RAMP Fire

Realistic Automatic Modeling and Propagation of Fire

A Maya Plug-in Development Tool

By:
John P. Falcone
Tim Pease

Based on:

Simulation and Visualization of Fire Using Extended Lindenmayer Systems
Tomasz Zaniewski and Shaun Bangay (2003)

Structural Modeling of Flames for a Production Environment
Arnauld Lamorlette and Nick Foster (2002)

CIS660: Advanced Topics in Computer Graphics and Animation
Instructor: Dr. Stephen Lane

PROJECT SUMMARY

The goal of this project is to design an interactive fire simulation development tool that will realistically propagate fire within a building and render the fire accurately. A tool of this nature has applications primarily in video game development and computer animation. Secondarily, it may be used as a fire safety tool to determine the safest parts of a building and safest evacuation routes in the case of a fire. In video game development it may be used to automatically propagate fire through a first-person shooter (FPS) level. As an animation tool it may be used to automatically animate a fire sequence. In the field of fire safety there is commercially available software endorsed by agencies such as the Building and Fire Research Laboratory (BFRL) and Fire Safety Engineering Group (FSEG). These tools create advanced simulations for a laboratory environment using techniques such as computational fluid dynamics (CFD). Our development tool will not provide this type of accuracy or data feedback for analysis. Therefore, this tool is not intended for direct use in a fire safety research laboratory. It is possible that this tool could be extended for these purposes.

We envision a technical artist using our tool, as it provides the artist with complete control over the environment. Our tool will provide the artist with the ability to create a custom building, or level, with user-friendly painting tools. The artist will have control over where the fire can propagate within this level. The artist will use a tool to paint fuel cells into areas of the level designating where fire can propagate. Finally, the artist will designate where the fire will ignite and then run the simulation to propagate the fire. A low-level of detail mode will grant the user the ability to simulate the fire propagation at low computational cost without realistic fire modeling. We project this mode will provide real-time, or near real-time, simulations. This will allow the artist to evaluate the propagation, quickly make adjustments to placement of fuel cells, and then re-simulate the fire propagation. Once fire propagation is to the artist's liking, a final production mode will be used to simulate the fire propagation with realistic fire modeling. We hope to achieve near real-time results in this mode as well. Run-time will be dependent upon the size of the level and number of fuel cells within it, and the level of realism desired by the artist for fire modeling.

The tool will be developed as a Maya plug-in using MEL scripting and the C++ API. Implementation of the fire propagation will be based on *Simulation and Visualization of Fire Using Extended Lindenmayer Systems* [Zaniewski and Bangay 2003]. The fire will be modeled using techniques from *Structural Modeling of Flames for a Production Environment* [Lamorlette and Foster 2002]. There are eight weeks allotted for development of the tool. The first few weeks will be dedicated to creation of a level editor tool and a paint tool for placing the fuel cells. Concurrently, implementation of the algorithms in the two papers mentioned above will begin. The next few weeks will be dedicated to completing and integrating the implementations of the two papers. The remainder of the time will be allotted for integration of the tools into one framework, debugging, and testing of the tool.

1. AUTHORING TOOL DESIGN

1.1. Production/Development Need

Simulation of fire and other natural phenomena has recently been the topic of much discussion within the field of computer graphics (CG). Many of these tools use computationally expensive techniques such as computational fluid dynamics (CFD) and the Navier-Stokes equations. Research focuses on optimizing these techniques to model and render correct behavior of fire. They do not aim to simulate the propagation of fire within an environment. Implementations of these techniques do not run at real-time and do not usually provide the user much control over the environment. Often, slight changes in the initial conditions of the system cause drastic differences in the fire animation.

Commercially available software endorsed by fire safety organizations such as the Building and Fire Research Laboratory (BFRL) and Fire Safety Engineering Group (FSEG) aims to accurately model propagation of fire. The tools endorsed by these organizations are advanced laboratory tools that are not concerned with accurate rendering and visualization of fire. They also have a relatively high cost, and are not suited for use by an artist.

There have not been many attempts at combining realistic fire propagation with realistic fire modeling. Veach et al [Veach et al 1994] used cellular automata to simulate the spread of fire in an outdoor environment. Lee et al [Lee et al 2001] propagate realistic fire in terms of its geometry, and Beaudoin et al [Beaudin et al 2001] use similar approaches. They are not able to provide breaking fronts of fire or determine where favorable conditions exist within the environment for the fire to propagate.

We intend to combine realistic fire propagation with accurate fire modeling for an indoor environment. We intend to use computationally inexpensive techniques to achieve real-time or near real-time results. We also place emphasis on providing an intuitive, user-friendly tool that grants the user control over the environment.

1.2. Technology

Two SIGGRAPH papers will be combined to develop our tool. Implementation of the fire propagation will be based on *Simulation and Visualization of Fire using Extended Lindenmayer Systems*, [Zaniewski and Bangay 2003]. In this paper, Zaniewski and Bangay use L-Systems to propagate the fire within a building. L-Systems have been extensively used for automatically modeling plants and emulating plant growth. L-Systems use a set of productions, or rules, to evolve a growth over time and an axiom to begin the growth. They can be made to be environmentally sensitive by allowing them to respond to parameters of the environment. In the case of plants, they may respond to available light in the environment. Zaniewski and Bangay adapt L-Systems for the use of fire propagation. They achieve realistic propagation by allowing the L-System to grow and respond to the environment (Figure 1). In particular, they respond to the amount of combustible fuel within each cell. They also use the L-System to update temperature for visual feedback of environment conditions.

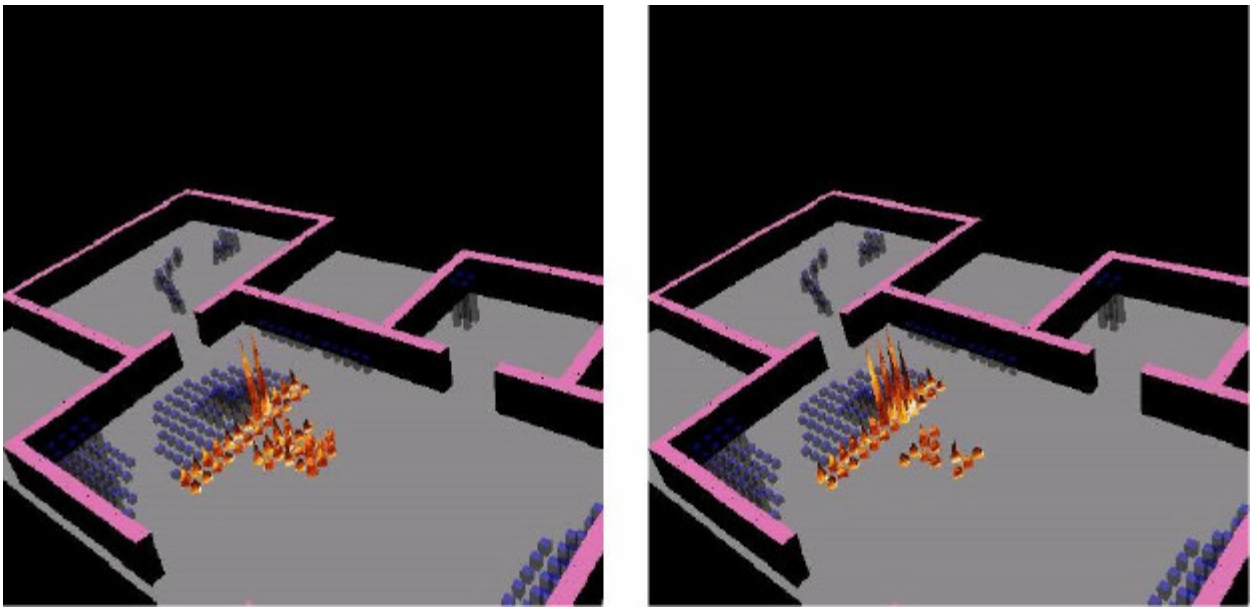


Figure 1 - Realistic fire propagation demonstrated by Zaniewski and Bangay. Note the breaking front achieved during propagation.

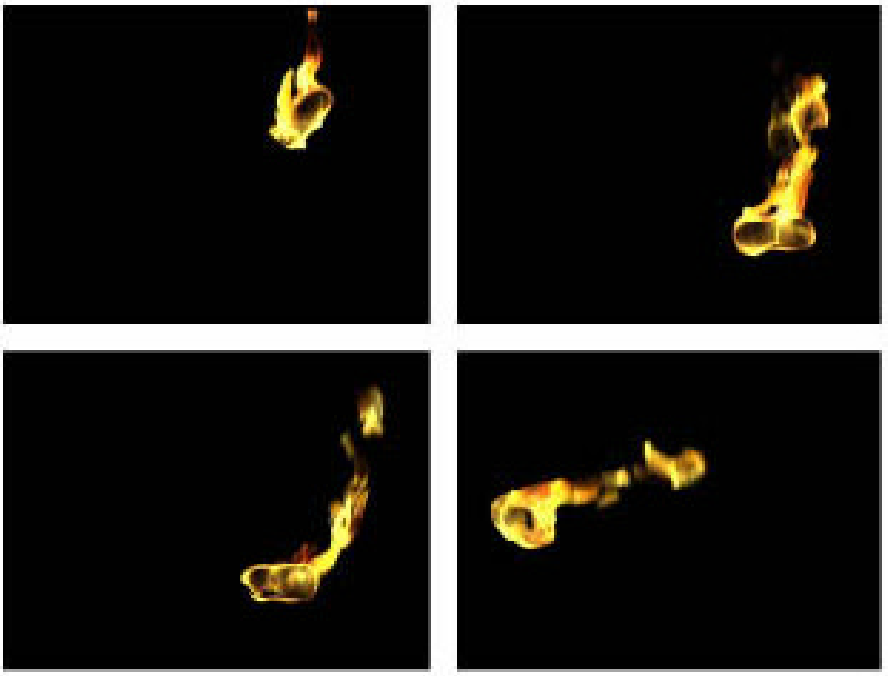


Figure 2 - Realistic fire produced with procedural techniques by Lamorlette and Foster. Note the flame flicker and breakage.

The fire will be modeled using techniques from *Structural Modeling of Flames for a Production Environment* [Lamorlette and Foster 2002]. In this paper, procedural techniques are presented for modeling and rendering fire. Lamorlette and Foster do not use highly computationally complex techniques such as CFD, yet they are still capable of creating realistic fire effects such as flame flicker and separation (Figure 2). The procedural fire is capable of being driven by an animator or by a physically based model. In our case, the L-System will drive the flames.

The papers we intend to implement use relatively inexpensive computational techniques and compliment each other. Zaniewski and Bangay achieve fire propagation that looks realistic, but do not put effort forth to create realistic fire. They model each element of fire as a pyramid with a texture map. While Zaniewski and Bangay lack realistic fire, Lamorlette and Foster create it. Therefore, we will use Zaniewski and Bangay's L-System fire propagation and drive the procedural fire with it, thus creating one fully integrated tool.

1.3. Design Goals

Our tool will realistically propagate fire within a building and render the fire accurately. There will be an emphasis on creating a tool that has a balance of interactive user control and automatic simulation. The user will be able to interactively create and modify a building, or level. They will have interactive control over where the fire ignites, where the fire can propagate, and how intense the fire will be in those areas. The simulation will automatically propagate the fire and model the fire realistically. We intend to use computationally inexpensive techniques to achieve real-time or near real-time results. This tool will be developed as a Maya plug-in using MEL scripting and the C++ API.

1.3.1 Target Audience.

We expect our main audience to be technical artists in the fields of game development and CG. Since this tool will be developed as a Maya plug-in, it is likely that a technical artist will already have familiarity with the platform (intrinsic features such as pan, zoom, and rotate will be used). We will be taking into consideration a user who has no experience in using Maya. It will be a user-friendly and interactive tool that does not require skills in 3D modeling. A possible secondary audience is researchers in the field of fire safety. It is not likely that our tool will be used in this capacity since it will lack the precision that exists in already available commercial software. Commercial software endorsed by professional agencies use CFD to precisely model fire propagation based on a wide variety of parameters such as temperature, moisture, pressure, and other physically based elements.

1.3.2 User goals and objectives

A technical artist will use this tool to create an indoor environment and then simulate fire propagation within it. This may be used for creating game content or a special effect. We hope to achieve near real-time results.

1.3.3 Tool features and functionality

The technical artist will have access to a level editing tool. This will grant the artist complete interactive control over design of the environment. This component is not the main focus of our tool and is not a part of the papers we are implementing. Much time could be spent on creating very advanced level editing tools. Instead, we will create a relatively basic level editor while noting that this component could be easily extended into an advanced tool.

The technical artist will also have access to a fuel cell painting tool. This will be used to designate areas of flammability within the level. As the artist paints with this tool, fuel cells (likely to be denoted by cylinders) will be placed into the environment. Painting over cells will increase the amount of fuel, thus increasing the height of the cylinder.

The technical artist will have a selection tool that will be used to designate the points where the fire will ignite.

Finally, the artist will have either toggle control, or slider control, over level of detail to render during simulation run-time. This will give the user faster editing control over the fire propagation. Low-level of detail can be used to create proxy simulations of the fire propagation with simple pyramid shaped flame elements. Editing of the fuel cell placement can then be conducted. For final rendering, production level detail will be used to model each flame element.

1.3.4 Tool input and output

Input to our tools will be provided via keyboard and mouse. The level editor component will initially require keyboard input to determine the level dimensions. The walls will be painted with a tool controlled by a mouse. The fuel cell paint tool, ignition point selection tool, and level of detail adjustor will be controlled with mouse input. Each tool will be selected by clicking on its corresponding button to initiate a dialog or begin using the tool. A button will be pressed to initiate the simulation. The user will also be able to pan, zoom, and rotate around the level using standard controls offered by Maya.

The tool will output a 3D simulation of fire propagation within the level created by the user. The output may be a series of key frames that are hardware rendered by Maya. We may also investigate how to create an MPEG output for the purposes of demonstration.

1.4. User Interface

The user interface will be a tool box of buttons. They will be used to initiate a dialog to create a new level, select each tool, and initiate a simulation. A proxy GUI appears in Figure 3 on the following page.

1.4.1 GUI Components and Layout

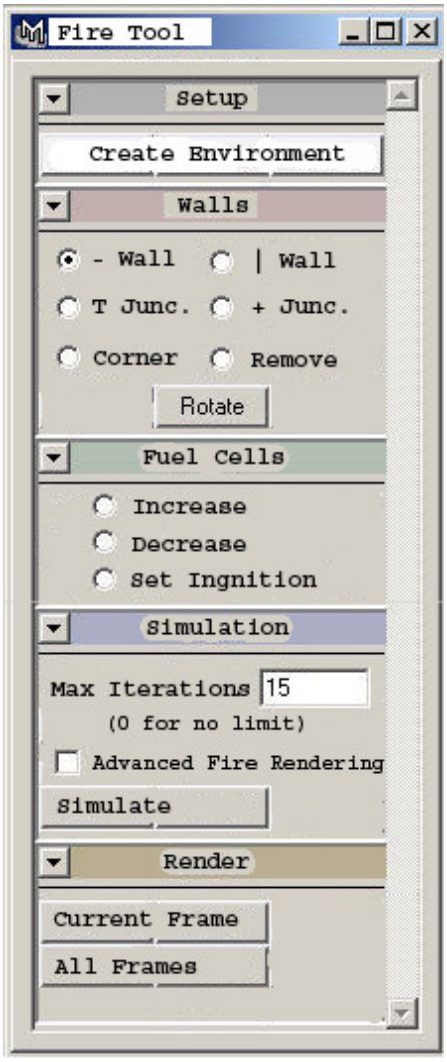


Figure 3 - A proxy layout of our GUI within Maya

The proposed GUI appears in Figure 3 to the left.

The "Create Environment" button will initiate a dialogue prompting the user for level dimensions. After return from this dialogue an empty level with those dimensions will be created.

The wall tools will be used for general editing of the environment. A variety of walls can be selected via radio buttons. The corresponding wall will then be placed, or painted, onto the level by clicking on the desired location within the level. The rotate button will be used to rotate a selected wall unit at increments of 90 degrees. Selecting the remove tool then clicking on an existing wall will remove that wall.

The "Fuel Cells" section has three paint tools. "Increase" and "Decrease" will be used to add or subtract flammability from a grid square, and the "Set Ignition" tool will be used to select the ignition point of the fire.

In the simulation section, the user will select the desired number of iterations for the fire simulation. If zero 0 is entered, the simulation will continue until its natural end (when all fuel has been consumed). Checking the "Advanced Fire Rendering" box will enable the advanced rendering of fire. If not checked, simple pyramids will be drawn for flame elements. The simulate button initiates the simulation.

The render section allows the user to render either the currently selected frame, or all frames of the simulation. In order to render, the user must have performed the simulation.

This is only a proxy layout of our GUI. This proxy shows the general types of tools that will be available, sections within the GUI, and the types of buttons we will use. We anticipate the GUI layout to evolve over time.

1.4.2 User Tasks

We intend to create a tool that allows user-friendly and interactive design. The user will not be required to have any knowledge of the physics of how fire

propagates. Nor will the user be required to have any knowledge of animation techniques, skills in 3D modeling, or a background in Fine Arts. While these skills are not necessary, familiarity with the Maya development platform will be beneficial as intrinsic Maya controls such as pan, zoom, rotate, and frame will be available. Since our tool is targeted mainly at technical artists in the field of CG, they will most likely already have experience with the Maya development environment.

1.4.3 Work Flow

Work flow will proceed as in Figure 4 below. The artist will begin by initiating a dialogue to create a level. In this dialogue they will enter the dimensions of the level. Then, the artist will proceed by designing the level by painting wall elements into the level. Next, the user will paint fuel cells into the level. Fuel cells will be denoted by cylinders. Painting over a fuel cell will increase its height to represent an increase of fuel within that cell. Finally, the user will designate the ignition point for the fire and begin the simulation. The user will be able to run the simulation in a low level of detail mode for faster run-time. This will allow the artist to evaluate the propagation, quickly make adjustments to placement of fuel cells, and then re-simulate the fire propagation. Once fire propagation is to the artist's liking, a final production mode will be used to simulate the fire propagation with realistic fire modeling.

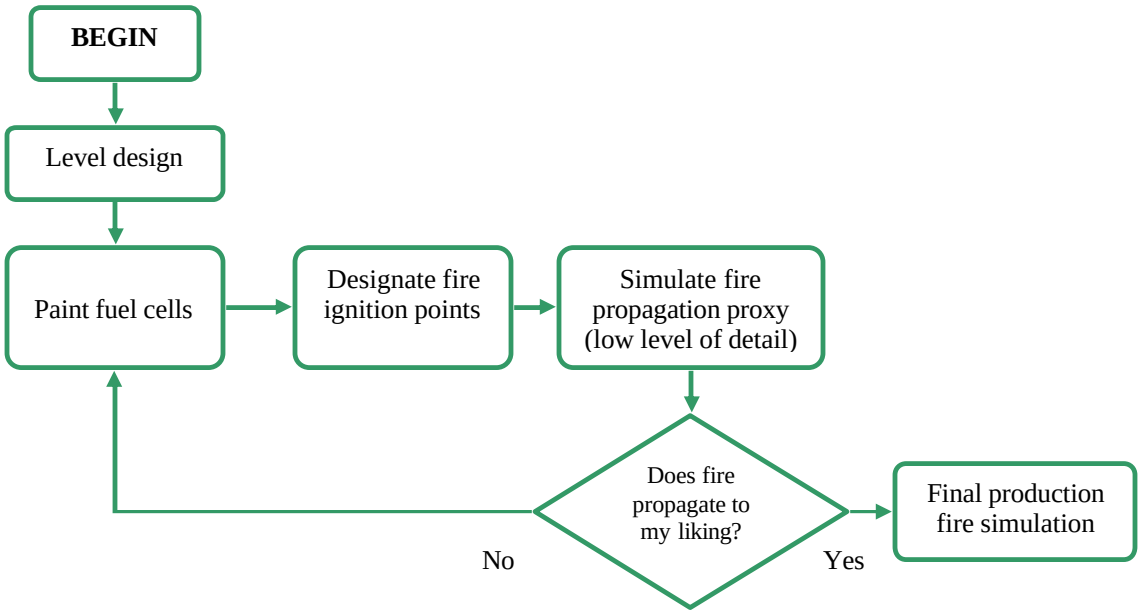


Figure 4 - Work flow chart

2. AUTHORING TOOL DEVELOPMENT

2.1. Technical Approach

2.1.1 Algorithm Details

Details regarding the level editor will not be discussed since this is not the main focus of our tool. This component is merely an editing tool that will be used to create data structures containing information about the environment.

The fire propagation algorithm will rely on L-Systems. An L-System contains three parts: an axiom, an alphabet, and rules (also called productions). The main idea of L-Systems is to use rewriting rules to generate a complex structure by replacing parts of a simple object. It is probably best to illustrate an L-System using an example.

Given:

Axiom: ab
Alphabet: {a, b}
Productions
p1: a -> ab
p2: b -> a

The characters to the right of the -> symbol replace the character to the left. This L-system would evolve as follows:

Iteration
n = 0 : ab
n = 1 : aba
n = 2 : abaab
n = 3 : abaababa
. . .

This L-System builds a new string at each new iteration n by traversing the string breadth first and applying each new production rule. Zaniewski and Bangay represent their L-System using a connectivity graph. They traverse the graph depth first to a desired depth, creating new productions along the way. They remark that this representation saves memory as the L-System evolves. In the simple example above, we can see how the breadth first L-System continually grows resulting in larger memory needs. This could be a problem in a large environment. Figure 5 on the following page shows an example of their graph representation. It shows an axiom, a production, and the resulting L-System after a traversal of the graph. Zaniewski and Bangay do not substantiate how this representation and traversal provides better performance, so we may chose to implement a breadth first traversal. We will experiment with each technique.

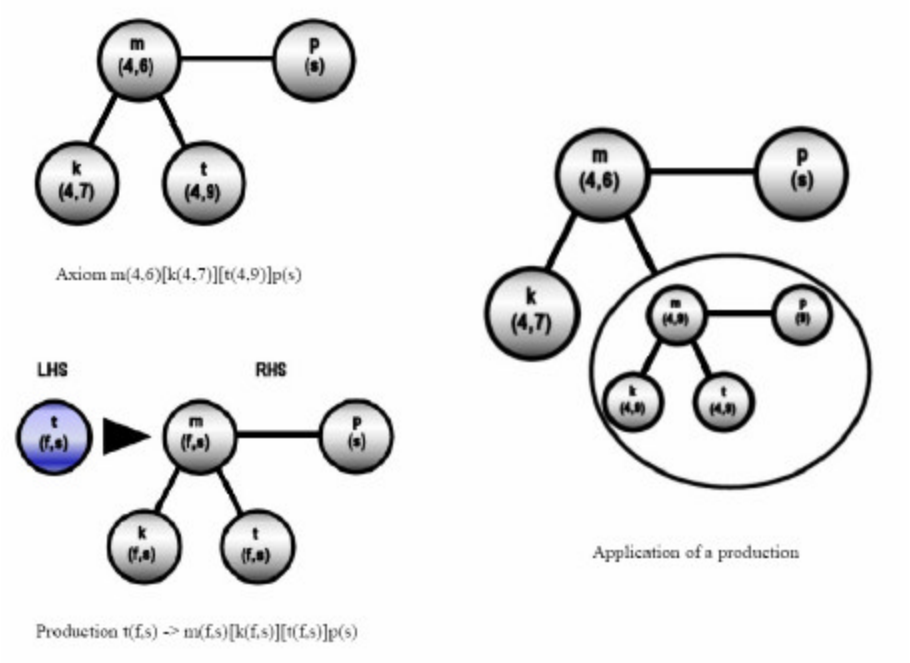


Figure 5 - L-System evolution using a connectivity graph

Zaniewski and Bangay extend their L-System in two ways. First, they use dedicated symbols for reading and writing from the environment. When these symbols are encountered they call the appropriate function and update the L-System or the environment. The second extension is the use of parametric L-Systems. In parametric L-Systems, each character carries a set of parameters that are used to control production. In their case they use the x, y, and z indices of their voxel space. This determines the location of the fire element within the level. A flow chart of the system appears in Figure 6 below.

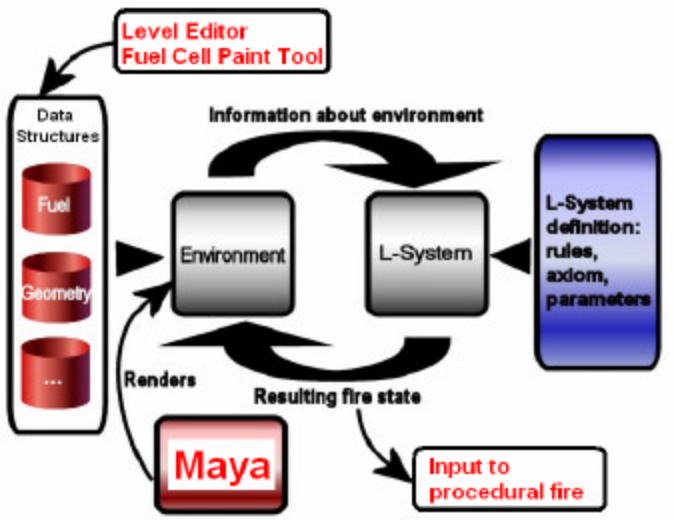


Figure 6 - The L-System continually reads from the environment and then updates it using dedicated symbols. The system is initialized by creation of a data structure through the use of a level editor and fuel cell paint tool.

We plan to make some modifications to Zaniewski and Bangay's system. To simplify the model and reduce data structure size, we are going to restrict our buildings to single level spaces. As a result, the symbols will need to carry one less parameter, i.e. one less index, as it evolves. This will remove the ability for fire to climb vertically similar to Zaniewski and Bangay's demonstration of fire climbing a wall.

In addition, we notice that Zaniewski and Bangay mention they receive "strange timing results". Early in the simulation, it takes longer for each new evolution of the L-System. As they get deeper into the simulation, each evolution begins to take less time despite the fact that the graph has grown. This is attributed to burned-out cells that no longer require flame elements to be rendered but still remain as nodes in the graph. We may attempt to exploit this fact using a breadth first traversal. We will create a production that replaces a burned out fuel cell with null, thereby removing it from the L-System. As the L-System grows, we will be removing burned out cells, helping to counter the quickly growing L-System and reduce memory needs.

One final simplification is the production rule that updates temperature. In their productions, they spend time writing temperature into each cell as a separate parameter within their data structure. Temperature is updated by multiplying 1.5 times the current fuel amount. They do not, however, use temperature as a physical feature that controls a production rule in the L-System. It is just a state variable that could be provided as data feedback in the form of an additional visualization mode. They do not demonstrate this visualization mode. Since we do not have any use for visualization of this parameter, and since it can be easily calculated from remaining fuel (1.5 times current fuel), we will remove this from the production rules. We will also experiment with creating other new production rules that propagate fire in multiple directions simultaneously.

The other main algorithm in our tool will follow the steps outlined by Lamorlette and Foster for creating flame elements in a procedural manner. The steps are outlined as follows and appear in Figure 7 on the following page.

1.) A parametric curve is created which will be used as the spine of the flame element.

2.) The curves evolve over time according to a combination of physics-based, procedural, and wind fields. Physical properties are based on statistical measurements of natural diffusion flames. The flame spine may also be re-sampled to ensure continuity.

3.) The curves can break, generating independently evolving flames with a limited lifespan. Engineering observations provide heuristics for both processes.

4.) A cylindrical profile is used to build an implicit surface representing the oxidization region, i.e. the visible part of the flame. Particles are point sampled close to this region using a volumetric falloff function.

5.) Procedural noise is applied to the particles in the parameter space of the profile. This noise is animated to follow thermal buoyancy.

6.) Steps 1- 3 can be viewed as one independent unit. Steps 4 and 5 can be viewed as a second independent unit. In this step the output of step 3 is used to transform the particles from step 5 into the parametric space of the curve from step 3. A second level of noise, with a Kolmogorov frequency spectrum, provides turbulent detail.

7.) The particles are rendered using either a volumetric, or a fast painterly method. The color of each particle is adjusted according to color properties of its neighbors, allowing flame elements to visually merge in a realistic way.

8.) To complete the system, we define a number of procedural controls to govern placement, intensity, lifespan, and evolution in shape, color, size, and behavior of the flames.

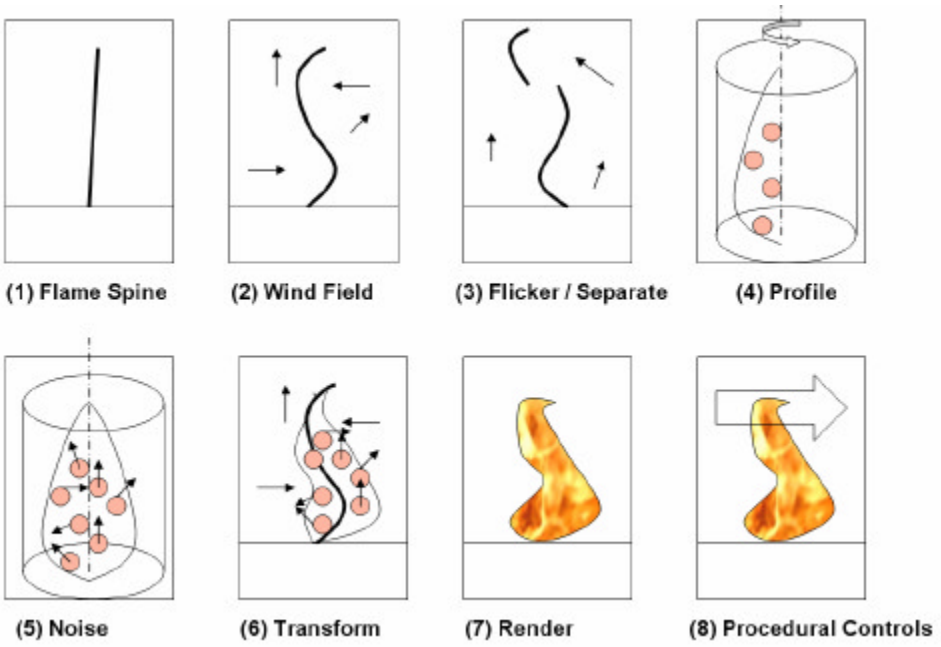


Figure 7 - The 8 step process for creating realistic fire outlined by Lamorlette and Foster

The procedural properties governing step 8 will be received from the environment data structure, which will be continually updated by the L-System. The environment variables will guide placement and intensity of the flame elements.

Intensity will be based on the quantity of fuel remaining in the cell. Size will be contained to one grid cell. Evolution in shape may be driven by the wind field and the change in intensity will result from fuel being consumed.

2.1.2 Maya Interface and Integration

Level Editor / Paint Tool:

The Level Editor and painting tools will be scripted mainly using MEL. We will need to make use of a plug-in programmed in C++ through the Maya API in order to store information about our environment and the fuel cells in a data structure. The environment data structure will be shared with the L-system simulator and fire visualization modules. The level editor will use basic Maya primitives, such as boxes, cylinders, and planes, to draw the environment.

L-system module:

The L-system module will be written using the C++ API. This module uses the environment data structure that was created via the level editor. Updated environment information generated by this module will be used by the fire visualization module. This module will not need to communicate with Maya; it is a stand-alone component.

Procedural Fire Module:

The procedural fire module uses the parameters of the environment data structure to model and render fire. We may use Maya's particle functions within this module, but this is still to be determined. If advanced fire rendering is turned off, this module will use Maya's cone or pyramid objects to represent the fire (along with a procedural shader). This module will be programmed using the Maya C++ API.

2.1.3 Software Design and Development

We plan to use the following data structures to share information between the modules and to perform some of the algorithms.

Map array

We will store information about the environment in a 2-dimensional array. Each element in the array will be a pointer to a map element. Zaniewski and Bangay used a voxel space, but we are simplifying our implementation to single level structures, thus requiring only two dimensions. The modules that will make use of this data structure are:

- Level Editor: The Level Editor will add a new map element (either a fuel cell or a wall unit) to the array whenever one is added to the map.

- L-System: The L-system module will access this array to run the propagation simulation. At each new epoch, it will sense parameters about the environment and then update the environment.
- Procedural Fire: The Procedural Fire module will use the L-System to guide parameters of the fire. The L-System, as stated in the algorithm details section, will be a parametric L-System. This means each symbol will carry parameters with it. In our implementation, the symbols will carry the indices that represent the element's x and y coordinates.

Map Element

The map element data structure will represent an object in the environment. The possible types are "wall" and "fuel cell". The data structure will be capable of holding other information about the element, such as quantity of fuel, temperature, and any other information we decide is necessary. It will be used by the following modules:

- Level Editor: The Level Editor will create the map element objects and add their pointers to the map array whenever the user adds an element to the environment.
- L-System: The L-system module will read information about each element on the map to determine its type and the corresponding data. It will update the temperature of
- Procedural Fire: The Procedural Fire module will access the temperature of each fuel cell and determine the size of the flame.

L-System

Although Zaniewski and Bangay use a connectivity graph to represent each epoch of their L-System, we may use a string. Alternatively we may use a linked list. We plan to implement breadth-first traversals of the L-System and use productions to remove symbols as fuel burns out. In this manner our L-System growth will be curtailed. The L-System will contain the alphabet, and productions. The axiom will be created by the user when they determine the fire ignition point.

B-Spline

An object will be created to store the necessary data for the B-spline used in the procedural modeling of fire. A B-spline is used to represent the base curve, or spine, of each flame element. This object will only be used by the Procedural Fire module.

Implicit Surface Object

An object will be created to store data of a revolved implicit surface that is used to represent the surface of the flame. Kolmogorav noise will be added to this surface to give the surface an irregular shape. The details of the data storage needs of this object are currently under investigation. This object will be used by the procedural fire module only.

2.2. Target Platforms

2.2.1 Hardware

The hardware requirements for running our tool are equivalent with the hardware requirements for running Maya. More memory and faster processing speeds will decrease computation time and speed up rendering, but are not necessary to make use of this tool. The requirements for running Maya as are the following:

- Windows: Intel® Pentium® III or higher, or AMD Athlon™ processor
- 512 MB RAM
- CD-ROM Drive
- Hardware-Accelerated OpenGL® graphics card
- 3-button mouse with mouse driver software
- 450 MB of hard disk space

Due to the memory requirements of the L-system, we recommend more than 512 MB of RAM.

2.2.2 Software

The tool is intended to run on a Windows platform. Since this tool will be developed as a Maya plug-in, it will require the user to own a copy of Maya 6.5 or greater.

2.3. Software Versions

2.3.1 Alpha Version Features (first prototype)

The alpha version will consist of three stand alone components. First, we will have created a Maya based level editor that includes a tool for painting fuel cells into the level. It will be capable of generating a data structure containing information about the environment, e.g. location of walls, fuel cells, amounts of fuel in each cell. As this is not the main focus of our project, we will need to have this component completed in the early stages of development (within the first two weeks). The second alpha version component will be a C++ stand alone project that demonstrates the propagation of fire using L-Systems that respond to the environment and renders the simulation using OpenGL. The third alpha version component will be a stand alone C++ project that implements procedural fire and renders it using OpenGL. It is likely that the alpha version of this component will use coarse sampling and not appear realistic. It will be important to have the fire propagation and procedural fire components completed near the midway point of development. This is because the following few weeks will be dedicated to converting them into Maya C++ API plug-ins and integrating them with the level editor component.

2.3.2 Beta Version Features

The beta version will include fire propagation and procedural fire integrated as a Maya C++ API plug-in. They will work in concert with the level editor and fuel cell paint tool. It will be the first integration of all components. It may not have full functionality. The beta version is scheduled for completion prior to the last week of development. The remainder of time will be used for testing and debugging.

2.3.3 Description of any demos or tutorials

Although our tool will be user-friendly and intuitive, we may provide a user's manual for installation instructions and basic instructions for use. In addition, or as an alternative, we may create a short training tutorial. The tutorial will be an MPEG demo showing level design, fuel cell painting, choosing the fire ignition point, and starting a simulation. The tutorial will also show a sample simulation. Alternatively, we may show a live in-class demonstration of our tool being used in the Maya development environment.

3. WORK PLAN

3.1. Tasks

We have broken down our project into seven distinct tasks.

Task 1: Level Editor (John)

The level editor is the basic editing environment necessary to initialize a simulation. It will provide an interface for the user to design a floor plan quickly and easily. We may use Alias' Level Tools plug-in as a guide and extend it for our purposes. This module will be implemented mainly using MEL scripts to create the level model and the C++ API to generate the data structures.

Task 2: Paint Tool (Tim)

The paint tool is a tool that allows for easy "painting" of fuel cells. By clicking somewhere on the floor plan the user can quickly place fuel cells (represented by cylinders). Painting over an area already occupied by a fuel cell will add fuel to that cell, increasing the flammability of that spot (represented by increasing the height of the cylinder). This will be implemented mainly using MEL scripts to place the fuel cells and the C++ API to update the data structures.

Task 3: Platform Integration (John and Tim)

The third task is to integrate the level designer and the paint tool into one tool. The user should be able to create and edit a floor plan, and then place fuel cells quickly and easily from within one user interface (UI).

Task 4: Lindenmayer System (John)

The L-system will be written in C++ using the Maya plug-in API. This task involves implementing the L-system described in *Simulation and Visualization of Fire using Extended Lindenmayer Systems* [Zaniewski and Bangay 2003]. This module will read information from data structures describing the environment. The evolution of the data structure will be handled by a repeated cycle of read and write phases. The L-system will read the data structures, the fire propagation will evolve, and then the system will update the data structures. The Procedural Fire Module will have access to the environment data structures and will use the data to procedurally create fire.

Task 5: Procedural Fire Module (Tim)

This module will also be programmed in C++ using the Maya API. This module will have access to environment data structures and each epoch of the L-System. The symbols of the L-System will carry parameters that give the indices representing the map cell on which we are working. The map cell will contain data such as the amount of fuel remaining in that cell. The procedural fire will use the location and fuel parameters to guide its development. Procedural techniques outlined in *Structural Modeling of Flames for a Production Environment*, Lamorlette and Foster [2002] will be used to model and render each flame element.

Task 6: Integration of modules (John and Tim)

The sixth task involves integrating the three main components: the Level Editor and Fuel Painting Module, the L-system Module, and the Procedural Fire Module. The channels of communication between the modules are:

- Level Editor UI to the L-system module (preferences set by user)
- The current scene to the L-system module (size and location of fuel cells)
- The L-system module to the Fire Modeling and Rendering module (temperature, location of flames)
- Procedural Fire module to the scene (rendering fire at each frame)
- Level Editor UI to the Fire Modeling and Rendering module (rendering coarseness settings)

Further Breakdown of tasks 5 and 6:

We further break down tasks 5 and 6 into sub-tasks, which will aid in the development of these modules:

1. Researching the concepts and algorithms necessary to build the module.
2. Programming a stand-alone version outside or inside of Maya to get a grasp of the necessary algorithms and data structures needed.

3. If not completed in part 2, migration of the module to the Maya API.
4. Preparation for integrating with other modules.

Task 7: Testing/Debugging (John and Tim)

The final task will be general testing and debugging of the integrated plug-in, as well as writing all necessary documentation and creating any demonstration material.

3.1.1 Alpha Version

The planned Alpha version of our tool involves completion of tasks 1 through 5. It is the first functioning version of the tool, but the modules remain separate. It will be important to have the fire propagation and procedural fire components completed near the midway point of development. This is because the following few weeks will be dedicated to converting them into Maya C++ API plug-ins and integrating them with the level editor component.

3.1.2 Beta Version

The Beta version of the simulator will require completion of task 6. The beta version should be a fully functional, integrated tool. The interface will not be refined at this point, and we estimate that the integration of the modules may not be seamless. We plan to complete the Beta version at least one week prior to the deliverable date. This is because the remainder of the time will be needed for debugging, testing, and documentation.

3.1.3 Final Version

The final version involves completion of task 7, as well as further refinement of each module. The interface should be refined to a point where an end-user can quickly and easily create a scene, paint it with fuel cells, and run a fire simulation. This version will be deliverable by April 24, 2006.

3.2. Schedule

The following is our proposed schedule of tasks over 8 weeks. The time table shows when each task should be completed. Milestones are indicated by an asterisk (*). Tasks marked by T, are tasks to be completed by Tim. Tasks marked by J, are tasks to be completed by John. Tasks marked by T&J will be a collaborative effort between Tim and John.

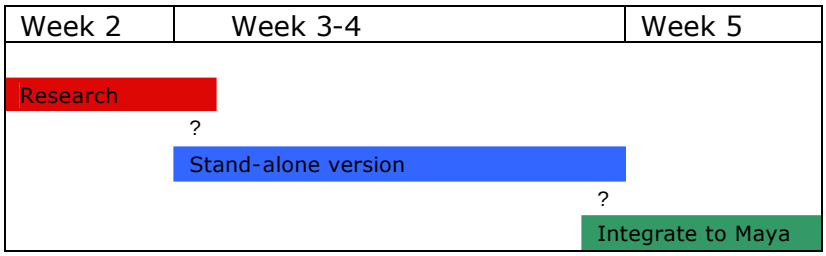
ID	Task Name	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8
0	Design Doc	T&J							
1	Level Editor		J						
2	Paint Tool		T						
3	Integrate 1&2				T&J	*1			
4	L-System		J				*1		
5	Procedural Fire		T				*1		
6	Integrate 3,4,&5						T&J		*2
7	Test/Debug								T&J *3

*[1] Alpha version release

*[2] Beta version release

*[3] Final version release

The following is a further breakdown of tasks 5 and 6, which are the two main modules of the project.



Works Cited

Beaudoin, P., Paquet, S., Poulin, P. 2001. Realistic and Controllable Fire Simulation. In *Proceedings of Graphics Interface 2001*. 159- 166.

Lamorlette, A., Foster, N. 2002. Structural Modeling of Flames for a Production Environment. *Association for Computing Machinery (ACM).*

Veach, M.S., Coddington, M., Fox, G.C. 1994. BURN: A Simulation of Forest Fire Propagation. <http://citeseer.nj.nec.com/119_736.html>

Zaniewski, T., Bangay, S. 2003. Simulation and Visualization of Fire using Extended Lindenmayer Systems. *Association for Computing Machinery (ACM).*