2008

DATA ASSIMILATION AND VISUALIZATION FOR ENSEMBLE WILDLAND FIRE MODELS

Soham Chakraborty
University of Kentucky, sohaminator@gmail.com

ABSTRACT OF THESIS

DATA ASSIMILATION AND VISUALIZATION FOR ENSEMBLE WILDLAND FIRE MODELS

This thesis describes an observation function for a dynamic data driven application system designed to produce short range forecasts of the behavior of a wildland fire. The thesis presents an overview of the atmosphere-fire model, which models the complex interactions between the fire and the surrounding weather and the data assimilation module which is responsible for assimilating sensor information into the model. Observation plays an important role in data assimilation as it is used to estimate the model variables at the sensor locations. Also described is the implementation of a portable and user friendly visualization tool which displays the locations of wildfires in the Google Earth virtual globe.

KEYWORDS: Observation function, Atmosphere-fire model, Dynamic Data-Driven Application System, Google Earth, Data Assimilation

Soham Chakraborty

04/18/2008

DATA ASSIMILATION AND VISUALIZATION

FOR ENSEMBLE WILDLAND FIRE MODELS

By

Soham Chakraborty

Dr. Craig C. Douglas

Director of Thesis

Dr. Raphael Finkel

Director of Graduate Studies

04/18/2008

Name	Date

THESIS

Soham Chakraborty

The Graduate School

University of Kentucky

2008

DATA ASSIMILATION AND VISUALIZATION FOR ENSEMBLE WILDLAND FIRE MODELS

THESIS

A thesis submitted in partial fulfillment of the requirements for the Master of Science in the College of Engineering at the University of Kentucky

By

Soham Chakraborty

Lexington, Kentucky

Director: Dr Craig C. Douglas, Professor of Computer Science

Lexington, Kentucky

2008

Acknowledgements

I would like to thank my Thesis Chair, Dr. Craig C. Douglas for his invaluable assistance and guidance at every stage of the thesis process. I also express my sincere gratitude to Dr. Janice Coen for her timely and insightful advice, at a very critical juncture, which allowed me to complete the project on schedule.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FILES

File 1. Thesis.pdf

Chapter One: Introduction

We describe a scheme for the implementation of an observation function and a visualization paradigm in a Dynamic Data-Driven Application System (DDDAS) to predict the behavior of wildland fires. Producing a short range forecast of the behavior of a wildfire is a challenging problem because it is a rapidly changing phenomenon [13]. There is also the added difficulty of assimilating data from disparate remote sensors, which is often out of order and of questionable accuracy, into the model and use it to drive the simulations [13].

At the heart of the DDDAS is a coupled atmosphere-fire model consisting of a numeric weather prediction model and a fire behavior model [12]. The coupled atmosphere-fire model simulates the complex interactions between a wildland fire and the local weather. The model does not directly use sensor data to make its predictions. Since sensors have errors and are unevenly distributed in time and space, the system uses data assimilation to compare the observed data with the last forecast (synthetic data) and make corrections to this forecast. The corrected forecast serves as the basis for the model's next forecast. Synthetic data, required by the data assimilation module, is extracted from the model state by the observation function. The observation function examines the positions of the sensors within the model grid and approximates the values of various quantities like temperature, wind velocity, and atmospheric pressure at these locations from the model variables located at the grid points. This forms the synthetic data that is compared with the real sensor data by the data assimilation module. A key component of this research is to create a wildland fire code independent observation function system.

The thesis also describes the implementation of a software tool to display the positions of wildfires in Google Earth. Google Earth is a free mapping tool released by Google and is available on many platforms. It presents a 3-D virtual globe which the users can rotate, zoom in, and zoom out of. Google Earth incorporates a powerful set of tools that allow customized images and objects to be displayed on the virtual globe. The visualization tool allows anyone with a Google Earth client to follow the progress of a wildfire in near real time.

Section 2 provides a lengthy overview of NCAR models, which have been developed over the past 40 years. In section 3 we see new research to translate a static model into a dynamic data-driven model. And in section 4 we present a new way of easily visualizing fires on a wide variety of devices and platforms.

Chapter Two: Background

This dissertation is part of an ongoing attempt to build a Dynamic Data-Driven Application System (DDDAS) for short term prediction of wild fire behavior. The system is built upon NCAR's (National Center for Atmospheric Research) coupled atmosphere-fire model which simulates the spread of the fire taking into consideration the effect of the fire on the surrounding atmosphere as well as the effects of the fire-induced winds on the fire itself [12]. This model's earliest ancestor dates back to 1990. The system has been adapted to accept real-time weather information, images and sensor streams, and changes the prediction, if necessary, when fresh data is made available [13]. In addition, components for saving, modifying, and restoring the state of an atmosphere-fire model have been added. Other features allow modification of ensemble member states using ensemble data assimilation algorithms by comparing the synthetic data generated from member states with real world data (from sensors) and adaptation of the computational results to suit different user needs.

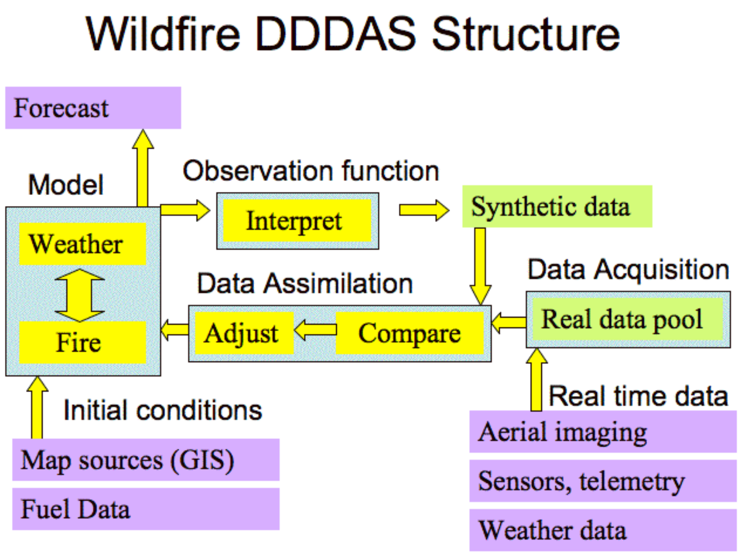


Figure 2.1. Schematic diagram of the wildfire DDDAS application [12]

2.1 Atmosphere-Fire Model

2.1.1 Model Overview

The coupled atmosphere-fire model has two components: a numerical weather prediction model and a fire behavior model that simulate the spread of a wildfire based on weather, fuel conditions, and topography [14, 15]. Heat and water vapor from wildfires affect the surrounding atmosphere, giving rise to fire winds. Atmospheric winds in turn drive the spread of the wildfire. Hence, a two-way coupled atmosphere-fire model is used to simulate this complex interaction between a wildfire and its surrounding atmosphere. Evolving atmospheric information predicted by the weather model drives the propagation of the simulated fire line. At the same time, heat and moisture information generated by the fire model is fed to the modeled atmosphere, thus affecting the atmospheric motions.

The weather prediction model is an extension of the atmospheric fluid dynamics model of Clark [14]. The model solves prognostic equations to predict the values of variables pertaining to momentum, thermodynamic energy, water vapor and precipitation *by advancing these fields in time at grid points in nested spatial domains*. The nested domains can be refined in both horizontal and vertical directions. The outermost domains deal with large scale features such as weather fronts and mesoscale convective systems. Inner domains concentrate on fine scale atmospheric features such as vortices and clouds within a fire line. The innermost domain deals with fluid dynamics and interacts directly with the fire model.

In each time step wind velocities from the fluid dynamics model are passed into the fire model. The fire model determines the spread of the fire and advances the fire line. Heat and moisture generated, because of the combustion, is fed into the fluid dynamics model as heat and moisture fluxes.

2.1.2 Tracking fire line propagation

For the purpose of tracking the fire line, the land surface within each atmospheric model cell is subdivided into grids called fuel cells. The fuel cells can be smaller than the atmospheric model cells by any integer ratio [14]. The characteristics of the fuel correspond to 13 different standard fuel types [16]. Four tracers are assigned to each of the fuel cells to keep track of the fire line. The tracers define the vertices of the polygon representing the burning region within each fuel cell.

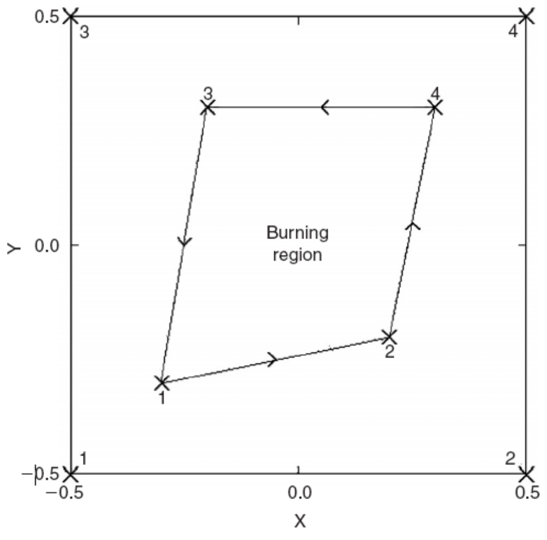


Figure 2.2. Basic fuel cell [14]

Tracers are capable of moving within the boundaries of the fuel cell. This allows for the smooth progression of the fire lines through the fuel cells without significant distortion effects because of the finite size and shape of the grid. The tracers must remain within the boundaries of its cell. However, temporary movement of a tracer outside its cell can occur. For example, it may cross the x = -0.5 boundary, while its y coordinate remains within the range -0.5 and 0.5. When this happens, the x coordinate of tracer 1 is fixed to corner 1 and remaining motion may only occur in the y-direction. Also, the neighboring cell may catch fire if it isn't already burning. The code is so formulated that the tracers move towards the grid corner with their corresponding number. The arrows joining the tracers give a directional sense to the fire line. As we move in the direction of the arrowhead the fire line will lie on our left. Figure 2.2 depicts the special case of a sub-grid fire. In such cases the tracer positions correspond to the coordinate points of the fire line. Most forest fires span across multiple fuel cells and the condition depicted in Figure 2.2 does not normally occur.

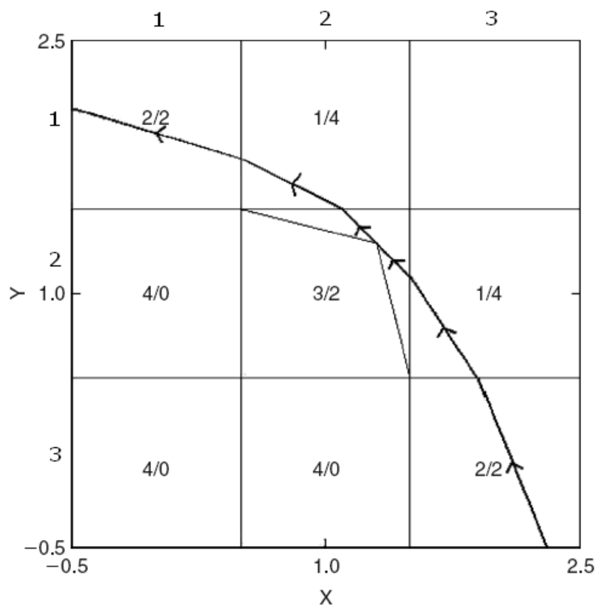


Figure 2.3. Fire line spanning multiple fuel cells [14]. Virtual coordinates

are used to define the fire line in the cell (2,2).

Figure 2.3 depicts a fire line which spans multiple cells. As in Figure 2.2, the arrows give a sense of direction to the fire line with the fires lying to the left of the arrows.

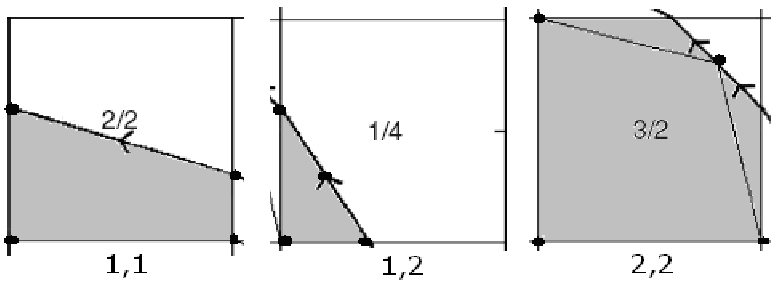


Figure 2.4. Cells (1,1), (1,2) and (2,2) from the grid in Figure 2.3

with the burning areas within each cell shaded in grey.

Figure 2.4 shows how the burning region within each cell is represented using tracers in situations where the fire line spans multiple cells. Notice that in cells (1,1) and (1,2) four tracers are sufficient to represent the area under fire. However, in cell (2,2) tracers from neighboring cells are needed to adequately represent the burning area.

Each of the fuel cells is assigned a class that is defined by two numbers (NCT/ICLS). NCT represents the number of tracers that are at their limiting corner positions. ICLS refers to the number of free coordinates. For example, in cell (1,1) tracers 1 and 2 are at their limiting corner positions whereas tracers 3 and 4 are free to move in the y-direction. Thus 2 coordinates (y coordinates of tracers 3 and 4) are free and two of the tracers are bound to their corners. Hence, NCT /ICLS for this cell is 2/2. Similarly in cell (2,2) three of the tracers (tracers 1, 2 and 3) are in their corner positions whereas tracer 4 is free to move in both the x and y directions. Hence the class of this cell is 3/2. The fuel cell depicted in Figure 2.2 is of class 0/8. This class has not been taken care of in the present code [14]. The active classes are 4/0, 3/1, 3/2, 2/2, 2/3, and 1/4.

Since virtual coordinates are used to define the burning region within each cell, two different formulae are used to determine the area. In the simplest case the burning region is a quadrilateral whose area Aq can be determined as follows:

$$A_q = \frac{1}{2}((x_4 - x_1)(y_3 - y_2) + (y_4 - y_1)(x_2 - x_3))$$

where x_1, .., x_4 and y_1, .., y_4 are the coordinates of tracers 1, 2, 3 and 4 respectively. For the calculating the area of the burning region in cell (2,2) we need take into account the area of the two triangles formed by the tracers of the cell and those of the neighboring cells along with the area of the quadrilateral formed by the tracers of the cell.

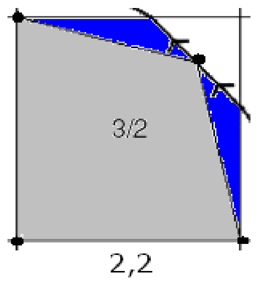


Figure 2.5. Figure depicting the 2 triangles (blue) and the quadrilateral (grey) that define the burning region within the cell.

The area of the burning region A_f is calculated as follows:

$$A_f = A_d + \frac{1}{2}\left((x_3 - x_4)(y_3 - y^O) + (y_3 - y_4)(x_4 - x^O)\right)$$

$$+\frac{1}{2}((x_4 - x_2)(y_4 - y^{OO}) + (y_4 - y_2)(x_4 - x^{OO}))$$

where the characters with superscripts represent the coordinates of the tracers from the neighboring cells. The area under combustion within each fuel cell is used to estimate the mass of fuel consumed in the last time step. Using the combustion coefficient data and the mass of fuel burned, the heat generated in the last time step is estimated. This in turn is converted into heat fluxes and entered into the atmospheric fluid dynamic model.

In order to define the fire line it is necessary to first identify the fuel cells that are a part of the fire line. A fuel cell is considered to be a part of the fire line if the area A_q under fire is less than one. In order to take into account cases where the fire line occurs along the grid boundary, cells with $A_q = 1$ are considered to be a part of the fire line if at least one of its neighbors have not been ignited. Fuel cells belonging to the fire line are marked with the flag NFL = 1. The remaining cells have the NFL flag set to 0.

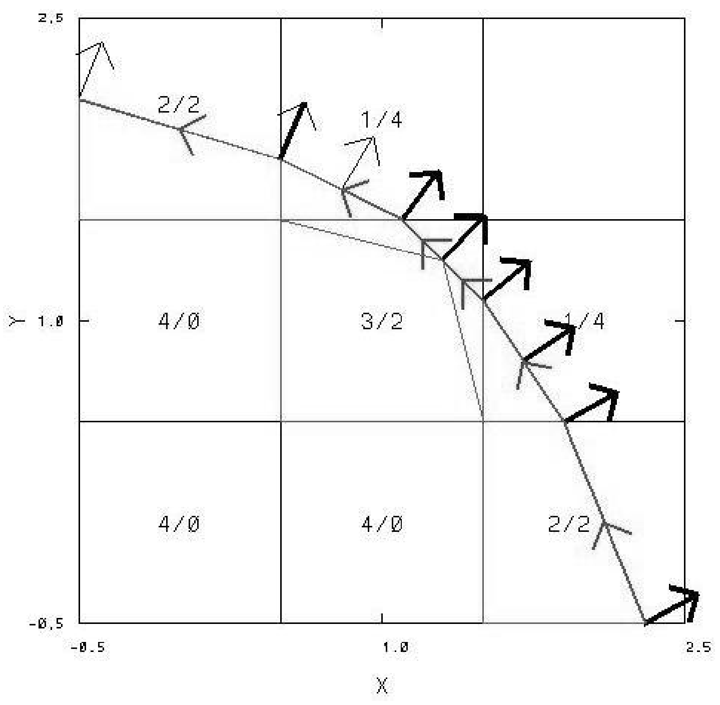


Figure 2.6. Figure showing the normal vectors directed away from the fire line

The next step is to determine the set of vectors directed away from the fire line (as shown in Figure 2.6). Normal vectors are used in calculating the spread of the fire.

Normal vectors are determined by fitting a circular arc to three points. The two end points of the fire line are used to determine the arc. The number of fire line points in a cell depends on its class. For example, classes 1/4 and 3/1 have three points each, whereas class 4/0 has two points.

2.1.2.1 Determining the rate of spread of the fire

The rate of spread of the fire depends upon wind speed, humidity, slope of the terrain, fuel type, and fuel moisture. This rate is determined at each coordinate point of the fire line. The Rothermel algorithm is used to calculate the rate of spread of the fire. The algorithm takes the form [26],

$$S_f = (1 + \phi_W + \phi_S)R_0,$$

where ϕ_W is the spread rate in zero wind and zero slope conditions, ϕ_W is the adjustment made for wind, and ϕ_S is the adjustment made for slope.

2.1.2.2 Identifying ignition of new fuel cells

This step involves cycling through all the fuel cells with NFL = 1 and checking to see if any of their tracers have moved into a neighboring cell. Once such occurrences have been identified, the algorithm checks to see if any of the new cells were already ignited, in which case it has to further investigate whether the situation represents a convergence of fire lines. For example, if the ignition occurs across the y= -0.5 boundary (west boundary) of a cell with NFL=1 and if corners 1 and 3 are not occupied by their respective tracers, then this is treated as convergence of a fire line from the west since the eastern part of the cell was already burning. After all the cases of fire line convergence have been treated, the algorithm looks at the cells which have been freshly ignited. In the case of freshly ignited cells, the code has to determine which fire line segment has to be used to ignite the cell. The fire line segment through a newly ignited cell can be ill defined if it represents the end of a sharply curving region of the fire line. In order to prevent premature ignition of fuel cells, the code only accepts the ignition of a fresh fuel cell if the extrapolated fire line covers only one corner of the cell. The code also accepts the ignition of a cell where the fire line covers more than one corner and the burning area of the igniter cell is greater than 95%.

2.2 Data Assimilation

2.2.1 An overview of data assimilation

Data assimilation may be defined as an *analysis technique in which observed information is accumulated into the model state by taking advantage of consistency constraints with laws of time evolution and physical properties* [19]. Data assimilation is an important technique to estimate the true initial state of the system especially in systems where measurements of the various model parameters are irregularly distributed over space and time [20]. The DDDAS Wildfire application is designed to work all the time and incorporate new measurement data as soon as it arrives. This, coupled with the complex and nonlinear nature of the problem, makes it particularly suitable for Bayesian filtering [18].

The state of the system is represented by set of physical variables and parameters mostly at mesh points [18]. To incorporate out of sequence data, a snapshot of the system state at various time intervals is saved in a time state vector x. Knowledge of the system's time state is represented by a probability density function p(x) which is represented in the model with an ensemble of time state vectors $x_1,\ldots.,x_n$ [18]. The number of system states maintained by the model will thus be equal to the number snapshots saved for incorporating out of order data times the ensemble size. Each of these system states are advanced over time through separate simulations. Sequential filtering involves successively updating the model state using sensor data via Bayes theorem.

The probability density $p^f(x)$ representing the current state of the model is called a *prior* or *forecast*. The data supplied to the model includes measurements y along with the information regarding the distribution of measurement errors and how the measured quantities are derived from the system state x. This information is represented by the vector y and the conditional probability density function p(y|x). The updated probability density $p^a(x)$, known as the *posterior* or *analysis*, is derived using Bayes theorem as follows:

$$p^a(x) = \left(p(y|x)p^f(x)\right)/\left(\int p(y|\xi)p^f(\xi)d\xi\right)$$

This then becomes the new state of the model. Advancing the model in time and injecting new data are decoupled operations.

2.3 Data Sources

Data measuring temperature, radiation, and local weather conditions come from fixed sensors. These sensors are designed to survive a burn over by a low intensity fire [13]. The sensor measurements are supplemented by data from fixed weather stations. The raw data is transmitted in comma separated ASCII format. Data in the form of images taken from satellite and aircraft based platforms are also collected. Images are processed using a variety of image processing algorithms that determine which of the pixels form a part of the fire and the energy radiated by it. The original pixel values, the computed probability of ignition of a pixel, and the geographic coordinate information is stored in the GeoTIFF image format or the HDF data format [13].

Information about previous fires are stored in a data center in a variety of formats (GeoTIFF, Ms Excel, text files or CSV). All the information coming in from sensors is timestamped to identify the time when the data was collected. If the incoming information does not have a timestamp, it is given one when it is received at the data center.

Information received at the data center may have to go through as many as 6 stages of processing. These are [13]:

(a) Retrieval – This entails getting the information directly from a sensor or via an intermediate computer system or storage device (eg an external hard disk drive).

(b) Extraction – This step involves extracting the relevant information from the raw data obtained from the sensors.

(c) Conversion – In the event that the received information is in a unit that is unsuited for the wildfire application; it is converted into a suitable unit.

(d) Quality Control – This step involves removing or repairing corrupted data.

(e) Store – Data must be archived to the right medium, for example, a tape drive or a disc drive. The archival process also takes into account the expected period of time for which the information will be archived (long term or short term storage).

(f) Notification – If a simulation is using data which is coming into the storage center, it must be notified whenever new data becomes available.

The sensor network operates either in active mode, in which case it actively streams data to registered end users, or in passive mode, wherein it only transmits data upon request.

Chapter Three: Observation Function

This section presents new research to develop an observation function which is independent of a specific wildfire code.

3.1 Overview of the observation function scheme

The observation function subroutine is passed a vector containing the model state and an integer, known as a tag, as input parameters. Depending on the value of the tag, it calls the appropriate weather observation function or the image observation function [24]. The subroutine definition for an observation function is as follows:

```
obs_function (syn_data,num_obs,nstate,tag)
```

Syn_data, which stands for synthetic data, is an output parameter containing the values computed by the observation function. Num_obs is also an output parameter indicating the length of the syn_data vector. The parameter state is a vector of length nstate containing the model state. Tag is an integer array of size tag_length (defined in the header file obs_params.h). The tag parameter contains a set of integers called tags which would be used to select the appropriate observation functions.

At this point it is important to note that the observation function cannot access the model state directly as it is not part of the model executable [24]. The model state is read from a checkpoint file as an input parameter. We restrict ourselves to the observation function dealing with weather station data, which is defined as follows:

```
weather_obs (ws_temp, ws_pressure, ws_windx, ws_windy,
```

ws_vapor, state, nstate, ws_lat, ws_lon, ws_alt, ws_time)

The parameters ws_temp, ws_pressure, ws_windx, ws_windy and ws_vapor contain the return values of the subroutine. The ws_temp parameter stands for weather station temperature and it is used to return the synthetic potential temperature determined by the subroutine. Similarly, parameters ws_pressure and ws_vapor are used to return the synthetic values for atmospheric pressure and the vapor mixing ratio. Parameters ws_windx and ws_windy correspond to the west-east and north-south velocity components [14]. The rest of the parameters provide the subroutine with a weather station's latitude, longitude, altitude, and the timestamp of the observation. The chief function of the weather_obs subroutine is to determine the offsets into the state vector where the pressure, temperature, vapor mixing ratio, and the u and v components of the wind velocity are located. The subroutine then calls another routine, weather_obs_w, which determines the actual synthetic data.

Most of the functionality for determining the synthetic data is present in the weather_obs_w subroutine. It processes the geographic coordinates of the sensor/weather station to determine its location with the model grid. Once the cell containing the sensor is determined, the synthetic values for the temperature, pressure, wind velocities, and other weather station data is determined by linearly interpolating the station location with respect to the grid points. The process for determining the position of the station within the model and calculating the values of the model variables at the station's location is detailed in the following sections.

3.2 Extracting the model variables

Before we discuss the procedure for extracting specific model variables we need to understand how variables are arranged in the model memory. The model memory is organized into three levels [14], which is explained in Section 3.2.1. Section 3.2.2 will explain the typical set of steps followed for extracting any variable from the model memory. Finally, in Section 3.3.3 we will see how the model variables pertaining to the weather station observation function are extracted.

3.2.1 Levels of data storage

As mentioned before, the model uses three levels of storage. These are as follows.

3.2.1.1 First level

The first storage level consists of variables which can be accessed directly by the model code. These are stored on the computer's main memory [25]. It contains only a fraction of the total model variables [14].

3.2.1.2 Second level

The second level is used to store data which may be word packed. Word packing compresses the data and frees up memory, allowing simulations of larger domains [14]. It also helps to reduce the number of disc accesses required to move data back and forth between level two and level three of the data storage. The second level is also stored in the main memory.

3.2.1.3 Third level

The third level refers to the model data stored on discs and tape drives. This level is only used for the storage of data and is not utilized during execution of the model code [25].

3.2.2 Procedure for extracting model variables

In this section we describe a particular code implementation for extracting data. Similar mechanisms can be implemented in other wildfire codes. We are primarily interested in extracting model variables during the execution of the model code. Therefore, our focus will be on the data storage levels one and two. Storage level two can be thought of as a table [25]. There are two forms of this table: (a) a table for the analysis cycle and (b) a table for the generator cycle. Since our focus is on the generator cycle, we have only included the table for the generator cycle in Appendix A. Each table has 16 file groups and each of these file groups contains three fields. The 16 file groups can be thought of as 16 three dimensional arrays. The face of the structure is partitioned into 3 NX by NZ slabs, corresponding to the 3 fields in each file group [25]. The depth is partitioned into NY/2 double slabs [25]. NX, NY and NZ represent the number of grid points in the x, y and z directions and is defined in the model code [14].

Variables stored in level two are not directly accessible by the model code. Before a file group can be read it must be prepared by calling the RDRSET subroutine. It accepts only one parameter, an integer containing the file group number (1 – 16). RDRSET brings the file group into a buffer from where it can be accessed by the model code [25]. Any previous version of a file group in the buffer will be over written if RDRSET is called again for the same file group name.

Data is read from the buffer using the RDDR3 subroutine. The use of RDDR3 and RDRSET will be made clear by the following code example, which describes the typical procedure for reading and extracting model variables from level two storage. Before looking at the code, we need to go over the following variable definitions [14]:

- NX, NY and NZ – The number of grid points in the x, y and z directions.
- N2 is defined as 2 * NX * NZ in the code
- N4 = 2 * N2 and N6 = 3 * N2
- NP1, …, NP15 – are the indices of the starting location minus one. NP1=0, NP2 = NY/2, NP3 = 2 NY/2, ……, NP15 = 14 NY/2
- JD is the slab index
- I, J and K are do-loop indices in the x, y and z directions respectively
- MX, MY and MZ are the maximum x, y and z direction dimensions in the model.

The following piece of FORTRAN 77 code demonstrates how data can be retrieved from file group 6.

```
1.  REAL A(MZ,MX,2)
2.  CALL RDRSET(6)
3.  DO 100 JJ=1,NYM,2
4.      JD=(JJ+1)/2
5.      CALL RDD3(A,N2,NP6+JD,1)
6.  100 CONTINUE
```

In line 1 we are declaring an array A of type real. MX and MZ represent the maximum x and z direction dimension in the calculation respectively [14]. Line 2 is a call to RDRSET which prepares file group 6 for reading. Lines 3 to 6 loops through the 3-D array and stores the extracted data in array A. As mentioned earlier N2 is defined as 2 * NX * NZ in the code. RDD3 (A, N2, NP6+JD, 1) reads N2 words from starting index NP6 of field 1. This code can be altered to extract the model variables required by the weather station observation function by substituting the starting index of the variable of interest in place of NP6 in the call to the function RDD3 and putting the variable's file group number in the call to RDRSET. The file group number and the starting index for a variable in the generator cycle can be determined from the table given in appendix A.

Currently only values that directly affect the weather station observation function are extracted. These include (a) horizontal wind velocity, (b) pressure, (c) theta (theta is the non-dimensional normalized potential temperature. This field or BUOYANCY field may be used to calculate temperature), (d) buoyancy, and (e) QV (QV is the water vapor mixing ratio. It is used to determine relative humidity).

3.3 Determining the sensor position

Here we discuss the process for converting the latitude and longitude of the sensor into the coordinate system used by the model. As mentioned earlier, the atmosphere-fire model has several domains 1, 2, 3, . . ., N (1 being the innermost domain). The first step is to find the distance (in kilometers) of the sensor from the center of domain 1. Let the station distances be represented by DXST and DYST. They are determined as follows:

$$DXST = \left(Longitude_{Station} - Longitude_{Center\ of\ Domain\ 1}\right) \times 111.319 \times \cos\left(Latitude_{Center\ of\ domain\ 1} \times \frac{\pi}{180}\right)$$

$$DYST = (Latitude_{Station} - Latidtude_{Center\ of\ Domain\ 1}) \times 110.942$$

The code for of the above equations is as follows:

DXST = (STLON (IST) – TLONUD) * 111.319 * COS (TLATUD* π /180)

DYST = (STLAT (IST) – TLATUD) * 110.942, where STLON is an array containing the longitudes of all the sensors and STLAT is the array containing the latitudes of all the sensors. IST is a loop counter and its values range from 1 to ISTA. ISTA is a constant denoting the maximum number of sensors. TLONUD and TLATUD denote the longitude and latitude of the center of domain 1. The sensors are sequentially numbered from 1 to ISTA. Given that the equatorial radius is 6378.137 kilometers, the distance between each degree along the equator is 111.319 kilometers. The polar radius is 6356.75 kilometers. Thus the distance between each degree along the north-south meridian is 110.942 kilometers. If we consider the north-south direction to be the y-direction, the distance along y between the model center and the sensor becomes $(Latitude_{Station} - Latidtude_{Center\ of\ Domain\ 1}) \times 110.942.$ While calculating DXST, we have to multiply 111.319 (the length of a degree along the equator) by a cosine factor. This is because as we move away from the equator the lines of latitude get smaller in length. To account for this we multiply the distance along the equator by the cosine of the angle of latitude at the model center. $Latitude_{Center\ of\ domain\ 1}$ times $\frac{\pi}{180}$ gives us the latitude in radians.

The next step is to calculate the distance of the center of domain 1 from the south-west corner of domain 1 (we call this the reference point), which is assumed to be the point of reference. This is carried out as follows:

XCNTR = (XOMDL (1) + (NXSET (1) – 2) * DXSET (1)/2)*SLNGTH*1.E-5

YCNTR = (YOMDL (1) + (NYSET (1) – 2) * DYSET (1)/2)*SLNGTH*1.E-5

Where XCNTR and YCNTR are the distances (in kilometers) of the center of domain 1 from the reference point in the x direction and the y direction respectively. XOMDL (k) is the displacement of domain k from the reference point in the x-direction. XOMDL for domain 1 is 0. NYSET (1) gives us the number of points in the x-direction in domain 1. We subtract 2 from NYSET (1) in order to account for the shadow domain. DXSET (k) contains the length of a grid side in domain k. Therefore, the expression (NXSET (1) – 2) * DXSET (1)/2 determines the length of one half of domain 1. Because values are stored in the model without a dimension, the expression XOMDL (1) + (NXSET (1) -2) * DXSET (1)/2 is multiplied by SLENGTH, which converts the value of the expression into centimeters. To convert this to kilometers we further multiply by10^{-5}.

Thus the distance of the sensor from the reference point, in the x-direction, can be obtained from the sum of XCNTR and DXST. This value is stored in the variable STX (STY for the distance along the y-direction). The only thing left to do now is to identify the atmospheric grid cell enclosing the sensor. This can be found by dividing STX (the distance of the sensor from the reference point in the x-direction) by the length of the grid (DXSET). The code for this is:

```
I_ISTX (IST, MODEL) = STX (IST)/ (DXSET (MODEL)*SLENGTH*1.E-5)

J_ISTX (IST, MODEL) = STY (IST)/ (DYSET (MODEL)*SLENGTH*1.E-5)
```

In the code both I_ISTX and J_ISTX are of type real so that we can use the fractional remainder to interpolate with neighboring points containing the model variables. This is further discussed in Section 3.4. To identify the grid cell containing the station we can assign I_ISTX and J_ISTX to integer variables so that the fractional part is truncated.

3.4 Determining the model variables at the sensor position

In Section 3.3 we discussed how to identify the atmospheric grid cell containing a sensor given its latitude and longitude. The position of a sensor within the model is given by a set of three real numbers representing the sensor’s distance (in km) from lower left corner of domain 1. Values of the various model variables such as the potential temperature, vapor mixing ratio, and u component of the wind velocity are available at specific points in an atmospheric grid cell. Scalar variables such as pressure and temperature are located at the center of the cell, whereas vectors such as the u and v components of the wind velocity are respectively centered on the right hand side and the top side of a cell. Since the position of a sensor can be randomly distribute in the cell and not necessarily at any of these points, we need to derive the value of a model variable at the sensor location from the nearby points. This is done by interpolating at the points where the model variables occur using the location and data of the sensor.

3.4.1 Determining values of vector model variables

Here we examine the procedure for determining the u and v components of the wind velocity at the sensor location. We will refer to the u and v components of the velocity at any point by U and V respectively. Let X_S and Y_S be the x and y coordinates of sensor S in the model. And let X and Y be the x and y coordinates of the lower left corner of the grid cell (EFGH) enclosing S.

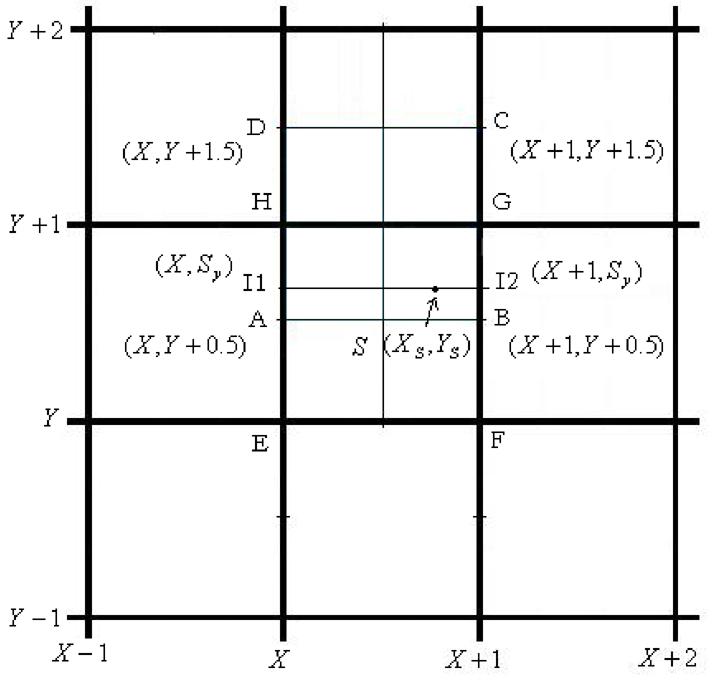


Figure 3.1. Figure showing the sensor S in the model grid

U_A and U_B represents the u component of the velocity in grid $(X-1, Y)$ and (X, Y) respectively. Recall that the u – component of the velocity is located on the right hand side of each grid box centered on the vertical. Therefore U_A and U_B are located at the coordinates $(X, Y+0.5)$ and $(X+1, Y+0.5)$ respectively (at positions A and B in Figure 3.1). Similarly the velocities at C and D are represented by U_C and U_D respectively.

One way to determine U at S is to do a linear interpolation including the points $(X, Y+0.5)$, $(X_S, Y+0.5)$ and $(X+1, Y+0.5)$ and assume that the value of U at $(X_S, Y+0.5)$ is the value of U at (X_S, Y_S). While this method is simple, it is inaccurate because we are using the value of U at $(X+0.5, Y+0.5)$ to approximate the value of U at S. In order to obtain a more accurate approximation of U at S, we have used an interpolation technique known as bilinear interpolation [27].

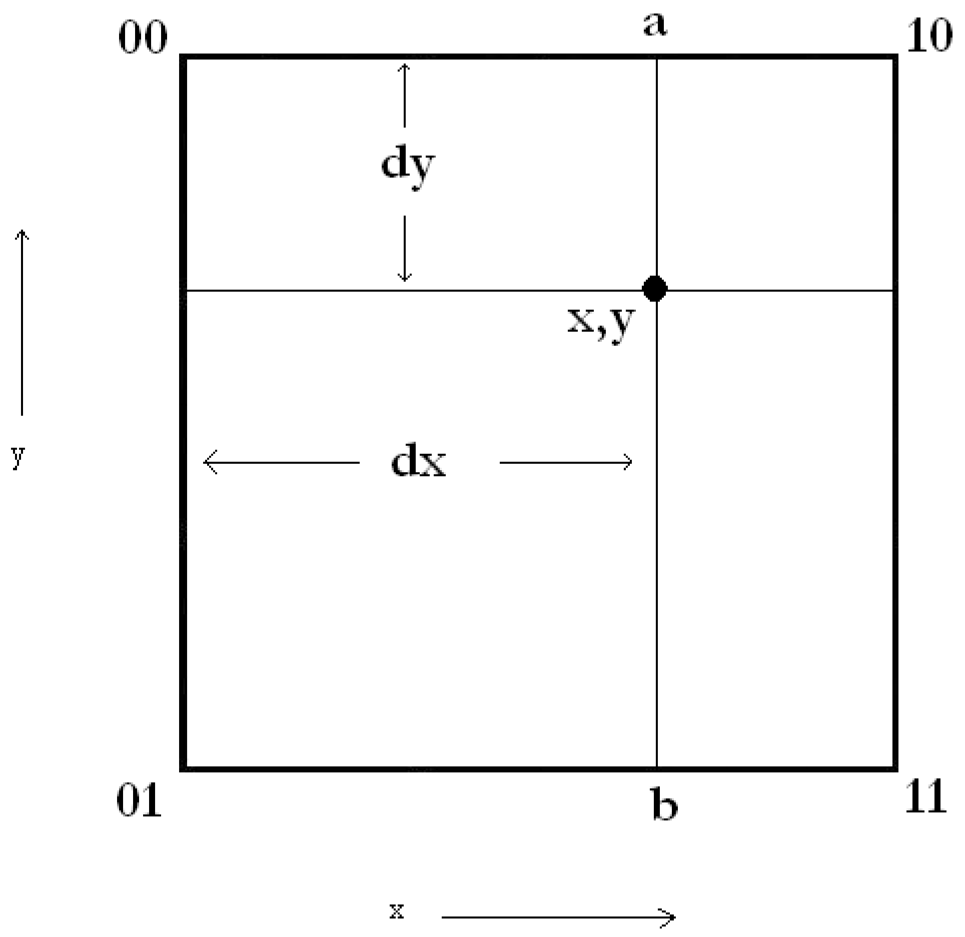


Figure 3.2. Layout of the bilinear interpolation scheme

Bilinear interpolation works as follows. With reference to Figure 3.2 assume that we have been tasked to find the value of some real quantity Q at the position x, y. We denote this value as $Q_{x,y}$. Similarly we denote the value of Qat the grid points by $Q_{i,j}$ where $i, j = 0{,}1$. The values of $Q_{i,j}$ will be available to us from the variables extracted from the model state. We can estimate $Q_{x,y}$ from the values of $Q_{i,j}$ as follows. First, we will find the value of Qat point a (we will call this Q_a).

$$Q_a = \frac{(Q_{10} - Q_{00})}{(10 - 00)} \times Q_{00} \qquad (1)$$

Since the sides of the grid cell are of length one $10 - 00 =$ (1) can be re written as

$$Q_a = (Q_{10} - Q_{00}) \times dx + Q_{00} \qquad (2)$$

Similarly we can calculate the value of Q_b as follows:

$$Q_b = (Q_{11} - Q_{01}) \times dx + Q_{01} \qquad (3)$$

Once we have determined the values of Q_a and Q_b we can determine $Q_{x,y}$ by

$$Q_{x,y} = (Q_b - Q_a) \times dy + Q_a \qquad (4)$$

In order to apply bilinear interpolation to the problem at hand we need to first identify the quadrilateral enclosing the sensor's location (to avoid confusion please note that when a reference is made to the quadrilateral enclosing S it refers to the square formed by the four neighboring points containing the model variables which surrounds S) . To do this we need to first identify the points neighboring S where U is defined in the model. In Figure 3.1 we see that these points are A, B, C, and D. The procedure for determining the coordinates of these points is as follows: first, we determine the x and y coordinates of the lower left corner of the grid box enclosing S (point E in Figure 3.1). This is accomplished by assigning X_S and Y_S to two integer variables say I_X and I_Y. This causes the decimal portion of (X_S, Y_S) to be truncated giving (X, Y), the coordinates of E. Next we compare Y_S to $Y + 0.5$ (assuming that the length of each side of a grid is one). If $Y_S > Y + 0.5$, then the coordinates of A are $(X, Y + 0.5)$. Coordinates of B are determined by adding one to the x coordinate of A. The coordinates of C and D are similarly determined by adding one to their coordinates. If $Y_S < Y + 0.5$, then A is $(X, Y - 0.5)$. B, C, and D are derived from the coordinates of A as explained above. Once we have determined the coordinates of quadrilateral enclosing S, we use equations (2) – (4) to determine the value of the model variable at S.

3.4.2 Determining values of scalar model variables

The procedure for approximating the values of scalar variables at S is almost identical to that of the vector variables. The main difference in the two cases is in the location of the model variables within the grid. Unlike vectors, the scalars occur at the center of each grid box. This leads to a slight difference in the procedure for determining the coordinates of the quadrilateral enclosing the sensor.

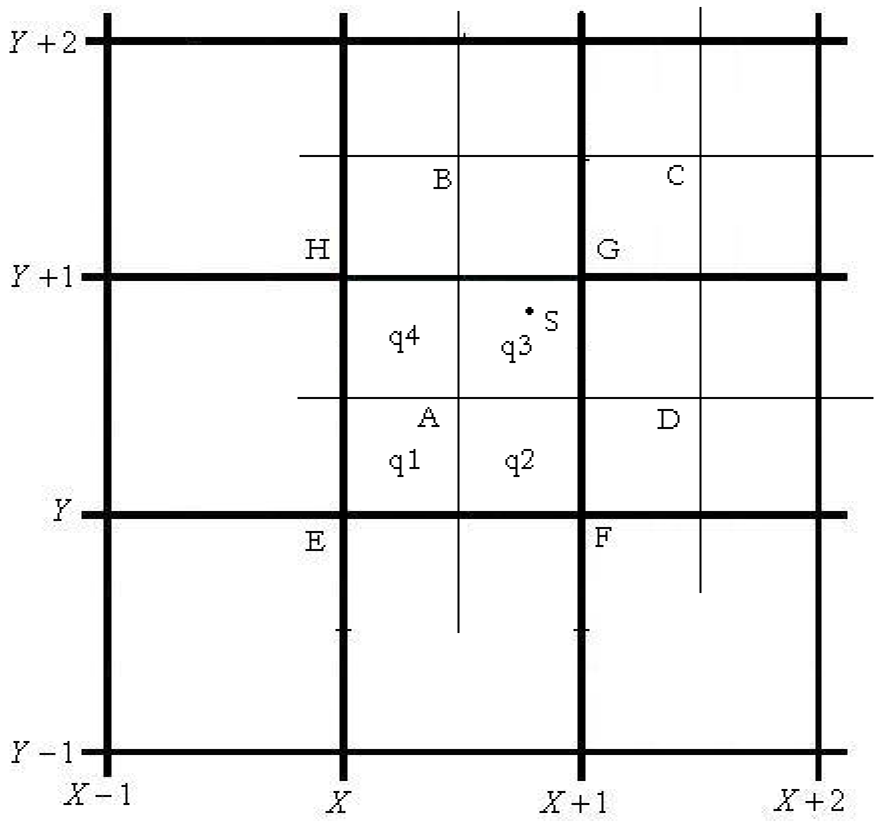


Figure 3.3. Determining the values of scalar model variables

In Figure 3.3, q1, q2, q3, and q4 represents the four quadrants of the grid box EFGH. Since scalars are located at the centers of the grid boxes we need to find the quadrant where sensor S is located in order to determine the four neighboring points containing the model variables. This is done by comparing both (X_S, Y_S) to the center coordinates of EFGH $(X+0.5, Y+0.5)$. Recall that (X_S, Y_S) is the coordinate of the sensor S. In Figure 3.3 we see that S lies in quadrant q3 therefore the four neighboring points containing the model variables must be the points denoted by A, B, C, and D. Once the quadrant is determined we can find the lower left corner of the quadrilateral enclosing S using the following logic: If the quadrant is q1, then the lower left corner is given by $(X-0.5, Y-0.5)$. In the case of q2, q3, and q4 it is given by $(X+0.5, Y-0.5)$,

$(X + 0.5, Y + 0.5)$, and $(X - 0.5, Y + 0.5)$ respectively. The other three corners of the enclosing quadrilateral are determined by adding the grid length to the x and y coordinates of the lower left corner, as explained in Section 3.4.1. Once coordinates of the quadrilateral enclosing the station is determined we can follow the procedure laid down in Section 3.4.1 to approximate the value of the model variable at sensor S.

Chapter Four: Visualization

4.1 Overview

Effective presentation of the data for consumption by the end users is an important aspect of any software system. Predictions made by the DDDAS wildfire program will ultimately be used by firemen to effectively combat forest fires. Due to the hazardous nature of fighting forest fires it is important to get the information to the users in an easy to use and timely manner.

Developing an intuitive graphical user interface for the visualization tool was an important design goal of our project. The interface should overlay the fire location information on the appropriate map and include the geographical coordinates. There should be a simple mechanism for zooming in and out of regions and scrolling along the map.

The visualization tool needs to have the ability to accept information from different sources. Information about fire location can come from various types of sensors or from fire fighters. Also there is the need to accept the forecast generated by the wildfire code.

Ease of installation is an important objective since the primary users of this program probably will not be computer experts. The goal is to provide an installation package that allows the user to install the program with minimal effort and technical knowledge.

An important requirement for the visualization tool is cross platform compatibility and portability. We envisaged a system which would work seamlessly on many platforms include mobile ones.

Another requirement of our project was to keep the overall cost of the system low. With this in mind, we decided to restrict ourselves to software tools that are open-source or are available free of charge.

Keeping the above constraints in mind we decided to display the wildfire information using the most basic form of Google Earth.

4.2 Google Earth

Google Earth is a desktop application that allows the user to seamlessly explore the Earth's surface. It presents a virtual globe by superimposing images from satellite imagery and aerial photography onto a 3-D sphere. Users can rotate the globe and zoom in or out of places of interest. Google Earth can be customized to display user defined objects (3-D models, images) on the virtual globe [1].

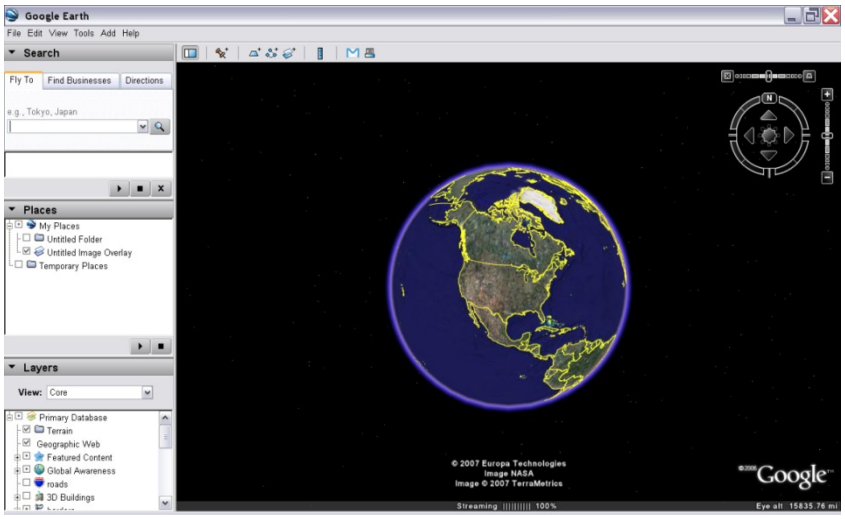


Figure 4.1. User interface for Google Earth

There are some distinct advantages in using Google Earth for displaying the wildfire information. Firstly, it has an intuitive GUI which allows the user to access information through a simple point and click mechanism. A map can be navigated by simply clicking and dragging it using a mouse or any other pointing device. It is also very easy to zoom in and out of locations.

Secondly, the Google Earth client software is available on many different platforms. Further, the basic version of Google Earth is available free of charge. The feature supported by this version meets all our requirements.

Third, the Google Earth client has the ability to check the server for updates and refresh the displayed information as often as every 4 seconds.

Lastly, Google Earth uses a XML style configuration file to store geospatial information. Information from any source can be displayed by simply entering the type of object (icons, images, or place-marks) to be displayed along with its geographic coordinates in this file. This allows us to easily incorporate data from many different sources.

4.3 Customizing Google Earth

Google Earth uses a configuration file with the extension .kml to describe the information displayed on the virtual globe. *KML (keyhole markup language) is a file format used to display geographic data in an Earth browser, such as Google Earth, Google Maps, and Google Maps for Mobile* [2]. It can be used to overlay images, display 3-D objects such as buildings, and control the rate at which the displayed information is updated. The structure of a kml file is based on the XML standard where information is presented in a tag based format with nested attributes and properties. For example, a

KML file entry for displaying a marker on a certain location on the virtual globe will look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">

 <Placemark>
  <name>Marker</name>
  <description>This is a marker.</description>
  <Point>
   <coordinates>20.0822,10,0</coordinates>
  </Point>
 </Placemark>

</kml>
```

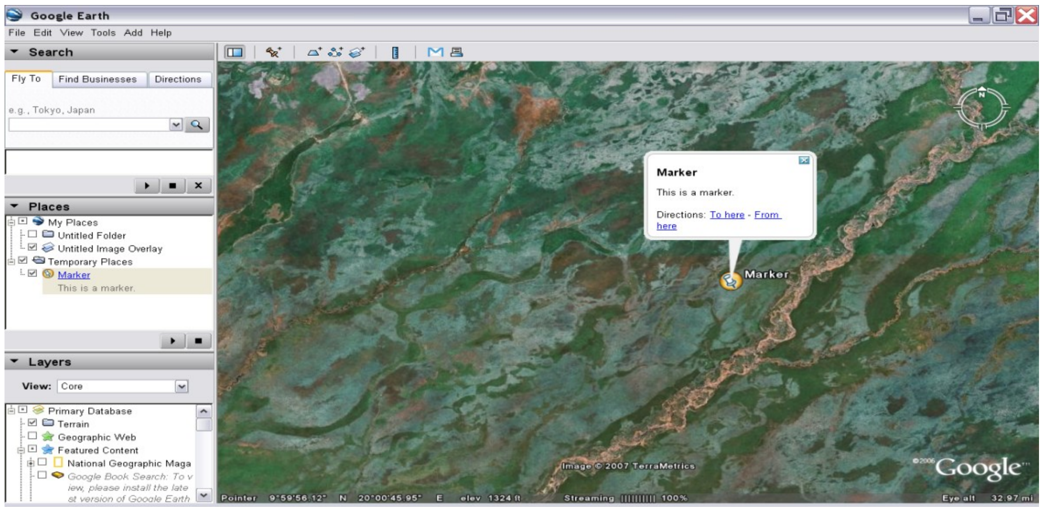


Figure 4.2. Screenshot showing the marker object as described by the sample KML file

See [7] for a complete reference to the various tags used in KML files.

4.4 Implementation overview

The current version of the visualization program is designed to display the locations of wildfires in North America. Details about this implementation are further discussed in Section 4.4.1 and Section 4.4.2.

4.4.1 Information gathering

All the information pertaining to the locations of wildfires is collated from the website of the National Oceanic and Atmospheric Administration or NOAA. NOAA's website provides a daily update of the locations of wildfires throughout North America. This data is stored in a file named modeislatest24hr.dbf and can be downloaded from the URL [8].

We have used the GNU Wget utility to download modeislatest24hr.dbf from NOAA's FTP site. GNU Wget is a free utility that permits non interactive downloads from the web. It supports HTTP, HTTPS, and FTP protocols [3]. We put a Wget command in a shell script and used a crontab file to run the script every 10 minutes. The Wget utility checks to see if there is any modification to the file and downloads it only if changes have been made. A line in a crontab file has the following syntax [4]:

```
Minute Hour Day-of-month Month Day-of-week command to be executed
```

Our crotab file has the entry:

```
0-50/10 * * * * /usr/local/GoogleEarth/downloadtrans >/dev/null 2>&1
```

The portion "0-50/10 * * * *" tells the Cron utility to execute the script every hour at minutes 0 to 50 skipping 10 values. Therefore the script will run every hour at 0, 10, 20, 30, 40 and 50 minutes [5]. Thus, the downloadtrans script will be executed every 10 minutes. Every time the Cron utility executes a job specified in the crontab file it sends an email to the user. To disable this feature the command ">/dev/null 2>&1" has been included.

The downloadtrans script file contains commands for the execution of the Wget utility and for running a Python script. It contains the following entries:

```
cd <path to the directory containing the code>

wget -N --limit-rate=20k <URL to the FTP site hosting the wildfire location data>

.<path to the directory containing python script>/Python run.py
```

For now we will limit our discussion to the entry dealing with execution of the Wget utility. The third line which executes a Python script is discussed in Section 4.3.2. Wget is an utility that permits non-interactive downloads from the web. It can be set up to run in the background and download files from the web while the user is not logged in. It supports HTTP, HTTPS, and FTP protocols [3]. The "-N" option turns on a feature known as timestamping. When timestamping is enabled, the Wget utility checks to see if the file being downloaded exists locally. If yes, it asks the server for the last modified date of its copy of the file. If this file is newer than the local version of the file then it is downloaded and the local file is overwritten with the new one [3]. The "--limit-rate" option is used to limit the download rate. Here the upper limit on the download rate has been set to 20K or 20000 bytes per second.

4.4.2 Data extraction and creation of KML files

In this section, we examine how data is extracted from the modislatest24hr.dbf file and how the KML data files are created. DBF stands for dBase File Structure. This format was originally developed to serve as the underlying file format for the dBase database management system [6]. DBF files are used by a number of applications which are collectively referred to as XBase. One such application is the Geographic Information System (GIS) [9]. The structure of the DBF file format is available at the URL [10]. We have used a Python script called dbfreader [11] to read and extract information from the modislatest24hr.dbf file. This script is available free of charge at the URL [11].

Recall, from Section 4.3.1, that the last line of the downloadtrans file contains an entry for the execution of a Python script called run.py. This script functions as the main subroutine of our program. It has two major functions:

1. Extract information from the modislatest24hr.dbf file.
2. Use the information extracted in 1 to generate the fires.kml file.

We will examine the working of the run.py subroutine with the aid of the pseudo code given below.

1. Open the modislatest24hr.dbf file and store a pointer the file in variable f.
2. Call subroutine dbfreader using variable f as the input parameter and store the return value in variable db. Variable db contains a set of records. The first and second records contain field names and field datatypes respectively. The records following record two contain the actual data.
3. Close modislatest24hr.dbf.
4. Store all the records, following record 2, in variable records.
5. Initialize an empty set of records called lon_lat.
6. Iterate through each record in variable records.
7. Since we are only interested in wildfire data pertaining to the North American continent, we only consider records for which the latitude lies between 20 and 70 degrees and the longitude lies between -135 and -30 degrees. If a record meets the above criteria it is append it to the set lon_lat. The system has been deliberately restricted to display only fire locations in North America because

inclusion of a larger geographical area (such as the entire planet) significantly slows down the system response time.

8. After the iterations are complete create a new file called fires.kml and store the file pointer in variable f.
9. Call subroutine writeKML using variables f and lon_lat as input parameters.
10. Close fires.kml.

Recall from Section 4.2 that the geographic information displayed on the Google Earth virtual globe is stored in the KML file format. Information stored in a KML file can be viewed on the virtual globe by simply opening the file using Google Earth. As mentioned earlier, we have employed Python scripts to generate a KML file (named fires.kml) using the wildfire location data extracted from modislatest24hr.dbf. All the user has to do in order to view the wildfire locations on the virtual globe is to open fires.kml in Google Earth.

However, this approach makes it difficult to continuously update the fire location information. In order to get the latest information, the user would have to periodically download new versions of the fires.kml file from the server and reopen it in Google Earth. To resolve this problem we decided to make two different KML files. One of them is the fires.kml file, described earlier, and the second one is called doc.kml. Unlike fires.kml which is updated periodically, the content of doc.kml does not change. The later essentially contains a link to the server location where the fires.kml file is stored. When doc.kml is opened in Google Earth, it downloads fires.kml from the link provided and displays its contents on the virtual globe. The doc.kml file includes a timer attribute which instructs Google Earth to download a new version of fires.kml from the server after a certain time interval and update the display. All of this is carried out automatically by Google Earth and relieves the user from having to manually check for new updates. A portion of doc.kml is given below.

```
<NetworkLink>

        <name>Fires</name>

        <open>1</open>

        <URL>

                <href><URL hosting the file fires.kml ></href>

                <refreshMode>onInterval</refreshMode>

                <refreshInterval>300</refreshInterval>

        </URL>

</NetworkLink>
```

The URL listed between the <href></href> tags points to the location of the fires.kml file.

The <refreshInterval> tag acts as a timer. It sets the time interval after which Google Earth has to download a new version of fires.kml. The tag <refreshMode> tells Google Earth on what event to refresh the screen. Here its value is set to onInterval, which means that the screen will be refreshed every time the time interval specified by the <refreshInterval> tag elapses. In other words the screen will be refreshed whenever Google Earth downloads a new copy of fires.kml.

Google Earth has a built in utility that allows the user to create a compressed file out of an ordinary KML file. This compressed file has the extension .kmz. A KMZ file may also be used to store any icons or images that maybe used by Google Earth. This has the advantage of allowing the KML file and all the images and icons used by it to be packaged in a single file. Once a KMZ file has been created, it can opened using Google Earth in the same manner as a KML file. There is no need to uncompress the KMZ file before using it. We have packed the doc.kml file and all the icons and images needed by our project in a file called fires.kmz. Users who wish to utilize our visualization tool need only to download this file.

4.4.3 Instructions

We will now go over the procedure to setup and use the wildfire visualization tool. The first step is to install the Google Earth client software. Google Earth can be downloaded free of charge from the URL [7]. A link to this site is provided on our application's website. Next, the user needs to download the fires.kmz file from the application's website. The user then has to open the fires.kmz file in Google Earth to start the visualization tool.

4.4.3.1 Virtual globe navigation

Next, we will familiarize ourselves with the Google Earth client and learn how to navigate the virtual globe.

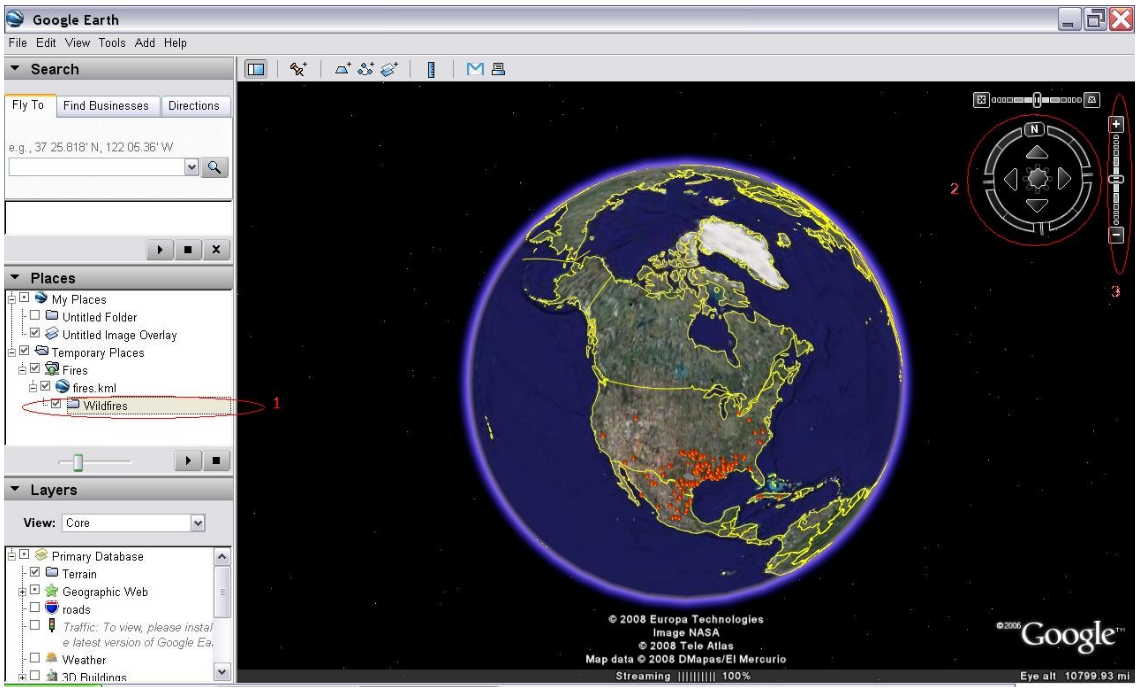
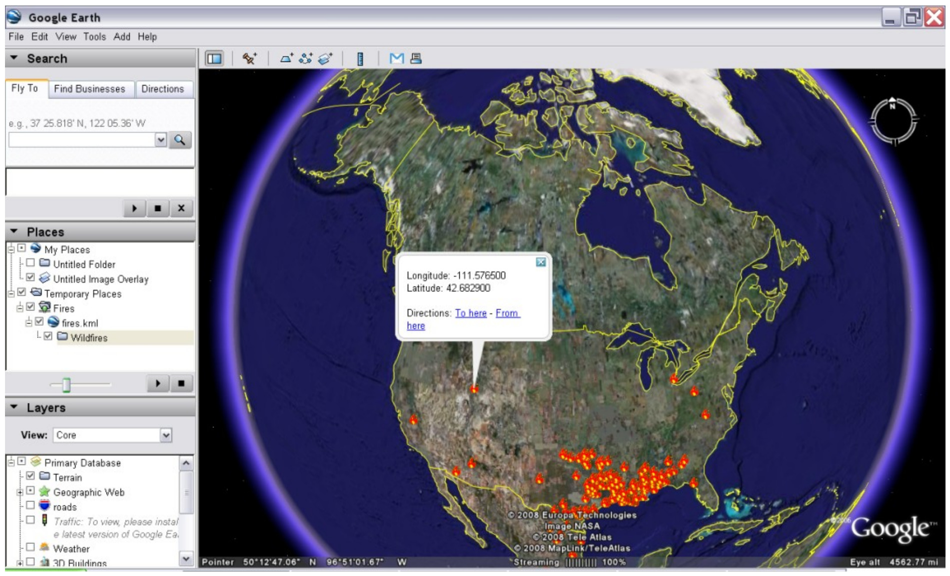


Figure 4.3. Navigation tools in Google Earth

Figure 4.2 shows controls for navigating the Google Earth virtual globe. To zoom in and out of the screen use the vertical slide bar indicated by control 3 in the figure. Users equipped with a mouse that has a scroll wheel can use it to control the zoom function. The globe can be rotated by left-clicking on the globe and dragging the pointer in the desired direction of rotation while keeping the left-click button pressed down. An alternate way to rotate the globe is to use control 2, shown in Figure 4.2.

4.4.3.2 Viewing wildland fire data

In order to bring up the wildland fire positions the user needs to double click on the node labeled wildfires in the navigation tree on the left hand side of the screen (designated as control 1 in Figure 4.2). This will bring up all the current wildfire locations in North America on to the screen. To view the details of a particular wildfire on the screen, click on the fire icon on the globe. This will bring up a box displaying the latitude and longitude of the fire. To zoom-in on the fire location, double click on the fire icon.



Longitude: -111.576500
Latitude: 42.682900

Directions: To here - From here

Figure 4.4. Figure showing a display box indicating the latitude and longitude of the wildfire

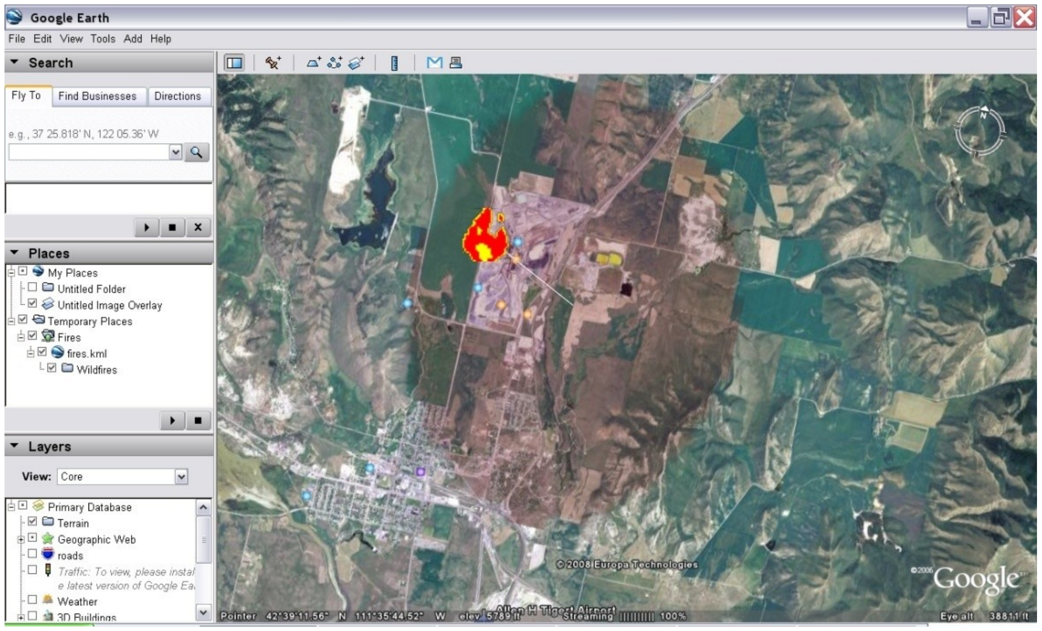


Figure 4.5. Figure showing the location of a wildfire after zooming in.

The area covered by the darkened circular region, immediately below the fire icon, roughly corresponds to 90 sq kilometers (about 22240 acres) and represents the area under fire. This is not an accurate representation of the burning region. Later versions of the visualization tool will be able to present a more accurate depiction of the region under combustion.

Chapter Five: Conclusion

This thesis has explained the software implementation of an observation function for the data assimilation module of the DDDAS wildfire simulation system. We have seen the important role played by data assimilation in determining the state of the model at the start of a forecast and in absorbing sensor data into the model. The main purpose of data assimilation is to provide the base condition that will produce the best possible model forecast. Data gathered from sensors contain inaccuracies and are often unevenly distributed in time and space. Also sensor data may arrive out of order. The DDDAS wildland fire system maintains snapshots of the system's state over time [18]. These are known as time-state vectors. A time-state vector contains a collection of physical variables and properties of interest. Time-state vectors permit the injection of out of order data into the system.

The observation function plays a critical role in data assimilation as it estimates the values of the model variables at the sensor locations. The data produced by the observation function from the model state is known as synthetic data. Synthetic data is compared with the sensor data by the data assimilation module and the result is used to adjust the initial state of the model prior to a forecast. The software implementation of the observation function is split into two executables. One part resides in the model executable and is called by the main function during the generator cycle. Currently this module extracts the model variables corresponding to the horizontal wind velocity, pressure, θ, buoyancy and the QV field. Temperature can be derived from either buoyancy or θ fields and the QV field is used to determine the relative humidity. The current implementation extracts these variables as they are required for the weather station observation function. These values are written into a checkpoint file.

The part of the observation function that resides in the second executable does the bulk of the processing. First, it maps the latitude and longitude of the weather stations to the model grid coordinates. Second, it reads the vectors containing the model variables from the checkpoint files and loads them into 3-D arrays. And third, it estimates values of the model variables at the weather station coordinates. A multipoint interpolation technique known as bilinear Equation is used to estimate the model variables at the weather station coordinates from the model variables at the neighboring grid points.

Our visualization tool makes use of Google Earth mapping tool to display the locations of wildfires in real time. The software presents a virtual globe, which the users can rotate and zoom in and out of. Google Earth allows users to overlay custom images and 3D models on the virtual globe. All the objects displayed on the virtual globe and their properties are recorded in a configuration file which has the extension .kml. The structure of a .kml is similar to XML and is used by applications such as Google Earth and Google Map to store geographic information.

The visualization tool uses the GNU Wget tool to gather information on wildfire locations from the Internet. The tool is executed automatically at set intervals using the Crond background daemon. Before fetching the new data, Wget checks to see if it differs from the data that was last fetched. If a change is detected, the new data is fetched and the fire location data recorded in the .kml file is updated.

Google Earth is made available free of charge by Google Inc. It is available on many platforms. The navigation tools in Google Earth are intuitive and user friendly. Currently the visualization tool is configured to refresh the information displayed in Google Earth once every ten minutes. However, the system can be configured to update the displayed information as often as once every minute. Thus the software meets the original design goals of developing a cost effective, portable and easy to use system for displaying wildfire location information on a virtual map in real time.

Even though the system currently limits itself to displaying wildfire location information gathered from the internet, it will be expanded to allow it to display the spread of a wildfire as predicted the DDDAS wildland fire simulation system. The output of the DDDAS will be in a netCDF file format. A server side script will download the output file generated by the system at set intervals. A netCDF file reader will extract the coordinates of the fire lines and write the information into the .kml file using the existing kml file writer.

Appendix A.

The table below represents the level 2 data structure for the analysis code [14].

File Group	Contents	Index Locations
1	$(B, \emptyset, \emptyset)$	(1, 2, . . , p)
2	(ξ_x, ξ_y, ξ_z)	(p+1, . . , 2p)
3	(U, v, ω)	(2p+1, . . , 3p)
4	$(\rho, \emptyset, \emptyset)$	(3p+1, . . , 4p)
5	(q_v, q_c, RH)	(4p+1, . . , 5p)
6	$(\theta, \Psi, \emptyset)$	(5p+1, . . , 6p)
7	(q_{IA}, N_{IA}, q_{IB})	(6p+1, . . , 7p)
8	$(N_{IB}, \emptyset, \emptyset)$	(7p+1, . . , 8p)
9	$(\emptyset, \emptyset, \emptyset)$	(8p+1, . . , 9p)
10	$(p, \emptyset, \emptyset)$	(9p+1, . . , 10p)
11	(u', v', C_d)	(10p+1, . . , 11p)
12	$(\rho u'w', \rho v'w', \rho w'\theta)$	(11p+1, . . , 12p)
13	$(\rho w'q_v', \rho w'q_c, q_R)$	(12p+1, . . , 13p)
14	$(c_pT + Lq_v, \emptyset, \emptyset)$	(13p+1, . . , 14p)
15	$(\emptyset, \emptyset, \emptyset)$	(14p+1, . . , 15p)
16	$(w, K_m, residual)$	(15p+1, . . , 16p)

References

1. Google Earth homepage,
   http://earth.google.com, 03/03/2008
2. KML Documentation Introduction,
   http://code.google.com/apis/kml/documentation/index.html, 03/03/2008
3. GNU Wget Manual,
   http://www.gnu.org/software/wget/manual, 03/03/2008
4. Crontab – quick reference,
   http://www.adminschoice.com/docs/crontab.htm, 03/03/2008
5. Information on crontab configuration,
   http://www.ss64.com/bash/crontab.html, 03/03/2008
6. Description of .dbf file format,
   http://filext.com/file-extension/dbf, 03/03/2008
7. KML Tutorial,
   http://code.google.com/apis/kml/documentation/kml_tut.html, 03/03/2008
8. FTP server URL for downloading the .dbf file containing the wildfire location data,
   ftp://gp16.ssd.nesdis.noaa.gov/pub/FIRE/MODIS/GIS/modislatest24hr.dbf, 03/03/2008
9. Data formats and GIS,
   http://coastwatch.noaa.gov/cw_form_shp.html, 03/03/2008
10. DBF File structure,
    http://www.dbf2002.com/dbf-file-format.html, 03/03/2008
11. Raymond Hettinger: *Reading Dbf file in Python*,
    http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/362715, 03/03/2008
12. Craig C. Douglas, Jonathan D. Beezley, Janice Coen, Deng Li, Wei Li, Alan K. Mandel, Jan Mandel, Guan Qin, and Anthony Vodacek: *Demonstrating the*

validity of a Wildfire DDDAS,

Computational Science – ICCS 2006. Volume 3993/2006.

Springer Berlin/Heidelberg, May 10, 2006. 522-529

13. Janice Coen, Jonathan Beezley, Lynn S. Bennethum, Craig C. Douglas, Minjeong Kim, Robert Kremens, Jan Mandel, Guan Qin and Anthony Vodacek: *A wildland fire dynamic data-driven application system*,

    11th symposium on integrated observing and assimilation systems for the atmosphere, oceans and land surface. American Meteorological Society, January 14-17, 2007. San Antonio. CD-ROM, Paper 3.12.

14. Terry L. Clark, William D. Hall, Janice Coen: *Source code documentation for the Clark-Hall Cloud-Scale Model,*

    Code version G3CH01. NCAR Tech. Note. NCAR/TN-426+STR, 1996, 174 pp

15. Janice Coen: *Simulation of the Big ELK Fire using coupled atmosphere-fire modeling*,

    International Journal of Wildland Fire. Volume 14, Number 1, 2005. 49-59

16. Anderson, H: *Aids to determining fuel models for estimating fire behavior*, USDA Forest Service, Intermountain Forest and range Experiment Station, INT-122 (1982)

17. http://www.metoffice.gov.uk/research/nw/analysis/, 03/03/2008

18. Jan Mandel, M. Chen, L.p. Franca, C. Johns, A. Puhalskii, J.L. Coen, C.C. Douglas, R. Kremens, A. Vodacek, W. Zhao: *A note on Dynamic Data Driven Wildfire Modeling,*

    Computational Science – ICCS 2004. Volume 3038/2006.

    Springer Berlin/Heidelberg, May 12, 2004. 725-731

19. F. Bouttier and P. Courtier: *Data Assimilation concepts and methods*, http://www.ecmwf.int/newsevents/training/rcourse_notes/DATA_ASSIMILATION/ASSIM_CONCEPTS/Assim_concepts2.html#962570, November 2007

20. http://www.pnas.org/cgi/content/full/97/21/11143, 03/03/2008

21. Geir Evensen: *Sampling strategies and square root analysis schemes for EnKF*, Ocean Dynamics (2004) 54: 539-560 DOI 10.1007/s10236-004-0099-2

22. Geir Evensen: *The Ensemble Kalman Filter: Theoretical formulations and practical implementation*,
Nansen Environmental and Remote Sensing Center, Bergen Norway.

23. Jan mandel and Craig W. Johns: *A two-stage Ensemble Kalman Filter for smooth data assimilation,*
Environmental and Ecological Statistics, Volume 15, Number 1, 2008. 101-110

24. Jan Mandel: *Overall documentation file for the wildfires project*

25. Janice Coen, C.C. Douglas and Adam Zones: *Access and Modification of Data in the Wildfire Code*

26. Terry L. Clark, Janice Coen and Don Lantham: *Description of a Coupled Atmosphere-Fire Model*

27. Gernot Hoffmann (University of Applied Sciences, Emden): *Interpolations for Image Wrapping*
http://www.fho-emden.de/%7Ehoffmann/bicubic03042002.pdf,
January, 2008

28. David Kidner, Mark Dorey and Derek Smith: *What's the point? Interpolation and extrapolation with a regular grid DEM*
http://www.geovista.psu.edu/sites/geocomp99/Gc99/082/gc_082.htm,
January 2008

29. W.Z. Shi, Q.Q. Li and C.Q. Zhu: *Estimating the propagation error of DEM from higher-order interpolation algorithms,*
International Journal of Remote Sensing. Volume 26, number 14, 2005. 3069-3084

30. Terry L. Clark, Mary Ann Jenkins, Janice Coen, and David Packham: *A Coupled Atmosphere-Fire Model: Convective Feedback on Fire-Line Dynamics*
Journal of Applied Meteorology, volume 35, June 1996, 887 -899

31. Jan Mandel, Jonathan D. Beezley, Lynn S. Bennethum, Soham Chakraborty, Janice L. Coen, Craig C. Douglas, Jay Hatcher, Minjeong Kim, and Anthony Vodacek: *A Dynamic Data Driven Wildland Fire Model,*
Computational Science – ICCS 2007. Volume 4487/2007.
Springer Berlin/Heidelberg, July 13, 2007. 1042-1049
32. Jan Mandel, Lynn S. Bennethum, Jonathan D. Beezley, Janice L. Coen, Craig C. Douglas, Minjeong Kim, and Anthony Vodacek: *A wildland fire model with data assimilation*,
CCM Report 233, revised March 2007
33. Jan Mandel, Lynn S. Bennethum, Mingshi Chen, Janice L. Coen, Craig C. Douglas, Leopoldo P. Franca, Craig J. Johns, Minjeong Kim, Andrew V. Knyazev, Robert Kremens, Vaibhav Kulkarni, Guan Qin, Anthony Vodacek, Jianjia Wu, Wei Zhao, and Adam Zornes: *Towards a Dynamic Data Driven Application System for Wildfire Simulation,*
Computational Science – ICCS 2005. Volume 3515/2005.
Springer Berlin/Heidelberg, May 04, 2005. 632-639
34. Jonathan D. Beezley and Jan Mandel: *Morphing Ensemble Kalman Filters*,
Tellus. Volume 60A, 2007. 131-140
35. Jan Mandel and Jonathan D. Beezley, *Predictor-Corrector and Morphing Ensemble Filters for the Assimilation of Sparse Data into High-Dimensional Nonlinear Systems*,
11th Symposium on Integrated Observing and Assimilation Systems for the Atmosphere, Oceans, and Land Surface (IOAS-AOLS), CD-ROM, Paper 4.12, 87th American Meteorological Society Annual Meeting, San Antonio, TX, January 2007

Vita

Soham Chakraborty

Date of Birth: 08/23/1981

Place of Birth: Calcutta, India

Education:

Manipal Institute of Technology, Manipal, India June, 2004

Degree: Bachelor of Engineering

Major: Computer Science and Engineering

Professional Positions:

Infosys technologies Ltd, Bhubaneshwer, India 08/ 2004- 12/2005

Position: Software Engineer

University of Kentucky, Lexington, KY 09/2006 - 05/2007

Position: Research Assistant

Epic Systems Corporation, Verona, WI 11/2007 - Present

Position: EDI Interface Analyst

Publications:

Jan Mandel, Jonathan D. Beezley, Lynn S. Bennethum, Soham Chakraborty, Janice L. Coen, Craig C. Douglas, Jay Hatcher, Minjeong Kim, and Anthony Vodacek: *A Dynamic Data Driven Wildland Fire Model,*
Computational Science – ICCS 2007. Volume 4487/2007.
Springer Berlin/Heidelberg, July 13, 2007. 1042-1049

Craig C. Douglas, Divya Bansal, Jonathan D. Beezley, Lynn S. Bennethum,
Soham Chakraborty, Janice L. Coen, Yalchin Efendiev, Richard E. Ewing, Jay Hatcher,
Mohamed Iskandarani, Christopher R. Johnson, Deng Li, Minjeong Kim,
Robert A. Lodder, Jan Mandel, Guan Qin and Anthony Vodacek,

Dynamic data-driven application systems for empty houses, contaminant tracking, and wild land fire line prediction,
Grid-based problem Solving Environments. Volume 239/2007.
Springer Berlin/Heidelberg, Nov 16, 2007. 255-272