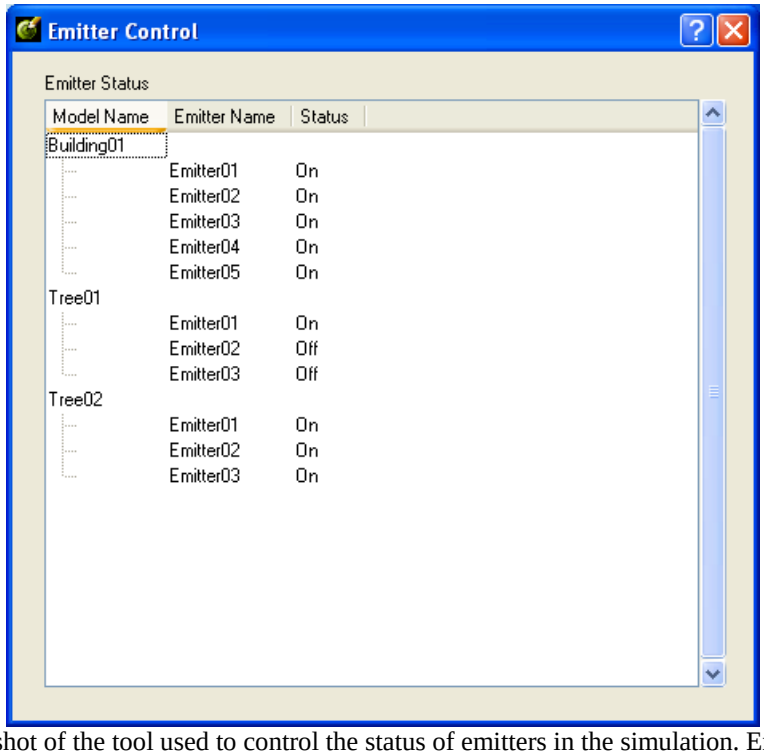

Fig. 14 A screen shot of the tool used to control the status of emitters in the simulation. Emitters control the flow of fire and smoke from an object.

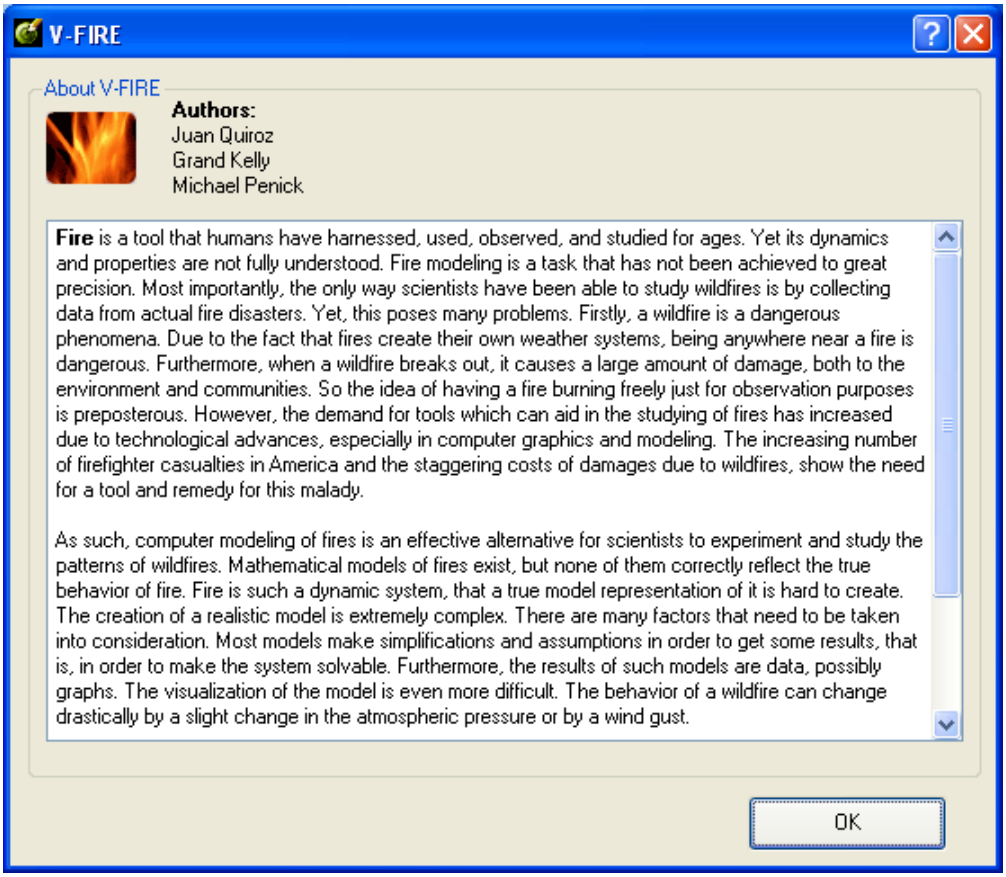

Fig. 15 A screen shot of the About dialog that provides information about the V-FIRE system.

References

Drysdale, Dougal. An Introduction to Fire Dynamics. New York: John Wyley & Sons, 2001.

The book An Introduction to Fire Dynamics, by Dougal Drysdale, discusses concepts, and ideas that are fundamental for the modeling and simulation of fire intended in V-FIRE. The modeling of fire must take into consideration numerous factors due to the chaotic nature of fire. The book discusses the following main topics: fire science and combustion, heat transfer, and limits of flammability and premixed flames, diffusion flames and fire plumes, steady burning of liquids and solids, ignition, spread of flame, spontaneous ignition within solids and smoldering combustion, and the pre-flashover compartment fire. On chapter 1, the author describes the physical chemistry of combustion in fires and the nature of fuels. This information will be used in order to be able to predict accurately how different objects ignite and burn, since such factors are dependent on the object composition. On chapter 3 the author describes the variation of burning velocity with experimental parameters. The experiments consisted of varying mixture composition, temperature, pressure, the addition of suppressants, and the effect of turbulence. This information is highly valuable due to the complexity of wildfires. Knowing how fire behaves with variations that are certain to be present on the simulation will help on determining how the fire will behave in the simulation. It will also help to make the fire realistic, without going too much into the details of fluid dynamics. Chapter 4 discusses the diffusion of flames and fire plumes. Several types of flames are discussed, including the characteristics of flames from natural fires.

Ahrens, James, James Bossert, Jon Reisner, Judith Winterkamp, Patrick McCormick. Case Study: Wildfire Visualization. 10 Feb 2005. <http://www.ccs.lanl.gov/ccs1/projects/Viz/pdfs/97wildfire.pdf>

This article describes the goal of creating a realistic simulation of wildfires, yet without compromising scientific data. The intended use of such a simulation is to be able to predict wildfire phenomena in order to reduce the cost of lives and damages. That is, to be able to forecast the evolution of a wildfire. The results described in the article state that the simulation runs slower than real-time, which is one of the main requirements for V-FIRE. It is also described that the volumetric rendering of smoke and fire is based on temperature data, taken from actual fires, which is inputted into the simulation. Topographic data from the sites of fires is also inputted into the simulation in order to create a polygonal mesh.

Enright, Doug, Duc Nguyen, Ron Fedkiw. Simulation and Animation of Fire and Other Natural Phenomena in the Visual Effects Industry. 5 Feb 2005. <http://graphics.stanford.edu/~fedkiw/papers/stanford2003-11.pdf>

This article describes methods used in the computer graphics industry for the simulation and animation of fire, smoke, and explosions. The methods described include the use of two-dimensional Euler equations, the Navier-Stokes model for incompressible flow, and a particle based animation of fire. All methods described would be of great use in our program, however, the math involved is quite advanced. Furthermore, having to compute the flow of fire with such methods in real time would create a large overhead. Furthermore, the results shown on the article are attractive and realistic, yet the differential equations used and the fluid model were too advanced for the team to understand.

Fernando, Randima. GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics. Boston: Addison-Wesley Professional, 2004.

This book describes advanced rendering techniques in the field of real-time graphics. The specific article "Vulcan Fire Simulation" outlines a method of creating very realistic fire in real-time. It specifically aimed at developers who might want to use this rendering method in games or real-time visualizations. The method describes uses volumetric data in the form of three dimensional textures which offers blending between frames of animation. The textures are placed on top of billboarding particles which are emitted off of a creature. Emitters are placed on three dimensional objects to simulate that they are on fire. This is one of the most influential articles that we will use to create the project.

Nguyen, Duc Quang, Ronald Fedkiw, Henrik Wann Jensen. Physically Based Modeling and Animation of Fire. 6 Feb 2005. <http://graphics.ucsd.edu/~henrik/papers/fire/fire.pdf>

This article describes the use of physically based modeling and animation of fire. It specifically focuses on the rendering of realistic fire, with little emphasis on smoke. Planck's formula is used to reproduce the color of fire, which computes the emitted spectral radiance. The model described can be used to render realistic "turbulent flames from both solid and gaseous fuels". Furthermore, the article provides good mathematical models to render fire. However, the model relies heavily on fluid dynamics. The simulations created include a flamethrower, a metal ball passing and interacting with a gas flame, and a flammable ball passing through a gas flame and catching on fire.

Contribution of Team Members

Grant Kelly
- High Level Design
- Detailed Design diagrams
- Design Document composition

Michael Penick
- Medium Level Design
- Class Diagram
- Class Method Descriptions
- User Interface Design

Juan Quiroz
- Introduction
- Medium Level Design
- Class Method Descriptions
- References

Glossary of Terms

Advection	The transfer of a property of the atmosphere, such a heat, cold, or humidity, by the horizontal movement of an airmass.
Billboarding	An efficient rendering method in which a textured quad-sided polygon always faces the camera to simulate a more complex three dimensional object.
Bouyancy	The upward pressure exerted upon a floating body by a fluid, which is equal to the weight of the body. This property affects the movement of fire upward through air.
Emitter	A three dimensional location where particles are spawned from. Emitters can be placed on objects to give the illusion of being on fire.
Flying Camera	A camera in which the point of view can be controlled by the user through a interactive hardware device i.e keyboard, mouse, or joystick.
Fire	The combustion of vaporized or gases fuel. The transformation of fuel into carbon based product yields an extreme release of energy in the form of light and heat.
Frustum	The part of a solid, such as a cone or pyramid, between two parallel planes cutting the solid, especially the section between the base and a plane parallel to the base. A Frustum is usually used in graphics to represent the visible part of scene as from a camera.
Frustum Culling	An algorithm or method of identifying objects not within the cameras view and omitting them from a scene before it is rendered.
Map	The culmination of different stationary objects in a scene the make up the landscape e.g. terrain, vegetation, buildings.
Mesh	A collection of polygons used to create a three dimensional surface.
Occlusion Culling	An algorithm or method of identifying non-visible objects to omit them from a scene before it is rendered.
OpenGL	A multi-platform software interface to graphics hardware, supporting rendering and imaging operations.
OpenSG	A multi-platform scene graph library built on top of OpenGL. It provides a higher level interface to graphics hardware aimed at developing highly complex three dimensional scenes.
Particles	An animated, transitory three dimensional point in space where geometric data is rendered usually in the form of a point, line, or billboard.

Point of View (POV)	The position from which a three dimensional scene is rendered.
Polygon	A plane based shape made up of three or more vertices. Polygons, along with lines and points, are the geometric primitives used by the OpenGL graphics system.
Real-time Graphics	A branch of graphics concerned with interactive and responsive display of complex two or three dimensional scenes.
Smoke	A carbon substance expelled form a burning body after the loss of fuel, especially from organic material such as vegetation.
Terrain	The physical layout and characteristics of a landscape.
Texture	A one, two, or three dimensional image that can be displayed on geometric primitives, usually polygons.
Thread	A light weight process used to simulate or execute with concurrency depending on the number of central processing units. In contrast to processes, threads share the same memory and data.
Turbulence	An eddying motion of the atmosphere that interrupts the flow of wind.
Vegetation	The plants of an area of region. Plant life such as trees, bushes, grass, etc.
Volumetric Rendering	A process of rendering or visualizing three dimensional data sets. The geometric primitive for volumetric rendering is a voxel as opposed to polygons or meshes which only describe a object's surface.
Vorticity	A measure of the spin of an air mass.
Voxel	An abbreviation for volume pixels. A voxel is a geometric primitive used to describe volumes.