Analytical Fire Modeling: Fire in its Environment

New Mexico
Supercomputing Challenge
Final Report

April 5th 2006

Team 77
Rio Rancho High School

Team Members
Christopher Morrison
Nicholas Kutac

Teachers
Debbie Loftin
Janet Penevolpe

Project Mentor
Nick Bennett

Table of Contents

Executive Summary

The original goal of this program was to create a model that attempted to accurately model fire flow. This year's goal was to improve upon the previous year's progress. The team achieved great success in more accurately modeling the spread of fire by incorporating concepts from Huygen's Principle along with many other fire-spread models. Historically, fires cause more damage every year than any other natural disaster. When attempting to model fire, several important factors must be considered in order to project for realistic fire flow. These factors include spread rates over different fuels and heat's effect on fire, as fire acceleration, wind, elevation, moisture contents, amounts of fuel, spot fires, and crown fires. However, accounting for all these variables requires an extremely complex model. Therefore, this project concentrates on the basis of fire flow in two dimensions limiting its factors to what was verifiable.

In building this year's model, rather than just modeling heat flow to demonstrate the advance of a fire as was done last year, both fire flow and heat flow have been differentiated to more realistically show a fire's expansion perimeter. The most important aspect of heat generated by a fire is the possibility that heat could lead to the rapid acceleration of a fires perimeter. The basis of the fire flow process is the Elliptical Fire Theory, which simply states that fire will form a perfect circle under perfect conditions. The fire flow process involves locating the fire's perimeter and extending new fire ellipses. The determining factor which establishes fire flow is the varying degrees of spread rates of the virtual-forest fuels. In the program, heat flow is accounted for using Newton's Law of Cooling, the Stefan Boltzmann Law of Radiation, and Fourier's Law of Conduction.

The environment also plays a crucial role in determining fire flow. In order for certain patches, which are locations in the environment, to burn, they must have reached an ignition point, or minimal burn temperature. Each patch owns different fuel types, amounts of fuel, spread rates, temperatures, and other independent and dependent variables that define fire flow in the program.

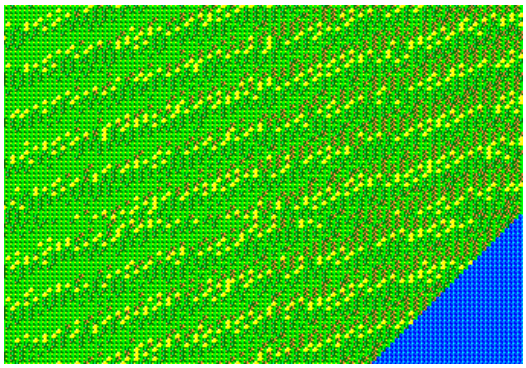
The basis of this program, the Elliptical Fire Theory, has been verified through empirical data collection. However, the limitations of this computer projection are many. These limitations are related to the multitude of variables within the fire flow process. Many of these variables could have been used, but much more time and research are needed to do this. Thus far, this project has yielded a foundation which can be built upon one step at a time.

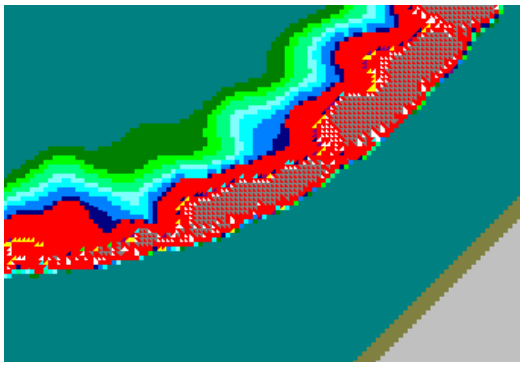
Introduction

Forest fires have threatened mankind for as long as man can remember. In the distant past, wild fires were free to demolish billions of acres of lush forest, and entire ecosystems have grown dependent on natural forest fires sweeping through and clearing out room for younger generations of plants to take over. This process was all very normal until humans, who inherently try to protect their lifestyles, began to intervene on the process. Today people do as much as possible to prevent the spread of fire into their settlements. Many of the most recent fires have reeked havoc on the near by populations. Some of these most recent fires include the Cerro Grande fire in 2000 which burned 47,650 acres and cost $10 million dollars to contain (Masse, 2003), the Yellowstone fire in 1983 which burned 793,000 acres (36% of the park) and cost $120 million dollars to contain (www.nps.gov); and even the Great Michigan fires of 1871 which burned 2.5 million acres, caused more than $200 million dollars in damage, and took the lives of more than 1,300 people (Heidorn, 2000). Even though these recent fires were relatively small they destroyed hundreds of homes and caused millions of dollars in damage. Ancient fires were much more massive by proportion, but they did not affect as many people as more recent fires.

Fire is a devastating force. Every year around six-and-a-half million acres of woodland forest are burned (O'Driscoll, 2005). By learning how fire flows and how the heat it generates allows it to spread, the team will be more able to predict where it will go and how much damage it can cause.

Last Years Progress





Last year's project was titled "Fire in the Bosque" and Christopher Morrison made remarkable progress on his own after his team member dropped out of the Supercomputing competition soon after the Glorieta kickoff. He was able to create a forest environment model with C++ that modeled how the heat generated from a fire in the Albuquerque Bosque would spread.

Last year's program was not as accurate at modeling fire flow because it attempted to model fire based upon principles that don't follow realistic fire flow. The basis of fire flow was founded upon the transfer of heat rather than the current program that differentiates the fire flow process from heat flow. Basically, last year's program used an algorithm which averaged temperatures and would mark the patch as a "fire" if the temperature in the patch was over the flashpoint of that particular patch. The program lacked the ability to track the fire's exact location across the landscape because it lacked the fire flow process which is this year's program's founding principle.

In essence, the inaccuracy of last year's program lies with its discord with the Elliptical Fire Theory through its algorithm. Its base flow formed a square or diamond shape rather than a perfect circle under perfect conditions. Therefore the basis of the old program has been disproven by its violation of the Elliptical Fire Theory.

Furthermore, last year's program was written in C++ for the SDL graphics emulator. It lacked the ability to graphically display the detail that this year's program has attained using C++ for the OpenGL graphics emulator. For further information on last year's project, please visit the final report at http://www.challenge.nm.org/archive/04-05/finalreports/50.pdf

Research

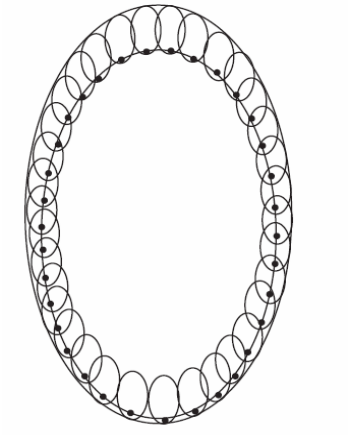
Contacts

The team's first contact was Andrea Rodriguez, a United State Forest Service Cartographer with the Geospatial Service and Technology Center. Andrea attempted to help by finding real forest-contour data derived from satellite data which could be incorporated into the program, but the program is not yet advanced enough to use the data. In the continuation next year, the program may become advanced enough for it to be possible to incorporate the available forest-contour data next year.

The next contact made was Nestor Pena who works as a battalion chief at the Rio Rancho Department of Public Safety. Nestor pointed out many of the assumptions made by the team and explained how firefighters battle large-scale forest fires. From Nestor, the team learned that because of the select few variables used in the program, the program can not yet be used as is for fire prevention and containment, but indicated the program showed promise.

FARSITE Program

The FARSITE Program is a fire perimeter expansion program developed by the United States Forest Service for fire path prediction. The most important thing learned from the FARSITE Program was Huygen's elliptical expansion principles. Huygen's Principles state that a fire will always expand in an elliptical pattern from a central ignition point. A fire's perimeter was further expanded by creating new ignition points along the original elliptical expansion to complete a circular pattern. This pattern of expansion can be seen in this picture. A new ignition points are created along the original elliptical pattern further expanding the fire.



The limitations of the FARSITE program include the limited factors which are taken into account by the program. These few factors include wind vectoring and elevation. Since it is believed by the team that the burned fuel type plays a significant role, the team does not agree that FARSITE is an all-inclusive model even though it is widely accepted by the

scientific community. Another difficulty with the FARSITE program is the very complex differential equations that made it hard to utilize in this project's program.

From FARSITE, the team further expanded upon the fire perimeter expansion concept by using an original ignition point and creating numerous rays which each expand depending upon the factors in the path of the ray. Basically, a fire flow process was developed that borrowed from Huygen's concept, but was altogether different.

Fire Methodology

From their research, the team was able to deduce several of the key factors which effect a fire. A key component in creating a successful fire model was first to differentiate between the heat flow of a fire and the actual fire spread. Fire cannot be explicitly defined and must be broken down into its component elements. When broken down, fire follows many known characteristics. Incorporated into this project's model are several of these characteristics. In order to break apart the fire phenomena and gain insight into these characteristics, fire must be broken down to its intrinsic nature by starting with as basic an assumption as possible. In order for fire to exist three factors must be present: **heat, oxygen, and fuel**. Differing amounts of these variables effect the ability of a fire to exist within a certain area, the spread speed of the fire, and the damage inflicted upon the environment.

- **Elliptical Fire Theory**

  The basis of this project is the concept that under perfect conditions, fire grows in the form of a perfect circle. Even in space, fire maintains the shape of a sphere. This is the Elliptical Fire Theory. The theory suggests that fire growth is circular and is only acted upon by external forces. While not proven, the Elliptical Fire Theory is the underlying assumption of this project. Once this basis has been determined it opens the door to incorporating many other characteristics.

- **Heat**

  If a wildfire is burning in the winter, it will not have the same ability to spread as a fire in the summer. This is due to heat's effect upon fire spread. There are three basic forms of

heat flow: **conduction, convection,** and **radiation.** Heat flow in the atmosphere occurs everyday and has highs and lows as the day passes but is very different when a fire produces heat.

- **Convection**

  Convection is relevant to this project because of heat flow and loss to the atmosphere and has been incorporated with Newton's Law of Cooling which states:

  T(1) = ambient temperature + (T(0) – ambient temperature) x e^{kt}

Whereas:
T(1) = New temperature *temperature units are interchangeable
T(0) = Old temperature
K = Negative variable constant dependent on the object's heat retaining rate
T = Time elapsed in minutes

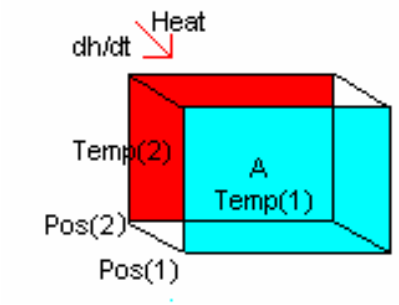
- **Conduction**

  Conduction is the heat flow into and from solid objects and integrated using Fourier's Law of Conduction

$$\frac{dh}{dt} = kA\left(\frac{Temp(2)-Temp(1)}{Pos(2)-Pos(1)}\right)$$

Whereas:
Temp(2) > **Temp(1)**
dh/dt is the amount of heat flow from Temp(2) to Temp(1)
k is the thermal conductivity of the material of transfer
A is the area^2 of the two materials in meters
Pos(2)-Pos(1) is the difference in meters between the two Temp()

Note: Air has been treated as a solid for conduction object with k=.026 (Wikipedia.com)



- **Radiation**

Radiation is the chief transport of heat flow from a fire. Radiation is the flow of heat from rays through space whether matter is present or not. Radiation has been defined with the Stephan Boltzmann Law of Radiation

$$\mathbf{E = KsAT^4}$$

Whereas
K = Stefan-Boltzmann Constant=5.67*10^(-8)
S = Emissivity
A = surface area (meters)
T = Kelvin Temperature of object
E = heat transfer in joules that is emitted

Note: This law doesn't account for radiation emitted from fire only from an object

- **<u>Fuel</u>**

  The heat fire produces is developed from rapidly oxidizing the fuel in a chemical reaction, and fire needs fuel to produce heat. This chemical reaction is a self-perpetuating reaction that breaks off the oxygen atoms from its fuel. Yet, before this reaction can take place, the water contained in the fuel must be evaporated. Once the moisture content is evaporated, fire needs only to attain the **piloted flashpoint or combustion temperature** to combust the fuel and begin to spread.

  - **Fuel types**

    In a forest there are many types of fuels. For example, the Russian olive tree and the cottonwood tree possess distinctive properties. When burned, the Russian olive produces a higher heat value and is consumed much faster than the cottonwood. There are thousands, if not millions, of varying kinds of fuels ranging from low grass to, piñon, and other deciduous hardwoods. Each fuel type governs different characteristics of fire flow when they are aflame and radically change the flow of fire. Also, some fuel types like oak are very thick and slow to burn. Others like redwoods have fire-retardant bark which prevents a crown fire. Depending on the forest, different fuel types must be defined empirically and entered into the program.

  - **Wet and Dry Fuels**

    Each of these above fuels can be further generalized into two groups: **wet fuels** and dry fuels. Wet fuels are living fuels, which contain a moisture content that must be

evaporated before igniting. **Dry fuels** consist of the dead underbrush that has lower flashpoints due to the minimal moisture.

- **Piloted Flash Point**

  The basic flash point, or piloted flashpoint, is the temperature the fire must attain to ignite the fuel and, with a fire present burst into flames. This flashpoint value is dependent upon the atomic structure of the material being burned and varies depending on the fuel source.

- **Unpiloted Flashpoint**

  The unpiloted flashpoint of a fuel is the temperature at which the fuel will be combusted, whether fire is present or not. Examples of such ignition without fire radiation include using friction to start a fire, a match on fire, rubbing two twigs together, or even focusing light under a magnifying glass to ignite a fire. This means the temperature of the heated area becomes hot enough to begin a fire without a fire present.

- **Crown Fires**

  The crown of a tree is the height at which its leaves or needles are located. The fire cannot burn the crown if it cannot reach that height. Tree trunks are generally too high for the heat of a ground fire to reach. Many forest fires behave in a pattern as if they were two separate fires a fast-moving crown fire and slow-moving ground fire. Crown fires constitute a major danger to forests and inhabitants because if a crown fire ignites a tree, it will likely determine whether the tree will survive the fire.

- **<u>Oxygen</u>**

  In order for the chemical reaction of rapid oxidization to take place, oxygen must be present. If there is no oxygen, then there is no fuel and thus no fire. As fire burns, it consumes the oxygen and produces carbon dioxide. The rate at which fire consumes oxygen can be partially attributed to convection movement of the oxygen-containing wind that fire produces. This is caused by the fact that when fire burns, it produces wind from convection, which further stimulates the fire because it brings in air rich in oxygen.

- **<u>Environmental Factors</u>**

In addition to the heat, fuel, and oxygen factors, environmental factors also affect fire. They are named environmental factors because they are derived from the forest conditions.

- **Wind speed and angle**

  Wind speed and wind angle will ultimately determine the prevailing path and intensity of the fire. Sometimes fire can travel much faster than a human can run because of the result of wind. To firefighters, this is known as a bad day.

- **Ambient Temperature**

  Temperature plays a large role in determining fire flow. A fire burning in cold temperatures will have a lower intensity than a fire burning in the hot temperatures of the afternoon, which is why a fire's intensity decreases after nightfall and increases in the afternoon. This suggests that heat has an affect upon fire flow.

- **Humidity**

  Humidity slows fire flow by increasing the moisture content of all fuels. The higher the moisture content, the longer and hotter a fire must burn to remove the moisture.

- **Elevation**

  Elevation plays a big role in fire flow. Fire will spread uphill faster than downhill to gain access to oxygen-rich air. Fire acts as a force that will oppose gravity. Therefore it affects fire flow in the third dimension and when elevation changes with slopes and cliffs, fire is also affected on the two-dimensional plane.

Fire is not totally inexplicable in its nature. This is because there are many variables acting upon fire that are yet to be defined. Those variables which have been defined and incorporated however, allow for a fire model to be constructed.

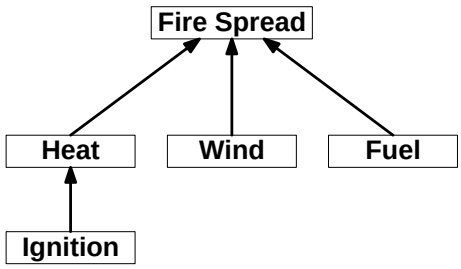
Fire Flow/Spread

As stated earlier, fire spread is something that, to date, can only be approximated. A commonly used approximation is based on Huygen's Principle. In accordance with the Elliptical Fire Theory, fire will spread in all directions as fast as it possibly can while still maintaining its three fundamental bases: heat, oxygen, and fuel. However, the exact determination of fire spread is still not as simple as that. There are so many factors

bearing upon fire that it is currently impossible to incorporate them all into some finite formula. To determine basic fire flow one must look at the basic intrinsic parts and develop an approximation. This is the goal of this year's project.

Problem Definition

There are three major challenges faced by this research team. The first is how to accurately model heat flow and fire flow, the second is how to successfully incorporate environmental factors like elevation and wind into fire flow; and the final issue to be solved is to verify that the fire spread program is successful. Fire flow is defined as the path the fire follows when exposed to an environment. Rather than bogging down in a marsh of factors that affect fire, this project decided to focus only on a few factors.



The independent variables of this project are wind, fuel, heat and fire ignition temperature, while the dependent variable is the fire spread. The general rule of a fire is that it will continue to expand until any one of the afore-mentioned variables are exhausted.

Heat flow and fire flow have been defined as separate entities because “fire”, as most people understand it, is actually a set of complex factors which all play a part in furthering fire expansion. Heat flow has been defined as the energy released by the fuel as it burns, as well as the ambient temperature flow within the forest. The higher the heat rating of a fire, the greater the energy the fire has, and the further it is likely to spread before dying out on its own. Furthermore, as the fire spreads, the heat on the perimeter, inside, and in the burned areas will change. The local temperature of a patch will get higher as the fire burns in it and will cool down as the fire moves on to the next patch.

The fuel modeling portion of the program is another variable involved in determining fire flow. As a fire spreads over different fuels it will spread at different rates. Basically, the fuel modeling is based upon the environment. The environment ultimately determines the shape of the fire assuming all other factors are consistent

Fuel may determine the shape of the fire, but wind decides the path through the forest that fire will follow. Wind works to push fire flow in one direction more so than another. This relationship is very important to firefighters because, in the wrong wind, fire will outrun ground crews, threatening their safety.

Problem Solution

Assumptions

In order to model fire flow, certain assumptions had to be made. This is due to the complex nature of fire itself. The base foundation of this project is the **Elliptical Fire Theory**. Because fire flows in a circle, to incorporate other factors would be a matter of shortening and lengthening the radii based on these factors. By adjusting each radii based on the given variables, a more random fire flow model would result. This would be effective for any two-dimensional model, especially since fire flows only on surface area. This flow pattern could later be adapted for a three dimensional model. From this cornerstone, other assumptions were made about the effects of factors on fire flow. The distinction between the heat flow and fire flow models, an improvement over last year's project, is an example of this. Though the exact relationship is undefined as of yet, fire is dependent upon heat to determine its flow.

Another founding assumption was that fire cannot exist where there is not ample fuel and heat. If fire encounters an area that cannot be burnt under certain conditions then it will be forced to stop. The other assumptions made by the research team in order to complete the program include: no humidity, no ground duff, wind speed. The team also assumed limited fuel types, and two-dimensional fire flow no precipitation, no convection currents produced by the heat of a fire, and no elevation change. These factors although essential to real-world fire spread, have each been simplified because each is a small piece of a larger picture.

Three-Dimensional Fire Spread

Fire does not move on a single plane, and can move in many different levels of forest, including the ground, small shrubs, and even in the canopy in large fires. Firefighters are most concerned about canopy fires because, once a fire reaches the forest crown, it gains speed and can quickly outrun and outmaneuver even the most experienced of fire fighters. Multi-dimensional fire flow was simplified to a two-dimensional plane. By ignoring the third dimension, gravity's effect on fire can be simplified and almost ignored.

- **Precipitation**

  Precipitation effects fire flow of a forest fire by increasing the fuel wetness, which raises the temperature required to maintain persistent fire. Rain will greatly diminish the

strength of a fire. Before a fire starts, precipitation will increase the humidity of the ambient environment and the wetness rating of the fuel types. Each of these variables dampens the likelihood of starting a fire and will decrease the "Fire Danger level" as measured by the United States Forest Service. The only related factor is moisture content of the fuel. If it has rained, then the moisture content can be adjusted likewise.

- **Fire-Generated Wind Currents**

  Since heat is a byproduct of fire, the heat generated by a fire causes convection currents in the vicinity of the fire and if the fire gets hot enough, the fire will create its own wind currents. Fire-generated wind is very unpredictable and is an important aspect of any fire program that very few models take into account. Fire-generated wind has been left out of this model as well because of the numerous complex aspects that would be involved in modeling heats effect on air currents and the air currents' effect on fire flow.

- **Elevation Change**

  When on a slope, fire will flow faster uphill, rather than downhill, because more of the heat will rise, which then increases the ambient temperature on that part of the slope, causing the fuels to ignite and burn. Elevation has been excluded from this model because the team was unable to verify how elevation affects a fire in the time available. In order to gain exact verifiable insight into elevation's effect upon fire, empirical data must be acquired.

- **Oxygen Modeling**

  Oxygen is one of the three vital factors involved in producing fire. However, it was decided to leave the oxygen component out because oxygen's effect upon fire must include the convection from the wind-producing fire.

- **Humidity**

  By simplifying humidity, a much more uniform fuel type is created, which allows for more even flow of fire. Humidity can vary greatly throughout a forest, depending on precipitation and proximity to a water source. Humidity effects a fire by raising the ignition temperature of the fuel because the fire must first burn away the moisture before it can successfully ignite the fuel. The solution to this is simplifying the assortment of humid fuels to two kinds: wet and dry fuels. The wet and dry fuels possess different flashpoints and can be manipulated according to environmental factors. For example, the

wet and dry fuel rating changes depending on whether it was a wet or dry year or if rain has just fallen upon the forest.

- **Wind**

  The programs wind assumption maintains that fire radii are dependent upon wind and its angle and based on vectoring to shorten and lengthen radii on the ellipse. Basically, fire is treated as a vector based upon the spread rate and wind direction and angle.

- **Ground Duff**

  Ground duff is the dead material at the bottom of a forest. In a coniferous forest, it is generally pine needles, in a deciduous forest, duff is generally composed of dead leaves and plant matter. It affects a fire by drastically increasing the amount of burnable fuel on the ground level. In the program, ground duff has been simplified along with the fuel types. For example, every cottonwood fuel patch will have the same amount of ground duff.

- **Numerous Fuel Types**

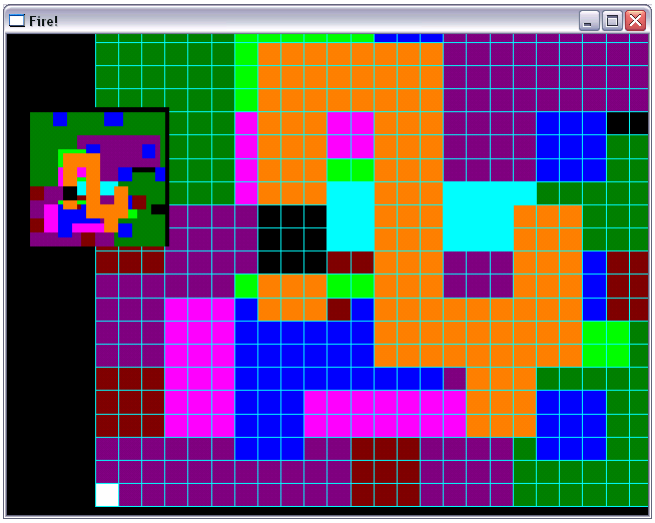
  Another basic assumption is that fire has the capacity to spread differentially over different fuel types because of the fuel's typical surface area and atomic structure. Every tree, shrub, and patch of grass has a different values which determine its specific fuel type. By reducing the amount of fuel types, the program is able to better model a simple forest fire with a few varieties of plants and is, therefore, able to be verified easily.

By simplifying these variables, a basic fire spread model has been created. For a future venture a continuation of this program would include many of these variables, The variables would need to be defined and incorporated to better account for the multitude of factors which make up fire.

Program

For this project an original computer program was written in the C++ language, using a technique that draws off Huygen's Principle, but it is original in itself. The program traces the fire perimeter across the forest environment. Implementing Newton's Law of Cooling, Fourier's Law of Conduction, and the Stefan-Boltzmann Law of Radiation, the program differentiates fire flow from heat flow. It also uses a patch class environment system that records the individual aspects of each individual area. The virtual forest is created from a forest environment editor that can be changed by the user to mimic any realistic forest.

The Basic Environment



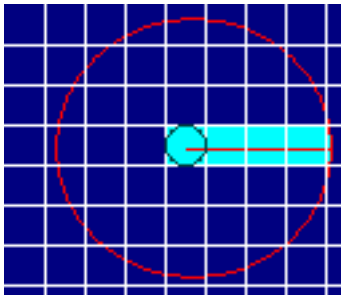
The environment in which the fire persists is a system of patches. Within those patches, there are more mini-patches. The patches each own their own specific set of variables which depend on the fuel type, wind speed, wind angle, ambient temperature, humidity, wet fuel, dry fuel, temperature, flash point, secondary flash point, produced heat, and crown height. Though, some of these variables have yet to have a place in the program. The collection of patches, as a whole, is known as the virtual forest. Each patch type is unique and causes a fire to either accelerate or slow down as it passes over the patch. The patches then have individual sections within them called mini-patches. These mini-patches are then used to further track the exact movement of the fire as it crosses the virtual forest.

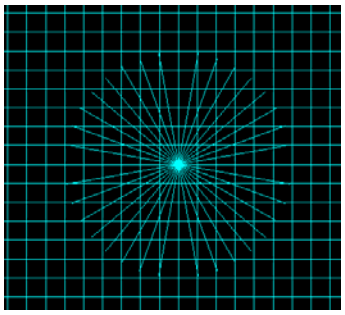
This is an overhead view of the virtual forest. Each different color represents a different fuel type. The rectangle top left is an overall view of the forest.

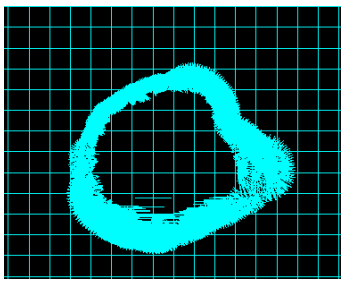
- **Virtual Forest Level Editor**

  The virtual forest was developed by the virtual forest level editor, which allows the user to design a forest from scratch through a Graphic User Interface.

The Algorithm Fire Spread/ Flow







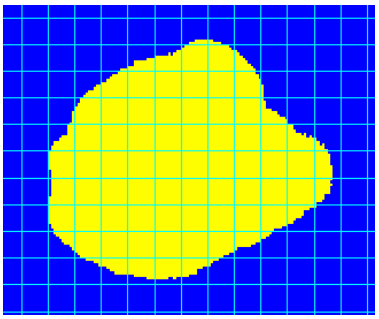
The program uses a hybrid of both Huygen's Principle and the Elliptical Fire Theory. The different patches in the virtual forest contain different values of what is called spread rate, which is the rate at which fire can spread through the patch. The spread rate can be manipulated by factors such as heat and humidity. For each time step, the fire spreads across the patches until the accumulated value of the time steps reaches a point called the maximum spread rate (top picture). This process is repeated until a fire arc is formed, making the perimeter of the fire for the certain time step. In this picture the teal patches have certain values. The fire line spreads across these patches, accumulating their values until the maximum value is reached for the fire.

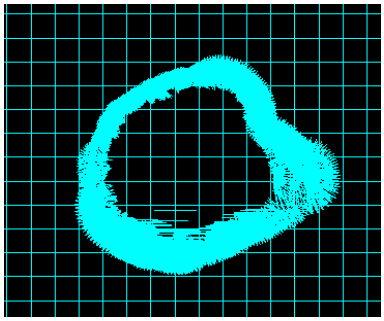
The **fire arc** shown here (second from top) is the same process as shown before, just taken out many more iterations. The endpoints of the **fire arc** are then stored, and during the next step are transformed into **fire arcs** themselves. This process is computed once per time step, producing **fire perimeter** results such as in the bottom picture. This picture depicts the result of several time steps in a non-uniform virtual forest and shows the created fire perimeter.

- **Fire Perimeter Reduction**

  As the fire grows larger, only the perimeter is needed to map the fire flow. If the computer took every fire endpoint and arced it, its efficiency would be x^N, whereas 'x' is the number of endpoints the fire arc contains and 'N' is the number of time steps. Therefore, a system was developed to record previously burned spots and reject new fire

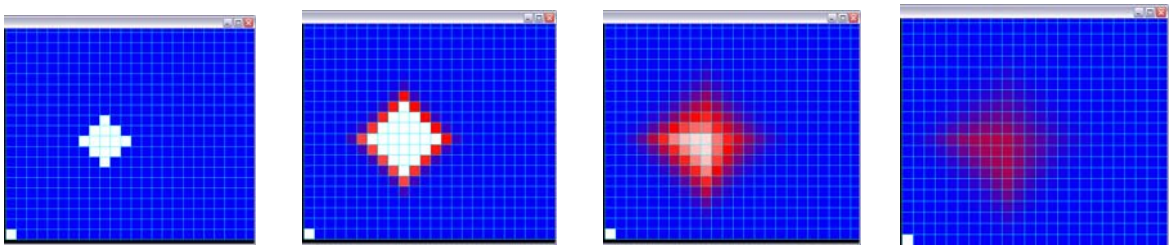
arcs in those burned spots. This would increase the efficiency of the program by preventing “burned” patches from being burned a second time. These two pictures show the fire perimeter and the burnt mini-patches that help with fire perimeter reduction.





The Algorithm Heat Flow

Heat flow was solved by incorporating Fourier’s Law of Conduction, Newton’s Law of Cooling, and the Stefan-Boltzmann law of Radiation. Fourier’s law of conduction is performed in order to account for heat conduction through air. The Stefan-Boltzmann Law of Radiation is then performed to account for heat radiation. It is not used, however, to account for fire radiation of heat. However, no heat loss is factored into these because Newton’s Law of Cooling accounts for heat loss through all three types of heat flow, convection, conduction, and radiation.

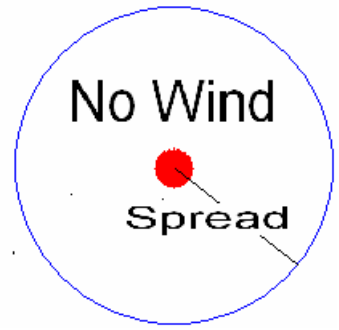


Heat created during the rapid oxidization process is not accounted for through radiation. The heat produced in a patch while burning is based upon the total percent of the patch burning and a pseudo value dictated by fuel types to calculate how much energy can be produced by burning the patch completely. So, for each iteration the heat produced by fire can be defined by multiplying the heat produced by the percent of the patch burning. This process will continue until the fuel is exhausted. The pictures above demonstrate their use.

The pictures shown above depict a very hot heat source (white) as it cools and transfers the heat (red) until the patch temperatures drop back to the ambient range (blue).

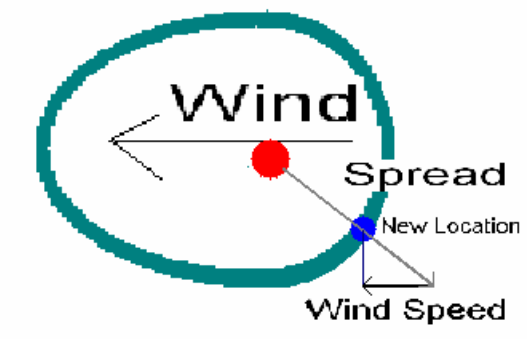
Environmental Factors

Fire is greatly affected by three certain environmental factors: wind, humidity, and ambient temperature.



- **Wind**

  Wind has been accounted for with the incorporation of vectoring. Each fire arc is subject to course correction by adjusting for wind using a vector which is based upon both the spread distance and the wind speed. However, this adjustment only takes place upon spread lengths emanating from the central ignition point. This is because, if the wind is blowing, the fire's shape would not produce the right shape as dictated by FARSITE and all other data upon which wind is based on. These pictures show the vectoring process in more detail.



- **Humidity**

  Humidity has not been incorporated because its effect on fire could not be easily defined. Humidity could possibly be accounted for by editing the spread rate based upon the ambient humidity; however, there is no finite scale upon which to make this assessment.

- **Ambient Temperature**

  Ambient temperature is the starting temperature of the virtual forest. As seen in the heat flow example, as the environment cools, it approaches the ambient temperature. This is the basis of Newton's Law of Cooling. The beginning ambient temperature must be input by the user in the beginning, and is handled by the program from then on.

Technical

This program was developed to be efficient. The use of advanced classes, vectors, header files, text file input and output, and such has successfully increased the efficiency of the program. The program, at its current state, is 1,403 lines of code, with the most important processes in the main header file (Please refer to Appendix C). However, the task is not completely done as of yet. Before the final presentations, there will be several more additions to the current code. There will also be a website developed detailing the project in its entirety. Code additions include: correction of the wind vectoring code, a status number box, a view that integrates the physical environment with the fire flow, and a java applet demo version of the program that will be placed on the website, as well as possibly incorporating parallel processing.

Program Validation

In order to validate the program, the team created ideal conditions, as much as possible in the laboratory and by creating ignition points on varying stocks of paper in a windless room using a lighter. The team chose paper because it is the most uniform material that can be burned. The Elliptical Fire Theory states that under ideal conditions, a fire will burn in a two-dimensional circle. Several different types of paper were chosen, which included printer paper, construction paper, and tissue paper. As the fire burned, the advance of the fire was filmed. As seen above, the fire moves in a very circular pattern because the fuel type is uniform, and there are no external factors such as wind which can adversely affect the fire flow. The next step was to input the conditions into the fire model.

When a uniform fuel type is input into the program, the "fire" moves in a perfectly circular pattern and matches the Elliptical Fire Theory. Since the program's fire moves in a similar pattern, then the program has been validated and the theory endorsed.

When a non-uniform surface was burned (see picture above), the fire flow upon the surfaces was different. This endorsed and validated the assumption that fire will flow at different rates over different fuel types. Appendix A section C contains the program's version of the experiment. The similarity between them is striking. This reflects the programs validity over the fire phenomena.

Since the basis of this project was validated, it is hoped that other factors cold also be validated in laboratory conditions.

Discussion

What has been learned during the course of this project is that fire is an entity which is very difficult to define, and in order to properly model it, many many variables must be incorporated. To date, no all inclusive fire model has ever been created. Although many groups such as the U.S. Forest Service have attempted to accurately model how a fire flows in a forest, no one has yet created a model which incorporates all the variables that can be observed in the real world.

Due to the limited time period in which to create this model, the team chose to model fuel types and wind vectoring because they believed that those two variables, in particular, affected the flow of fire the most. Although the model does not take into account many of the variables associated with fire, those few variables which have been modeled, have created an accurate replication of how a fire flows n a realistic environment. The variables which have been incorporated significantly advance the program beyond the progress that was achieved last year.. Many concepts which were intangible when the project was first begun have been defined and will be essential in future fire modeling research. The most important of these is the distinction between fire flow and heat flow.

The Heat Flow Model

The incorporation of Newton's Law of Cooling and Fourier's Law of Conduction account for convection and conduction, and the introduction of these two laws into the project gives the project a strong foundation. However, the radiation model has eluded the modeling process. The Stefan-Boltzmann Law of Radiation accounts for only fire radiation given off by heat and not the chemical reaction of fire. Fire's main method of heat creation is radiation. Therefore, the heat flow model requires more development, most specifically the radiation model. The actual fire model is also affected, because if heat cannot be properly defined, then the effect of temperature upon fire cannot be properly modeled.

The Wind Model

The wind vectoring process makes logical sense. Fire flow can be treated as a vector of the proportion of the wind speed and its direction and the spread of the fire across an area. However, this has not been validated yet and can only remain suspicion.

The Fuel Model

The fuel model generalizes many factors. It breaks the forest and its individual traits into a square grid with meters as its base unit. This generalization makes fire flow less accurate. Yet, you cannot model the forest exactly in a computer program. There are factors that cannot be perfectly defined no matter how precise you become.

The division of the patches' fuel into wet and dry fuel also pulls the fire flow process from the mark. Yet, it is this simplification that gives the project a basis, flawed, but workable, upon which it can solve the important part, the fire flow. The program can then easily characterize wet and dry fuels.

The Fire Flow Model

The fire flow process is the biggest success of this project. The Elliptical Fire Theory gives the program a validated base upon which to build. As more variables are added to the model, one must know the beneficial or detrimental effects of the variable upon fire spread to find the growth of the fire. This is a standard benchmark that depends upon all the incorporated variables the fire will travel a certain distance at a certain degree.

Variables and Empirical Data

Fire is never exposed to perfect circumstances. In the real world, fire is affected by many variables, some yet to be defined. This project attempted to take into account three specific variables that effect fire flow; heat, fuel type, and wind. However, to determine these variables, other variables must be known, and eventually integrated. For example, it is known fire spreads at different rates over different fuels, but it remains unknown exactly at what rate per fuel type. Empirical data is needed to increase the accuracy of this project. There are many variables which could become constants with more research and experimentation.

Though this program may not account for all environmental factors, it does provide a basis for later addition of these factors when the variables' effects are known a can be integrated into the program.

Validation

Looking back, this project has come a long way since last year. The foundation of the Elliptical Fire Theory has evolved into the modeling of a fire. This project and the program cannot be disproven, yet cannot be proven at their current state. This is suggested by the research and data collected.

This program follows basic fire expansion. In a future venture however, further validation of this project needs more real-world data. This data must then be collected, compared, and the program adjusted to take into account realistic, imperfect fire conditions. Possibly a static test could be done to compare the effects of a real forest fire against the program's version of the fire. Or perhaps this program could be compared against FARSITE. The basis of validation in this project was to validate the Elliptical Fire Theory by burning materials in relatively perfect conditions. Although in tune with the program, in wildfires the conditions of the forest are not perfect. For this project to be of any practical use, the program must be validated against more than perfect conditions.

Overview

This program is the basis upon which a more advanced program can be built. Because there are so many variables that must be taken into account in order to ensure a true fire flow program, only a select few have been incorporated as discussed above. In the future, many more important fire variables must be incorporated in order to ensure an even more accurate fire flow program. In addition, further verification will be attempted using actual fire data collected by the United States Forest Service and other organizations. The team will attempt to verify that, based on the given environmental conditions, the program would flow in the same manner and speed evidenced by a past fire. If the program can be validated by this method using additional variables, then a true to life fire model, which may be even more accurate than current fire models, could be created for use in the real world. Fire is a devastating force, and by understanding it, and using the wondrous technology around us to model and predict it, we may one day live in a world in which forest fires no longer threaten our homes and our lives.

Conclusion

In conclusion, this project's foundation has been validated. The Elliptical Fire Theory has been validated in the real world and has been successfully incorporated into the program. With this basis validated, other factors that bear on fire can then be incorporated when their effect on fire radii are known.

This project was named Analytical Fire for a reason. After all, forests are far too precious to be burned at a whim. However, now that a basis has been laid, it needs real world values to be placed in variable areas. The fuel type assumption, that fire will spread at different rates over different fuel types, has been validated. However, there is no data yet for the rate at which fire moves over different fuels. Heat's effect upon fire is known, but not precisely defined. Therefore, this project needs an empirical basis as well as an analytical basis to become a more accurate fire flow forecaster.

Another inaccuracy of this project involves the sheer number of factors involved in determining fire flow. Fire flow cannot be accurately modeled without knowing these factors and their affect upon fire. However, the nature of the Elliptical Fire Theory allows for these variables to be accounted for easily once their affects are known. As more and more variables are accounted for, more accurate fire flow approximation can be included in the program. Therefore, this program allows for a core around which a realistic approximation can be built.

The conclusion is that this project is a success. The amount of progress accomplished in such a short amount of time is remarkable. The flow of fire across the virtual environment is rational. The results of this project are viable. The program only needs more work, time, research, and an empirical basis to become an asset which could be used in real world applications.

Bibliography

Bonsor, K. (2004) *How Wild Fires Work* Retrieved October 3rd 2005 from
http://science.howstuffworks.com/wildfire.htm

Diaz, J.C. (2002). Fourier's law retrieved on December 6th 2005 from
http://www.mcs.utulsa.edu/~class_diaz/cs4533/flowheat/node4.html#SECTION00121000000000000000

EFunda.com (2005). Heat Transfer. *EFounda.com*. Retrieved December 12th 2005 from
http://www.efunda.com/formulae/heat_transfer/home/overview.cfm

Finey, M. (2002) Australian Mathematical Society *Fire Growth Using Minimum Travel Time Methods*

Fire.org Public Domain Software For the Wildland Fire Comunity

Harris, T. (2005). *How Wild Fire Works* Retrieved December 13th 2005 from
http://science.howstuffworks.com/fire.htm

Heidorn, Keith C. (2000) Weather Almanac for October 2000; The Great Fires of October 1871. Retrieved December 8th, 2005 http://www.islandnet.com/%7Esee/weather/doctor.htm

Masse, B. & Nisengard, J. (2003). *Cerro Grange Fire Assessment Project* Cultural Resources Report No. 211.

O'Driscoll, P. (2005). Studies at odds over logging after wildfires. *USA Today* Nov. 2nd

Rona, A. (2003). *Conduction.* Retrieved December 6th, 2005
http://www.le.ac.uk/engineering/ar45/eg1100/eg1100w/node12.html

Taftan Data (1998). *Fourier's Law of Conduction.* Retrieved December 6th, 2005
http://www.taftan.com/thermodynamics/FOURIER.HTM

The Official Website of Yellowstone National Park, Wildland Fire
http://www.nps.gov/yell/nature/fire/index.htm

Wikipedia: The Free Encyclopedia
http://en.wikipedia.org/wiki/Newton's_law_of_cooling
http://en.wikipedia.org/wiki/Thermal_conductance
http://en.wikipedia.org/wiki/Heat_conduction

www.nifc.gov (last updated, 2002). Historical Wildfire Statistics. Retrieved December 6th 2005
http://www.nifc.gov/stats/historicalstats.html

Acknowledgments

This project would not have been possible without the help of many hard working individuals including;

Nick Bennet – Supercomputing Challenge consultant who was an instrumental aid in the development of the program.

Debbie Loftin – Primary teacher/facilitator at Rio Rancho Mid-High who made sure the team met deadlines and made all travel arrangements for trips with the RRHS administration.

Nadyne Shimada – Helped the team to stay on task, get in touch with contacts, do research, and edit reports.

Jannet Penevolpe – Teacher contact at Rio Rancho High School who helped team make travel arrangements with Deborah Loftin, ensured team was on track, and got the team in touch with Nester Pena.

Nester Pena – Helped the team to better understand how fire is fought and understand the basic assumptions being made.

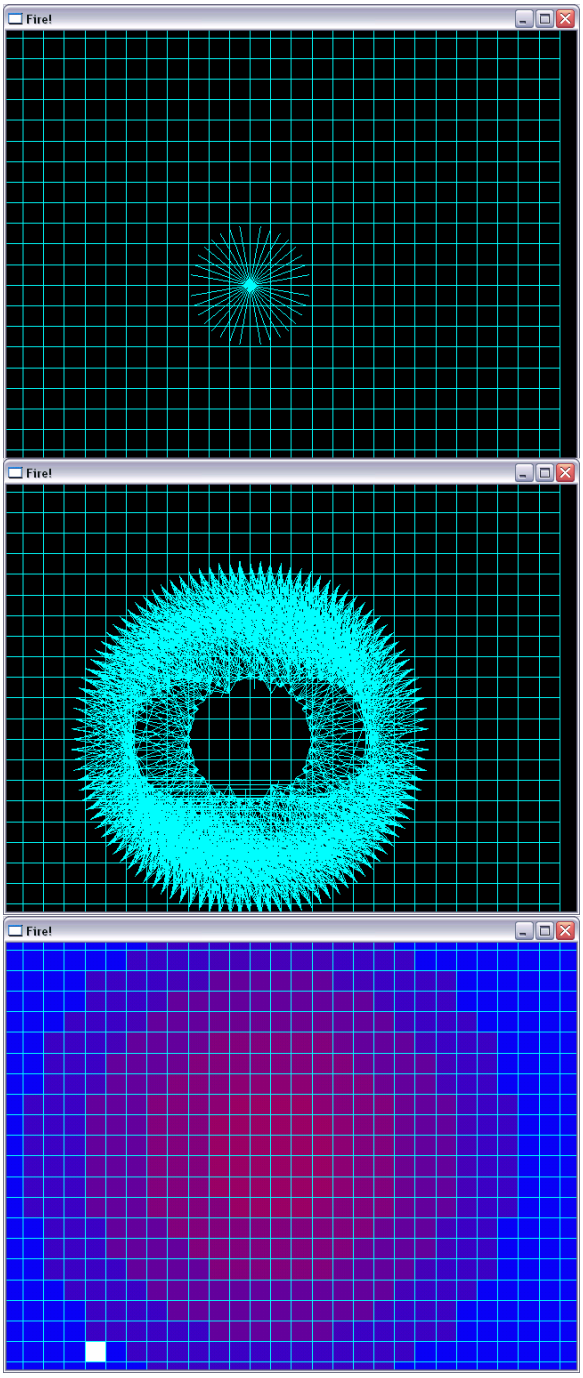
Andrea Rodriguez – Assisted the team by supplying them with real forest-contour data from the Geospatial Service & Technology Center which would have helped with program verification on elevation and terrain change.

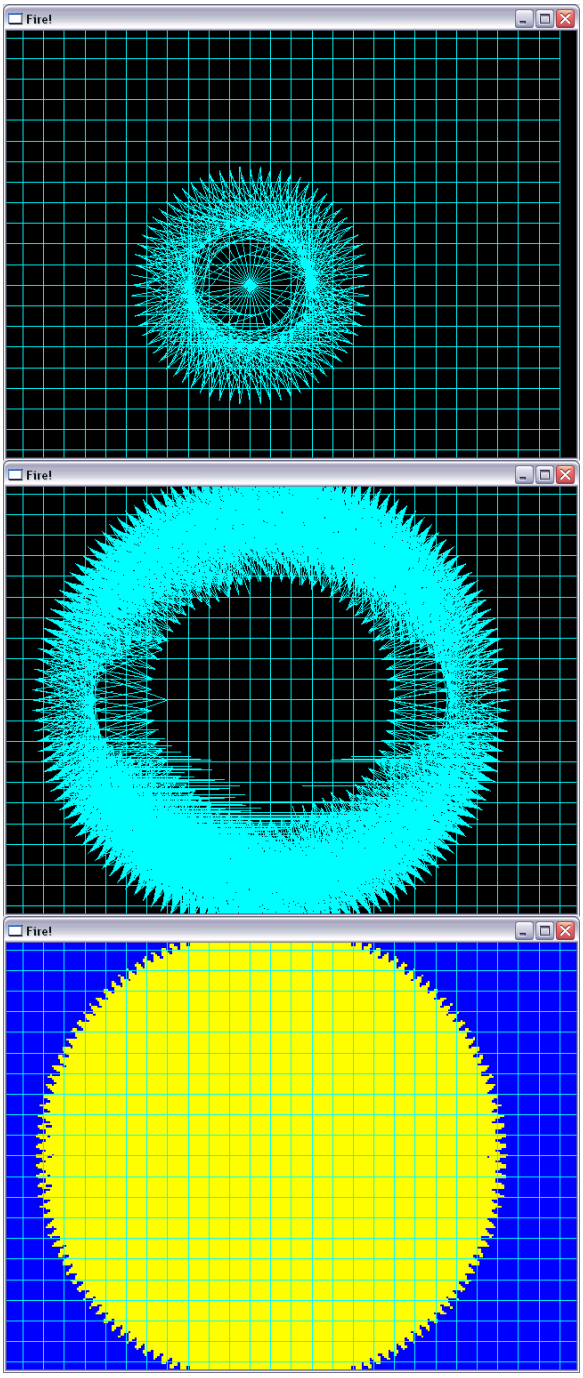
Team's Parents – To the team's parents for allowing them to go on all the trips and putting up with all of the late nights working on the program and reports.

Appendix

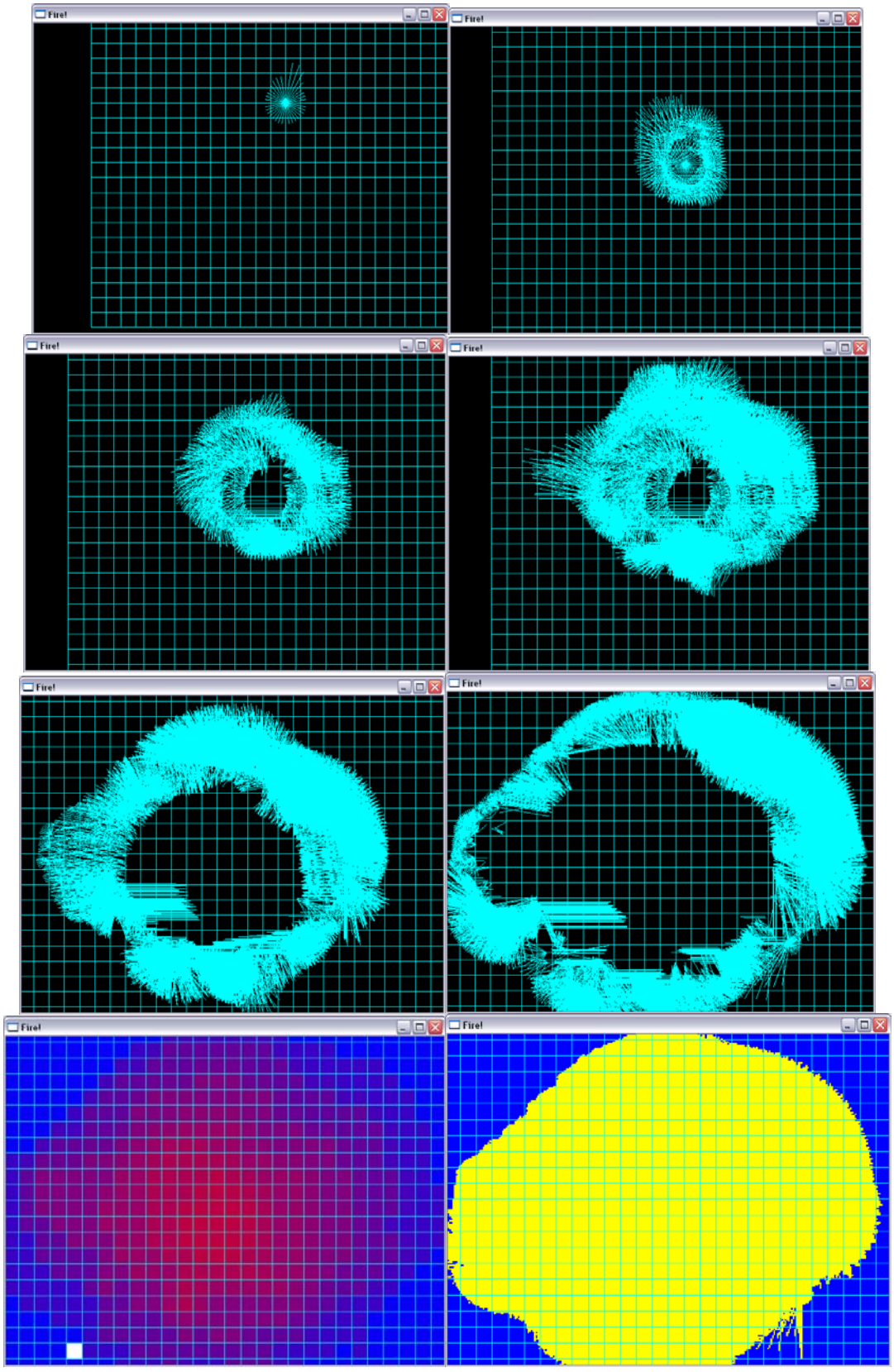
Appendix A - Program Screen Shots

A. Uniform Fire: This execution shows the application of the Elliptical Fire Theory in the program.

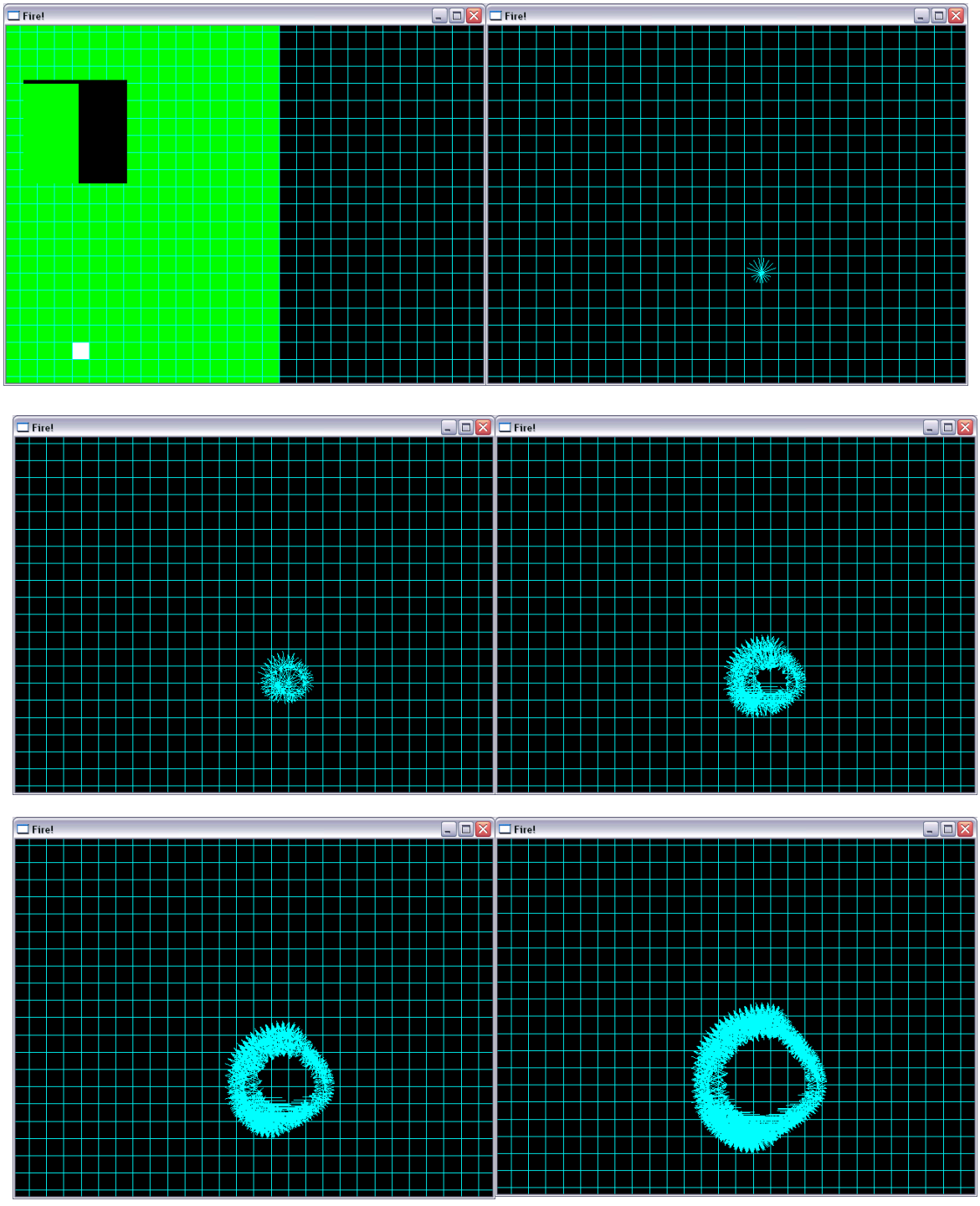


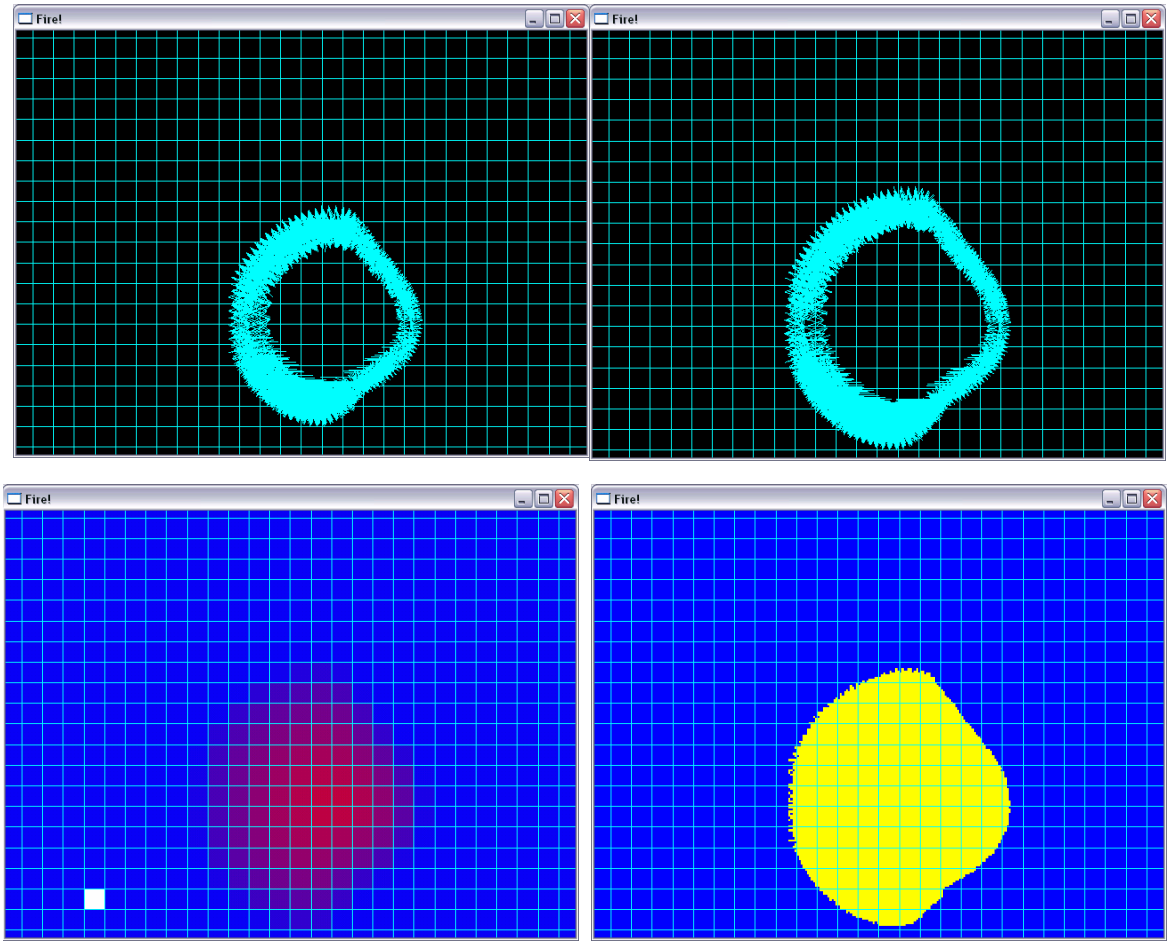


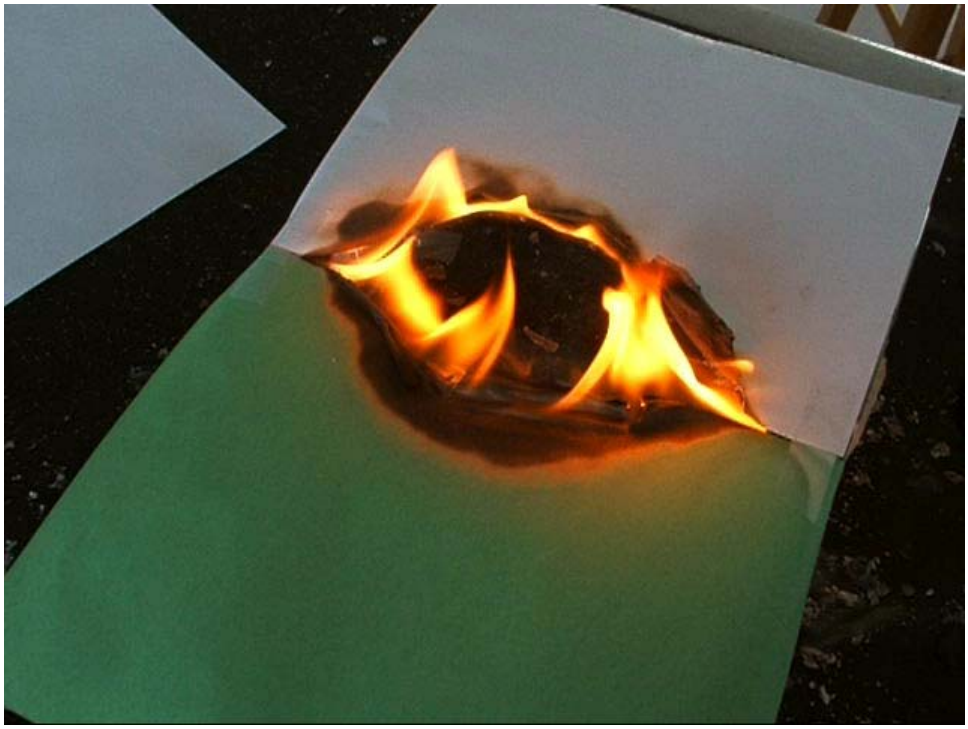
B. Fuel Type Emphasis: This execution of the program emphasizes the effect of fuel types on fire growth



C. A repeated experiment of the validation only in the computer program. The similarity between the program and real fire is remarkable.




Fire!
Fire!
Fire!
Fire!



Appendix B - Glossary

Ambient Heat – Starting from the forest's ambient temperature and raising as a fire moves approaches, and decreasing as the fire passes by or dies our for lack of fuel or oxygen.

Ambient Temperature- Temperature to which environment will cool based upon *Newton's Law of Cooling*

Dry Fuel- Fuel classified as having a lower moisture content requiring a lower *flash point* than *wet fuel.*

Elevation Assumption- Changes in ground level affect fire because of differences in oxygen

Elliptical Fire Theory- Theory that fire will form a perfect ellipse under perfect burn conditions. This implies that to properly account for *fire flow,* the polar radii can be shortened and lengthened accounting to different variables' effects on fire.

FARSITE- Forestry Service's fire approximation program based on *Huygen's Principle*.

Fire – The chemical reaction of fuel, ignition heat, and oxygen which generates additional heat burning fuel and oxygen. Fire is self-perpetuating as long as sufficient fuel and oxygen remain.

Fire Acceleration- As fire grows larger, it accelerates faster. The heat assumption may account for this phenomenon.

Fire Arcs- The collection extensions of the *fire radii* from a single point.

Fire Flow – The most commonly understood definition of a fire, defined as the actual, observable movement of a fire through a forest.

Fire Perimeter- Collection of *fire arcs* that, with *fire perimeter reduction,* form a perimeter around the fire.

Fire Perimeter Reduction- For better efficiency, unneeded *fire arcs* are deleted in the middle of the fire.

Fire Radii- Polar extensions from a central ignition point. For example, 3 units at 30 degrees. Fire radii are susceptible to changes in lengths based upon factors involved in *fire flow*.

Fourier's Law of Conduction- Law to describe conduction. Heat flow through air can be considered conduction.

Fuel- Every patch has fuel with different moisture contents. These moisture contents are simplified into two types of fuels requiring different prerequisites to burn then.

Fuel Type – Different types of fuels where each contain its own heat generation factors, ignition point values, dry and wet fuel ratings, etcetera. Some examples include trees and shrubs.

Fuel Type Assumption- As fire flows over different types of ground coverage, it will spread faster or slower depending upon the fuel type's atomic structure and typical surface area.

Heat Flow – The transfer of heat in a forest as a fire moves, as well as, during everyday circumstances. Generally connected to *fire flow* because of the natural properties of fire which gives off heat as a byproduct, but can also be associated with ambient warming and cooling with the time of day.

Heat Assumption- Different amounts of heat cause fire to spread differentially. The hotter it is, the faster the *fire flow*.

Huygen's Principle- *FARSITE* equation that uses wind and elevation to determine an ellipse that the fire will follow.

Ignition Point – Minimum temperature at which a fuel will ignite and become self sustaining based on fuel type and environmental factors.

Moisture Assumption- More moisture will slow fire spread.

Mini-Patches- *Patches* divided into 100 equal parts to better keep track of certain aspects in the program.

Newton's Law of Cooling- Variable equation that accounts for cooling of patches during radiation, convection, and conduction.

Patches- Division of fuel type exactness and area. One *patch* is approximately 1 sq meter.

Piloted Flash Point/ Instantaneous – Point at which patch will combust whether or not fire is present.

Rapid Oxidization- The scientific term for the chemical reaction of fire. Rapidly taking of the oxygen in the fuel.

Spot Fires- A term for sparks that fly into the air and create fires in new places down wind.

Spread Rate- The rate at which fire can spread across the ground. Influenced by many environmental factors, heat, wind, and other factors.

Stefan-Boltzmann Law of Radiation- Law to give the amount of radiation emitted from an object based upon its emmisivity and temperature.

Unpiloted Flash Point/ Basic – Temperature which is required to burn a certain fuel.

Virtual Forest- Collection of patches that make up the forest environment.

Wet Fuel- Fuel classified as having a higher moisture content requiring a higher *flash point* than *dry fuel* .

Wind Assumption- Wind's affect upon fire can be described as a vector of wind speed and fire spread.

Wind Vectoring- The process of correcting fire flow for wind.

Appendix C Program Code

Main.h

The bulk of the useful program code

```
//Main.h  where it all happens
//Written by: Chris Morrison and Nick Kutac
//Not perfected version
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
#include <gl\gl.h>            //gl libraries
#include <gl\glu.h>
#include <math.h>
#include<vector.h>
#include<iomanip.h>
#include "Load.h"
#include "Data.h"
#include "Wind.h"

void DrawForest(double ypers, double xpers,double sf, int selected[3]);
void DrawHeat  (double ypers, double xpers,double sf, int selected[3]);
void DrawFire  (double ypers, double xpers,double sf, int selected[3]);
void DrawBurnt (double ypers, double xpers,double sf, int selected[3]);
void MakeFire();
void Continue(int sf);
void disContinue();
void Record();

double increment=.1;             //fire step length
double MSR=1;                   //interval*fuela legnth per step
double s=0;                     //current interval*fuela

int degrees=18;                  //number of extensionsor radii on fire arc
int boundaries;
//-------------------------------------------------------------------
  class Enviornment            //forest environment
{
 public:
   double WindSpeed;           //global env factors
   double WindAngle;
   double AmbientTemperature;

    class Patches
  {     public:               //specific to patch env factors
        int type;
        float DryFuelHum;
        float WetFuelHum;
        float oxygen;
        double CrownWetFuel;
        double CrownDryFuel;
        double FloorWetFuel;
        double FloorDryFuel;
        long double CelsiusTemperature;
        double WetFlashPoint;
```

```
      double DryFlashPoint;
      double InstantaneousFlashPoint;
      double ProducedHeat;
      double CrownHeight;
      double emmisivity;
      bool active[10][10];  //minipatches

  void Ignite(int x,int y)   //ignite mini-patches
  {
   active[x][y]=true;
  }

  bool PreviousFire(int x,int y)//check if a fire has previously been
  {                          //in the patch
     return active[x][y];
  }

  bool Sustainable()           //if the conditions in the patch are
  {                          //able to sustain fire

if((CelsiusTemperature>DryFlashPoint&&FloorDryFuel>0)||(CelsiusTemperature>WetFlashPoint&&FloorWetFuel
>0))
      {return true;}
       return false;
  }

}Heat[500][500][2];          //500x500 patches supportable x2 for heat records
//
void Rationalize(int sf)     //rationalize temperatures
 {    for(int x=0;x<=sf;x++)
       {for(int y=0;y<=sf;y++)
          {Heat[x][y][0].CelsiusTemperature+=Heat[x][y][1].CelsiusTemperature;}
       }
 }
//
void HeatFlow(int x,int y)     //heat flow algorithm
 {       double k=-.25;        //Newton's Law of Cooling Varible
        double e=2.718281828; //e
        double K=1;           //Fouriers Law of Conductivity
        double joulesToCelsius=.5;
                        //Joules required to raise

    Heat[x][y][0].CelsiusTemperature=AmbientTemperature+(Heat[x][y][0].CelsiusTemperature-
AmbientTemperature)*pow(e,k);
    //Newton's Law of Cooling
    if(Heat[x-1][y][0].CelsiusTemperature<Heat[x][y][0].CelsiusTemperature)
      {Heat[x-1][y][1].CelsiusTemperature=+K*(((Heat[x][y][0].CelsiusTemperature-Heat[x-
1][y][0].CelsiusTemperature))/4);}
    if(Heat[x+1][y][0].CelsiusTemperature<Heat[x][y][0].CelsiusTemperature)
      {Heat[x+1][y][1].CelsiusTemperature=+K*(((Heat[x][y][0].CelsiusTemperature-
Heat[x+1][y][0].CelsiusTemperature))/4);}
    if(Heat[x][y-1][0].CelsiusTemperature<Heat[x][y][0].CelsiusTemperature)
      {Heat[x][y-1][1].CelsiusTemperature=+K*(((Heat[x][y][0].CelsiusTemperature-Heat[x][y-
1][0].CelsiusTemperature))/4);}
    if(Heat[x][y+1][0].CelsiusTemperature<Heat[x][y][0].CelsiusTemperature)
```

```
      {Heat[x][y+1][1].CelsiusTemperature=+K*(((Heat[x][y][0].CelsiusTemperature-
Heat[x][y+1][0].CelsiusTemperature))/4);}
   //Fourier's Law of conductivity
    Heat[x-
1][y][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTemperature+273
,4)*joulesToCelsius/8;

Heat[x+1][y][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTemperat
ure+273,4)*joulesToCelsius/8;
    Heat[x][y-
1][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTemperature+273,4)
*joulesToCelsius/8;

Heat[x][y+1][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTemperat
ure+273,4)*joulesToCelsius/8;
    Heat[x-
1][y+1][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTemperature+
273,4)*joulesToCelsius/16;

Heat[x+1][y+1][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTempe
rature+273,4)*joulesToCelsius/16;
    Heat[x-1][y-
1][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTemperature+273,4)
*joulesToCelsius/16;

Heat[x+1][y+1][1].CelsiusTemperature=+.0000000567*Heat[x][y][1].emmisivity*pow(Heat[x][y][1].CelsiusTempe
rature+273,4)*joulesToCelsius/16;
    //Stefan Boltzmann Law of Radiation
   double total=0;
   for(int xa=0;xa<=9;xa++)
     { for(int ya=0; ya<=9;ya++)
       {   if(Heat[x][y][0].active[xa][ya])
         {total+=.01;}
       }
     }
  Heat[x][y][0].CelsiusTemperature+=Heat[x][y][0].ProducedHeat*total;

   //Burn loop
 }
//
void LoadForest(int sf)  //initializes forest and its values
 { AmbientTemperature=getAmbient();
  WindSpeed=getWind();
  WindAngle=getWindAngle();
  //Load for environment class
  for(int x=0; x<=sf-1;x++)
    {for(int y=0; y<=sf-1;y++)
       {Heat[x][y][0].CelsiusTemperature=AmbientTemperature;
        Heat[x][y][0].ProducedHeat=Produced(Heat[x][y][0].type)*3;
        Heat[x][y][0].FloorWetFuel=WetFuel(Heat[x][y][0].type);
        Heat[x][y][0].FloorDryFuel=DryFuel(Heat[x][y][0].type);
        Heat[x][y][0].WetFlashPoint=wetBasicFlash(Heat[x][y][0].type);
        Heat[x][y][0].DryFlashPoint=dryBasicFlash(Heat[x][y][0].type);
        Heat[x][y][0].InstantaneousFlashPoint=instantFlash(Heat[x][y][0].type);
        Heat[x][y][0].emmisivity=Emmisivity(Heat[x][y][0].type);
        //load values for patch class
```

```
            for(int xa=0; xa<=9;xa++)
               {for(int ya=0; ya<=9;ya++)
                  {Heat[x][y][0].active[xa][ya]=false;
                 }}}//set mini-patches unburned
     }
  //Heat[10][10][0].CelsiusTemperature=150000;
  }

}FOREST;
//---------------------------------------------------------------------
class FirePerimeter //keep track of all points to arc next iteration
{public:
 vector<double> xloc;
 vector<double> yloc;
 vector<double> zloc;
}Perimeter;
//---------------------------------------------------------------------
class Draw  //records last iteration to display it with less processing
{public:
 double xstart;
 double ystart;
 double zstart;
 vector <double> xloc;
 vector <double> yloc;
 vector <double> zloc;

 Draw(int num,vector <double> aloc,vector <double> bloc,vector <double> cloc,double xs,double ys,double zs)
    {//constructor
     xloc=aloc;
     yloc=bloc;
     zloc=cloc;
     xstart=xs;
     ystart=ys;
     zstart=0;
    }

 void Display() //function to display the past step
 {   glBegin(GL_LINES);
    if(xloc.size()!=0)
    {for(int x=0; x<=xloc.size()-1; x++)
       {
        glVertex3f(xloc[x],yloc[x],0);
        glVertex3f(xstart,ystart,0);
       }
       glEnd();
    }
 }
};
vector<Draw> DRAWING;//make it dynamic
//---------------------------------------------------------------------
class Fire  //handles fire procreation
{public:
 double xstart;
 double ystart;
 double zstart;
```

```
Fire(double x,double y, double z)
{ xstart=x;ystart=y; zstart=z;}
//constructor
void FireFlow()
 { bool done;
       vector<double> xloc,yloc,zloc;
     for(int x=0;x<=180;x+=(360/degrees))
     { //increments the degree of the radii
      double spread=0;s=0;
         do
         {
          //double b=ystart-tan(convert(x))*xstart;
          //double xa=xstart+(s/sqrt(1+pow(tan(convert(x)),2)));
          //double ya=(xstart+(s/sqrt(1+pow(tan(convert(x)),2))))*tan(convert(x))+b;
          //before and after polar corrections
          double xa= xstart + s*(cos(convert(x) + pi/2));
          double ya= ystart + s*(sin(convert(x) + pi/2));
          //keeps track of exact coordinates
          double za=0;
         if(xa<1){xa=1;}if(ya<1){ya=1;}
         if(xa>30){xa=30;}if(ya>30){ya=30;}
         //keeps flow within boundaries
         int aa=floor(xa);
         int ba=floor(ya);
         //casts as ints to ignite flow

         spread=spread+(increment*Geta(FOREST.Heat[aa][ba][0].type));
         //adds spread  until reaxhes maximum
         if(spread>=MSR)
            {done=true;
            //if it has reached the maximum time flow
               double newxa=xa,newya=ya,newza=0;
                   if(FOREST.WindSpeed!=0)
                   { newxa=WindVectoring('x',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                     newya=WindVectoring('y',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                     newza=WindVectoring('z',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                   } //computes wind vectoring
                   if(newxa<1){newxa=1;}if(newya<1){newya=1;}
                     bool record=firepath(newxa,newya,x);
                     if(record)
                     {Perimeter.xloc.push_back(newxa);
                      Perimeter.yloc.push_back(newya);
                      //keep track of fire end points and perimeter
                      xloc.push_back(newxa);
                      yloc.push_back(newya);
                      zloc.push_back(0);
                     }

            }
         else
         {done=false;  }

         if(!FOREST.Heat[aa][ba][0].Sustainable())
         {done=true;  //if conditions aren't suitable for fire
         double newxa=xa,newya=ya,newza=0;
                   if(FOREST.WindSpeed!=0)
```

```
                { newxa=WindVectoring('x',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                  newya=WindVectoring('y',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                  newza=WindVectoring('z',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                }
                if(newxa<1){newxa=1;}if(newya<1){newya=1;}
                  bool record=firepath(newxa,newya,x);
                  if(record)
                  {Perimeter.xloc.push_back(newxa);
                   Perimeter.yloc.push_back(newya);
                   //keep track of spread in vectors
                   xloc.push_back(newxa);
                   yloc.push_back(newya);
                   zloc.push_back(0);

        }}

         s=s+(increment*heatFactor(FOREST.Heat[aa][ba][0].CelsiusTemperature));
       }while(!done);
     spread=0; s=0;
       do
        { //computes the same just for the opposite quadrant
          //double b=ystart-tan(convert(x))*xstart;
          //double xa=xstart-(s/sqrt(1+pow(tan(convert(x)),2)));
          //double ya=(xstart-(s/sqrt(1+pow(tan(convert(x)),2))))*tan(convert(x))+b;
          double xa = xstart - s * (cos(convert(x) + pi/2));
          double ya = ystart - s * (sin(convert(x) + pi/2));
          double za=0;
         if(xa<1){xa=1;}if(ya<1){ya=1;}

          int aa=floor(xa);
          int ba=floor(ya);

       spread=spread+(increment*Geta(FOREST.Heat[aa][ba][0].type));

          if(spread>=MSR)
            {done=true;
                  double newxa=xa,newya=ya,newza=0;
                  if(FOREST.WindSpeed!=0)
                  {
newxa=WindVectoring('x',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,FOREST.WindSpeed,x+180);

newya=WindVectoring('y',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,FOREST.WindSpeed,x+180);

newza=WindVectoring('z',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,FOREST.WindSpeed,x+180);
                  }
                   if(newxa<1){newxa=1;}if(newya<1){newya=1;}
                   bool record=firepath(newxa,newya,x+180);
                   if(record)
                   {Perimeter.xloc.push_back(newxa);
                    Perimeter.yloc.push_back(newya);
                    xloc.push_back(newxa);
                    yloc.push_back(newya);
                    zloc.push_back(0);
                    }
            }
          else
```

```
        {done=false;
        }

        if(!FOREST.Heat[aa][ba][0].Sustainable())
        {done=true;  double newxa=xa,newya=ya,newza=0;
                if(FOREST.WindSpeed!=0)
                { newxa=WindVectoring('x',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                  newya=WindVectoring('y',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                  newza=WindVectoring('z',xa,ya,za,xstart,ystart,zstart,FOREST.WindAngle,2,x);
                }
                if(newxa<1){newxa=1;}if(newya<1){newya=1;}
                  bool record=firepath(newxa,newya,x);
                  if(record)
                  {Perimeter.xloc.push_back(newxa);
                   Perimeter.yloc.push_back(newya);

                   xloc.push_back(newxa);
                   yloc.push_back(newya);
                   zloc.push_back(0);       }}
       s=s+(increment*heatFactor(FOREST.Heat[aa][ba][0].CelsiusTemperature));
      }while(!done);
   }

  Draw tmp(degrees,xloc,yloc,zloc,xstart,ystart,zstart);
  DRAWING.push_back(tmp);

}
bool firepath(double xa,double ya,int degslope)//keeps track of the fire path
{  double legnth=sqrt(pow(xstart-xa,2)+pow(ystart-ya,2));
  int aa=0,ba=0,ca=0,da=0;
   for(double x=0; x<=legnth; x+=.09)
   { //tells the computer to burn the minipatches in the way of the fire
     double feta;
      if((degslope>=90&&degslope<=180)||(degslope>=270&&degslope<=360))
      {feta=90-(degslope%90);}
      if((degslope<90)||(degslope>180&&degslope<270))
      {feta=degslope%90;}

      if(degslope<=90)
      {aa=floor(xstart+x*cos(convert(feta)));
       ca=floor(10*(xstart+x*cos(convert(feta))));
       ba=floor(ystart+x*sin(convert(feta)));
       da=floor(10*(ystart+x*sin(convert(feta))));
      }

      if(degslope>90&&degslope<180)
      {aa=floor(xstart-x*cos(convert(feta)));
       ca=floor(10*(xstart-x*cos(convert(feta))));
       ba=floor(ystart+x*sin(convert(feta)));
       da=floor(10*(ystart+x*sin(convert(feta))));
      }
      if(degslope>=180&&degslope<=270)
      {aa=floor(xstart-(x*cos(convert(feta))));
       ca=floor(10*(xstart-(x*cos(convert(feta)))));
       ba=floor(ystart-(x*sin(convert(feta))));
       da=floor(10*(ystart-(x*sin(convert(feta)))));
```

```
        }
        if(degslope>270&&degslope<=360)
        {aa=floor(xstart+x*cos(convert(feta)));
         ca=floor(10*(xstart+x*cos(convert(feta))));
         ba=floor(ystart-x*sin(convert(feta)));
         da=floor(10*(ystart-x*sin(convert(feta))));
        }
       if(!FOREST.Heat[aa][ba][0].PreviousFire(ca%10,da%10)&&x<legnth-sqrt(.02))
         {FOREST.Heat[aa][ba][0].Ignite(ca%10,da%10);}
     }
  if(!FOREST.Heat[aa][ba][0].PreviousFire(ca%10,da%10))
    {return true;}
   return false;
  }
}fire(14,16,0),fIre(5,5,0);

vector<Fire> FIRE;
//*********************************************************************
bool Main(double ypers, double xpers,const double sf,int selected[3],int status,long timer,int TimeChange)
{   //initializes
    if(timer==0)
     {boundaries=sf+1;
    for(int x=0;x<=sf;x++)
      {for(int y=0; y<=sf; y++)
         { FOREST.Heat[x][y][0].type=getType(x,y);}
      }
     FOREST.LoadForest(sf);
     }
    switch(TimeChange)
    {//if there is a time step or not
    case 0:
      {          disContinue();
       if(status==1){ DrawForest(ypers,xpers,sf,selected);}
       if(status==2){ DrawHeat(ypers,xpers,sf,selected);}
       if(status==3){ DrawFire(ypers,xpers,sf,selected);}
       if(status==4){ DrawBurnt(ypers,xpers,sf,selected);}
       break;//which screen to display
      }

    case 1:
      { //if there is a time change
       if(status==1){ DrawForest(ypers,xpers,sf,selected);}
       if(status==2){ DrawHeat(ypers,xpers,sf,selected);}
       if(status==3){ DrawFire(ypers,xpers,sf,selected);}
       if(status==4){ DrawBurnt(ypers,xpers,sf,selected);}
    //which screen to display
       DRAWING.clear();
       fire.FireFlow();
       //fIre.FireFlow();
       if(FIRE.size()!=0)
         {for(int x=1; x<=FIRE.size()-1;x++)
            {
             FIRE[x].FireFlow();
            }
         }
       //compute the fire flow
```

```
        Continue(sf);
        Record();
        //record valid data to a text file
        break;
       }

   default:{break;}
  }
  return true;
}

//*****************************************************************
void DrawForest(double ypers, double xpers,double sf,int selected[3])
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            glLoadIdentity();

            glTranslatef(xpers,ypers,-25.0f);
   glBegin(GL_QUADS);
   {        glColor3f(0,.75,.5);
      glVertex3d(1,1,0);
      glVertex3d(1,sf,0);
      glVertex3d(sf,sf,0);
      glVertex3d(sf,1,0);



   }glEnd();

   for(int x=1; x<=sf; x++)
   { for(int y=1; y<=sf; y++)
     { glBegin(GL_QUADS);
            glColor3f( blue(FOREST.Heat[x][y][0].type)
,yellow(FOREST.Heat[x][y][0].type),red(FOREST.Heat[x][y][0].type));
      glVertex3d(x,y,0);
      glVertex3d(x+1,y,0);
      glVertex3d(x+1,y+1,0);
      glVertex3d(x,y+1,0);
   glEnd();
   }}
  //draw the fuel types

    glBegin(GL_QUADS);
   {        glColor3f(1.0f,1.0f,1.0f);
      glVertex3d(selected[0],selected[1],0);//selected[3]);
      glVertex3d(selected[0]+1,selected[1],0);//selected[3]);
      glVertex3d(selected[0]+1,selected[1]+1,0);//,selected[3]);
      glVertex3d(selected[0],selected[1]+1,0);//selected[3]);
   }glEnd();
    glBegin(GL_LINES);
            for(int x=1; x<=sf; x++)
   {                    glColor3f(0.0f,1.0f,1.0f);
         glVertex3d(1,x,0);                                     // Left Center Of Player
                                 glVertex3d(sf,x,0);
         glVertex3d(x,1,0);                                     // Left Center Of Player
```

```
                                glVertex3d(x,sf,0);
    }
            glEnd();

       for(int x=1; x<=sf; x++)
       { for(int y=1; y<=sf; y++)
      {
        glBegin(GL_QUADS);
            glColor3f( blue(FOREST.Heat[x][y][0].type)
,yellow(FOREST.Heat[x][y][0].type),red(FOREST.Heat[x][y][0].type));
       glVertex3d(((6/sf)*x)-xpers-11-2,((6/sf)*y)-ypers-11+12,0);
       glVertex3d(((6/sf)*x)-xpers-11-2+6/sf,((6/sf)*y)-ypers-11+12,0);
       glVertex3d(((6/sf)*x)-xpers-11-2+6/sf,((6/sf)*y)-ypers-11+12+6/sf,0);
       glVertex3d(((6/sf)*x)-xpers-11-2,((6/sf)*y)-ypers-11+12+6/sf,0);
       }
       }
       //draw the minimap                                                    // Done Drawing The Quad
    glEnd();


}
//*************************************************************
void DrawHeat(double ypers, double xpers,double sf, int selected[3])
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(xpers,ypers,-25.0f);

    for(int x=1; x<=sf; x++)
   { for(int y=1; y<=sf; y++)
     { glBegin(GL_QUADS);
            glColor3f( BLUE(FOREST.Heat[x][y][0].CelsiusTemperature)
,GREEN(FOREST.Heat[x][y][0].CelsiusTemperature),RED(FOREST.Heat[x][y][0].CelsiusTemperature));
       glVertex3d(x,y,0);
       glVertex3d(x+1,y,0);
       glVertex3d(x+1,y+1,0);
       glVertex3d(x,y+1,0);
   }}
    glEnd();

    glBegin(GL_QUADS);
   {        glColor3f(1.0f,1.0f,1.0f);
      glVertex3d(selected[0],selected[1],0);//selected[3]);
      glVertex3d(selected[0]+1,selected[1],0);//selected[3]);
      glVertex3d(selected[0]+1,selected[1]+1,0);//selected[3]);
      glVertex3d(selected[0],selected[1]+1,0);//selected[3]);
   }        glEnd();
   glBegin(GL_LINES);
            for(int x=1; x<=sf; x++)
   {                glColor3f(0.0f,1.0f,1.0f);
         glVertex3d(1,x,0);                                    // Left Center Of Player
                                glVertex3d(sf,x,0);
         glVertex3d(x,1,0);                                    // Left Center Of Player
                                glVertex3d(x,sf,0);
   }
```

```
   glEnd();
   DrawNum(1,FOREST.Heat[selected[0]][selected[1]][0].CelsiusTemperature,ypers+12,xpers+12);



}
//*****************************************************************
void DrawFire(double ypers, double xpers,double sf, int selected[3])
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(xpers,ypers,-25.0f);
    glBegin(GL_LINES);
           for(int x=1; x<=sf; x++)
   {                   glColor3f(0.0f,1.0f,1.0f);
         glVertex3d(1,x,0);                                    // Left Center Of Player
                           glVertex3d(sf,x,0);
         glVertex3d(x,1,0);                                    // Left Center Of Player
                           glVertex3d(x,sf,0);
   }
          glEnd();


   //DRAWING.clear();
   if(DRAWING.size()!=0)
  {for(int x=0; x<=DRAWING.size()-1;x++)
   {
         DRAWING[x].Display();
   }
  }    //displays all saved fire arcs
}
//*****************************************************************
void DrawBurnt (double ypers, double xpers,double sf, int selected[3])
{ glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glTranslatef(xpers,ypers,-25.0f);
    for(int x=1; x<=sf; x++)
   { for(int y=1; y<=sf; y++)
     { for(int xa=0; xa<=9; xa++)
       { for(int ya=0; ya<=9; ya++)
         {
              double a=xa,b=ya;
     glBegin(GL_QUADS);
           if(FOREST.Heat[x][y][0].active[xa][ya]){glColor3f(1,1,0);}
      if(!FOREST.Heat[x][y][0].active[xa][ya]){glColor3f(0,0,1);}
      glVertex3d(x+(a/10),y+(b/10),0);
      glVertex3d(x+.1+(a/10),y+(b/10),0);
      glVertex3d(x+.1+(a/10),y+.1+(b/10),0);
      glVertex3d(x+(a/10),y+.1+(b/10),0);
   glEnd();
   }}}}


    glBegin(GL_LINES);
```

```
          for(int x=1; x<=sf; x++)
   {                  glColor3f(0.0f,1.0f,1.0f);
         glVertex3d(1,x,0);                                    // Left Center Of Player
                              glVertex3d(sf,x,0);
         glVertex3d(x,1,0);                                    // Left Center Of Player
                              glVertex3d(x,sf,0);
   }
          glEnd();

}
//*****************************************************************
void MakeFire()
{if(Perimeter.xloc.size())
   { for(int x=0; x<=Perimeter.xloc.size()-1; x++)
     {     Fire Temporary(Perimeter.xloc[x-1],Perimeter.yloc[x-1],0);
          FIRE.push_back(Temporary);
     }
   }  //add fires to vectors
Perimeter.xloc.clear();
Perimeter.yloc.clear();
Perimeter.zloc.clear();
}
//*****************************************************************
void Continue(int sf)
{
    for(int x=1; x<=sf-1; x++)
    {   for(int y=1; y<=sf-1; y++)
          {FOREST.HeatFlow(x,y);}
    }
    FOREST.Rationalize(sf);
    //create new fires
    FIRE.clear();
    MakeFire();
}
//*****************************************************************
void disContinue()
{
Perimeter.xloc.clear();
Perimeter.yloc.clear();
Perimeter.zloc.clear();
}//crear extra vectors
//*****************************************************************
void Record()
{ int num=0;
for(int w=0; w<=30-1;w++)
  {for(int x=0; x<=30-1;x++)
    {for(int y=0; y<=9;y++)
     {for(int z=0; z<=9;z++)
       {if(FOREST.Heat[w][x][0].PreviousFire(y,z))
       {num++;}}}}}
       PutOut(num,num);
}
```

Data.h

Data Storehouse

```
//Data.h Where data is stored for patches mainly
//based on fuel type values

#include<math.h>
#define pi 3.141592654

double inline convert(double con)
{  return con*pi/180;}

float red(int ft)
{
   if(ft==0){return 0;}
   if(ft==1){return 0;}
   if(ft==2){return 0.5;}
   if(ft==3){return 1;}
   if(ft==4){return 0;}
   if(ft==5){return 0;}
   if(ft==6){return 0;}
   if(ft==7){return 1;}
   if(ft==8){return 0;}
   if(ft==9){return 1;}
   if(ft==10){return 0;}
   return 0;
}
float yellow(int ft)
{
   if(ft==0){return 0;}
   if(ft==1){return 0;}
   if(ft==2){return 0;}
   if(ft==3){return 0;}
   if(ft==4){return 0.5;}
   if(ft==5){return 1;}
   if(ft==6){return 0;}
   if(ft==7){return 0;}
   if(ft==8){return 0.5;}
   if(ft==9){return 1;}
   if(ft==10){return 0;}
   return 0;
}
float blue(int ft)
{
   if(ft==0){return 0;}
   if(ft==1){return 0;}
   if(ft==2){return 0.5;}
   if(ft==3){return 1;}
   if(ft==4){return 0;}
   if(ft==5){return 0;}
   if(ft==6){return 0.5;}
   if(ft==7){return 0;}
   if(ft==8){return 1;}
   if(ft==9){return 0;}
   if(ft==10){return 1;}
   return 0;
```

```
}
//spread values based on fuel type
 float Geta(int ft)
{
    if(ft==0){return 1.5;}
    if(ft==1){return 1;}
    if(ft==2){return .5;}
    if(ft==3){return 3;}
    if(ft==4){return 2;}
    if(ft==5){return 1;}
    if(ft==6){return .5;}
    if(ft==7){return 2;}
    if(ft==8){return 1;}
    if(ft==9){return 2.5;}
    if(ft==10){return 1;}
    return 0;
}
//RGB heat colors
  float RED(double Heat)
{  if(Heat<0)
      {return 1;}
   if(Heat<=1500&&Heat>=0)
      {return 1-Heat/1500;}
   else {return (Heat-1500)/1500;}
}

 float BLUE(double Heat)
{  if(Heat<0)
      {return 0;}
   if(Heat<=1500&&Heat>=0)
      {return Heat/1500;}
   else {return 1;}
}
  float GREEN(double Heat)
{  if(Heat>1500)
   {return (Heat-1500)/1500;}
   else
   {return 0;}
}
float Produced(int ft)
{   if(ft==0){return 100;}
    if(ft==1){return 100;}
    if(ft==2){return 100;}
    if(ft==3){return 100;}
    if(ft==4){return 100;}
    if(ft==5){return 100;}
    if(ft==6){return 100;}
    if(ft==7){return 100;}
    if(ft==8){return 100;}
    if(ft==9){return 100;}
    if(ft==10){return 100;}
    return 0;
}
int WetFuel(int ft)
{   if(ft==0){return 1;}
    if(ft==1){return 1;}
```

```
    if(ft==2){return 1;}
    if(ft==3){return 1;}
    if(ft==4){return 1;}
    if(ft==5){return 1;}
    if(ft==6){return 1;}
    if(ft==7){return 1;}
    if(ft==8){return 1;}
    if(ft==9){return 1;}
    if(ft==10){return 1;}
    return 0;
}
int DryFuel(int ft)
{   if(ft==0){return 1;}
    if(ft==1){return 1;}
    if(ft==2){return 1;}
    if(ft==3){return 1;}
    if(ft==4){return 1;}
    if(ft==5){return 1;}
    if(ft==6){return 1;}
    if(ft==7){return 1;}
    if(ft==8){return 1;}
    if(ft==9){return 1;}
    if(ft==10){return 1;}
    return 0;
}
double wetBasicFlash(int ft)
{   if(ft==0){return 200;}
    if(ft==1){return 200;}
    if(ft==2){return 200;}
    if(ft==3){return 200;}
    if(ft==4){return 200;}
    if(ft==5){return 200;}
    if(ft==6){return 200;}
    if(ft==7){return 200;}
    if(ft==8){return 200;}
    if(ft==9){return 200;}
    if(ft==10){return 200;}
    return 0;

}
double dryBasicFlash(int ft)
{   if(ft==0){return 0;}
    if(ft==1){return 0;}
    if(ft==2){return 0;}
    if(ft==3){return 0;}
    if(ft==4){return 0;}
    if(ft==5){return 0;}
    if(ft==6){return 0;}
    if(ft==7){return 0;}
    if(ft==8){return 0;}
    if(ft==9){return 0;}
    if(ft==10){return 0;}
    return 0;
}
double instantFlash(int ft)
{   if(ft==0){return 500;}
```

```
   if(ft==1){return 500;}
   if(ft==2){return 500;}
   if(ft==3){return 500;}
   if(ft==4){return 500;}
   if(ft==5){return 500;}
   if(ft==6){return 500;}
   if(ft==7){return 500;}
   if(ft==8){return 500;}
   if(ft==9){return 500;}
   if(ft==10){return 500;}
   return 0;
}
//MONITERS HEATS EFFECT UPON FIRE FLOW
float heatFactor(int temp)
{
 return 1;
}
float Emmisivity(int ft)
{
   if(ft==0){return .5;}
   if(ft==1){return .5;}
   if(ft==2){return .5;}
   if(ft==3){return .5;}
   if(ft==4){return .5;}
   if(ft==5){return .5;}
   if(ft==6){return .5;}
   if(ft==7){return .5;}
   if(ft==8){return .5;}
   if(ft==9){return .5;}
   if(ft==10){return .5;}
   return 0;
}
//draws numbers to the screen (incomplete)
void DrawNum(double size,double temp,double xl,double yl)
{ glBegin(GL_QUADS);
    int ones=floor(temp);
  for(int x=0;x<=7;x++)
  {
        for(int y=0; y<=7;y++)
        {/*if(getNum(ones%10-1,x,y)==1)
         //{
           glColor3f(1.0f,1.0f,1.0f);
      glVertex3d(10+x/8,    10+y/8    ,0);
      glVertex3d(10+x/8+1/8,10+y/8    ,0);
      glVertex3d(10+x/8+1/8,10+y/8+1/8,0);
      glVertex3d(10+x/8,10+y/8+1/8    ,0);
     // glVertex3d(10,10+1,0);
      //glVertex3d(10,10,0);
      //glVertex3d(10+1,10,0);
      //glVertex3d(10+1,10+1,0);

        }*/}
  }

    glEnd();
}
```

Fire.cpp

Initializatione Code borrowed and edited from nehe.gamedev.net
You don't want to reinvent the wheel
//Chris Morrison
//Initialization Source borrowed and edited from nehe.gamedev.net
//2.5d fire flow program

```
//**********************************************************************
#include <windows.h>                // Header File For Windows
#include <gl\gl.h>                  // Header File For The OpenGL32 Library
#include <gl\glu.h>                 // Header File For The GLu32 Library
#include "Main.h"
#include<iostream.h>
#include<fstream.h>
#define sf size4feet



HDC             hDC=NULL;           // Private GDI Device Context
HGLRC           hRC=NULL;           // Permanent Rendering Context
HWND            hWnd=NULL;          // Holds Our Window Handle
HINSTANCE       hInstance;          // Holds The Instance Of The Application

bool    keys[256];                  // Array Used For The Keyboard Routine
bool    active=TRUE;                // Window Active Flag Set To TRUE By Default
bool    fullscreen=TRUE;            // Fullscreen Flag Set To Fullscreen Mode By Default
bool   Return=false;
int status=1;
int selected[3]={0,0,0};

double xpers=-11;
double ypers=-10;
double size4feet=30;

unsigned long timer=0;

LRESULT     CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
// Declaration For WndProc
//**********************************************************************
GLvoid ReSizeGLScene(GLsizei width, GLsizei height)
        // Resize And Initialize The GL Window
{
        if (height==0)      // Prevent A Divide By Zero By
        {
                height=1;           // Making Height Equal One
        }

        glViewport(0,0,width,height);       // Reset The Current Viewport

        glMatrixMode(GL_PROJECTION);            // Select The Projection Matrix
        glLoadIdentity();                   // Reset The Projection Matrix

        // Calculate The Aspect Ratio Of The Window
        gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,35.0f);

        glMatrixMode(GL_MODELVIEW);             // Select The Modelview Matrix
```

```
	glLoadIdentity();				// Reset The Modelview Matrix
}
//*********************************************************************
int InitGL(GLvoid)			// All Setup For OpenGL Goes Here
{
	glShadeModel(GL_SMOOTH);		// Enable Smooth Shading
	glClearColor(0.0f, 0.0f, 0.0f, 0.5f);	// Black Background
	glClearDepth(1.0f);				// Depth Buffer Setup
	glEnable(GL_DEPTH_TEST);		// Enables Depth Testing
	glDepthFunc(GL_LEQUAL);				// The Type Of Depth Testing To Do
	glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
	// Really Nice Perspective Calculations
	return TRUE;					// Initialization Went OK
}
//*********************************************************************
GLvoid KillGLWindow(GLvoid)			// Properly Kill The Window
{
	if (fullscreen)					// Are We In Fullscreen Mode?
	{
		ChangeDisplaySettings(NULL,0);	// If So Switch Back To The Desktop
		ShowCursor(TRUE);				// Show Mouse Pointer
	}

	if (hRC)			// Do We Have A Rendering Context?
	{
		if (!wglMakeCurrent(NULL,NULL))
	// Are We Able To Release The DC And RC Contexts?
		{
		MessageBox(NULL,"Release Of DC And RC Failed.","SHUTDOWN ERROR",MB_OK |
		MB_ICONINFORMATION);
		}

		if (!wglDeleteContext(hRC))		// Are We Able To Delete The RC?
		{
			MessageBox(NULL,"Release Rendering Context Failed.","SHUTDOWN
			ERROR",MB_OK | MB_ICONINFORMATION);
		}
		hRC=NULL;					// Set RC To NULL
	}

	if (hDC && !ReleaseDC(hWnd,hDC))	// Are We Able To Release The DC
	{
		MessageBox(NULL,"Release Device Context Failed.","SHUTDOWN ERROR",MB_OK |
MB_ICONINFORMATION);
		hDC=NULL;					// Set DC To NULL
	}

	if (hWnd && !DestroyWindow(hWnd))// Are We Able To Destroy The Window?
	{
		MessageBox(NULL,"Could Not Release hWnd.","SHUTDOWN ERROR",MB_OK |
		MB_ICONINFORMATION);
		hWnd=NULL;					// Set hWnd To NULL
	}

	if (!UnregisterClass("OpenGL",hInstance))  // Are We Able To Unregister Class
	{
```

```
                MessageBox(NULL,"Could Not Unregister Class.","SHUTDOWN ERROR",MB_OK |
                MB_ICONINFORMATION);
                hInstance=NULL;                         // Set hInstance To NULL
        }
}

/*This Code Creates Our OpenGL Window.  Parameters Are:                 *
 *title                 - Title To Appear At The Top Of The Window              *
 *      width           - Width Of The GL Window Or Fullscreen Mode             *
 *      height          - Height Of The GL Window Or Fullscreen Mode            *
 *      bits            - Number Of Bits To Use For Color (8/16/24/32)          *
 *      fullscreenflag  - Use Fullscreen Mode (TRUE) Or Windowed Mode (FALSE)*/
//*********************************************************************
BOOL CreateGLWindow(char* title, int width, int height, int bits, bool fullscreenflag)
{
        GLuint          PixelFormat;    // Holds The Results After Searching For A Match
        WNDCLASS        wc;             // Windows Class Structure
        DWORD           dwExStyle;      // Window Extended Style
        DWORD           dwStyle;// Window Style
        RECT            WindowRect;     // Grabs Rectangle Upper Left / Lower Right Values
        WindowRect.left=(long)0; // Set Left Value To 0
        WindowRect.right=(long)width;   // Set Right Value To Requested Width
        WindowRect.top=(long)0; // Set Top Value To 0
        WindowRect.bottom=(long)height; // Set Bottom Value To Requested Height
        fullscreen=fullscreenflag;  // Set The Global Fullscreen Flag

        hInstance= GetModuleHandle(NULL);   // Grab An Instance For Our Window
        wc.style  = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
        // Redraw On Size, And Own DC For Window.
        wc.lpfnWndProc= (WNDPROC) WndProc;  // WndProc Handles Messages
        wc.cbClsExtra     = 0;                          // No Extra Window Data
        wc.cbWndExtra= 0;                               // No Extra Window Data
        wc.hInstance= hInstance;            // Set The Instance
        wc.hIcon= LoadIcon(NULL, IDI_WINLOGO); // Load The Default Icon
        wc.hCursor= LoadCursor(NULL, IDC_ARROW);     // Load The Arrow Pointer
        wc.hbrBackground= NULL;                         // No Background Required For GL
        wc.lpszMenuName= NULL;                          // We Don't Want A Menu
        wc.lpszClassName= "OpenGL";     // Set The Class Name

        if (!RegisterClass(&wc))            // Attempt To Register The Window Class
        {
                MessageBox(NULL,"Failed To Register The Window
                Class.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                return FALSE;                           // Return FALSE
        }

        if (fullscreen)                         // Attempt Fullscreen Mode?
        {
                DEVMODE dmScreenSettings;    // Device Mode
                memset(&dmScreenSettings,0,sizeof(dmScreenSettings));
                // Makes Sure Memory's Cleared
                dmScreenSettings.dmSize=sizeof(dmScreenSettings);
                // Size Of The Devmode Structure
                dmScreenSettings.dmPelsWidth= width;   // Selected Screen Width
                dmScreenSettings.dmPelsHeight= height; // Selected Screen Height
                dmScreenSettings.dmBitsPerPel= bits;     // Selected Bits Per Pixel
```

```
dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;
                // Try To Set Selected Mode And Get Results.  NOTE: CDS_FULLSCREEN Gets Rid Of Start
                Bar.
        if(ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=DISP_CH
        ANGE_SUCCESSFUL)
                {// If The Mode Fails, Offer Two Options.  Quit Or Use Windowed Mode.
                        if (MessageBox(NULL,"The Requested Fullscreen Mode Is Not Supported By\nYour
                        Video Card. Use Windowed Mode Instead?","NeHe
                        GL",MB_YESNO|MB_ICONEXCLAMATION)==IDYES)
                        {
                        fullscreen=FALSE;                       // Windowed Mode Selected.  Fullscreen = FALSE
                        }
                        else
                        {
                // Pop Up A Message Box Letting User Know The Program Is Closing.
                        MessageBox(NULL,"Program Will Now Close.","ERROR",MB_OK|MB_ICONSTOP);
                        return FALSE;                   // Return FALSE
                        }
                }
        }

        if (fullscreen)                                 // Are We Still In Fullscreen Mode?
        {
                dwExStyle=WS_EX_APPWINDOW;
        // Window Extended Style
                dwStyle=WS_POPUP;
        // Windows Style
                ShowCursor(FALSE);
        // Hide Mouse Pointer
        }
        else
        {
                dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;
        // Window Extended Style
                dwStyle=WS_OVERLAPPEDWINDOW;
        // Windows Style
        }

        AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);
        // Adjust Window To True Requested Size

                // Create The Window
        if (!(hWnd=CreateWindowEx(
                dwExStyle,                              // Extended Style For The Window
                "OpenGL",                               // Class Name
                title,                                  // Window Title
                dwStyle |                               // Defined Window Style
                WS_CLIPSIBLINGS |       // Required Window Style
                WS_CLIPCHILDREN,        // Required Window Style
                0, 0,                                   // Window Position
                WindowRect.right-WindowRect.left,       // Calculate Window Width
                WindowRect.bottom-WindowRect.top,// Calculate Window Height
                NULL,                           // No Parent Window
                NULL,                                   // No Menu
```

```
        hInstance,                      // Instance
NULL)))                         // Dont Pass Anything To WM_CREATE
{
        KillGLWindow();                 // Reset The Display
        MessageBox(NULL,"Window Creation
        Error.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                   // Return FALSE
}

static  PIXELFORMATDESCRIPTOR pfd=
// pfd Tells Windows How We Want Things To Be
{
        sizeof(PIXELFORMATDESCRIPTOR),
// Size Of This Pixel Format Descriptor
        1,                                      // Version Number
        PFD_DRAW_TO_WINDOW |    // Format Must Support Window
        PFD_SUPPORT_OPENGL |            // Format Must Support OpenGL
        PFD_DOUBLEBUFFER,               // Must Support Double Buffering
        PFD_TYPE_RGBA,                  // Request An RGBA Format
        bits,                           // Select Our Color Depth
        0, 0, 0, 0, 0, 0,               // Color Bits Ignored
        0,                              // No Alpha Buffer
        0,                              // Shift Bit Ignored
        0,                              // No Accumulation Buffer
        0, 0, 0, 0,                     // Accumulation Bits Ignored
        16,                             // 16Bit Z-Buffer (Depth Buffer)
        0,                              // No Stencil Buffer
        0,                              // No Auxiliary Buffer
        PFD_MAIN_PLANE,                 // Main Drawing Layer
        0,                              // Reserved
        0, 0, 0                         // Layer Masks Ignored
};

if (!(hDC=GetDC(hWnd)))                 // Did We Get A Device Context?
{
        KillGLWindow();                 // Reset The Display
        MessageBox(NULL,"Can't Create A GL Device
        Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                   // Return FALSE
}

if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))  // Did Windows Find A Matching Pixel Format?
{
        KillGLWindow();                         // Reset The Display
        MessageBox(NULL,"Can't Find A Suitable
        PixelFormat.","ERROR",MB_OK|MB_ICONEXCLAMATION);        return FALSE;
                                // Return FALSE
}

if(!SetPixelFormat(hDC,PixelFormat,&pfd))
// Are We Able To Set The Pixel Format?
{
KillGLWindow();                         // Reset The Display
MessageBox(NULL,"Can't Set The PixelFormat.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                   // Return FALSE
}
```

```
	if (!(hRC=wglCreateContext(hDC)))
	// Are We Able To Get A Rendering Context?
	{
		KillGLWindow();				// Reset The Display
		MessageBox(NULL,"Can't Create A GL Rendering
		Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
		return FALSE;				// Return FALSE
	}

	if(!wglMakeCurrent(hDC,hRC))	// Try To Activate The Rendering Context
	{
		KillGLWindow();			// Reset The Display
		MessageBox(NULL,"Can't Activate The GL Rendering
		Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
		return FALSE;			// Return FALSE
	}

	ShowWindow(hWnd,SW_SHOW);// Show The Window
	SetForegroundWindow(hWnd);	// Slightly Higher Priority
	SetFocus(hWnd);			// Sets Keyboard Focus To The Window
	ReSizeGLScene(width, height);	// Set Up Our Perspective GL Screen

	if (!InitGL())				// Initialize Our Newly Created GL Window
	{
		KillGLWindow();			// Reset The Display
		MessageBox(NULL,"Initialization Failed.","ERROR",MB_OK|MB_ICONEXCLAMATION);
		return FALSE;			// Return FALSE
	}

	return TRUE;				// Success
}
//***************************************************************************
LRESULT CALLBACK WndProc(HWND
		hWnd,				// Handle For This Window
		UINT	uMsg,			// Message For This Window
		WPARAM		wParam,// Additional Message Information
		LPARAM		lParam)	// Additional Message Information
{
	switch (uMsg)				// Check For Windows Messages
	{
		case WM_ACTIVATE:	// Watch For Window Activate Message
		{
			if (!HIWORD(wParam)) // Check Minimization State
			{
				active=TRUE;		// Program Is Active
			}
			else
			{
				active=FALSE; // Program Is No Longer Active
			}

			return 0;		// Return To The Message Loop
		}

		case WM_SYSCOMMAND:		// Intercept System Commands
```

```
		{
			switch (wParam)  // Check System Calls
			{
				case SC_SCREENSAVE:
		// Screensaver Trying To Start?
				case SC_MONITORPOWER:
		// Monitor Trying To Enter Powersave?
				return 0;
		// Prevent From Happening
			}
			break;    // Exit
		}

		case WM_CLOSE:			// Did We Receive A Close Message?
		{
			PostQuitMessage(0);	// Send A Quit Message
			return 0;		// Jump Back
		}

		case WM_KEYDOWN:    // Is A Key Being Held Down?
		{
			keys[wParam] = TRUE;// If So, Mark It As TRUE
			return 0;		// Jump Back
		}

		case WM_KEYUP:			// Has A Key Been Released?
		{
			keys[wParam] = FALSE;
		       Return=true;				// If So, Mark It As FALSE
			return 0;			// Jump Back
		}

		case WM_SIZE:			// Resize The OpenGL Window
		{
			ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));  // LoWord=Width,
			HiWord=Height
			return 0;			// Jump Back
		}
	}// Pass All Unhandled Messages To DefWindowProc
	return DefWindowProc(hWnd,uMsg,wParam,lParam);
}
//*********************************************************************
int WINAPI WinMain(
	HINSTANCE	hInstance,		// Instance
	HINSTANCE	hPrevInstance,		// Previous Instance
	LPSTR lpCmdLine,			// Command Line Parameters
	int nCmdShow)			// Window Show State
{
	MSG	msg;				// Windows Message Structure
	BOOL	done=FALSE;			// Bool Variable To Exit Loop
	// Ask The User Which Screen Mode They Prefer
	if (MessageBox(NULL,"Are you cool?",
"Coolness",MB_YESNO|MB_ICONQUESTION)==IDNO)
```

```
  {
        fullscreen=FALSE;                   // Windowed Mode
  }

  // Create Our OpenGL Window
  if (!CreateGLWindow("Fire!",640,480,16,fullscreen))
  {
        return 0;                   // Quit If Window Was Not Created
  }
  Load();
  while(!done)                              // Loop That Runs While done=FALSE
  {
        if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))  // Is There A Message Waiting?
        {
              if (msg.message==WM_QUIT)
              // Have We Received A Quit Message?
              {
              done=TRUE;                    // If So done=TRUE
              }
              else                          // If Not, Deal With Window Messages
              {
                    TranslateMessage(&msg); // Translate The Message
                    DispatchMessage(&msg);  // Dispatch The Message
              }
        }
        else                                // If There Are No Messages
        {
         if ((active && !Main( ypers, xpers, sf,selected,status,timer,0)) || keys[VK_ESCAPE])     //
        Active?  Was There A Quit Received?
              {
                    done=TRUE;
}
          else                      // Not Time To Quit, Update Screen
              {
                    SwapBuffers(hDC);         // Swap Buffers (Double Buffering)
              }
      if(keys[VK_SHIFT]&&keys[VK_UP])
 {ypers--;}
if(keys[VK_SHIFT]&&keys[VK_DOWN])
 {ypers++;}
if(keys[VK_SHIFT]&&keys[VK_LEFT])
 {xpers++;}
if(keys[VK_SHIFT]&&keys[VK_RIGHT])
 {xpers--;}
if(keys[VK_UP])
 {selected[1]++;}
if(keys[VK_DOWN])
 {selected[1]--;}
if(keys[VK_LEFT])
 {selected[0]--;}
if(keys[VK_RIGHT])
 {selected[0]++;}
if(keys[VK_RETURN]&&Return)
 {timer++;
 Main(ypers, xpers, sf,selected,status,timer,1);
 Return=false;
```

```
          }
        if(selected[0]<1)
          {selected[0]=1;}
        if(selected[1]<1)
          {selected[1]=1;}
        if(selected[0]>sf-1)
          {selected[0]=sf-1;}
        if(selected[1]>sf-1)
          {selected[1]=sf-1;}
        if (keys[VK_F1])
                                {status=1;
          Main(ypers, xpers, sf,selected,status,timer,2);
          }
        if(keys[VK_F2])
                                {status=2;
         // Main(ypers, xpers, sf,selected,status,timer,2);
          }
                        if(keys[VK_F3])
                                {status=3;
        //   Main(ypers, xpers, sf,selected,status,timer,2);
          }
        if(keys[VK_F4])
                                {status=4;
      //     Main(ypers, xpers, sf,selected,status,timer,2);
          }
                        }
           }

           // Shutdown
           KillGLWindow();                                                                                  // Kill
The Window
           return (msg.wParam);                                                                  // Exit The Program
}
```

Load.h

```
//loads basis of patches and such
//from text Files
#include<iostream.h>
#include<fstream.h>

vector<int> record;
struct Factors
{ double AmbientTemp;
  double WindSpeed;
  double WindAngle;
}Forest;

int level[300][300];
int num[10][8][8];
//*********************************************************
 void Load()
 { ifstream coolestfile;
   ifstream coolfile;
   ifstream Numbers;
   coolfile.open("forest.txt");
   for(int x=0; x<=29; x++)
   {for(int y=0; y<=29; y++)
    {
            coolfile>>level[x][y];
    }
   }

 coolestfile.open("Factors.txt");

     coolestfile>>Forest.AmbientTemp>>Forest.WindSpeed>>Forest.WindAngle;

Numbers.open("numbers.txt");  //the following loads the text 0 and 1 renderings of numbers for this program
    for(int n=0;n<=9; n++) //0-9
     {
     for(int y=0;  y<=7; y++) //8x8
        {
     for(int x=0; x<=7;x++)
        {
            Numbers>>num[n][x][y];
        }
      }
    }
     coolestfile.close();
     coolfile.close();
     Numbers.close();
}
//*********************************************************
int getType(int x,int y)
{
   return level[x][y];
}
//*********************************************************
int getNum(int x,int y,int z)
{
```

```
   return num[x][y][z];
}
//***********************************************************
inline double getAmbient()
{return Forest.AmbientTemp;}
//***********************************************************
inline double getWind()
{return Forest.WindSpeed;}
//***********************************************************
inline double getWindAngle()
{ return Forest.WindAngle;}
//***********************************************************
void PutOut(int minifires, int totalfires)
{    record.push_back(minifires);
    ofstream coolerfile;
     coolerfile.open("record.txt",ios::out);
    for(int x=0; x<=record.size()-1; x++)
    {
      coolerfile<<record[x]<<'\n';
    }
    //coolerfile<<totalfires;
    coolerfile.close();
}
//***********************************************************
```

Wind.h

```
//computes wind vectoring
//This version is incomplete
#include<Math.h>

double WindVectoring(char Component,double xa,double ya,double za,double xstart,double ystart,double zstart,int
WindAngle,double WindSpeed,int xb)
 {    WindSpeed=0;
     double newxa=xa,newya=ya,newza=za;
     double legnth=sqrt(pow(xstart-xa,2)+pow(ystart-ya,2));
     double x;
     int feta;
     int diff=abs(xb-WindAngle);
     bool neg;
     if(xb-WindAngle<0)
     {neg=true;}
     if(xb-WindAngle>=0)
     {neg=false;}
     if(diff<=90)
     {feta=diff;}
     if(diff>90&&diff<=180)
     {feta=90-diff%90;}
     if(diff>180&&diff<=270)
     {feta=diff%180;}
     if(diff>270)
     {feta=90-diff%270;}
     x=WindSpeed*cos(convert(feta));
     if(neg)
     {x*=-1;}
     double newlength=legnth;//+x;
     double xc=xstart+newlength*(cos(convert(xb)));
     double yc=ystart+newlength*(sin(convert(xb)));
     double zc=0;
     newxa=xc;
     newya=yc;
     newza=zc;

     if(Component=='x')
     {return newxa;}
     if(Component=='y')
     {return newya;}
     if(Component=='z')
     {return newza;}

 }
```