A PHYSICAL MODEL OF DRY RAVEL MOVEMENT

By

XIANGYANG FU

A thesis submitted in partial fulfillment of

the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING

WASHINGTON STATE UNIVERSITY

College of Engineering and Architecture

DECEMBER 2004

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of XIANGYANG FU find it satisfactory and recommend that it be accepted.

_____

Chair

_____

_____

ACKNOWLEDGMENTS

I would like to thank all the people who have given me advice and helped me with my degree. My sincere appreciation to my major professor Dr. Joan Q. Wu for her guidance, patience, and support throughout my graduate program at Washington State University. I must thank Dr. Peter R. Robichaud and Dr. Sunil Sharma, without their support and invaluable advice, I would not have completed the research. I would like to acknowledge the staff in the Forest Fire Laboratory, Pacific Southwest Research Station, USDA Forest Service, Riverside, CA. Peter Wohlgemuth provided useful insights into the real world application of this research, and Lynne Casal provided the topographic data of the San Dimas Experimental Forest. I am grateful to Mr. Roger Nelson for his advice and assistant in developing the dry ravel model program. Funding for this project was in part provided by the U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station; and the U.S. Department of Interior, U.S. Department of Agriculture, Forest Service Joint Fire Science Program through a Research Join Venture Agreement.

Especially, I would like to acknowledge the assistance of my lovely wife, Jian Ling, to prepare the manuscript for the thesis and publication during the last years. It was her love encouraged me overcome all the difficulty of the research and finally complete my thesis.

A PHYSICAL MODEL OF DRY RAVEL MOVEMENT

Abstract

by Xiangyang Fu, M.S.
Washington State University
December 2004

Chair: Joan Q. Wu

Dry ravel is a gravity-induced downslope surface movement of soil grains, aggregates, and rock materials that commonly occurs on steep hillslopes after disturbances such as wildfires. A quantitative analysis and model of dry ravel is needed to understand the behavior and effects of dry ravel movement. In this study, a physical-based model is developed based on classic mechanical laws and particle flow theory. The model predicts source locations, movement path, and deposition areas of dry ravel produced after wildfires. The short-term dry ravel process is computed with theoretical calculations, and the long-term effects are described with both theoretical calculations and empirical functional characterization. The sensitivity of the input coefficients, and provide a suitable range for these input coefficients are assessed. To calibrate and validate the model, the estimation of dry ravel from a wildfire area in the San Dimas Experimental Forest in southern California was compared with experimental results. The comparison shows that the model is suitable for analyzing dry ravel on steep slope, with easily-weathered parent material.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1. Overview

Dry ravel is a gravity-induced downslope surface movement of soil grains, aggregates, and rock materials [*Anderson et al.*, 1959]. It is a common form of hillslope erosion in semiarid regions with steep topography [*Wells*, 1981]. When the land slope angle is greater than the kinetic friction angle of soil particles, the soil particles will move (roll, slide, and bounce) downslope due to gravity [*Rice*, 1982]. Vegetation on the slope tends to obstruct the movement of the soil particles and can retain a portion of these soil particles [*Rice*, 1982]. In most cases, soil particles held by vegetation on a hillslope form the major source of dry ravel [*Gabet*, 2003].

After a wildfire, soil on a slope becomes dry and friable and soil particles initially held behind the vegetation will likely flow downslope if a significant portion of the vegetation is removed due to the fire [*Wells*, 1985]. Often, such downslope movement of dry ravel can also be activated by other natural disturbances such as wind, earthquakes, or animal activities. Dry ravel tends to occur over a long period of time (e.g., several months or one year before the vegetation regrowth) and across large areas where wildfires frequently occur [*Krammes*, 1960]. Although the majority of dry ravel movement occurs immediately after wildfires, the remnant process will continue until new vegetation is established. Over a long time period, the individual particles collide and press each other, and their movement is largely related to each other, resembling the behavior of a system. Therefore, the complex dynamic processes of dry ravel may be viewed and described as the mechanical behavior of a system comprised of a collection of irregular shaped soil particles.

1.2. Related Research

Many studies have been conducted to gain a better understanding of dry ravel processes. Most of these studies, however, are focused on qualitative, rather than quantitative, analysis of dry ravel. *Anderson et al.* [1959] conducted experiments to investigate the type, amount, and source of downslope movement of debris in the Los Angeles River Watershed. They found that the main source of the debris was from the south-facing slopes along the mountain front. They also observed more debris deposition during the dry season than during the wet season. *Krammes* [1960] performed an experiment to measure the amount of debris after a wildfire and found that most of the debris was produced immediately after the fire and that the amount of postfire debris was much larger than the amount of prefire debris. *Wells* [1981] investigated the causes of dry ravel occurrence in the mountainous area of Southern California. He identified four important factors that combine to facilitate dry ravel processes: steep slopes; easily-weathered parent material; highly flammable vegetation; and the area's Mediterranean climate typified by short wet winters and long dry summers. He concluded that wildfires have played a significant role in sediment movement in the mountains of Southern California.

Little effort has been made to quantify the dry ravel movement. *Cundall and Stack* [1979] developed a discrete numerical model to calculate the movement of particles of granular materials. The model was primarily based on Newton's second law and a force-displacement law, and could be used to calculate the velocity, position and interaction force of each individual particle of a granular material. The model is useful for improving the understanding of dry ravel, yet the applicability of the model has been limited primarily because field conditions often involve extremely large quantities of soil particles and the calculation of the velocity and position of each

individual particle demands impractically long computer run time.

Gabet [2003] derived equations for calculating the downslope mass flux of dry ravel and conducted flume experiments and field measurements to evaluate the derived equations. These equations include most of the key factors controlling dry ravel, such as the slope angle, coefficient of kinetic friction, amount of soil available to move and the characteristics of the disturbances(e.g., animal activities). Many other important conditions and factors, however, were not explicitly represented, (e.g., the distribution of the initial velocities and the specific dry ravel sources). Instead, these conditions were described using a single parameter, which was assumed to be invariant with time and space. As the parameter has no explicit physical meaning and is difficult to determine, this method is therefore of limited applicability.

1.3. Objectives

The main goal of this research is to develop a physical model for field-scale dry ravel movement. The specific objectives are: (1) to develop a physical model that simulates the processes of dry ravel movement after a wildfire within a typical watershed under natural topographic, soil and vegetation conditions; and, (2) to evaluate the model performance by testing its adequacy under a conceptual setting; and (3) to further evaluate and refine the model by testing it using field experimental data.

CHAPTER TWO

METHODS

During the process of dry ravel, soil particles will translate, rotate, and collide with each other while moving downslope. Mathematically, it is difficult to obtain the exact solution of these mechanical movements due to the difficulties in accounting for (1) the dynamic and complicated interactions among the large number of individual soil particles involved in the process, and (2) the dynamic and complex ambient field and environmental conditions that can impact dry ravel. To simplify the problem, the model in this study will calculate the quantity of soil moving downslope as a whole and will not track the exact movements of individual soil particles. The simulation of continuous propagation of the soil flux will depend largely on the topography within the model domain, surface conditions (e.g., densely populated with vegetation versus bare soil), and the mechanical properties of the soil (e.g., the kinetic friction coefficient). The model will hereafter be referred to as a mass-flux model.

2.1. Model Conceptualization

In the mass-flux model, dry ravel movements are treated as a series of mass fluxes. The model domain is divided into isometric square cells for calculation (Figure 1). The laws of conservation of mass, momentum and energy are used to govern the magnitude and direction of the mass fluxes.

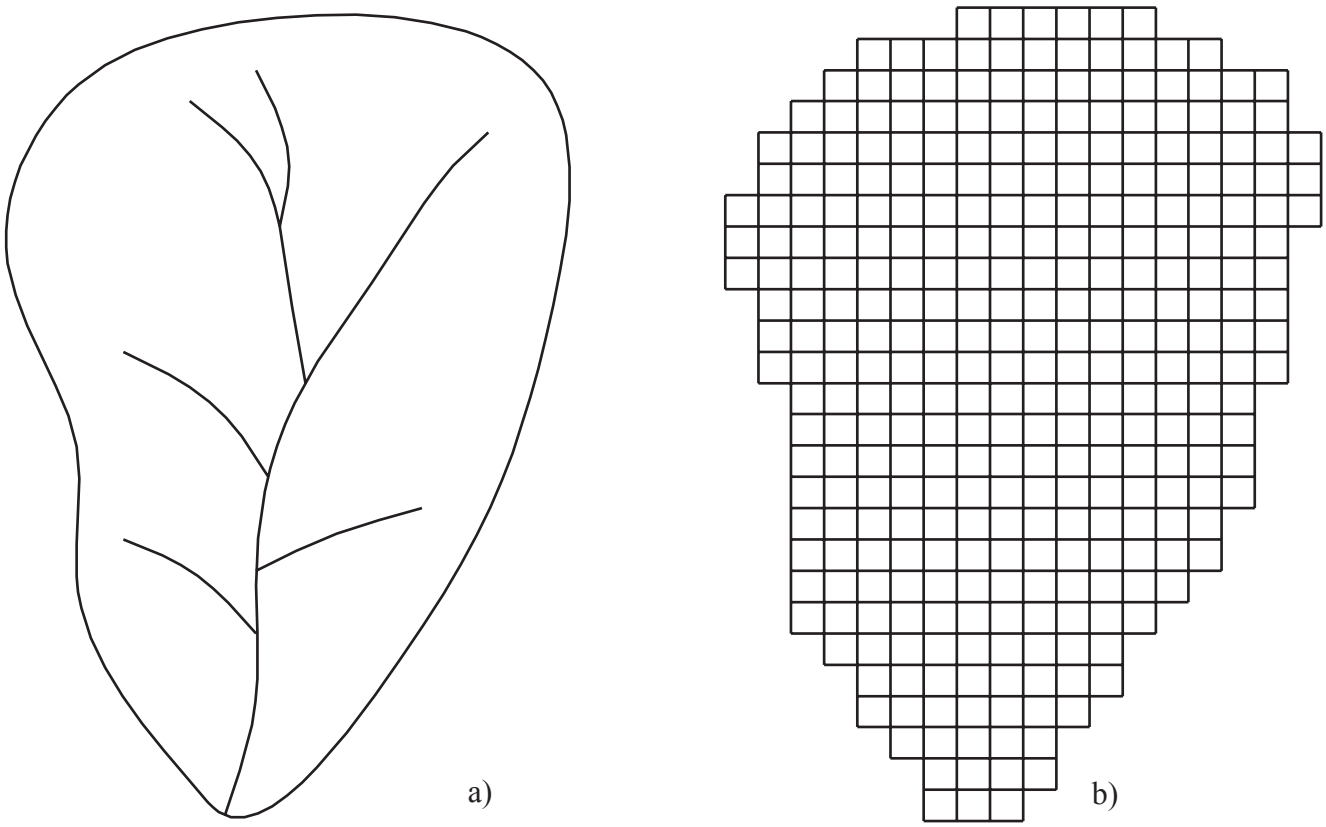

Figure 1. Conceptual model domain with a channel system of a) the actual watershed, b) The represented model domain with isometric cells.

After a wildfire, some cells within the model domain will become completely burned. Soil particles in these cells are detached from the original position and move downslope. The occurrence of soil particles being detached from a single cell due to one wildfire is defined as a ravel event, and the burned cells are defined as fire-impact cells. We also define the fire-impact depth as the depth within which the roots of the vegetation will be fully consumed by fire, therefore soil particles within that layer available to move. The location of the fire-impact cells and fire-impact depth are required as user input.

In the mass-flux model, we consider two types of ravel events: one is short-term ravel event, and the other is long-term ravel event. Short-term ravel events occur within one day immediately after a wildfire, in these cases the dry ravel primarily consists of the soil particles originally held by the vegetation stems and the root systems of vegetation above the fire-impact depth. However, not all of the soil particles held by the root systems of the vegetation are detached during and immediately after the fire. Part of them may remain in place while becoming unstable. During a long period after a wildfire (e.g., several months to one year before the vegetation is re-established), these remaining soil particles are sporadically mobilized by natural disturbances, such as wind and animal activities, thereby producing long-term ravel events.

For each short-term ravel event, soil particles detached from the original fire-impact cell will be distributed to one, or at most, two neighboring cells depending on the elevation of the four nodes of the cell. The soil mass flux entering and leaving a cell are called inflow and outflow velocities. Each mass flux from an upslope cell will be again distributed to one, or at most two, neighboring cells, and this process is repeated until there is no new mass flux produced, indicating that all dry ravel has been deposited to some new locations within, or has reached, the boundary of the model domain.

After all of the ravel events are evaluated, the short-term simulation of dry ravel is completed. For the long-term ravel events, an empirical function is used to estimate the temporal variation of the source of dry ravel, and subsequently the dry ravel movement with time.

The production, deposition and net mass change of dry ravel in each cell and the paths of the mass flux across the model domain are the main outputs of the model. The cumulative amount of dry ravel moving out of the boundary of the model domain is also generated for both short- and long-term ravel events.

2.2. Assumptions

The following assumptions are made in order to improve the computation efficiency of the mass-flux model. Whenever an assumption may be in violation of the general field conditions or may lead to serious errors, approaches to reducing potential errors are suggested and discussed.

Within each cell in the model domain, a planar surface is assumed and the slope angle and soil bulk density are constants. This assumption may become inadequate when the size of the model cells becomes large (e.g., 20 m) and the change in surface elevation within a individual cell is significant. To reduce the model errors, the user should use small cell sizes (ranging from 1 to 10 m depending on site-specific topographic conditions) although the smaller cell size will lead to increased total number of cells and computation time. Reducing the size of the model domain when using small size of cells is a compromise to avoid overly long computation time while assuring adequate accuracy.

The soil flux across a boundary of an individual cell is assumed to be constant along the boundary, and the resultant velocity acts on the center of the boundary (Figure 2). In reality, the soil flux varies along the boundary and the resultant velocity does not necessarily act on the center point.

This assumption is regarded reasonable when the size of the model cells is much smaller compared to that of the whole model domain (e.g., less than 1% in size ratio), in which case the variation of velocity along a cell's boundary is typically insignificant.

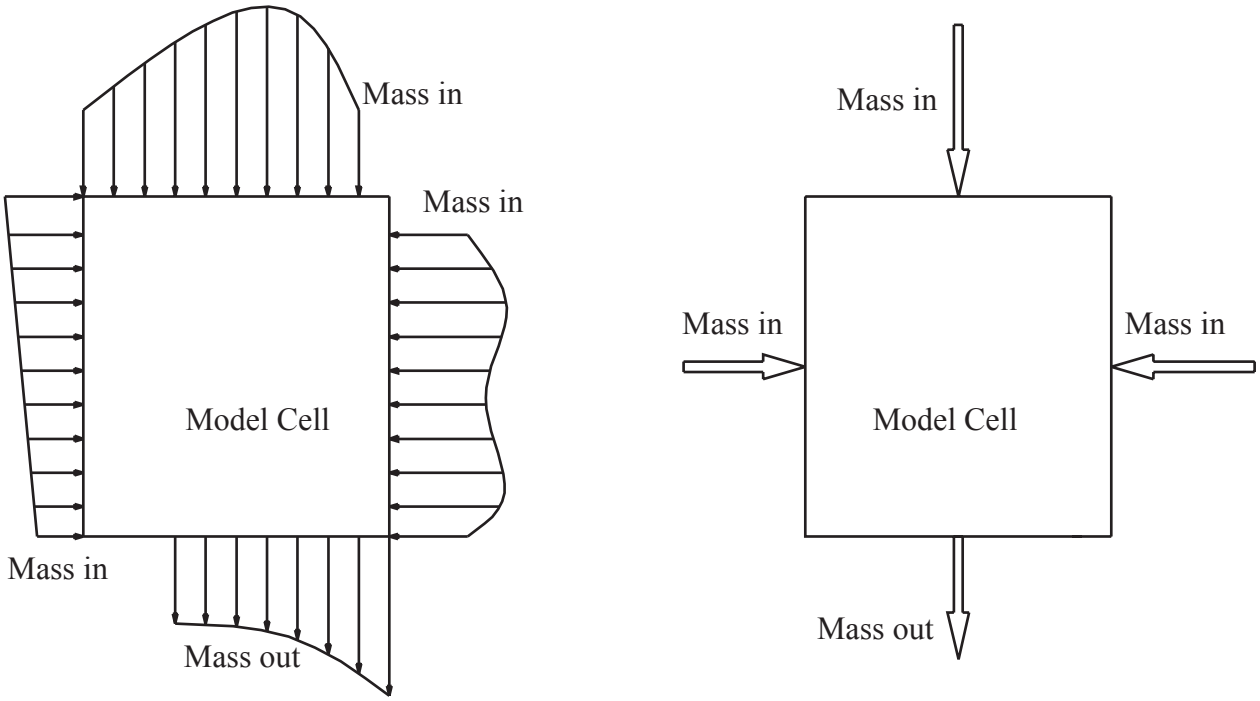

Figure 2. Model simplification of mass flux distribution.

2.3. Sources of Flowing Soil

Two major sources of dry ravel are considered: the soil particles initially held by the vegetation stems, and those held by fine roots of the vegetation (Figure 3). These soil particles become mobile after the vegetation is consumed by fire. For dry ravel held by stems in each cell (Figure 4), we can determine the volume V using Equations (1).

$$V_s = \frac{D H H_{bs}}{6} \tag{1}$$

$$H_{bs} = \frac{h \sin(90 + \gamma)}{\sin(\alpha - \gamma)} \tag{1a}$$

$$h = \frac{D \tan\gamma}{2} \tag{1b}$$

where V_s (m^3) is the volume of the tetrahedron shaped pile of dry ravel held by stems (Figure 4 and 5), D (m) is the diameter of the plants, $H = h \times \cos\alpha$ (m) is the height of the tetrahedron (Figure 6), H_{bs} (m) is the height of the base of the tetrahedron, h (m) is the height of the side (assumed vertical) of the tetrahedron that is in direct contact with the vegetation, α (°) is the slope angle which is defined as the angle between the surface of a cell and the horizontal plane, and γ (°) is the static friction angle, the angle between the horizontal and the surface slope of the pile of dry ravel formed by free falling soil particles.

The second dry ravel source comprises soil particles within the fire-impact depth. The total volume of this portion of the dry ravel depends on the extent and thickness of the fine roots consumed by the fire. In the mass-flux model, the fire-impact depth is assumed the same for all fire-impact cells (Figure 6). The volume of dry ravel held by fine roots in each cell can be determined

by the product of the fire-impact depth and the surface area of the cell.

The total mass of dry ravel produced in each cell may be determined by Equation (2) and then can be used with the governing equations for dry ravel transport as presented in the following section.

$$M = \rho V \tag{2}$$

where M (g) is the mass of dry ravel produced in each cell, and ρ (g m^{-3}) is the bulk density of the soil, and V (m^3) is the volume of dry ravel produced in each cell.

The velocity of the mass flux flowing out of the original cell depends on the coefficient of kinetic friction β, the size and the slope of cell, and can be determined by Equations (3) and (4)

$$v_y = \sqrt{Y a_y} \tag{3}$$

$$a_y = (g \sin\alpha - g \tan\beta \cos\alpha) \frac{\tan\alpha_y}{\sqrt{\tan^2\alpha_x + \tan^2\alpha_y}} \tag{3a}$$

$$v_x = a_x \sqrt{\frac{Y}{a_y}} \tag{4}$$

$$a_x = (g \sin\alpha - g \tan\beta \cos\alpha) \frac{\tan\alpha_x}{\sqrt{\tan^2\alpha_x + \tan^2\alpha_y}} \tag{4a}$$

where V_y (m s^{-1}) is the surface velocity in y-direction, V_x (m s^{-1}) is the surface velocity in x-direction, Y is the cell size in y-direction, a_x (m s^{-2}) and a_y (m s^{-2}) are the accelerations in x- and y-directions, respectively, g (m s^{-2}) is the gravitational acceleration, α (°) is the slope angle, α_x (°) and α_y (°) are the slope angles in x- and y- directions, respectively, and β (°) is the kinetic friction angle of the soil.

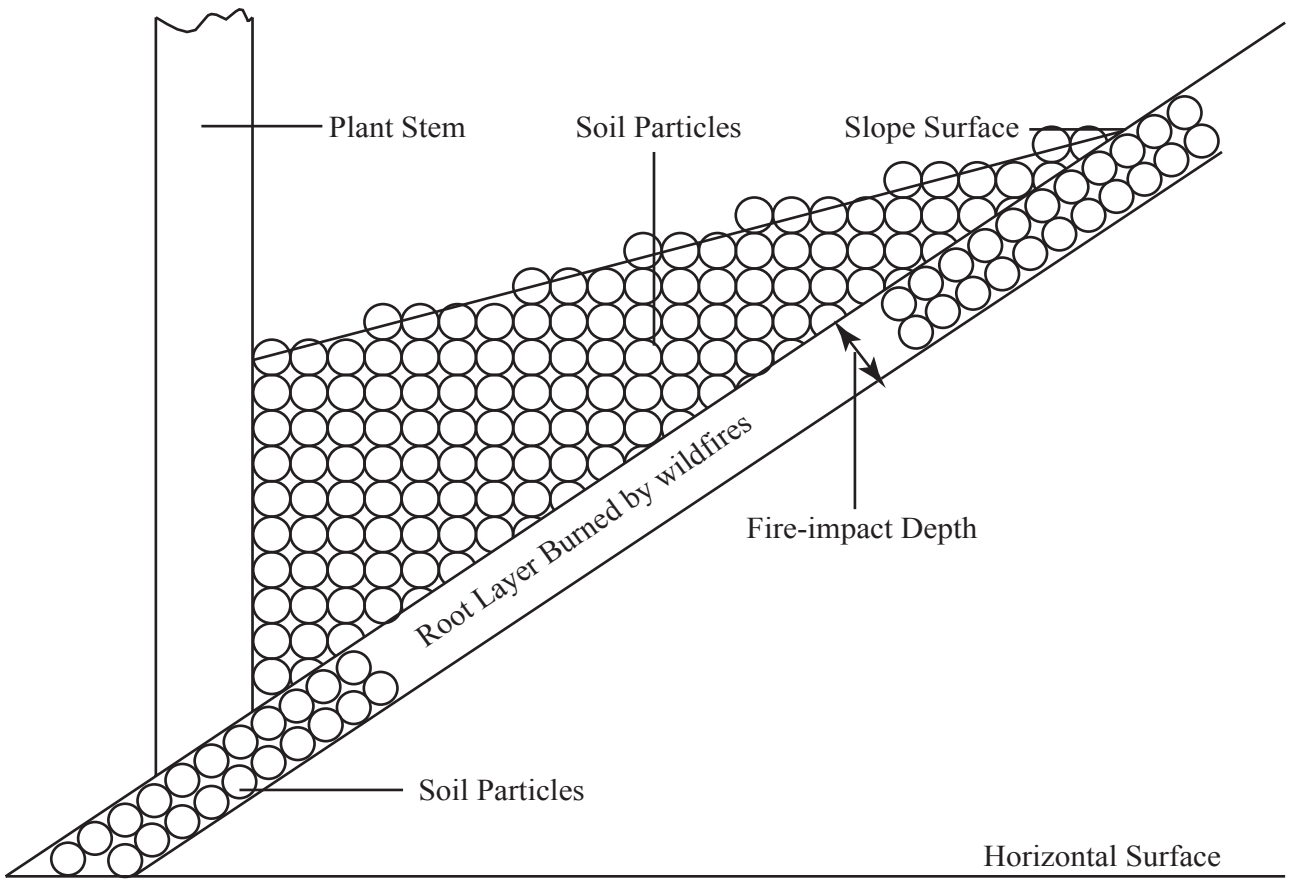

Figure 3. Two types of dry ravel sources , the soil particles held by plant stems and those held by fine plant roots.

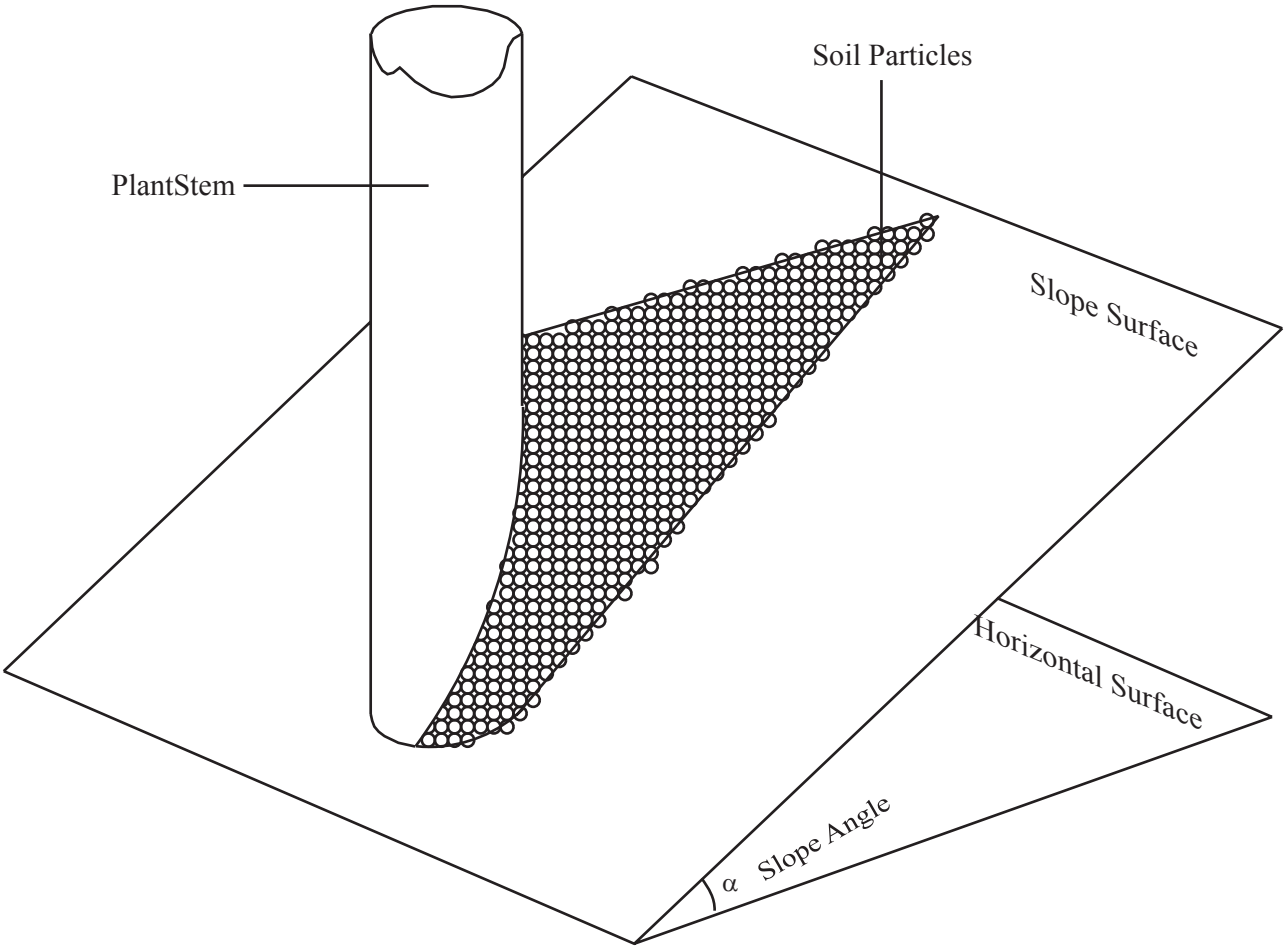

Figure 4. Sketch of dry ravel held by plant stems.

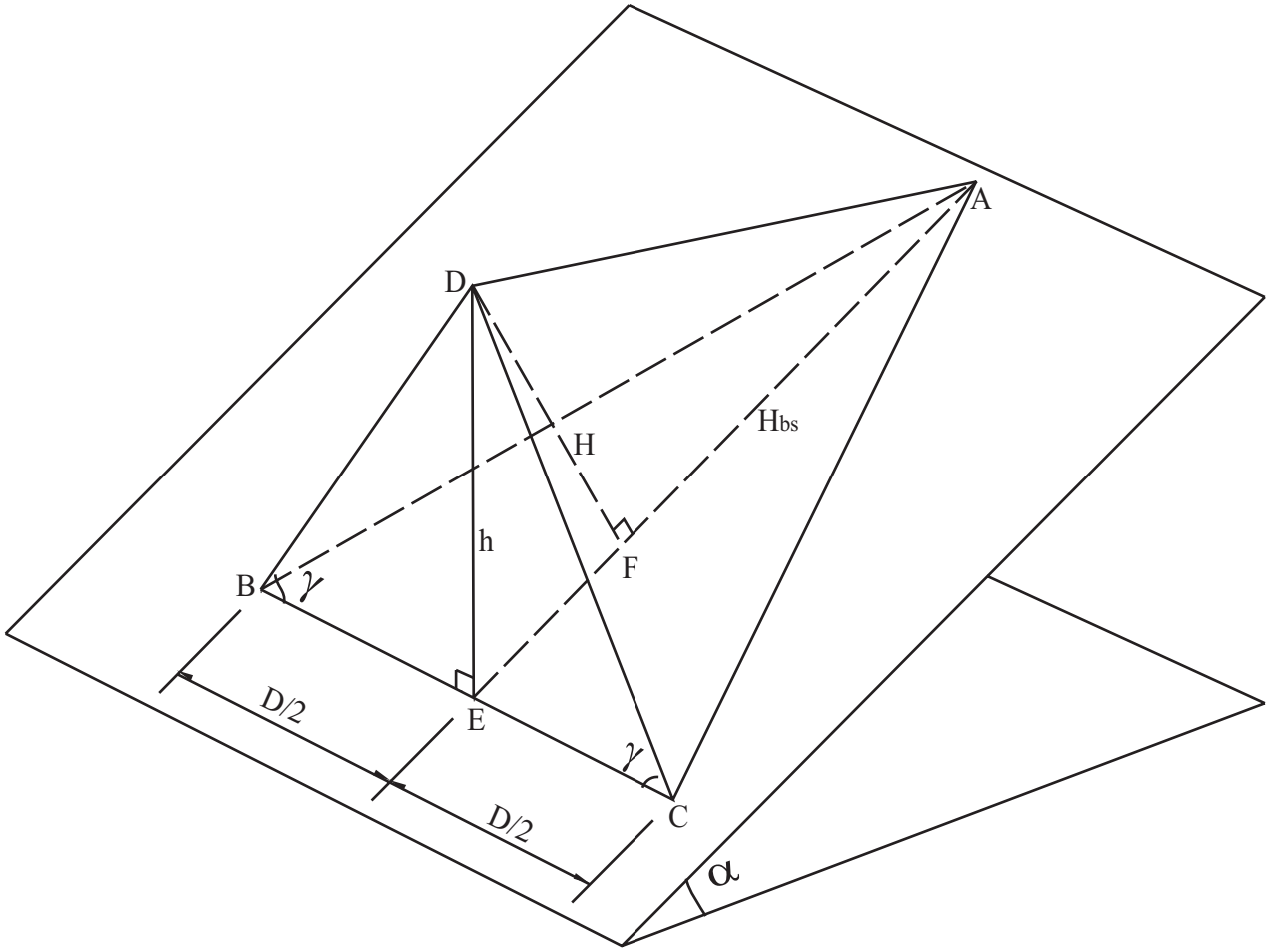

Figure 5. Representative volume of dry ravel held by plant stems. α is slope angle, γ is the angle of repose, D (m) is the diameter of plant stems, h (m) is the height of dry ravel accumulated behind plant stems.

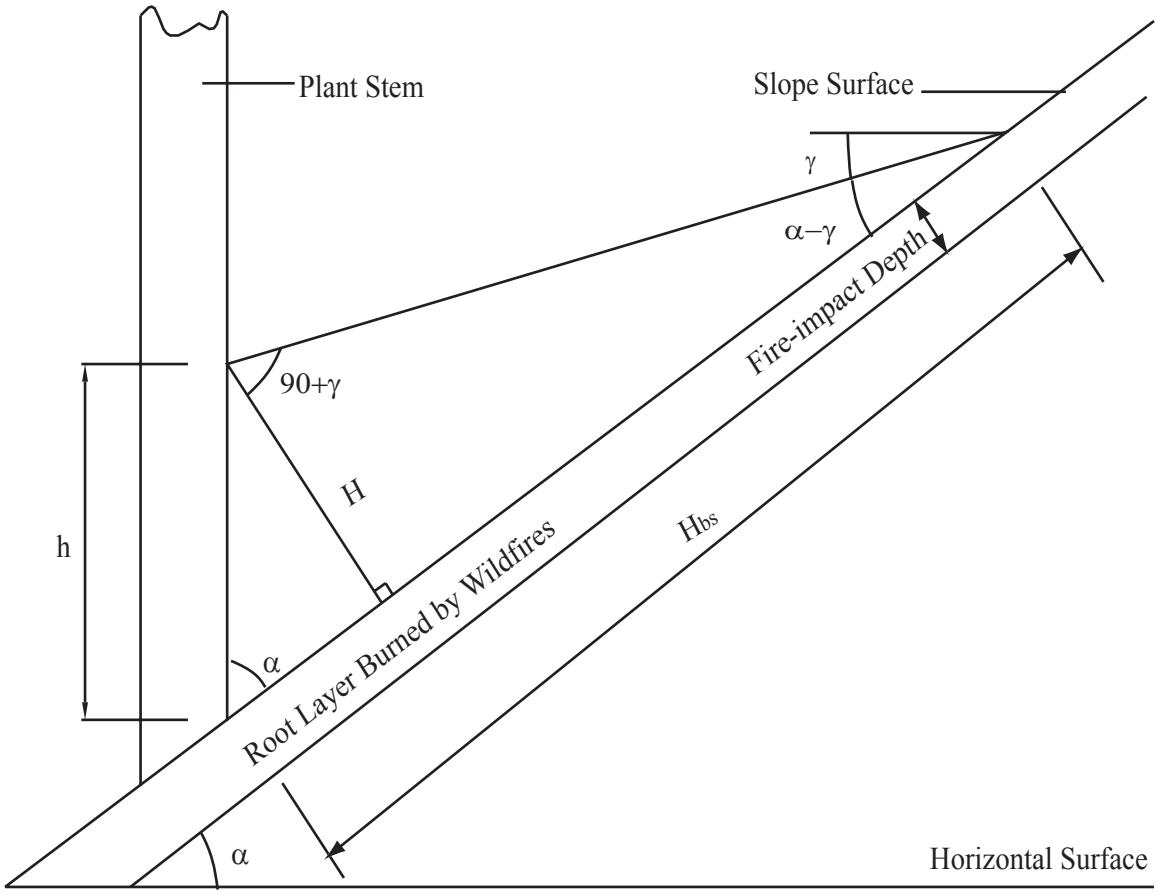

Figure 6. Representative volume of dry ravel held by plant stems (2D). α is slope angle, γ is the soil static friction angle, D (m) is the diameter of plant stems, h (m) is the height of dry ravel accumulated behind plant stems.

2.4. Governing Equations of Mass-flux Velocity and Mass in a Model Cell

For each individual cell within the model domain, the velocity of the outflowing mass flux can be determined by the inflow velocities, slope angle, coefficient of kinetic friction and vegetation density of the cell. With the assumptions specified in the earlier section, the following governing equations were developed to calculate the change in mass and velocities of mass flux in each model cell for two scenarios. The two scenarios evaluated are: 1) soil particles flowing into the cell from the top and 2) soil particles flowing into the cell from the side (Figure 7).

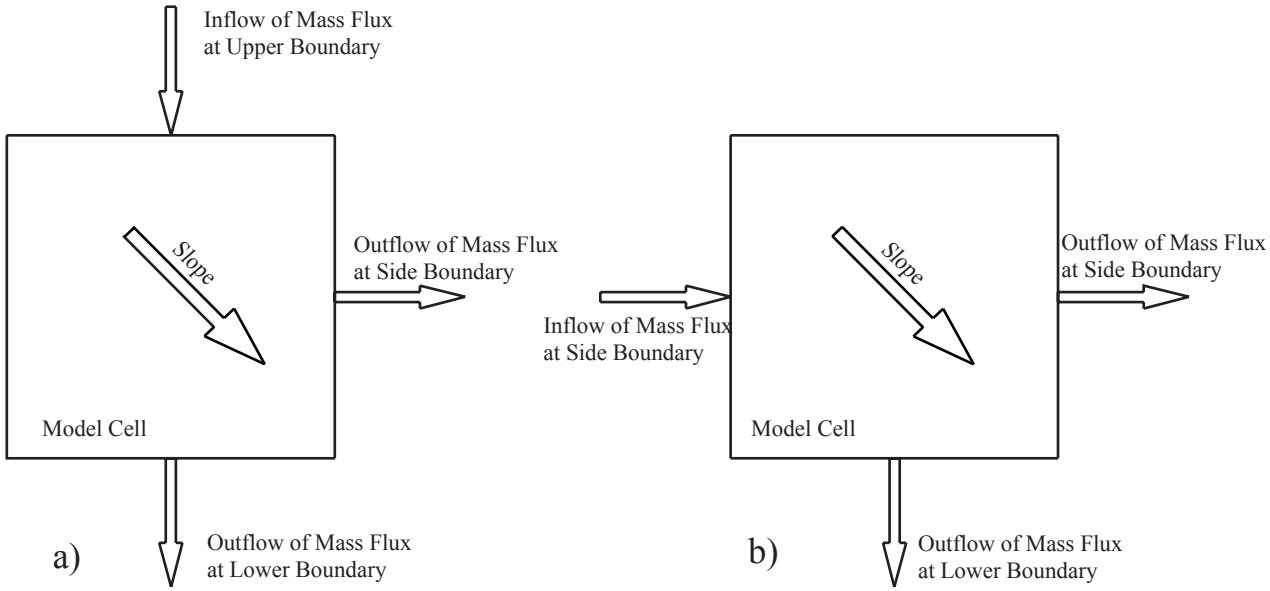

Figure 7. Mass flux entering a) the top and b) side boundaries. Each inflow mass flux is divided into at most two mass fluxes depending on cell topography.

2.4.1. Soil particles flowing in through the top boundary

The first scenario evaluated is the case in which the soil particles are only flowing into a cell through the top boundary. In this case, the outflow velocities across the bottom boundary (Figure 7A) can be described by:

$$v_{yout} = v_{yin} + a_y t \tag{5}$$

$$v_{xout} = v_{xin} + a_x t \tag{6}$$

where v_{xout} (m s^{-1}) and v_{yout} (m s^{-1}) are the outflow velocities and v_{xin} (m s^{-1}) and v_{yin} (m s^{-1}) are the inflow velocities in the x- and y-directions, respectively. The accelerations in the x- and y-directions, a_x (m s^{-2})and a_y (m s^{-2}), are defined as

$$a_y = (g \sin\alpha - g \tan\beta \cos\alpha) \frac{\tan\alpha_y}{\sqrt{\tan^2\alpha_x + \tan^2\alpha_y}} \tag{5a}$$

$$a_x = (g \sin\alpha - g \tan\beta \cos\alpha) \frac{\tan\alpha_x}{\sqrt{\tan^2\alpha_x + \tan^2\alpha_y}} \tag{6a}$$

The time, t, taken by the soil particles to flow through the cell is expressed as

$$t = \frac{\sqrt{v_{yin}^2 + 2 a_y Y} - v_{yin}}{a_y} \tag{7}$$

where Y (m) is the cell size in y-direction. The outflow velocities on the side boundary are given by:

$$v_{yout} = v_{yin} + a_y t \tag{8}$$

$$v_{xout} = v_{xin} + a_x t \tag{9}$$

$$t = \frac{\sqrt{v_{xin}^2 + a_x X} - v_{xin}}{a_x} \tag{10}$$

where the variables are as previously defined.

With the assumption that the resultant velocities act at the centers of the boundaries (Figure 8), the mass M_{yout} (g) flowing through the lower boundary is determined by Equation (11)

$$M_{yout} = (1 - \frac{D_x}{X}) M_{yin} \tag{11}$$

$$D_x = v_x t + \frac{1}{2} a_x t^2 \tag{11a}$$

where D_x (m) is the distance between the center of outflow and the center of the cell's lower boundary, and X (m) is the cell size in x-direction. The mass M_{xout} flowing through the side boundary is determined by Equation (12)

$$M_{xout} = \frac{D_x}{X} M_{yin} \tag{12}$$

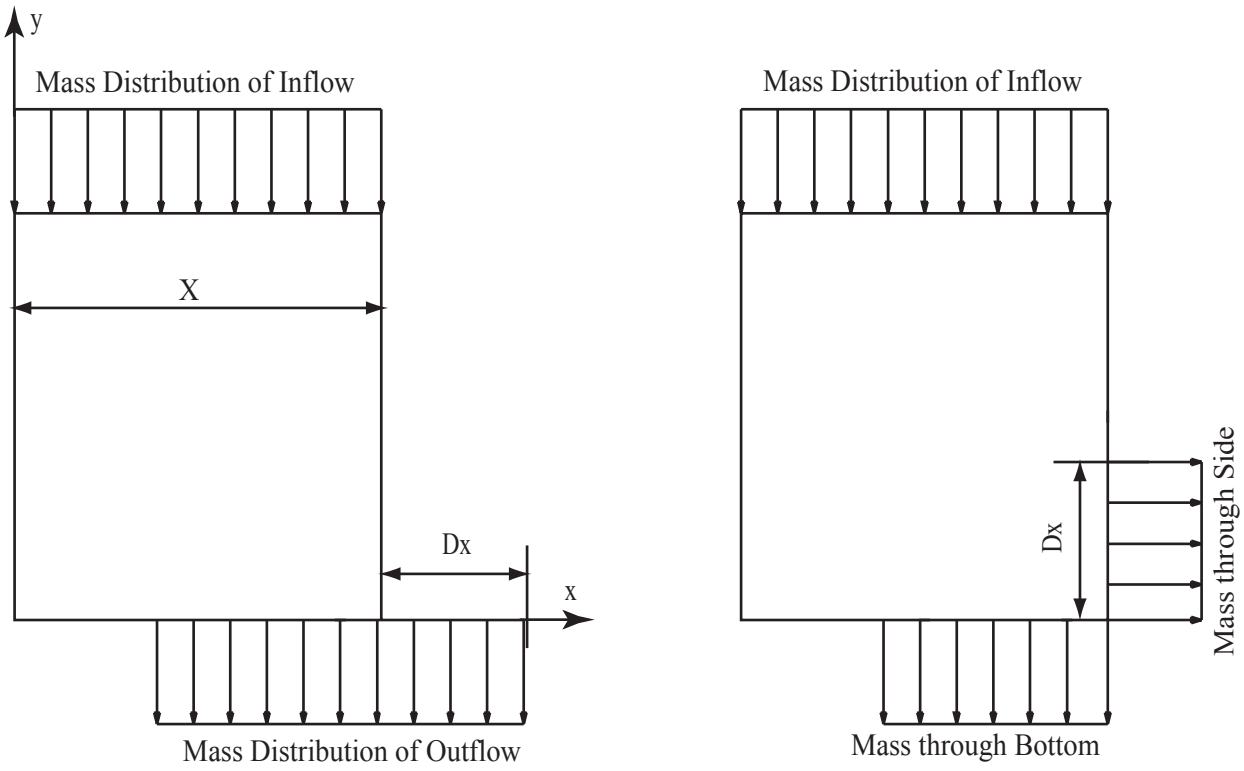

Figure 8. Distribution of mass flux. a) actual situation, b) an model representation.

2.4.2. Soil particles flowing in through the side boundary

The second scenario evaluated is the case in which the soil particles are only flowing into a cell through the side boundaries (Figure 7b). Under this condition, the outflow velocities on the lower boundary are determined by:

$$v_{yout} = v_{yin} + a_y t \tag{13}$$

$$v_{xout} = v_{xin} + a_x t \tag{14}$$

$$t = \frac{\sqrt{v_{yin}^2 + a_y Y} - v_{yin}}{a_y} \tag{15}$$

The outflow velocities on the side boundary are determined by:

$$v_{yout} = v_{yin} + a_y t \tag{16}$$

$$v_{xout} = v_{xin} + a_x t \tag{17}$$

$$t = \frac{\sqrt{v_{xin}^2 + 2 a_x X} - v_{xin}}{a_x} \tag{18}$$

The mass flowing through the lower boundary is determined by:

$$M_{yout} = (1 - \frac{D_x}{X}) M_{yin} \tag{19}$$

$$D_x = v_x t + \frac{1}{2} a_x t^2 \tag{19a}$$

where D_x (m) is the distance between the center of the soil flow and the center of the cell's lower

boundary. The mass flowing through the side boundary is determined by Equation (20).

$$M_{yout} = \frac{D_x}{X} M_{yin} \tag{20}$$

It should be noted that Equation (5) through (20) were developed for the cases in which acceleration in the y-direction is greater than the acceleration in the x-direction ($\alpha_y > \alpha_x$). If $\alpha_y < \alpha_x$, Equation (5) through (20) can be modified by simply interchanging the velocities in the *x*- and *y*-directions to properly describe the actual conditions.

2.5. Long-term Fire Effects of Dry Ravel

In order to compute the production and deposition of dry ravel for a longer period of time (e.g., several months to one year before the vegetation is re-established), an empirical function was used according to the field observations,

$$M_L = M_0(1 + a(1 - e^{-bt})) \tag{21}$$

where M_L (g) is the accumulated mass of dry ravel produced during time t, M_0 (g) is the short-term model results of dry ravel, t (days) is time after the wildfire, a and b are empirical coefficients determined from a regression analysis of experimental data. The coefficient, a, is the ratio of the difference between the amount of long-term and short-term dry ravel production, and the coefficient, b, is a decay coefficient with a dimension of $[T^{-1}]$.

2.6. Model Initialization and Boundary Conditions

An important part of the model initialization is to define the model domain. Digital Elevation Models (DEMs) can be processed in Geographic Information System (GIS) software to obtain a discrete grid system. With the discrete grid system, the topography and boundary of the model domain and the detailed topographic information of each model cell can be explicitly defined.

Other initial information, including the density and the average diameter of the vegetation stems (dependent on vegetation type and conditions), the fire-impact depth (dependent on the fire severity and vegetation), the angle of repose(dependent on soil properties), and the coefficient of kinetic friction (dependent on both the soil and surface conditions). All of this information can be obtained from field investigations and the literature. In this research, the results of laboratory experiments and field observation are used to estimate the information [*Wohlgemuth*, 2004 unpublished data]. All of the input parameters are regarded as spatially variant, and depending on the available data, the user may provide constant average values for the entire model domain or use fully distributed values. Here, we used the constant average values for the entire model domain. According to the fire severity (which is one of the most important factors to evaluate after a fire), the fire-impact depth (frequently ranging from 0–15 mm) can also be estimated by the user. The coefficient of kinetic friction, a critical input to the model, can be obtained from the literature or through laboratory experiments. Here the average slope angle of the entire model domain is used to approximate the angle of kinetic friction in this case.

Typically, all the boundaries are no-flow boundary conditions. The exception to that is the topographically low (or the lower) boundary which is set as a time-variant flow boundary allowing dry ravel to leave the model domain.

2.7. Code Development

A computer program for the mass-flux model was developed in Microsoft® Visual Studio® .net (version 2003). The program requires two input files. The first file contains the following input parameters: general simulation information (simulation time (s), size of grid cell (m)); soil 's kinetic friction coefficient; vegetation characteristics (vegetation density, average diameter of vegetation stem); fire severity (location of impact cells, fire-impact depth). The second file contains the detailed topographic information with elevations for each node of the model cells. Table 1 shows the parameters needed in the first input file, including the angle of repose, kinetic friction angle, diameter of plant stems, vegetation density, cell size, and simulation time. The values in Table 1 were estimated for the dry ravel applicable in Southern California. The input parameters need to be adjusted according to the characteristics of the target domain.

Table 1. The model inputs and the range of the input factors

	Fire-impact Depth	Static Friction Angle	Kinetic Friction Angle	Diameter of Plant Stems	Vegetation Density	Cell Size	Simulation Time
Range	0-15 mm	29°-34°	Slope angle±5°	5-25 cm	1-3 stems/m^2	1-10 m	1day-1year
Model Inputs	0-1 mm	30°-34°	Average slope angle	5 cm	1 stems/m^2	1,2,5,10 m	1 day

The program uses an iterative algorithm to represent the dynamic behavior of dry ravel movement. The iterative algorithm requires the repeated application of the governing equations (1) to (20) to calculate the dry ravel mass and varying velocities for each fire-impact cell and the subsequent distribution of the soil fluxes. Accordingly, the production, deposition, and the net changes in dry ravel mass in each cell are also calculated. A flow chart illustrating the framework of the model and the major algorithms is shown in Figure 9.

For a short-term simulation, each of the fire-impact cells is evaluated sequentially. At the first-level of the iteration, the governing equations (1) to (4) are applied to a fire-impact cell to calculate the dry ravel source. One, or at most two, mass fluxes entering the downslope neighboring cell(s) are produced. The resultant mass fluxes are then evaluated by the second-level iterative algorithm with different governing equations (5) to (20), and another group of mass fluxes entering more downslope cells are produced. These mass fluxes are further evaluated with the same, second-level iteration algorithm, and this iterative process is repeated until there is no new mass flux produced. When this iterative process terminates, the mass fluxes of dry ravel produced from the original source cell are fully simulated and the program proceeds to the computation for the second fire-impact cell. Similarly, the first- and second-level iterations will be executed and this procedure is repeated until all the fire-impact cells have been evaluated. After the short-term simulation is complete, the long-term simulation of dry ravel may be performed depending on the user's choice. The long-term simulation is based on the short-term results and a user-parameterized function, Equation (21), for long-term dry ravel production.

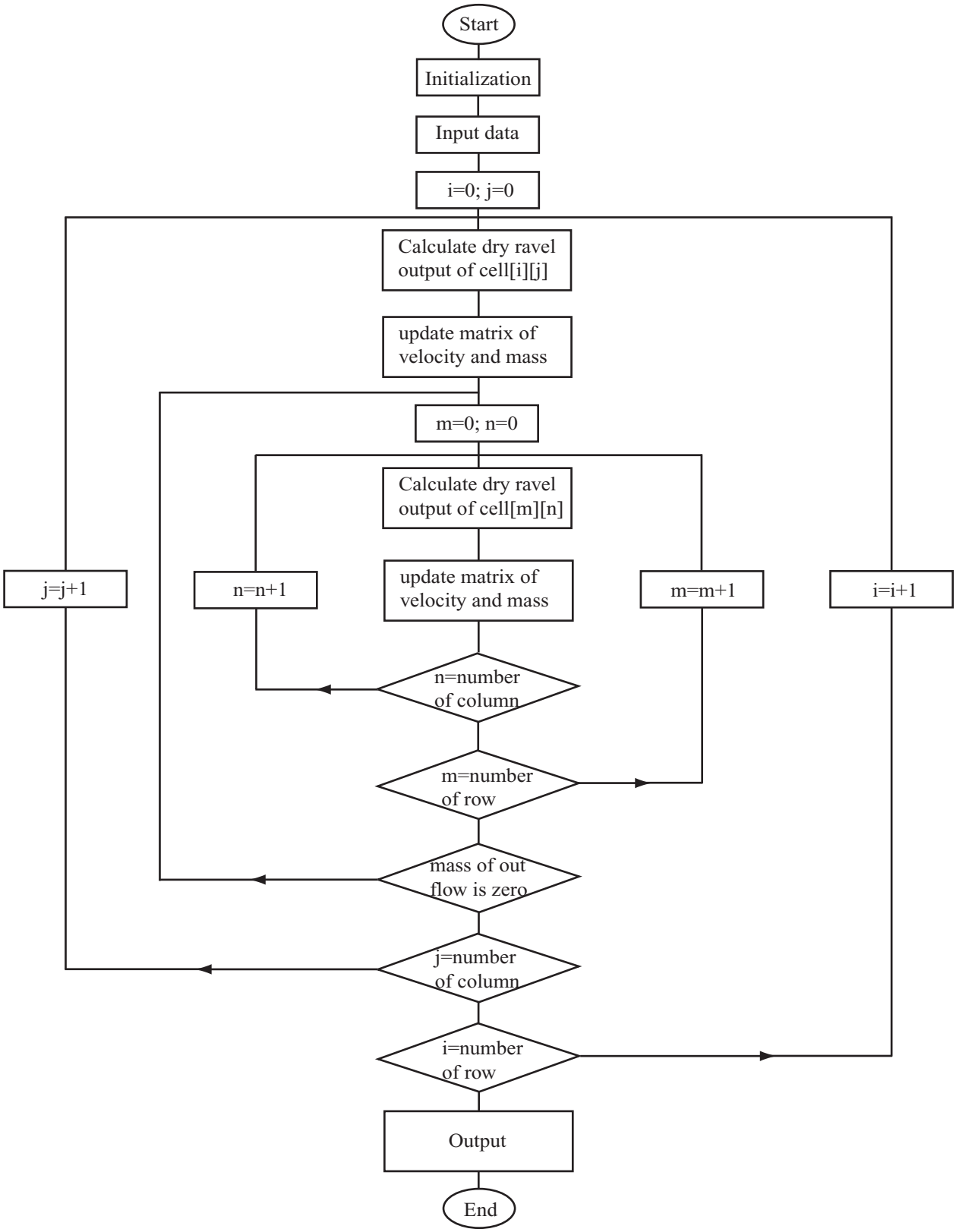

Figure 9. Flowchart for the code of the mass-flux model.

The computer program generates an output file containing (i) the production, deposition, and net changes in dry ravel mass for each cell in the model domain, (ii) cell numbers representing the paths of mass fluxes, and (iii) the total mass leaving the boundary of the model domain, for a short-term simulation. Additionally, the program will generate a series of the deposition of dry ravel as a function of time after the fire as specified by user for a long-term simulation.

2.8. Model Evaluation

2.8.1. A previous laboratory experiment

A previous laboratory experiment was conducted to better understand the mechanisms that govern dry ravel by USDA Forest Service Pacific Southwest Research Station [*Wohlgemuth*, 2004 unpublished data]. Part results of the experiment are used here as the value of the model input parameters.

In this laboratory experiment, the effects of soil texture, slope angle, and vegetation density on dry soil movement were evaluated using a tilting table that consisted of a soil tray that was 1 m × 1 m in area and 5 cm deep and a supporting frame pivoted on upright posts, a cable and pulley system, and a gutter fastened to the downhill end of the tray to catch the soil material as it rolled off the tray. Five different soil materials were used in this study and all came from the surface soil layer of the burn sites in the Southern California mountains. Among these five soil materials, one was obtained from the USDA Forest Service San Dimas Experiment Forest (SDEF), the site where the model predictions are evaluated with the field observation.

Soil properties, including the particle size distribution and dry bulk density, were measured first. The static friction angle, which is the degree of the table inclination at which the soil started to move,

was then determined. The angle of repose and dry soil bulk density are inputs for estimating the dry ravel source in the mass-flux model.

2.8.2. A representative setting

The performance of the mass-flux dry ravel model was first assessed using a representative setting. A 1000 m × 1500 m rectangular model domain near the city of Glendora and vicinity in the Riverside County was selected and clipped from the DEM (Figure 10). This model domain was intended to be representative of the mountainous regions of Southern California prone to wildfire and dry ravel. The topographic information used here was acquired from the USGS website http://data.geocomm.com/catalog/US/61069/group4-3.html.

Simple spatial analysis was performed on the DEM data and the results indicated that the slopes within the model domain were typically steep, with slope angle normally exceeding 45°. No rainfall events were considered, and dry ravel was assumed to be the primary erosion process occurring during the simulation time. The model domain was divided into 10 m × 10 m cells, producing a total of 15,000 cells. Across the domain, chaparral was assumed to be the dominant vegetation, following *Wells* [1981]. The density of vegetation was assumed to be one plant m^{-2} and the average diameter of an individual plant was set to 10 cm.

For the first analysis, it was assumed that all the cells were burned by fire. For this analysis, the model predicted the production, the paths, and the deposition of dry ravel. Based on the output results, a figure showing the deposition in each cell is used to assessed the model prediction. For the second analysis, twelve cells that were randomly chosen across the domain were picked out as the source cells. The complete processes of detachment of dry ravel from these original source cells,

movement downslope, and deposition in the valley and flat areas were simulated. The source cells and the paths of dry ravel are plotted to qualitatively evaluate the model prediction of movement of dry ravel.

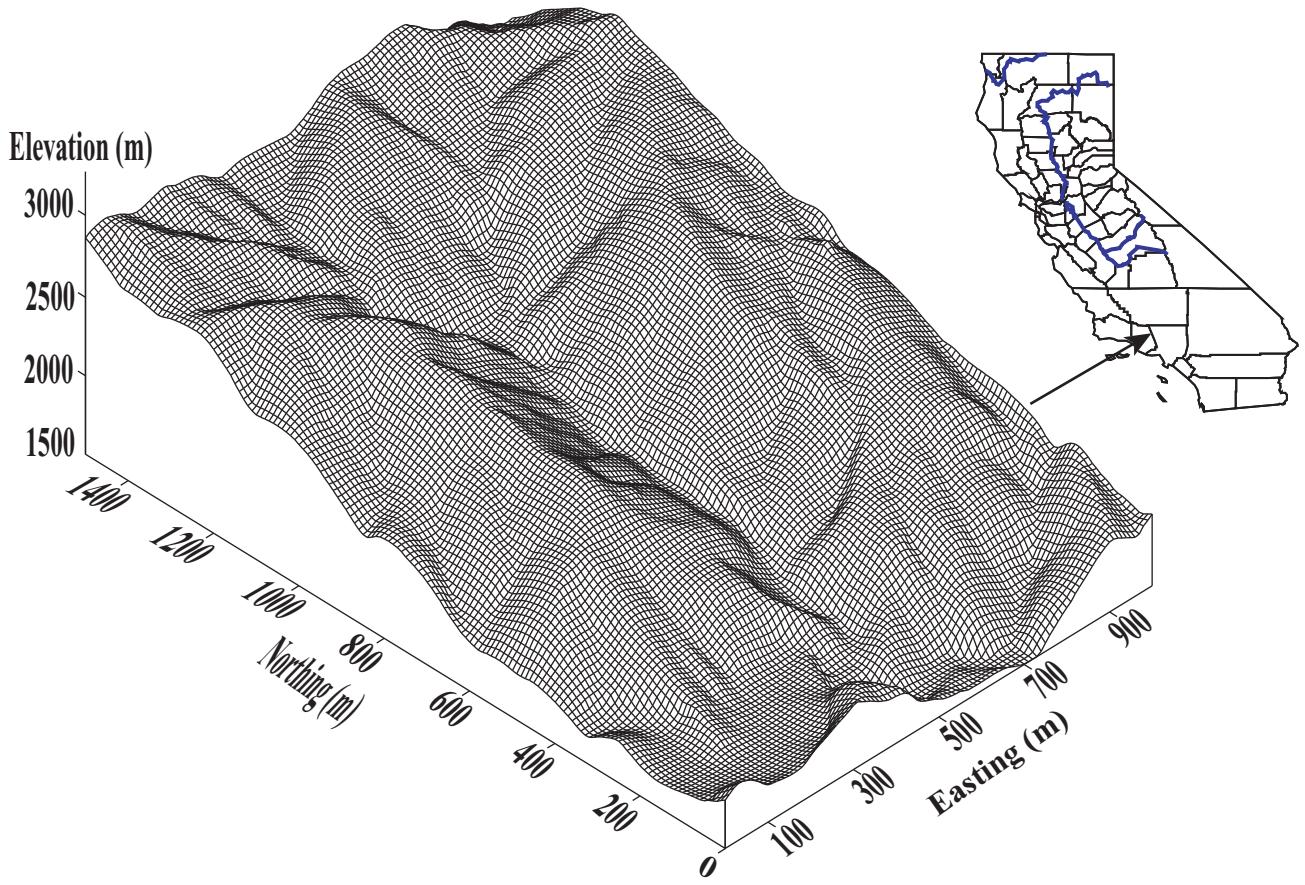

Figure 10. Topography of the representative setting, Riverside County, California.

2.8.3. Model application to the San Dimas Experimental Forest

As mentioned earlier, the Riverside Fire Laboratory Pacific Southwest Research Station, USDA Forest Service has been continually monitoring surface erosion processes in the SDEF for over seven years (1996-2003) [*Wohlgemuth*, 2004 unpublished data]. Like many other areas in the Southern California mountains, the SDEF is characterized by a Mediterranean climate, steep slopes frequently exceeding 100%, and flammable chaparral vegetation cover, and easily weathered parent bedrock.

Table 2. Characteristics of the four small watersheds at the SDEF[a]

Watershed Nr.	Position	Average Slope (%)	Channel Length (m)	Vegetation	Fire Characteristics
0507	117°45′56.19″W 34°12′13.70″N	97.2	782	Grass with sparse sage and scrub	Wildfire (Williams Fire)
0508	117°45′56.19″W 34°12′14.49″N	106.1	739	Woody chaparral	Wildfire
0542	117°45′27.08″W 34°12′33.24″N	105.7	831	Woody chaparral	Wildfire
0560	117°45′50.35″W 34°12′43.96″N	106.5	446	Woody chaparral	Prescribed fire

[a]Soil texture at all four watersheds is loamy sand and the underlain bedrock is Precambrian crystalline.

The monitoring site at the SDEF is centered on 34°12' 30"N and 117° 45' 45"W, covering about one square mile. Four small watersheds (1–2 ha in size, Table 2) were monitored for surface erosion. In each of these four watersheds, 25 sediment collectors (each with a 30 cm rectangular aperture) were installed along fall line transects (perpendicular to slope contours) and another 50 were installed along the hillslope and channel interfaces. The collectors catch all types of debris moving down the slope. Yet the debris captured during the summer time of the field monitoring mainly comprises dry ravel because contribution from other sources (e.g., water erosion, soil creep) during this season is typically negligible [*Anderson et al.*, 1959] and any rain-induced erosion would be

separated in the analysis [*Wohlgemuth*, personal communication]. The collectors were examined periodically and samples were taken to the Riverside Fire Laboratory, Riverside, CA for weighing and analysis of the contents.

Evaluations of the sensitivity of the input parameters were conducted for a range of values for watershed 0508. The mass-flux model was applied to all four watersheds at the SDEF monitoring site with the fire-impact depth (FDI), the most sensitive parameters in the mass-flux model, calibrated first for dry ravel production on hillslopes (mass of dry ravel per unit area). Comparison was then made between observed and predicted dry ravel deposition along the valley (mass per unit length of valley) using analysis of variance (ANOVA) in SAS (SAS Institute, 2004). Finally, the long-term accumulated dry ravel produced for a specific watershed (Watershed 0508) was evaluated.

CHAPTER THREE

RESULTS AND DISCUSSIONS

3.1. Sensitivity Analysis of Model Parameters

3.1.1. Vegetation size and density

The vegetation size and density are two important factors affecting the amount of dry ravel held behind the vegetation stems (Figure 11). The dry ravel source mass increased cubically with the increasing diameter of the vegetation stem, and linearly with the vegetation density. Theoretically, there is no limitation on the dry ravel source mass when the vegetation size increases. In reality, however, any specific vegetation has its specific ranges of size and density (in SDEF, the size and density of chaparral are 5-20 cm and 1-3 stem m^{-2}, respectively). Thus, the dry ravel source behind a vegetation stem is physically limited by the vegetation size and density. Equation (1) can be used to determine this limitation according to the vegetation factors specified by the users.

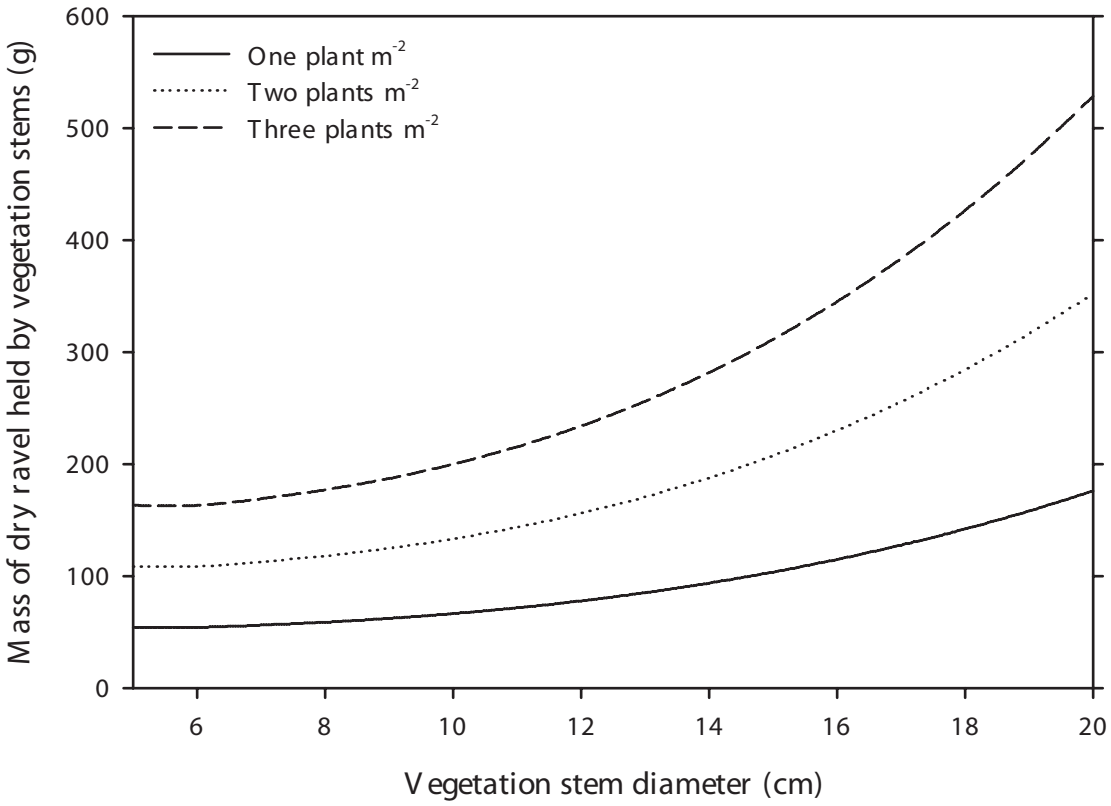

Figure 11. Relationship of the mass of dry ravel held by vegetation stems and the vegetation size. The different curves represent the different vegetation density. Soil kinetic friction angle and the slope angle are taken as 30° and 60°, respectively.

3.1.2. Topography

For the mass-flux model, the topography within the model domain is one of the most important factors that affect the amount of dry ravel. The amount of dry ravel source held by vegetation stems decreases with an increase in slope angle (Figure 12). From Equation (1), the volume of dry ravel held by vegetation stems depends on three factors, the diameter of the vegetation stem, the height of the tetrahedron of dry ravel (Equation 1b), and the height of the base of the tetrahedron (Equation 1a) (Figure 5). The diameter of the vegetation stems is assumed to be a constant for all cells, while the other two factors decrease with the increase in the slope angle. Hence, the volume of dry ravel held by the vegetation stems decreases with increasing slope angle.

When the slope angle becomes close to the soil kinetic friction angle, the dry ravel mass held by a vegetation stem increases dramatically and tends to be unlimited. In such a situation, the height of the tetrahedron base increases greatly and approaches infinity (refer to Equation (1) and Table 3). In reality, however, an upper limit of the dry ravel must be set due to the fact of an existing adjacent, uphill vegetation, and the limited source of dry ravel. This limitation can be readily estimated according to the vegetation density and model grid size. For example, if the vegetation density is 1 stem m^{-2}, the upper limit of the height of the dry ravel tetrahedron base would be 1 m. Consequently, the upper limit of the dry ravel held by a vegetation stem would be 186 g m^{-2} (refer to Equation (1)).

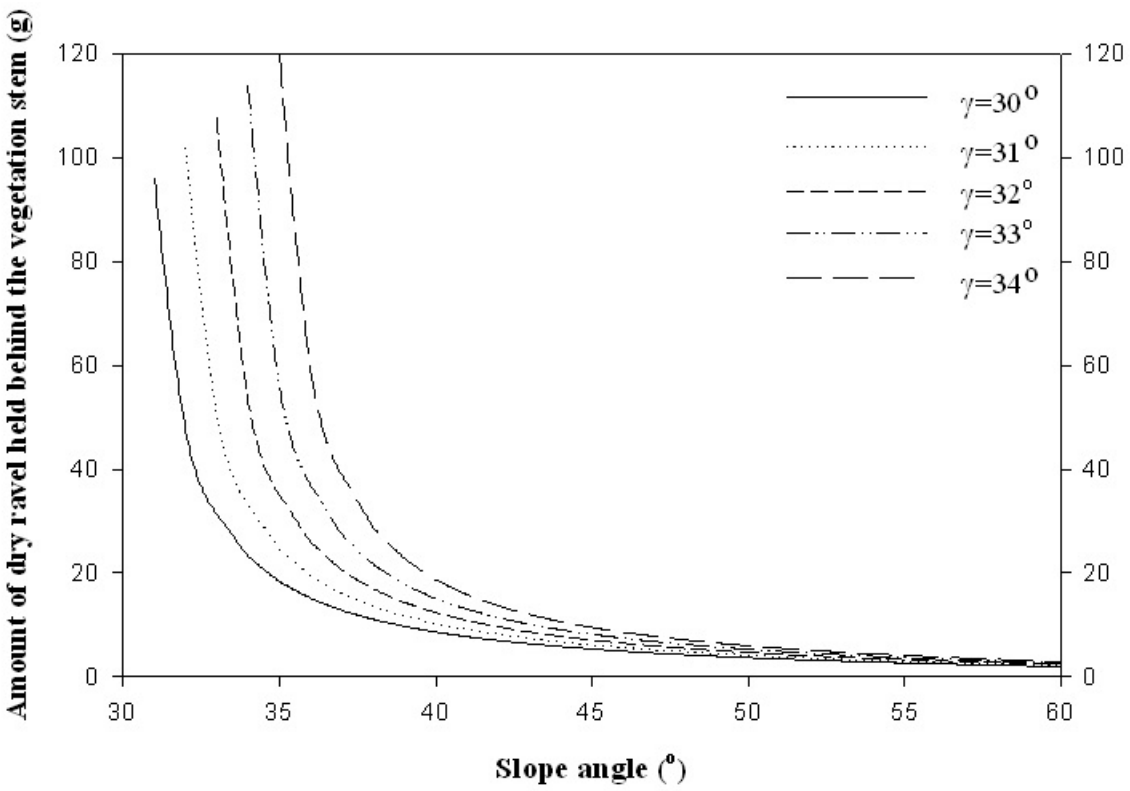

Figure 12. Relationship of the mass of dry ravel held by vegetation stems and slope angle. The different curves represent the different soils with different static friction angles (γ). The diameter of the vegetation stem is 5 cm.

Table 3. The variation of the volume and mass of the dry ravel source held by plant stems with different slope angles and soil static friction angles

Slope Angle, α (°)	Soil Static Friction Angle, γ (°)	Height of Side of Tetrahedron, h (m)	Height of Base, H_{bs} (m)	Height of Tetrahedron, H (m)	Volume of Tetrahedron, V (cm^3)	Mass of Dry Ravel Held by Stem, M (g)
31	30	1.4	71.6	1.2	74	96
35	30	1.4	14.3	1.2	14	18
40	30	1.4	7.2	1.1	7	9
45	30	1.4	4.8	1.0	4	5
50	30	1.4	3.7	0.9	3	4
55	30	1.4	3.0	0.8	2	3
60	30	1.4	2.5	0.7	2	2
32	31	1.5	73.8	1.3	78	102
35	31	1.5	18.5	1.2	19	25
40	31	1.5	8.2	1.2	8	10
45	31	1.5	5.3	1.1	5	6
50	31	1.5	4.0	1.0	3	4
55	31	1.5	3.2	0.9	2	3
60	31	1.5	2.7	0.8	2	2
33	32	1.6	75.9	1.3	83	108
35	32	1.6	25.3	1.3	27	35
40	32	1.6	9.5	1.2	9	12
45	32	1.6	5.9	1.1	5	7
50	32	1.6	4.3	1.0	4	5
55	32	1.6	3.4	0.9	3	3
60	32	1.6	2.8	0.8	2	2
34	33	1.6	78.0	1.3	88	114
35	33	1.6	39.0	1.3	43	56
40	33	1.6	11.2	1.2	12	15
45	33	1.6	6.5	1.1	6	8
50	33	1.6	4.7	1.0	4	5
55	33	1.6	3.6	0.9	3	4
60	33	1.6	3.0	0.8	2	3
35	34	1.7	80.1	1.4	92	120
40	34	1.7	13.4	1.3	14	19
45	34	1.7	7.3	1.2	7	9
50	34	1.7	5.1	1.1	5	6
55	34	1.7	3.9	1.0	3	4
60	34	1.7	3.2	0.8	2	3

[a]Diameter (D) of plant stem is 5cm and soil bulk density is 1300 Kg m^{-3}.

On the other hand, when the slope angle becomes larger than the soil kinetic friction angle, the height of the dry ravel tetrahedron base decreases dramatically, resulting in a rapid decrease of the dry ravel held by the vegetation stems. When the slope angle is greater than 45°, this part of dry ravel source may become negligible. Furthermore, Figure 12 indicates that, for the different soil kinetic friction angles, the mass of dry ravel held by vegetation varies considerably depending on the angle of repose when the angle of repose is close to the slope angle. For example with a slope angle of 35°, and angle of repose varying from 30° to 34°, dry ravel varies from 18 to 110 g, respectively. As the slope angle increases, however, the dry ravel mass held by the vegetation stems decreases rapidly in general and all the curves tend to converge.

3.1.3. Static friction angle

According to Equation (1), the angle of repose can only affect the amount of dry ravel held by vegetation stems. To understand the effects of static friction angle, the dry ravel production and deposition in watershed 0508 was calculated for different static friction angles. In this case, the fire-impact depth was taken as 0 cm and thus the production of dry ravel was only due to the plant stems. Both production and deposition of dry ravel increase with the increase of static friction angle. The portion of dry ravel deposition on the production is constant because the angle of repose has no effect on the transportation of dry ravel (Figure 13).

The angle of repose can be determined by the laboratory experiments or field investigation. For the dry soil of mountainous Southern California, the angle of repose is generally within the range of 29° to 34° according to a previous laboratory test [*Wohlgemuth*, 2004 unpublished data].

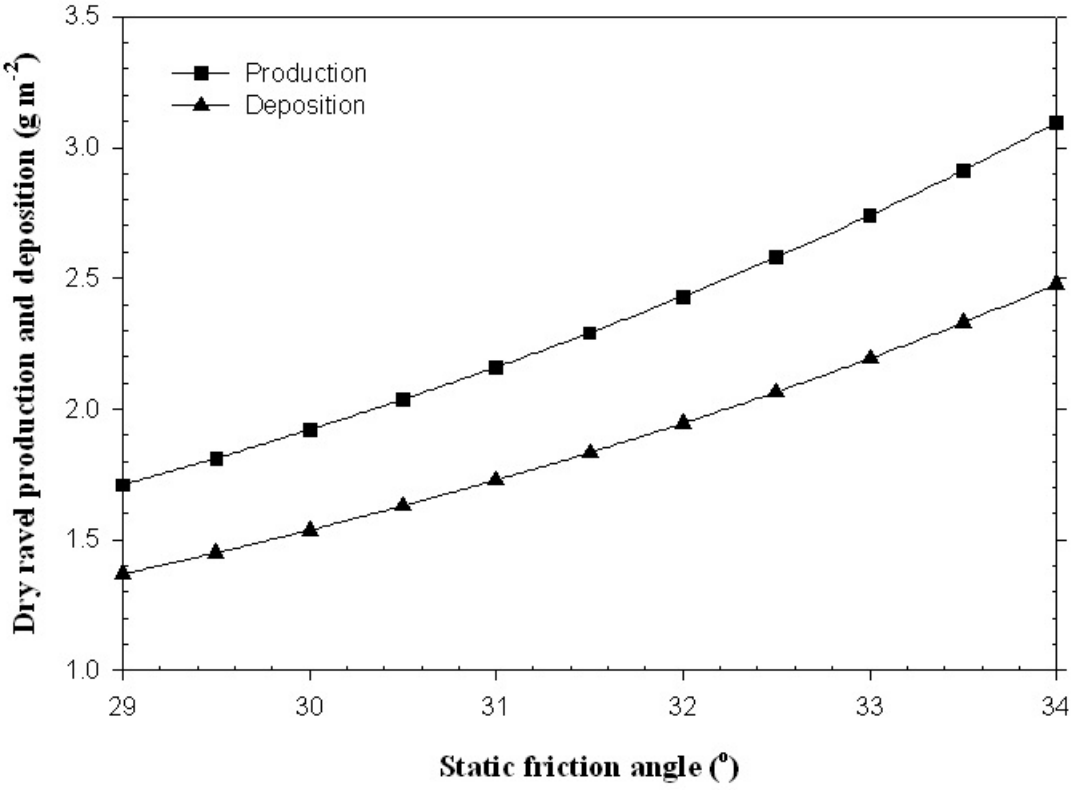

Figure 13. Model prediction of dry ravel production and deposition with static friction angle.

3.1.4. Fire-impact depth

The amount of dry ravel held by vegetation roots can be determined by the product of the fire-impact depth and the surface area while the amount of dry ravel held by vegetation stems is independent of the fire-impact depth. When the fire-impact depth is greater than 0.1 mm, the amount of dry ravel held by vegetation roots is much larger than the portion held by vegetation stems. Thus the source of dry ravel held by the vegetation stems can usually be neglected if the fire-impact depth is larger than 0.1 mm. Figure 14 shows an linear relationship between the dry ravel production and fire-impact depth. Because the fire-impact depth has no effect on the movement of dry ravel, the ratio of deposition to the production does not change with the fire-impact depth. And thus the dry ravel deposition increased linearly with the increase of the fire-impact depth.

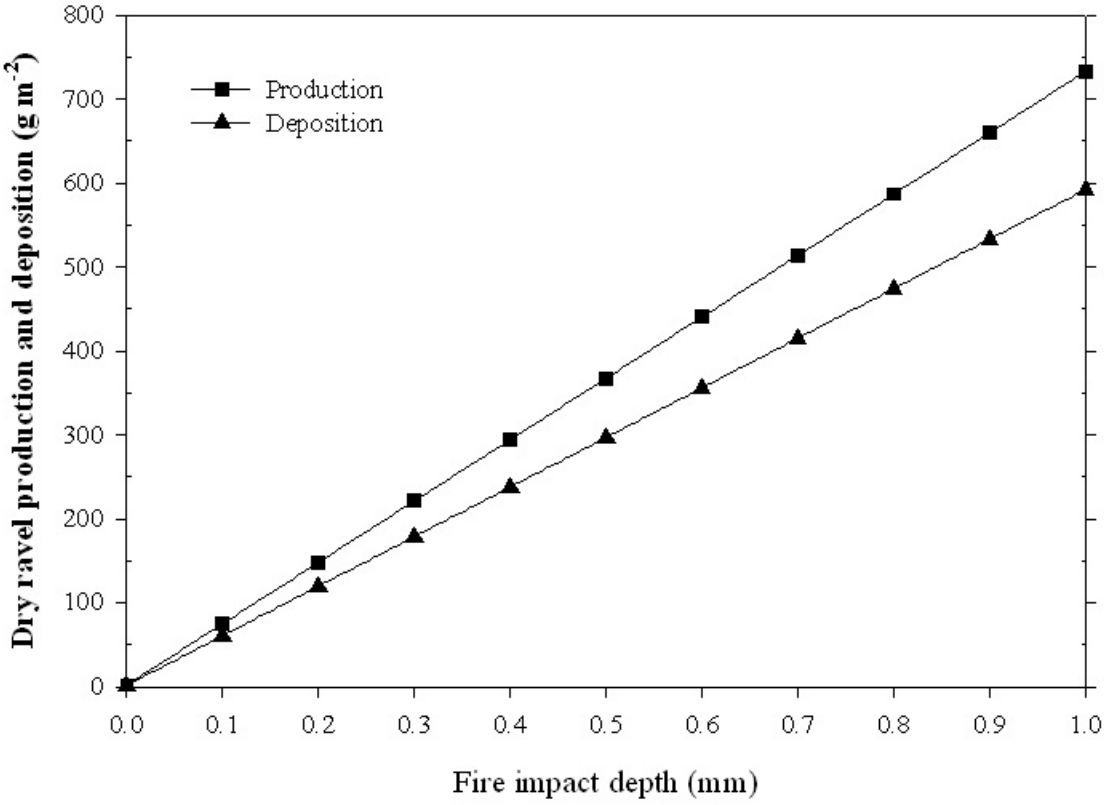

Figure 14. Model prediction of dry ravel production and deposition with fire impact depth.

3.1.5. Kinetic friction angle

The kinetic friction angle is one of the most important factors that affect the movement of dry ravel. The kinetic friction angle used in the mass-flux model is a modified coefficient from the kinetic friction angle of soil materials. It depends not only on the soil properties, (e.g., particle size, soil type, soil moisture), but also on the micro-topography in each cell, the vegetation density and size, the obstacles (e.g., rocks, sticks) along the path of dry ravel movement, and the dry ravel velocity. According to Figure 15, dry ravel production and deposition decreased with the increase of kinetic friction angle. The decrease of production was caused by the reduction of the number of source cells of dry ravel, because no dry ravel will be produced from a cell if the slope angle is less than the kinetic friction angle. Some cells will become non-source cells as the kinetic friction angle is increased and become larger than the slope angle of these cells.

The kinetic fiction angle affects the dry ravel deposition in two opposite ways. The decreasing dry ravel production reduces the amount of dry ravel which is the potential source of the deposition. When the kinetic friction angle is increased, the potential source of deposition decreased and consequently caused the reduction of dry ravel deposition. On the other hand, the increase of kinetic friction angle can lower the velocities of dry ravel, and leading to the reduction of the amount of off-site dry ravel, and consequently the increase of the deposition within the domain. Comparing the two effects of kinetic friction angle, dry ravel deposition is more sensitive to the effect of kinetic friction angle than is dry ravel production. Dry ravel deposition decreases with the increase of kinetic friction angle but the decrease rate is less than the decrease rate of dry ravel production (Figure 15). In this figure, the small fluctuations from a linear trend are the effects of the topography because the cell slope angles are randomly distributed within a range of 0° to 90° and there is a non-linear

distribution of cell slope angle within the domain.

The kinetic friction angle is difficult to determine due to the complexity of the influential factors. Further field investigation is needed to obtain an accurate value of this coefficient. Here, we use the average slope angle of the whole model domain to approximate the kinetic friction angle. This approximation is based on the fact that the kinetic friction angle can not be much different from the average slope angle of the whole model domain. Otherwise, the hillslope of the model domain is unstable resulting from a change of the topography.

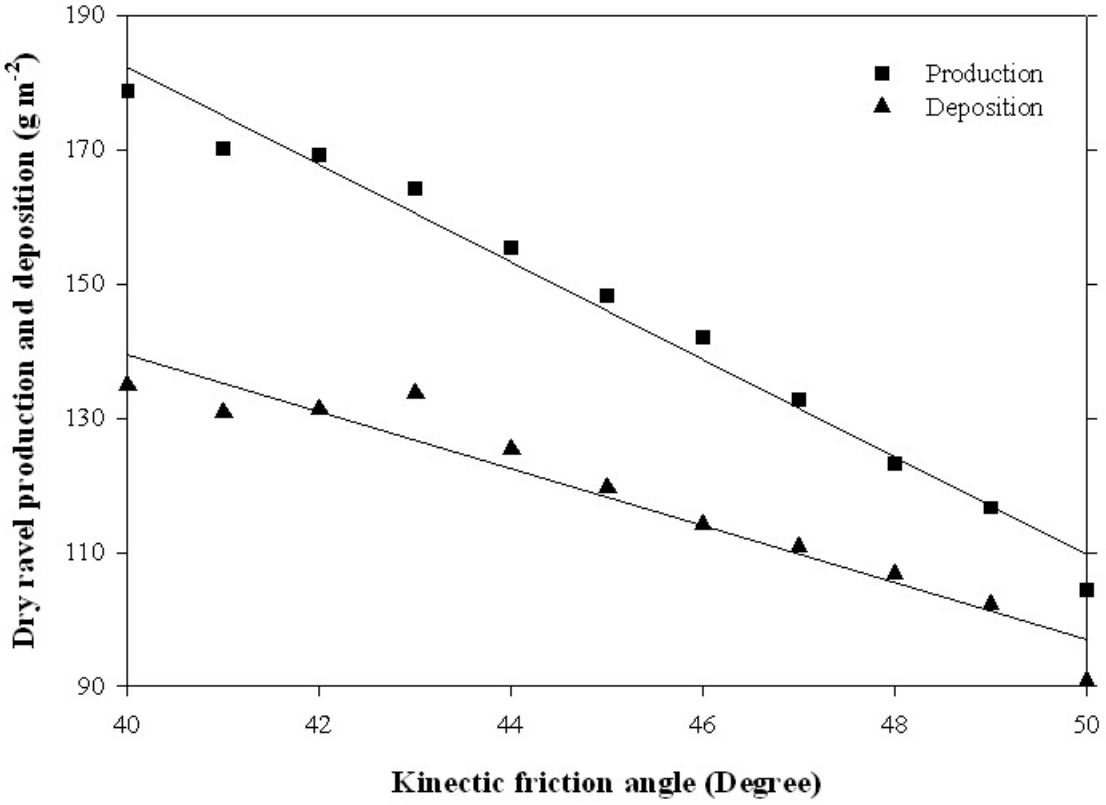

Figure 15. Model prediction of dry ravel production and deposition with kinetic friction angle.

3.1.6. Cell size

Figure 16 shows the impact of cell size selections on predicted dry ravel. The average production increased with the decrease of cell size. An approximate linear decrease relationship between the dry ravel production and deposition and cell size was obtained with a regression method. This linear decrease can be explained by the decrease of total surface area and the changes of slope angle within each cell. In the mass-flux model, the planar surface was used for each cell through the model domain. In reality, the cell surface is a curved surface and the area of the curve surface is larger than the area of the planar surface. Therefore, the model under-estimates the surface area of the model domain with the planar assumption. As the size of cell decreases, the model can more accurately simulate the curved surface leading to an increase of the surface area, and consequently resulting in the increase of both dry ravel production and deposition.

Additionally, the slope angle of each cell will change with cell size. Because the dry ravel will be produced only if the slope angle is larger than the kinetic friction angle, the more cells with a slope angle greater than kinetic friction angle will produce more dry ravel. A smaller cell size will cause an increase of the average slope angle, that means the number of cells with larger slope angle will increase. Therefore, the production of dry ravel increases as the cell size decreases.

Although the smaller cell size can predict the topography better than the larger cell size, a very small cell size (less than 1 m) is not suitable in application. Since we assumed that all parameters do not change within each cell and part of these parameters do not change through the whole domain, a small cell size will cause obvious conflict with the assumption (e.g. the vegetation density and size will be different for different cells if the cell size is close to the vegetation size). A cell size within 1 m to 10 m is recommended in the mass-flux model.

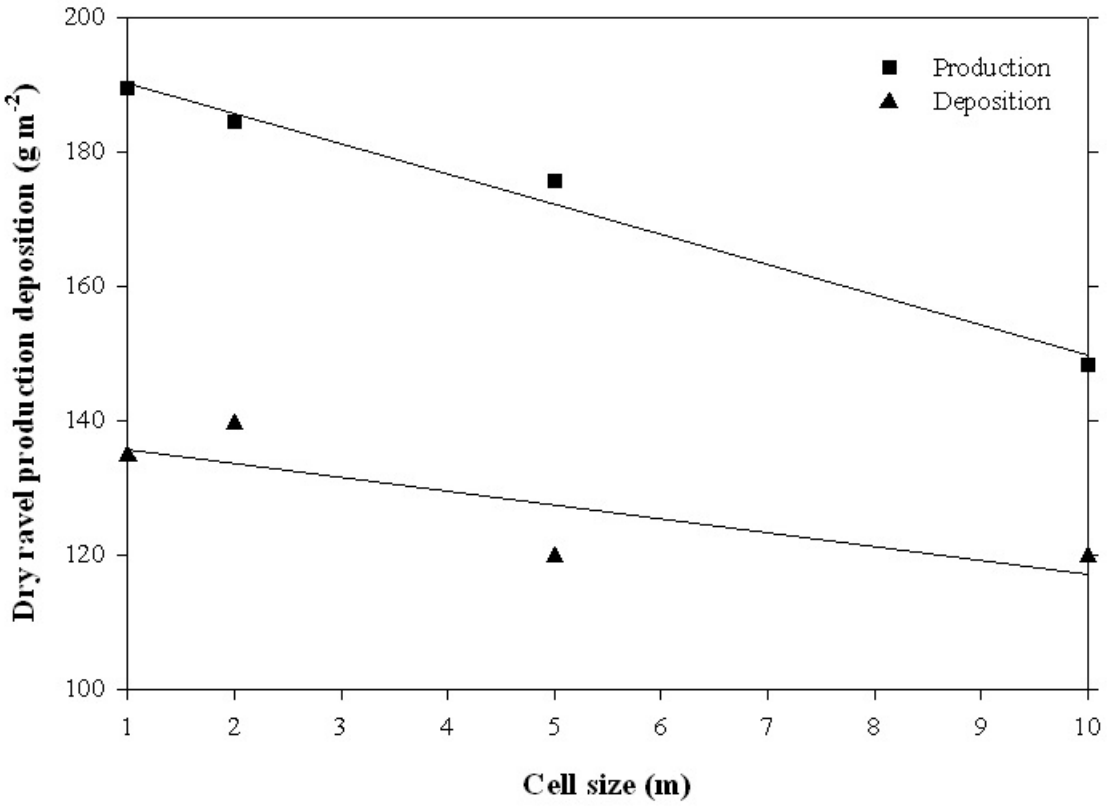

Figure 16. Model prediction of dry ravel production and deposition with model cell size

3.2. Model Evaluation for a Representative Setting

Figure 17 illustrates the paths of mass fluxes of dry ravel from 12 arbitrarily assigned source cells. Most of these source cells are located on steep slopes, with a few exceptions that are on relatively flat areas. For the latter sources, no mass flux was generated because the slope angle is much less than the kinetic friction angle of the soil. From the former sources, the mass fluxes flow along the steepest slope which is perpendicular to the contours and become dispersed over certain areas according to the topography. In the processes of dry ravel movement, the mass fluxes of dry ravel disperse more broadly on a convex slope, and tend to concentrate along a narrow path on a concave slope, and eventually deposit on flat areas or move out the boundary of the model domain.

Figure 18 shows the predicted results of dry ravel deposition on the representative setting. Most of the dry ravel deposits along the valley of the watershed or on flat areas. The area (cells) with high deposition are in topographic lows (i.e., valleys and stream channels). The area (cells) with very high deposition (spikes) occur in topographic lows that also have a large contribution.

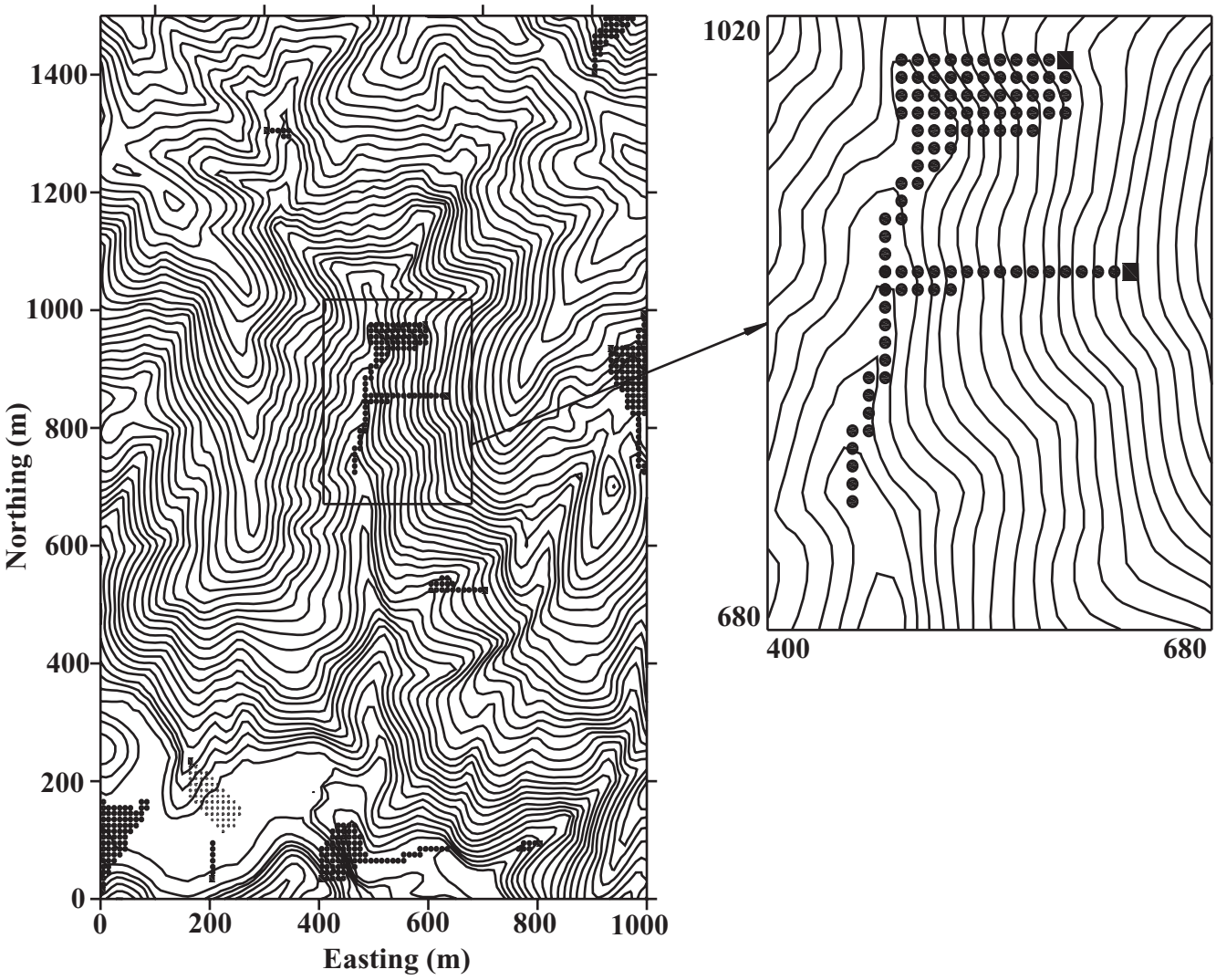

Figure 17. Movement of dry ravel. The squares are the source cell and the circles represent the path of dry ravel.

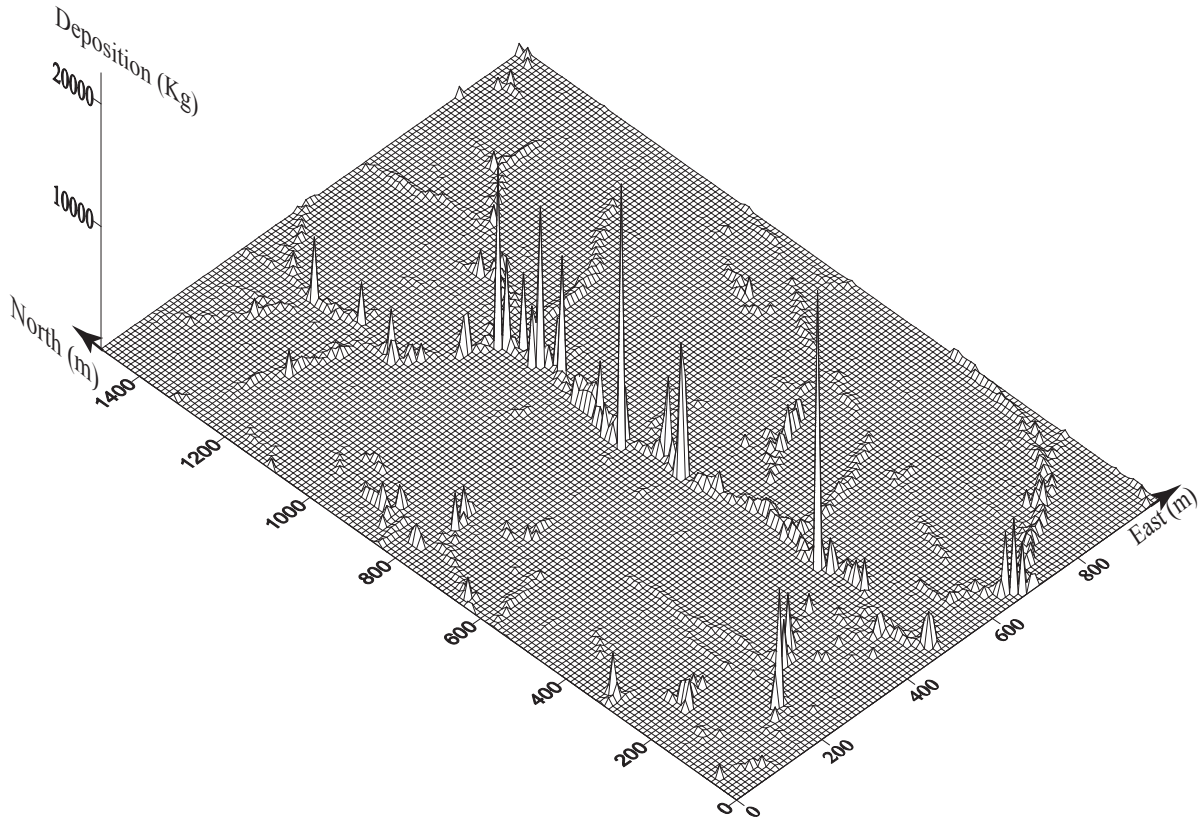

Figure 18. The deposition of dry ravel in representative setting for Figure 17.

3.3. Model Evaluation for the San Dimas Experimental Forest

The mass-flux model has been applied to the four small watersheds in the SDEF for a short-term prediction of both dry ravel production and deposition. The comparison between the model prediction of dry ravel and field observations at the four small watersheds over a short term (1 day) at the SDEF is shown in Table 4. All observed productions and most depositions were within the range predicted by the model.

Table 4. Comparison of model prediction and field observation

Nr.	Fire Impact Depth(mm)	Field Observation		Model Prediction[a]	
		Production (g/m^2)	Deposition(kg/m)	Production (g/m^2)	Deposition (kg/m)
0507	0-1	69	1.27	1-522	0.016-8.96
0508	0-1	100	39.0	1-516	0.010-5.80
0542	0-1	322	2.02	1-663	0.011-6.12
0560	0-1	131	2.26	1-791	0.016-9.28

[a]In model prediction, the vegetation density is 1 stem/m^2 and diameter of plant stem is 5cm, respectively.

The watershed 0507 had the smallest production among the four watersheds because the vegetation in this watershed mainly consists of the grass with sparse sage brush and shrubs. In addition, the fire severity was less in watershed 0507 than the other watersheds. With chaparral covering watershed 0508, the fire severity was higher, resulting in greater dry ravel production. The watershed 0542 had the highest production of dry ravel caused by high fire severity and steeper topography. The field data shows that the deposition of dry ravel along the valley had no obvious relationship with the production. This is because the deposition is more dependent on the topography rather than the production of dry ravel. The deposition in watershed 0508 is much greater than the

other watersheds because one of the collectors in the watershed has collected about 200 kg dry ravel deposition while the other collectors can only collect about a few hundred grams of dry ravel deposition. This collection limitation greatly affected the measurement of the average deposition for the whole watershed.

The model prediction shows a range of dry ravel production and deposition along the valley. Because the exact fire-impact depth was not available from the field observation, a range of the fire-impact depth from 0 to 1 mm was used for model prediction. When the fire-impact depth is set at 0 , the dry ravel production is about 1 g m^{-2} for all four watersheds and the deposition of dry ravel was also small. This is because only the source of dry ravel held by plant stems can contribute to the total amount of dry ravel, while the source of dry ravel held by plant roots is 0. As mentioned in an earlier section, the amount of dry ravel increased linearly with the increase of the fire-impact depth. When the fire-impact depth is set at 1 mm, both of the production and deposition increase greatly. For most watersheds, the amount of dry ravel deposition was consistent with the increase of the production with the exception of the watershed 0507. In this watershed, there was a higher deposition despite low production. This may be caused by the gentle topography of the watershed.

Table 5. Observed and predicted means and standard deviations (sample size 25) of dry ravel production (mass per unit area). There is no significant difference between the observed and predicted means at a significance level of 0.1 for all four watersheds.

Watershed	FID[a] (mm)	Means of Production (g/m^2)	
		Observed	Predicted
0507	1.1	167.7 (169.4)[b]	168.0 (80.9)
0508	1.2	234.9 (223.3)	235.0 (60.7)
0542	2.0	428.6 (616.2)	429.0 (180.5)
0560	1.0	172.2 (231.8)	172.3 (48.1)

[a]The fire-impact depth (FID) was manually calibrated for the dry ravel production at each watershed.
[b]Included in parentheses is the standard deviation.

Table 6. Observed and predicted means and standard deviations (sample size 50) of dry ravel deposition for individual collector measuring 30-cm wide.

Watershed	FID[a] (mm)	Means of Deposition (kg)		ANOVA test
		Observed	Predicted	
0507	1.1	0.25 (0.40)[b]	0.48 (0.83)	NS[c]
0508	1.2	6.09(28.50)	0.40 (0.55)	NS
0542	2.0	049 (0.71)	0.58 (1.15)	NS
0560	1.0	0.38 (0.83)	0.33 (0.40)	NS

[a]The fire-impact depth (FID) was calibrated for the dry ravel production as in Table 5.
[b]Included in parentheses is the standard deviation.
[c]The effect of the factor "method" (observed and predicted) is not significant at a significance level 0.05.

The statistical comparisons between the observed and predicted dry ravel production on hillslopes and deposition along the valley are shown in Table 5 and 6. Note that, the fire-impact depth for each watershed was first calibrated for dry ravel production on hillslopes, and this fire-impact depth indeed varied among the watersheds. Observed and predicted dry ravel deposition along the valley at each watershed were then compared with the ANOVA test indicating no significant difference at a significance level of 0.05. Although the observed mean of dry ravel

deposition in watershed 0508 is much higher than the predicted value, the two means do not differ statistically due to the large standard deviation in the field observations. Among the fifty collectors from watershed 0508, one collector (#725) collected 200 kg dry ravel over a 18-day time period while all other collectors collected no more than 1 kg of dry ravel. Bare soil occurred on the hillslope terracing immediately above the collector #725 in the 1960's. Some revegetation took place with time; however, the patchy bare soil appears to have always been a substantial source of dry ravel with collector #725 consistently receiving larger amount of soil materials compared to other collectors, even during the pre-burn time. It is therefore presumed that the wildfire further augmented the instability of the bare soil, leading to an extreme case of post-fire dry ravel. Given the fact that the model was able to quantify the amount of dry ravel which fits with the case of the field observations with a variable, fire-impact depth.

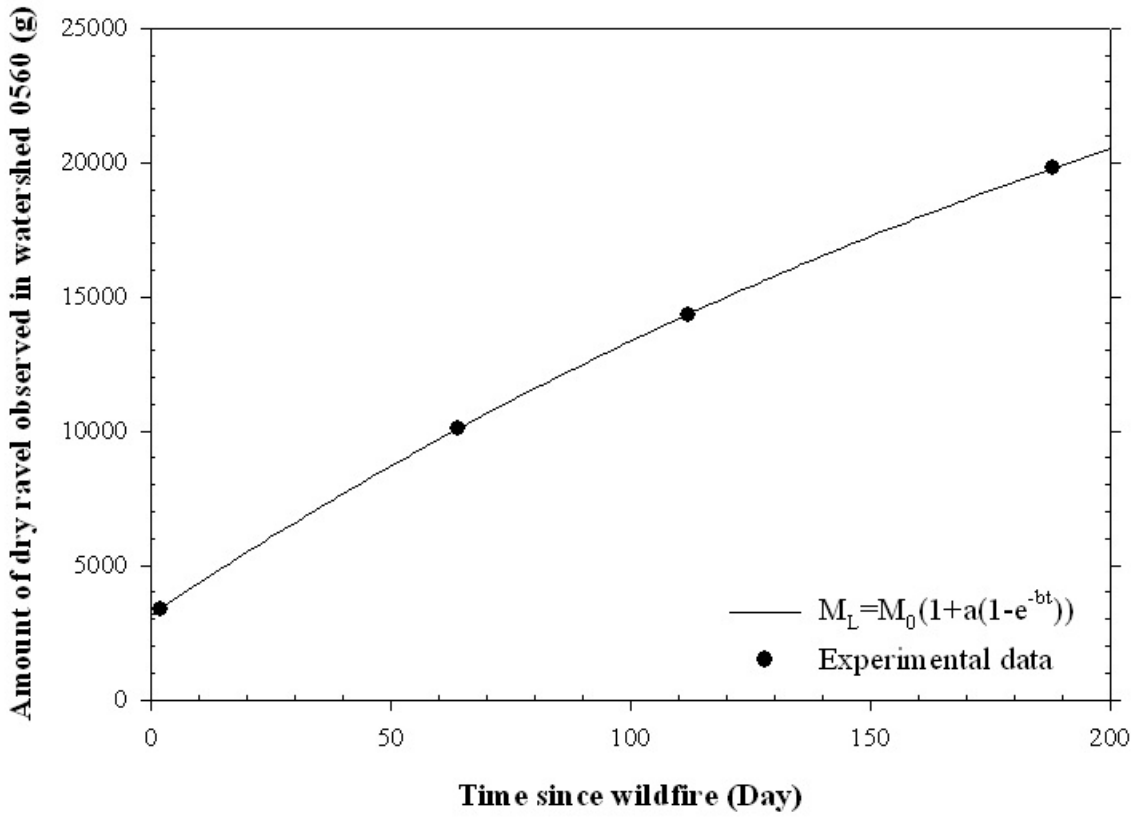

Figure 19 Experimental results and model simulation of long term effects of dry ravel.

3.4. Long-term dry ravel prediction

Figure 19 shows the prediction curve (Equation (21)) for long term dry ravel, which is based on the field observed dry ravel for the small watershed 0560 at the SDEF over 200 days after the fire. The curve is drawn according to Equation (21), and the curve parameters are M_0 =3,151 g, and α =34,420, and β =0.0035.

3.5. Future Efforts

From the conceptual model assessment and application to the SDEF, continuous development and improvement of the model is needed. First, more field tests and data are required to further assess the model's adequacy and reliability. This is particularly important for long-term dry ravel prediction. For the SDEF, we only had field observation data for certain times after the fire. More detailed temporal data with shorter time intervals (e.g., on a daily basis) are required to accurately determine the coefficients of the empirical equation for long-term dry ravel production. Additionally, the interrelationship between these coefficients and the environmental factors (e.g., wind, earthquake, and animal movement) should be established through field investigation..

Second, additional field efforts are needed for evaluating the kinetic friction coefficient. During the dry ravel transport, the friction force is not only controlled by the soil properties but also affected by the micro-topography, vegetation characteristics, and other environmental factors. Field and laboratory experiments may be designed to establish the relationships between the kinetic friction coefficient and these factors.

Third, the interrelationship between the extent of the fire-impact depth and the fire severity, vegetation characteristics, and soil properties should also be better understood in order to improve

the adequacy of the model parameters and thus the model predictions. The part of dry ravel held by plant roots is typically much greater than the part held by the plant stems. Therefore, a proper estimate of the depth of burned roots will directly improve adequacy of the model results.

Fourth, additional field observations are needed to calibrate the coefficients α and β in Equation (21). The coefficients, α and β, are a function of the topography, the fire severity, vegetation properties, soil properties, and other environmental factors. Therefore, additional field investigations and modeling efforts are needed to predict the influence of the two coefficients according to the natural situation.

CHAPTER FOUR

SUMMARY

This paper describes the concepts and methods of a physical mass-flux model, for simulating the processes of dry ravel and predicting the amount of source of dry ravel. In principle, this mass-flux model applies the classic laws of mechanics to predict source area, movement paths, deposition areas, and off-site transport of dry ravel produced after wildfires.

The short-term dry ravel process is computed with theoretical calculations, and the long-term effects are described based on empirical relationships. The assessment of the model performance was conducted by applying the mass-flux model to a conceptual setting as well as the San Dimas Experimental Forest (SDEF) in southern California where wildfires and dry ravel frequently occur. Compared to the experimental results, the model performed adequately in predicting the source location, movement path, and deposition areas. On the other hand, the accuracy of predicting the mass of dry ravel production and deposition, and the long-term ravel events of dry ravel is dependent on the specific coefficients.

While the mass-flux model shows promise as an effective tool to predict dry ravel, the model has certain limitations in practical application. These limitations are caused by the assumptions necessary for simplifying the calculations, lack of information on spatially varying input data, the unclear relationship of fire-impact depth and the fire severity, and limit knowledge of long-term effect of dry ravel. These limitation need to be investigated in future studies.

REFERENCES

Anderson, H. W., G. B. Coleman, and P. J. Zinke (1959), Summer slides and winter scour—wet–dry erosion in southern California mountains, *Gen. Tech. Rep.,* PSW–36, Berkeley, CA.

Bardet, J. P. and Q. Huang (1992), Numerical modeling of micropolar effects in idealized granular materials, *Mech. Granular Matr. Powder Sys., 37*, 85–92.

Bardet, J. P. and J. Proubet (1991), A numerical investigation of the structure of persistent shear bands in granular media, *Geotechnique, 41*, 599–613.

Cundall, P. A. and O. D. L. Strack (1979), A discrete numerical model for granular assemblies, *Geotechnique*, *29*, 47–65.

Gabet, E. J. (2003), Sediment transport by dry ravel, *J. Geophys. Res., 108*(B1), 2049, doi:10.1029/2001JB001686.

Goodman, L. E. (1962), Contact stress analysis of normally loaded rough spheres, *J. Appl. Mech.*, *30*, 515–522.

Huhlhaus, H. B. and I. Vardoulakis (1987), The thickness of shear bands in granular materials, *Geotechnique, 37*, 271–283.

Krammes, J. L. (1960), Erosion from mountain side slopes after fire in Southern California, *Res. Note, No. 171, 8p,* Pacific Southwest Forest and Range Experiment Station, U. S. Dep. Agric., Forest Service, Berkeley, CA.

Kalker, J. J. (1970), Transient phenomena in two elastic cylinders rolling over each other with dry friction, *J. Appl. Mech.*, *11*, 677–688.

Rice, R. M. (1982), Sedimentation in the chaparral: How do you handle unusual events?, in *Sediment Budget and Routing in Forested Drainage Basins*, edited by F. J. Wsanson et al., pp. 39–49, Forest Service, U.S. Dep. Agric., Washington, D.C.

SAS Institute Inc. 2004. Base SAS 9.1 Procedures Guide, Volumes 1,2, and 3. Cary, NC.

Wells, W. G. and N. R. Palmer. (1979), Role of vegetation in sedimentation processes, in *Sediment Management for Southern California Moutains, Coastal Plains and Shoreline*, EQL Rep. No. 16, Environmental Quality Lab., Calif. Inst. of Tech. Pasadena, CA, Appendix D.

Wells, W. G. (1981), Some effects of brushfires on erosion processes in coastal southern California, In: *Erosion and Sediment Transport in Pacific Rim Steeplands*. Christchurch, New Zealand: IASH Publication No. 132:305–342.

Wells, W. G. (1985), The influence of fire on erosion rates in California chaparral, in *Proc. Chaparral Ecosystem Res. Conf.*, edited by J. J. Devries, *Rep. 62, Calif.* Water Resour. Cent., Santa Barbara, CA.

APPENDIX

A. Mass-flux Model Code

```
// The file is a header file which indicate the c++ library files to be used
also define
// the constant numbers.

// Set the file to be compile only once
#pragma once

//Indicate the files of the standard c++ library used here
#include <iostream>
#include <fstream>
#include <cstring>
#include <cmath>

//Using the std namespace
using namespace std;

// Define constant pi
const double PI=3.14159265358979323846;
```

```
#pragma once

//Include the header file "headerfile.h" to be used in this file
#include "headerfile.h"

//This part codes declare the class "node"
//the class "node" represents the node of the cell
class node
{
public:
      node();                                    //declare the public function
node()
                                                 //which is used to initiate
the class "node"
      ~node();                             //declare the public function
~node()

//declare three input functions
      void setx(double xvalue);     //setx function set the x to be xvalue
      void sety(double yvalue);     //sety function set the y to be yvalue
      void setz(double zvalue);     //setz function set the z to be zvalue

//declare three output function
      double getx();                       //getx return the x value
      double gety();                       //gety return the y value
      double getz();                       //getz return the z value

private:
// declare the parameters of node x, y, and z which are the coordination of
the node
      double x;
      double y;
      double z;
};
```

```
#pragma once
#include "headerfile.h"

//declare the class "event" which represent the mass flux through a cell
boundary
class event
{
public:
      event();                              //initial function
      ~event();                             //terminal function

// declare the function of setting the total mass of a mass flux
      void settotalmass(double);

// declare the function of setting the starting time of a mass flux
      void setstarttime(double);

// declare the function of setting the x-direction velocity of a mass flux
      void setxvelocity(double);

// declare the function of setting the y-direction velocity of a mass flux
      void setyvelocity(double);

// declare the function of output the total mass of a mass flux
      double gettotalmass();

// declare the function of output the starting time of a mass flux
      double getstarttime();

// declare the function of output the x-velocity of a mass flux
      double getxvelocity();

// declare the function of output the y-velocity of a mass flux
      double getyvelocity();

private:
      double totalmass;             // declare the variable of total mass of
a mass flux
      double starttime;             // declare the variable of start time of
a mass flux
      double xvelocity;             // declare the variable of x-velocity of
a mass flux
      double yvelocity;             // declare the variable of y-velocity of
a mass flux
};
```

```
#pragma once
#include "headerfile.h"
#include "node.h"
#include "event.h"

// Declare the class "element" which represents the cell of model domain.
// The "element" class mainly has two functions "eventproduct" and
"eventoccur".
// The function "eventproduct" calculate the total mass and velocities of
each mass flux
// flowing out the original cell;
// The function "eventoccur" calculate the total mass and veocities of each
mass flux
// flowing out any cell depending on the inflow mass flux and topography
class element
{
public:
//declare the initial function
      element(int elementno,double,double,double,double,int Size);
//declare the terminal function
      ~element();

// declare function "eventproduct" which is used to calculate total mass and
velocities
// of each mass flux flowing out the original cell, input parameters are the
vegetation
// size/diameter, vegetation density, the fire-impact depth, the friction
angle, and the
// kinetic friction angle
      void eventproduct(double vegetationsize, double vegetationdensity,
            double FireImpactDepth,double FrictionAngle,double
KineticFrictionAngle);

// declare the function "setmassflowin" to set the mass and velocities of
each mass flux
// flowing in the cell
      void setmassflowin(double mass[4],double velocity[4][2]);

// declare the function "eventoccur" which is used to calculate total mass
and veocities
// of each mass flux flowing out any cell depending on inflow mass flux and
topography,
// the input parameters are the minimun mass and kinetic friction angle
      void eventoccur(double, double KineticFrictionAngle);

//declare the array of inflow mass flux with type "event"
      event massflowin[4];

//declare the array of outflow mass flux with type "event"
      event massflowout[4];

private:
      int Size;                           //declare the cell size
      int elementno;                      //declare the cell number
      node elementnode[4];          //declare four node for each cell
    double slope[2];                //declare slope of cell in x and y
direction };
```

```
// This part of code define the functions of class "node"
#include "node.h"

node::node(){}                              //define the initial function
node::~node(){}                             //define the terminal function

//define the function of setting x value
void node::setx(double xvalue)      {x=xvalue;}

//define the function of setting y value
void node::sety(double yvalue)      {y=yvalue;}

//define the function of setting z value
void node::setz(double zvalue)      {z=zvalue;}

//define the function of outputting x value
double node::getx()     {return x;}

//define the function of outputting y value
double node::gety()     {return y;}

//define the function of outputting z value
double node::getz()     {return z;}
```

```
// This part of code is used to define the functions of class "event"
#include "event.h"

// define the initial function which set total mass, start time, x-velocity,
y-velocity
// of mass flux to be zero
event::event()    {     totalmass=0;      starttime=0;      xvelocity=0;
      yvelocity=0;      }

//define the terminal function
event::~event()   {     }

//define the function to set the total mass value
void event::settotalmass(double value)    {     totalmass=value;  }

//define the function to set the start time value
void event::setstarttime(double value)    {     starttime=value;  }

//define the function to set the x-velocity
void event::setxvelocity(double vx)       {     xvelocity=vx;           }

//define the function to set the y-velocity
void event::setyvelocity(double vy)       {     yvelocity=vy;           }

//define the function to output the total mass
double event::gettotalmass()              {     return totalmass; }

//define the function to output the start time
double event::getstarttime()              {     return starttime; }

//define the function to output the x-velocity
double event::getxvelocity()              {     return xvelocity; }

//define the function to output the y-velocity
double event::getyvelocity()              {     return yvelocity; }
```

```
// This part of code is use to define the functions of class "element"

// include the file element.h
#include "element.h"

//define the initial function setting the cell number, the elevation of cell
node,
//and the cell size
element::element(int number,double elevation0,double elevation1, double
elevation2,double elevation3,int size)
{
      this->elementno=number;                   //set the cell number as
"number"
      this->Size=size;                          //set the cell size
      elementnode[0].setz(elevation0);    //set the elevation of upper-left
node of cell
      elementnode[1].setz(elevation1);    //set the elevation of upper-right
node of cell
      elementnode[2].setz(elevation2);    //set the elevation of lower-right
node of cell
      elementnode[3].setz(elevation3);    //set the elevation of lower-left
node of cell

//set all the variables of four mass flux to be 0
      for(int i=0;i<4;i++)
      {
            massflowout[i].setstarttime(0);
            massflowout[i].settotalmass(0);
            massflowout[i].setxvelocity(0);
            massflowout[i].setyvelocity(0);

            massflowin[i].setstarttime(0);
            massflowin[i].settotalmass(0);
            massflowin[i].setxvelocity(0);
            massflowin[i].setyvelocity(0);
      }

//calculate the slope of a cell in x and y direction (slope[0] and slope[1]
respectively)
      if(Size!=0)
      {
        slope[1]=((elementnode[0].getz()+elementnode[1].getz())/2-
(elementnode[2].getz()+elementnode[3].getz())/2)/Size;
        slope[0]=((elementnode[0].getz()+elementnode[2].getz())/2-
(elementnode[1].getz()+elementnode[3].getz())/2)/Size;
      }
      else
      {
            cout << "Error!!! Size is 0!!!";
      }
}

// Define the terminal function
element::~element()     {     }

//define the function calculating the mass flux produced from an original
cell
```

```
void element::eventproduct(double vegetationsize, double vegetationdensity,
double FireImpactDepth,double FrictionAngle,double KineticFrictionAngle)
{
//declare and initialize the variables and arrayes
      double xsurfaceangle=fabs(atan(slope[0]));      // declare slope angle
in x-direction
      double ysurfaceangle=fabs(atan(slope[1]));      // declare slope angle
in y-direction
      double slopeangle=atan(sqrt(slope[0]*slope[0]+slope[1]*slope[1]));
      // declare slope angle

      double outputtime1=0;         //declare the shorter time of mass flux
flowing out cell
      double outputtime2=0;         //declare the longer time of mass flux
flowing out cell

      double outputmass=0;          // outputmass is the total mass produced
from this cell

      double acceleration=0;        // declare the acceleration
      double xacceleration=0;       // declare the acceleration component in
x-direction
      double yacceleration=0;       // declare the acceleration component in
y-direction

      double validheight=vegetationsize*tan(FrictionAngle)/sqrt(2);
      // declare the valid height to which soil particles accumulate just
behind vegetation

      double divergence=0;          // declare the divergence of mass flux

      double length;                      // length is height of base of the
soil particles pile

      // for the case that slope angle is not same as the friction angle
      if (slopeangle!=FrictionAngle)
      {
            length=validheight*sin(PI/2+FrictionAngle)/sin(slopeangle-
FrictionAngle);
      }
      // for the case that slope angle equal to the friction angle
      else
      {
            cout << "Error!!! Slope angle is equal to friction angle";
      }

      double maxlength;             // declare the upper limit of length

      // if there is vegetation on the cell, calculate the maximun length
between the two vegetation
      if (vegetationdensity!=0)
      {
            maxlength=1/sqrt(vegetationdensity);
      }
      // if there is no vegetation in the cell
      else
      {
```

```
            maxlength=1;
      }

// Calculate the total mass of dry ravel produced from the cell
      if(length<maxlength && slopeangle!=(PI/2))

outputmass=1000*Size*Size*(FireImpactDepth/cos(slopeangle)+vegetationdensity*
vegetationsize*validheight*length*sin(PI/2-slopeangle)/6);
      else if(slopeangle!=PI/2)

      outputmass=1000*Size*Size*(FireImpactDepth/cos(slopeangle)+vegetationde
nsity*vegetationsize*validheight*maxlength*sin(PI/2-slopeangle)/6);
      else
      {
            outputmass=0;
            cout << "Error!!! Slope angle is 90";
      }


//    calculate the surface acceleration within a cell
      acceleration=9.8*(sin(slopeangle)-
tan(KineticFrictionAngle)*cos(slopeangle));
      if(acceleration<0) acceleration=0;

//    calculate the x- and y- acceleration
      if (sqrt(slope[0]*slope[0]+slope[1]*slope[1])!=0)
      {

xacceleration=acceleration*fabs(slope[1]/sqrt(slope[0]*slope[0]+slope[1]*slop
e[1]));

yacceleration=acceleration*fabs(slope[0]/sqrt(slope[0]*slope[0]+slope[1]*slop
e[1]));
      }
      else
      {           xacceleration=0;  yacceleration=0;  }

// if the acceleration is zero, there is no mass flux produced
      if(acceleration==0)     {outputmass=0;}

// if the acceleration is not zero, calculate the output mass from a cell
      else
      {

// this sentence is use to judge which mass flux (from side or bottom
boundary) is faster
        if(xacceleration*cos(xsurfaceangle) >=
yacceleration*cos(ysurfaceangle))
                  // here, mass flux in x-direction is faster
            {
//                calculate the time of mass flux flowing out in x-direction
                  if(xacceleration!=0)
outputtime1=sqrt(Size)/sqrt(cos(xsurfaceangle)*xacceleration);

//                calculate the time of mass flux flowing out in y-direction
                  if(yacceleration!=0)
outputtime2=sqrt(Size)/sqrt(cos(ysurfaceangle)*yacceleration);
```

```
//                calculate the divergence of mass flux (the "Dx" in the
paper)

      divergence=fabs(cos(ysurfaceangle)*yacceleration*outputtime1*outputtime
1/(2*Size));

//                calculate the output mass and velocity
                  if(slope[0]>=0 && slope[1]>=0)      // judge the slope
direction, here, the slope
                                                                        //
directions is positive x- y- direction
                  {

                        // set the mass flux flowing through the boundary "2"
                        massflowout[2].setstarttime(outputtime1); //set the
starting time of the mass flux
                        massflowout[2].settotalmass(outputmass*(1-
divergence));

            //set the total mass of the mass flux

      massflowout[2].setxvelocity(fabs(xacceleration*outputtime1)>0.1? 0.1 :
xacceleration*outputtime1);

            //set the x-direction velocity

      massflowout[2].setyvelocity(fabs(0.01*yacceleration*outputtime1)>0.1?
0.1 : 0.01*yacceleration*outputtime1);

            // set the y-direction velocity

                        // set the mass flux flowing through the boundary "3"
                        massflowout[3].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[3].settotalmass(outputmass*divergence);//set the total mass
of the mass flux

      massflowout[3].setxvelocity(fabs(0.01*xacceleration*outputtime2)>0.1?
0.1 : 0.01*xacceleration*outputtime2);//set the x-direction velocity

      massflowout[3].setyvelocity(fabs(yacceleration*outputtime2)>0.1? 0.1 :
yacceleration*outputtime2);// set the y-direction velocity

                  }
                  else if(slope[0]>=0 && slope[1]<0)  //x-slope is on the
positive x-direction
                                                                        //y-
slope is on the negative y-direction
                  {
                        massflowout[2].setstarttime(outputtime1);//set the
starting time of the mass flux
                        massflowout[2].settotalmass(outputmass*(1-
divergence));//set the total mass of the mass flux
```

```
      massflowout[2].setxvelocity(fabs(xacceleration*outputtime1)>0.1? 0.1 :
xacceleration*outputtime1);//set the x-direction velocity
                        massflowout[2].setyvelocity(fabs(-
0.01*yacceleration*outputtime1)>0.1? 0.1 : -
0.01*yacceleration*outputtime1);// set the y-direction velocity


                        massflowout[0].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[0].settotalmass(outputmass*divergence);//set the total mass
of the mass flux

      massflowout[0].setxvelocity(fabs(0.01*xacceleration*outputtime2)>0.1?
0.1 : 0.01*xacceleration*outputtime2);//set the x-direction velocity
                        massflowout[0].setyvelocity(fabs(-
yacceleration*outputtime2)>0.1? 0.1 : -yacceleration*outputtime2);// set the
y-direction velocity

                  }
                  else if(slope[0]<0 && slope[1]>=0)  //x-slope is on the
negative x-direction
                                                                        //y-
slope is on the positive y-direction
                  {
                        massflowout[1].setstarttime(outputtime1);//set the
starting time of the mass flux
                        massflowout[1].settotalmass(outputmass*(1-
divergence));//set the total mass of the mass flux
                        massflowout[1].setxvelocity(fabs(-
xacceleration*outputtime1)>0.1? 0.1 : -xacceleration*outputtime1);//set the
x-direction velocity

      massflowout[1].setyvelocity(fabs(0.01*yacceleration*outputtime1)>0.1?
0.1 : 0.01*yacceleration*outputtime1);// set the y-direction velocity

                        massflowout[3].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[3].settotalmass(outputmass*divergence);//set the total mass
of the mass flux
                        massflowout[3].setxvelocity(fabs(-
0.01*xacceleration*outputtime2)>0.1? 0.1 : -
0.01*xacceleration*outputtime2);//set the x-direction velocity

      massflowout[3].setyvelocity(fabs(yacceleration*outputtime2)>0.1? 0.1 :
yacceleration*outputtime2);// set the y-direction velocity
                  }
                  else                                          //x-slope
is on the negative x-direction
                                                                        //y-
slope is on the negative y-direction
                  {
                        massflowout[1].setstarttime(outputtime1);//set the
starting time of the mass flux
```

```
                        massflowout[1].settotalmass(outputmass*(1-
divergence));//set the total mass of the mass flux
                        massflowout[1].setxvelocity(fabs(-
xacceleration*outputtime1)>0.1? 0.1 : -xacceleration*outputtime1);//set the
x-direction velocity
                        massflowout[1].setyvelocity(fabs(-
0.01*yacceleration*outputtime1)>0.1? 0.1 : -
0.01*yacceleration*outputtime1);// set the y-direction velocity

                        massflowout[0].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[0].settotalmass(outputmass*divergence);//set the total mass
of the mass flux
                        massflowout[0].setxvelocity(fabs(-
0.01*xacceleration*outputtime2)>0.1? 0.1 : -
0.01*xacceleration*outputtime2);//set the x-direction velocity
                        massflowout[0].setyvelocity(fabs(-
yacceleration*outputtime2)>0.1? 0.1 : -yacceleration*outputtime2);// set the
y-direction velocity
                  }
            }
            else                    // here, mass flux in y-direction is
faster
            {
                  if(yacceleration!=0)
outputtime1=sqrt(Size)/sqrt(cos(ysurfaceangle)*yacceleration);
                  if(xacceleration!=0)
outputtime2=sqrt(Size)/sqrt(cos(xsurfaceangle)*xacceleration);


      divergence=fabs(cos(xsurfaceangle)*xacceleration*outputtime1*outputtime
1/(2*Size));

                  if(slope[0]>=0 && slope[1]>=0)
                  {
                        massflowout[3].setstarttime(outputtime1);//set the
starting time of the mass flux
                        massflowout[3].settotalmass(outputmass*(1-
divergence));//set the total mass of the mass flux

      massflowout[3].setxvelocity(fabs(0.01*xacceleration*outputtime1)>0.1?
0.1 : 0.01*xacceleration*outputtime1);//set the x-direction velocity

      massflowout[3].setyvelocity(fabs(yacceleration*outputtime1)>0.1? 0.1 :
yacceleration*outputtime1);// set the y-direction velocity

                        massflowout[2].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[2].settotalmass(outputmass*divergence);//set the total mass
of the mass flux

      massflowout[2].setxvelocity(fabs(xacceleration*outputtime2)>0.1? 0.1 :
xacceleration*outputtime2);//set the x-direction velocity
```

```
      massflowout[2].setyvelocity(fabs(0.01*yacceleration*outputtime2)>0.1?
0.1 : 0.01*yacceleration*outputtime2);// set the y-direction velocity
                  }
                  else if(slope[0]>=0 && slope[1]<0)
                  {
                        massflowout[0].setstarttime(outputtime1);//set the
starting time of the mass flux
                        massflowout[0].settotalmass(outputmass*(1-
divergence));//set the total mass of the mass flux

      massflowout[0].setxvelocity(fabs(0.01*xacceleration*outputtime1)>0.1?
0.1 : 0.01*xacceleration*outputtime1);//set the x-direction velocity
                        massflowout[0].setyvelocity(fabs(-
yacceleration*outputtime1)>0.1? 0.1 : -yacceleration*outputtime1);// set the
y-direction velocity

                        massflowout[2].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[2].settotalmass(outputmass*divergence);//set the total mass
of the mass flux

      massflowout[2].setxvelocity(fabs(xacceleration*outputtime2)>0.1? 0.1 :
xacceleration*outputtime2);//set the x-direction velocity
                        massflowout[2].setyvelocity(fabs(-
0.01*yacceleration*outputtime2)>0.1? 0.1 : -
0.01*yacceleration*outputtime2);// set the y-direction velocity
                  }
                  else if(slope[0]<0 && slope[1]>=0)
                  {
                        massflowout[3].setstarttime(outputtime1);//set the
starting time of the mass flux
                        massflowout[3].settotalmass(outputmass*(1-
divergence));//set the total mass of the mass flux
                        massflowout[3].setxvelocity(fabs(-
0.01*xacceleration*outputtime1)>0.1? 0.1 : -
0.01*xacceleration*outputtime1);//set the x-direction velocity

      massflowout[3].setyvelocity(fabs(yacceleration*outputtime1)>0.1? 0.1 :
yacceleration*outputtime1);// set the y-direction velocity

                        massflowout[1].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[1].settotalmass(outputmass*divergence);//set the total mass
of the mass flux
                        massflowout[1].setxvelocity(fabs(-
xacceleration*outputtime2)>0.1? 0.1 : -xacceleration*outputtime2);//set the
x-direction velocity

      massflowout[1].setyvelocity(fabs(0.01*yacceleration*outputtime2)>0.1?
0.1 : 0.01*yacceleration*outputtime2);// set the y-direction velocity
                  }
                  else
                  {
```

```
                        massflowout[0].setstarttime(outputtime1);//set the
starting time of the mass flux
                        massflowout[0].settotalmass(outputmass*(1-
divergence));//set the total mass of the mass flux
                        massflowout[0].setxvelocity(fabs(-
0.01*xacceleration*outputtime1)>0.1? 0.1 : -
0.01*xacceleration*outputtime1);//set the x-direction velocity
                        massflowout[0].setyvelocity(fabs(-
yacceleration*outputtime1)>0.1? 0.1 : -yacceleration*outputtime1);// set the
y-direction velocity

                        massflowout[1].setstarttime(outputtime2);//set the
starting time of the mass flux

      massflowout[1].settotalmass(outputmass*divergence);//set the total mass
of the mass flux
                        massflowout[1].setxvelocity(fabs(-
xacceleration*outputtime2)>0.1? 0.1 : -xacceleration*outputtime2);//set the
x-direction velocity
                        massflowout[1].setyvelocity(fabs(-
0.01*yacceleration*outputtime2)>0.1? 0.1 : -
0.01*yacceleration*outputtime2);// set the y-direction velocity
                    }
              }
      }
}

//    define the mass flux changing when it flow through a cell
void element::eventoccur(double MinMassOut,double KineticFrictionAngle)
{
      int noutputside=0;      //declare the boundary of outflow mass flux in
normal direction
      int soutputside=0;      //declare the boundary of outflow mass flux in
shear direction
                                    // 0-top boundary 3-buttom boundary 1-
left side 2-right side

      double ssurfaceangle=0; //declare the slope angle in normal direction
      double nsurfaceangle=0; //declare the slope angle in shear direction
      double slopeangle=0;    //declare the slope angle of a cell

      double noutputtime=0;   //declare time of mass flux flowing out in
normal direction
      double soutputtime=0;   //declare time of mass flux flowing out in
shear direction

      double nstoptime=0;     //time in which mass flux stop within cell in
normal direction
      double sstoptime=0;     //time in which mass flux stop within cell in
shear direction

      double ns=Size;         //size of cell in normal direction
      double ss=Size;         //size of cell in shear direction

      double nslope=0;  //slope in normal direction
      double sslope=0;  //slope in shear direction
```

```
      double nacceleration=0; //acceleration in normal direction
      double sacceleration=0; //acceleration in shear direction

      double nbacceleration=0;      //acceleration in normal direction if
flux turn back
      double sbacceleration=0;      //acceleration in shear direction if flux
trun back

      double divergence=0;    //divergence of flux, same as "Dx" in the paper

      double massin=0;  //total mass of inflow mass flux
      double nmassout=0;      //total mass of outflow mass flux in normal
direction
      double smassout=0;      //total mass of outflow mass flux in shear
direction

      double nvin=0;    //velocity of inflow mass flux in normal direction
      double svin=0;    //velocity of inflow mass flux in shear direction
      double nnvout=0;//velocity of outflow in normal direction through
normal boundary
      double nsvout=0;//velocity of outflow in shear direction through normal
boundary
      double snvout=0;//velocity of outflow in normal direction through shear
boundary
      double ssvout=0;//velocity of outflow in shear direction through shear
boundary

      double ndircoef=0;//the direction coefficient in normal direction which
is used to
                              //determine the direction of friction
      double sdircoef=0;//the direction coefficient in shear direction

      double outputmass[4]={0};     //array of mass of flux flowing out the
each boundary
      double outputvelocity[4][2]={0};    //velocity array of each flux

      for(int i=0;i<4;i++)    // do a loop for each boundary
      {
            massin=massflowin[i].gettotalmass();      // set initial mass of
inflow mass flux

            if(massin==0) continue;       // there is no outflow if the
inflow mass is 0

//          choose and set the mass flux direction
            if(i==0 || i==3)  // inflow through the top or bottom boundary
            {
//                calculate the surface angle of the cell in the shear
direction
                  ssurfaceangle=fabs(atan(slope[0]));
//                calculate the surface angle of the cell in the normal
direction
                  nsurfaceangle=fabs(atan(slope[1]));

//                calculate the inflow velocity of the cell in the shear
direction
                  svin=massflowin[i].getxvelocity();
```

```
//                calculate the inflow velocity of the cell in the normal
direction
            nvin=massflowin[i].getyvelocity();

//                calculate the slope in shear direction and normal direction
                  nslope=slope[1];
                  sslope=slope[0];

                  noutputside=(i==0)? 3:0; //determine the normal boundary of
outflow

                  //determine the shear boundary of outflow
                  if(svin>0) soutputside=2;
                  else if(svin==0 && sslope >0) soutputside=2;
                  else soutputside=1;
            }

            else        //inflow through the side boundary
            {
//                calculate the surface angle of the cell in the shear
direction
                  ssurfaceangle=fabs(atan(slope[1]));
//                calculate the surface angle of the cell in the normal
direction
                  nsurfaceangle=fabs(atan(slope[0]));

//                calculate the inflow velocity of the cell in the shear
direction
                  nvin=massflowin[i].getxvelocity();
//                calculate the inflow velocity of the cell in the normal
direction
            svin=massflowin[i].getyvelocity();

//                calculate the slope in shear direction and normal direction
                  nslope=slope[0];
                  sslope=slope[1];

                  noutputside=(i==1)? 2:1 ;//determine the normal boundary of
outflow

//                determine the shear boundary of outflow
                  if(svin>0) soutputside=3;
                  else if(svin==0 && sslope >0) soutputside=3;
                  else soutputside=0;
            }

//          calculate the slope angle
            slopeangle=atan(sqrt(slope[0]*slope[0]+slope[1]*slope[1]));

//          calculate the normal and shear direction coefficient
            ndircoef=fabs(nvin/sqrt(nvin*nvin+svin*svin));
            sdircoef=fabs(svin/sqrt(nvin*nvin+svin*svin));

            // calculate the surface size of cell in shear and normal
direction
            ss=Size/(2*cos(ssurfaceangle));
            ns=Size/cos(nsurfaceangle);
```

```
            nstoptime=0;      // initialization
            sstoptime=0;      // initialization

// this part of code is used to determine mass and velocity of outflow in
each boundary
            if((nvin*nslope)>=0 && (svin*sslope)>=0)//the x- and y- velocity
of inflow flux
                                                                  //are same
as the x- and y- slope direction
            {
                  // calcualte the acceleration in normal direction
                  nacceleration=9.8*(sin(nsurfaceangle)-
ndircoef*cos(slopeangle)*tan(KineticFrictionAngle));
                  // calcualte the acceleration in shear direction
                  sacceleration=9.8*(sin(ssurfaceangle)-
sdircoef*cos(slopeangle)*tan(KineticFrictionAngle));

                  // calcualte the acceleration in normal direction
                  nbacceleration=-
9.8*(sin(nsurfaceangle)+ndircoef*cos(slopeangle)*tan(KineticFrictionAngle));
                  // calcualte the acceleration in shear direction
                  sbacceleration=-
9.8*(sin(ssurfaceangle)+sdircoef*cos(slopeangle)*tan(KineticFrictionAngle));

                  // determine the time of flux flowing out the cell in
normal direction
                  if(nacceleration>0)     // when the mass flux is
accelerated in normal direction
                  {
                        noutputtime=(-
fabs(nvin)+sqrt(nvin*nvin+2*nacceleration*ns))/nacceleration;
                  }
                  else if(nacceleration<0 &&
fabs(nvin)>sqrt(2*fabs(nacceleration)*ns))
                                          // when the mass flux is slow down
but can move out the cell
                  {
                        noutputtime=(-
fabs(nvin)+sqrt(nvin*nvin+2*nacceleration*ns))/nacceleration;
                  }
                  else if(nacceleration==0 && nvin!=0)      //when the
velocities of mass flux

      // keep constant
                  {
                        noutputtime=ns/fabs(nvin);
                  }
                  else  // when the mass flux can not move out the cell
                  {
                        noutputtime=-1;

                        if(nacceleration!=0)    nstoptime=-
fabs(nvin)/nacceleration;
                        else  nstoptime=-1;
                  }
```

```
                  // determine the time of flux flowing out the cell in shear
direction
                  if(sacceleration>0)// when the mass flux is accelerated in
normal direction
                  {
                        soutputtime=(-
fabs(svin)+sqrt(svin*svin+2*sacceleration*ss))/sacceleration;
                  }
                  else if(sacceleration<0 &&
fabs(svin)>sqrt(2*fabs(sacceleration)*ss))
                        // when the mass flux is slow down but can move out
the cell
                  {
                        soutputtime=(-
fabs(svin)+sqrt(svin*svin+2*sacceleration*ss))/sacceleration;
                  }
                  else if(sacceleration==0 && svin!=0)//when the velocities
of mass flux

      // keep constant
                  {
                        soutputtime=ss/fabs(svin);
                  }
                  else
                  {
                        soutputtime=-1;

                        if(sacceleration!=0) sstoptime=-
fabs(svin)/sacceleration;
                        else sstoptime=-1;
                  }
            }

            else if((nvin*nslope)>=0 && (svin*sslope)<0)//the velocity
direction is same as
                                    // slope direction in x-direction and
opposite in y-direction
            {
                  // calcualte the acceleration in normal direction
                  nacceleration=9.8*(sin(nsurfaceangle)-
ndircoef*cos(slopeangle)*tan(KineticFrictionAngle));
                  // calcualte the acceleration in shear direction
                  sacceleration=-
9.8*(sin(ssurfaceangle)+sdircoef*cos(slopeangle)*tan(KineticFrictionAngle));

                  // calcualte the acceleration in normal direction
                  nbacceleration=-
9.8*(sin(nsurfaceangle)+ndircoef*cos(slopeangle)*tan(KineticFrictionAngle));
                  // calcualte the acceleration in shear direction
                  sbacceleration=9.8*(sin(ssurfaceangle)-
sdircoef*cos(slopeangle)*tan(KineticFrictionAngle));

                  // determine the time of flux flowing out the cell in
normal direction
                  if(nacceleration>0)// when the mass flux is accelerated in
normal direction
                  {
```

```
                        noutputtime=(-
fabs(nvin)+sqrt(nvin*nvin+2*nacceleration*ns))/nacceleration;
                  }
                  else if(nacceleration<0 &&
fabs(nvin)>sqrt(2*fabs(nacceleration)*ns))
                        // when the mass flux is slow down but can move out
the cell
                  {
                        noutputtime=(-
fabs(nvin)+sqrt(nvin*nvin+2*nacceleration*ns))/nacceleration;
                  }
                  else if(nacceleration==0 && nvin!=0)//when the velocities
of mass flux

      // keep constant
                  {
                        noutputtime=ns/fabs(nvin);
                  }
                  else
                  {
                        noutputtime=-1;

                        if(nacceleration!=0) nstoptime=-
fabs(nvin)/nacceleration;
                        else nstoptime=-1;
                  }

                  soutputtime=-1;
                  sstoptime=-1;
            }

            else if((nvin*nslope)<0 && (svin*sslope)>=0)//the x- and y-
velocity of inflow flux
                                                                  //are same
in the y- and opposite in x- slope direction
            {
                  // calcualte the acceleration in normal direction
                  nacceleration=-
9.8*(sin(nsurfaceangle)+ndircoef*cos(slopeangle)*tan(KineticFrictionAngle));
                  // calcualte the acceleration in shear direction
                  sacceleration=9.8*(sin(ssurfaceangle)-
sdircoef*cos(slopeangle)*tan(KineticFrictionAngle));

                  // calcualte the acceleration in normal direction
                  nbacceleration=9.8*(sin(nsurfaceangle)-
ndircoef*cos(slopeangle)*tan(KineticFrictionAngle));
                  // calcualte the acceleration in shear direction
                  sbacceleration=-
9.8*(sin(ssurfaceangle)+sdircoef*cos(slopeangle)*tan(KineticFrictionAngle));

                  noutputtime=-1;
                  nstoptime=-1;

                  // determine the time of flux flowing out the cell in
normal direction
                  if(sacceleration>0)// when the mass flux is accelerated in
shear direction
```

```
                  {
                        soutputtime=(-
fabs(svin)+sqrt(svin*svin+2*sacceleration*ss))/sacceleration;
                  }
                  else if(sacceleration<0 &&
fabs(svin)>sqrt(2*fabs(sacceleration)*ss))
                        // when the mass flux is slow down but can move out
the cell
                  {
                        soutputtime=(-
fabs(svin)+sqrt(svin*svin+2*sacceleration*ss))/sacceleration;
                  }
                  else if(sacceleration==0 && svin!=0)//when the velocities
of mass flux

      // keep constant
                  {
                        soutputtime=ss/fabs(svin);
                  }
                  else
                  {
                        soutputtime=-1;

                        if(sacceleration!=0) sstoptime=-
fabs(svin)/sacceleration;
                        else sstoptime=-1;
                  }
            }

            else                                          //the x- and y-
velocity of inflow flux
                                                                  //are
opposite in the x- and y- slope direction
            {
                  // determine the time of flux flowing out the cell in
normal direction
                  noutputtime=-1;
                  soutputtime=-1;
                  nstoptime=-1;
                  sstoptime=-1;
            }

            if(sbacceleration<0) sbacceleration=0;// calculate the shear
acceleration

            // this part of code is used to calculate the mass and velocity
of outflow

            // if there is no mass flowing out a cell
            if(noutputtime<=0 && soutputtime<=0)
            {
                  nmassout=0;
                  smassout=0;

                  nnvout=0;
                  nsvout=0;
                  snvout=0;
```

```
                ssvout=0;
                // add code for mass accumulated limitation
            }

            // if the mass flowing out the cell in normal direction is
dominant
            else if(noutputtime>0 && soutputtime<=0)
            {
                nmassout=massin;
                smassout=0;
                snvout=0;
                ssvout=0;

                // calculate the normal velocity
                if(nvin>=0)
                {
                    nnvout=nvin+nacceleration*noutputtime;
                }
                else
                {
                    nnvout=nvin-nacceleration*noutputtime;
                }

                //calculate the shear velocity
                if(svin>=0)
                {
                    if(sstoptime>noutputtime)
nsvout=svin+sacceleration*noutputtime;
                    else
nsvout=svin+sacceleration*sstoptime+sbacceleration*(noutputtime-sstoptime);
                }
                else
                {
                    if(sstoptime>noutputtime) nsvout=svin-
sacceleration*noutputtime;
                    else nsvout=svin-sacceleration*sstoptime-
sbacceleration*(noutputtime-sstoptime);
                }
            }

            // if the mass flowing out the cell in shear direction is
dominant
            else if(noutputtime<=0 && soutputtime>0)
            {
                smassout=massin;
                nmassout=0;
                nnvout=0;
                nsvout=0;

                //calculate the shear velocity
                if(svin>=0)
                {
                    snvout=svin+sacceleration*soutputtime;
                }
                else
                {
                    snvout=svin-sacceleration*soutputtime;
```

```
                }

                //calculate the normal velocity
                if(nvin>=0)
                {
                        if(nstoptime>soutputtime)
ssvout=nvin+nacceleration*soutputtime;
                        else
ssvout=nvin+nacceleration*nstoptime+nbacceleration*(soutputtime-nstoptime);
                }
                else
                {
                        if(nstoptime>soutputtime) ssvout=nvin-
nacceleration*soutputtime;
                        else ssvout=nvin-nacceleration*nstoptime-
nbacceleration*(soutputtime-nstoptime);
                }

            }

            //if mass flowing out a cell in both shear and normal direction
            else
            {
                  // calculate the mass outflow a cell in shear and normal
direction
                  if(noutputtime<=soutputtime)
                  {

      divergence=fabs((fabs(svin)*noutputtime+sacceleration*noutputtime*noutp
uttime/2)/(2*ss));
                        if(divergence<0) divergence=0;
                        nmassout=massin*(1-divergence);
                        smassout=massin*divergence;
                  }
                  else
                  {

      divergence=fabs((fabs(nvin)*soutputtime+nacceleration*soutputtime*soutp
uttime/2)/ns);
                        if(divergence<0.5) divergence=0.5;
                        nmassout=massin*(divergence-0.5);
                        smassout=massin*(1.5-divergence);
                  }

                  // calculate the normal and shear velocity of outflow
massflux
                  if(nvin>=0)
                  {
                        nnvout=nvin+nacceleration*noutputtime;
                        ssvout=nvin+nacceleration*soutputtime;
                  }
                  else
                  {
                        nnvout=nvin-nacceleration*noutputtime;

                        ssvout=nvin-nacceleration*soutputtime;
                  }
```

```
                  if(svin>=0)
                  {
                        snvout=svin+sacceleration*soutputtime;
                        nsvout=svin+sacceleration*noutputtime;
                  }
                  else
                  {
                        snvout=svin-sacceleration*soutputtime;
                        nsvout=svin-sacceleration*noutputtime;
                  }

                  if(nnvout>0.1)nnvout=0.1;
                  if(nsvout>0.1)nsvout=0.1;
                  if(snvout>0.1)snvout=0.1;
                  if(ssvout>0.1)ssvout=0.1;
            }

            // if the mass flowing out a cell is less than the minimun
tolerance, ignore this part of mass and set the mass outflowing a cell to be
zero
            if(nmassout<=MinMassOut)
            {
                  nmassout=0;
                  nnvout=0;
                  nsvout=0;
            }

            // if the mass flowing out a cell is less than the minimun
tolerance, ignore this part of mass and set the mass outflowing a cell to be
zero
            if(smassout<=MinMassOut)
            {
                  smassout=0;
                  snvout=0;
                  ssvout=0;
            }

            // this part of code is used to put the mass and velocity of
outflow into the global matrix
            if(i==0 || i==3) // if the mass flow in the top or bottom
boundary
            {
                  if(outputmass[noutputside]==0) // if the mass flow out
through the top boundary
                  {
                outputvelocity[noutputside][0]=nsvout;
                outputvelocity[noutputside][1]=nnvout;
                outputmass[noutputside]=nmassout;
                  }
                  else // if the mass flow out through the bottom boundary
                  {

      outputvelocity[noutputside][0]=(nmassout*nsvout+outputvelocity[noutputs
ide][0]*

      outputmass[noutputside])/(nmassout+outputmass[noutputside]);
```

```
outputvelocity[noutputside][1]=(nmassout*nnvout+outputvelocity[noutputside][1
]*

      outputmass[noutputside])/(nmassout+outputmass[noutputside]);;
                outputmass[noutputside]+=nmassout;
                  }

                  if(outputmass[soutputside]==0) // if the mass flow out
through the top boundary
                  {
                outputvelocity[soutputside][0]=snvout;
                outputvelocity[soutputside][1]=ssvout;
                outputmass[soutputside]=smassout;
                  }
                  else // if the mass flow out through the bottom boundary
                  {

      outputvelocity[soutputside][0]=(smassout*snvout+outputvelocity[soutputs
ide][0]*

      outputmass[soutputside])/(smassout+outputmass[soutputside]);

outputvelocity[soutputside][1]=(smassout*ssvout+outputvelocity[soutputside][1
]*

      outputmass[soutputside])/(smassout+outputmass[soutputside]);;
                outputmass[soutputside]+=smassout;
                  }
            }

            else  // if the mass flow in the side boundary
            {
                  if(outputmass[noutputside]==0) // if the mass flow out
through the top boundary
                  {
                outputvelocity[noutputside][0]=nnvout;
                outputvelocity[noutputside][1]=nsvout;
                outputmass[noutputside]=nmassout;
                  }
                  else // if the mass flow out through the bottom boundary
                  {

      outputvelocity[noutputside][0]=(nmassout*nnvout+outputvelocity[noutputs
ide][0]*

      outputmass[noutputside])/(nmassout+outputmass[noutputside]);

outputvelocity[noutputside][1]=(nmassout*nsvout+outputvelocity[noutputside][1
]*

      outputmass[noutputside])/(nmassout+outputmass[noutputside]);;
                outputmass[noutputside]+=nmassout;
                  }

                  if(outputmass[soutputside]==0) // if the mass flow out
through the top boundary
```

```
                  {
                outputvelocity[soutputside][0]=ssvout;
                outputvelocity[soutputside][1]=snvout;
                outputmass[soutputside]=smassout;
                  }
                  else // if the mass flow out through the bottom boundary
                  {

      outputvelocity[soutputside][0]=(smassout*ssvout+outputvelocity[soutputs
ide][0]*

      outputmass[soutputside])/(smassout+outputmass[soutputside]);

outputvelocity[soutputside][1]=(smassout*snvout+outputvelocity[soutputside][1
]*

      outputmass[soutputside])/(smassout+outputmass[soutputside]);;
                outputmass[soutputside]+=smassout;
                  }
            }
      }

      // put the mass and velocity of outflow into the global matrix
      for(int i=0;i<4;i++)
      {
            massflowout[i].settotalmass(outputmass[i]);
        massflowout[i].setxvelocity(outputvelocity[i][0]);
        massflowout[i].setyvelocity(outputvelocity[i][1]);
      }
}

//    define the function setmassflowin which is used to set the mass and
velocity of inflow
void element::setmassflowin(double mass[4],double velocity[4][2])
{
      for(int i=0;i<4;i++)
      {
            this->massflowin[i].settotalmass(mass[i]); // set the inflow mass
            this->massflowin[i].setxvelocity(velocity[i][0]); // set the
inflow velocity in x-direction
            this->massflowin[i].setyvelocity(velocity[i][1]); // set the
inflow velocity in y-direction
      }
}
```

```
// This program is used to calculate the amount mass of dry ravel
// produced by wild fire
// Include the head file
// headerfile.h include the namespace and files used in program
// headerfile.h also include the definition of constant
#include "headerfile.h"
// node.h declare the class node
// node.cpp define the class node
#include "node.h"
// element.h declare the class element
// element.cpp define the class element
#include "element.h"
// event.h declare the class event
// event.cpp define the class event
#include "event.h"
// the main program begin
int main()
{
//    char fire;
//define the number of the row and column of the
//calculation domain
      int NumberofRow=100;
      int NumberofColumn=100;
// define the size of each cell
      int Size=1;
//    int numberoffire=0;
//    define the minimun mass flowing out one cell.
//    if the mass flowing out the cell less than MinMassOut,
//    the program will ignore this part of mass
      double MinMassOut=0.001;
      double vegetationsize=0.1;    //average diameter of vegetation
      double vegetationdensity=20;  //amount of vegetation in 1 square meter
      double averageproduction=0;   //declare the average dry ravel
production in each cell
      double averagedeposition=0;   //declare the average dry ravel
deposition in each cell
      double averagevariation=0;    //declare the average dry ravel variation
in each cell
      double FireImpactDepth=0;

      //Define the friction angle
      double FrictionAngle;
      double KineticFrictionAngle;


//    open the file which including the input parameters
      ifstream paraminput("paraminput.txt", ios::in);

//    input parameters from the file "paraminput.txt"
      paraminput >> NumberofRow >> NumberofColumn >> Size >> MinMassOut
                     >> vegetationsize >> vegetationdensity >>
FireImpactDepth >> FrictionAngle >> KineticFrictionAngle;

//    calculate the static friction angle (repose angle) of soil
      FrictionAngle=FrictionAngle*PI/180;
//    calculate the kinetic friction angle of soil
      KineticFrictionAngle=KineticFrictionAngle*PI/180;
```

```
//    close the file which including the input parameters
      paraminput.close();

//    declare a pointer "nodeelevation" which point to array
//    storing the elevation of node point
      double **nodeelevation = new double * [NumberofRow+1];
      for (int i = 0; i < NumberofRow+1; i++)
            nodeelevation[i] = new double [NumberofColumn+1];

//    declare a pointer "massvariationation" which point to array
//    storing the information of mass changes in each cell
      double **massvariation = new double * [NumberofRow];
      for(int i=0;i<NumberofRow;i++)
            massvariation[i] = new double [NumberofColumn];

//    declare a pointer "massloss" which point to array
//    storing the information of mass loss from each cell
      double **massloss = new double * [NumberofRow];
      for(int i=0;i<NumberofRow;i++)
            massloss[i]= new double [NumberofColumn];

//    declare a pointer "elementmassin" which point to array
//    storing the information of mass flowing in each cell
      double **elementmassin = new double * [NumberofRow];
      for(int i=0;i<NumberofRow;i++)
            elementmassin[i]= new double [NumberofColumn];

//    declare a pointer "elementmassout" which point to array
//    storing the information of mass flowing out from each cell
      double **elementmassout= new double * [NumberofRow];
      for(int i=0;i<NumberofRow; i++)
            elementmassout[i]= new double [NumberofColumn];

//    declare a pointer "massin" which point to array
//    storing the information of in-flow mass of mass flux on each side of
cells
      double *** massin = new (double ** [NumberofRow]);
      for(int i=0;i<NumberofRow;i++)
      {
            massin[i]=new (double * [NumberofColumn]);
            for(int j=0;j<NumberofColumn;j++)
                  massin[i][j]=new (double [4]);
      }

//    declare a pointer "massout" which point to array
//    storing the information of out-flow mass of mass flux on each side of
cells
      double *** massout = new (double ** [NumberofRow]);
      for(int i=0;i<NumberofRow;i++)
      {
            massout[i]=new (double * [NumberofColumn]);
            for(int j=0;j<NumberofColumn;j++)
                  massout[i][j]=new (double [4]);
      }

//    declare a pointer "xinvelocity" which point to array
```

```
//    storing the information of in-flow x-velocity of mass flux on each side
of cells
      double *** xinvelocity =new (double ** [NumberofRow]);
      for(int i=0;i<NumberofRow;i++)
      {
            xinvelocity[i]=new (double * [NumberofColumn]);
            for(int j=0;j<NumberofColumn;j++)
                  xinvelocity[i][j]=new (double [4]);
      }

//    declare a pointer "yinvelocity" which point to array
//    storing the information of in-flow y-velocity of mass flux on each side
of cells
      double *** yinvelocity =new (double ** [NumberofRow]);
      for(int i=0;i<NumberofRow;i++)
      {
            yinvelocity[i]=new (double * [NumberofColumn]);
            for(int j=0;j<NumberofColumn;j++)
                  yinvelocity[i][j]=new (double [4]);
      }

//    declare a pointer "xoutvelocity" which point to array
//    storing the information of out-flow y-velocity of mass flux on each
side of cells
      double *** xoutvelocity =new (double ** [NumberofRow]);
      for(int i=0;i<NumberofRow;i++)
      {
            xoutvelocity[i]=new (double * [NumberofColumn]);
            for(int j=0;j<NumberofColumn;j++)
                  xoutvelocity[i][j]=new (double [4]);
      }

//    declare a pointer "youtvelocity" which point to array
//    storing the information of mass of mass flux on each side of cells
      double *** youtvelocity =new (double ** [NumberofRow]);
      for(int i=0;i<NumberofRow;i++)
      {
            youtvelocity[i]=new (double * [NumberofColumn]);
            for(int j=0;j<NumberofColumn;j++)
                  youtvelocity[i][j]=new (double [4]);
      }

// Input elevation of every node point
      for(int i=0;i<NumberofRow+1;i++)
      {
            for(int j=0;j<NumberofColumn+1;j++)
            {
                  nodeelevation[i][j]=0;
            }
      }

//    set the initial value of the arraies to be zero
      for(int i=0;i<NumberofRow;i++)
      {
            for(int j=0;j<NumberofColumn;j++)
            {
                  massvariation[i][j]=0;
```

```
                massloss[i][j]=0;
                elementmassin[i][j]=0;
                elementmassout[i][j]=0;
                for(int k=0;k<4;k++)
                {
                      massin[i][j][k]=0;
                      massout[i][j][k]=0;
              xinvelocity[i][j][k]=0;
                      yinvelocity[i][j][k]=0;
                      xoutvelocity[i][j][k]=0;
                      youtvelocity[i][j][k]=0;
                }
          }
      }

// open the file pointinput.txt which store the elevation
// of node point
      ifstream pointinput("pointinput.txt", ios::in);

//    input the elevation of each node
      for(int i=0;i<NumberofRow+1;i++)
      {
            for(int j=0;j<NumberofColumn+1;j++)
            {
                  pointinput >> nodeelevation[i][j];
            }
      }

//    close the elevation file
      pointinput.close();


// open the file pointoutput.txt for output data
      ofstream pointoutput("pointoutput.xls", ios::out);

//    start the loop to calculate the dry ravel movement of each fire impact
cell
      for(int i=0;i<NumberofRow;i++)
      {
            for(int j=0;j<NumberofColumn;j++)
            {

                  // output some imformation of program running processes
                  cout << endl << i << ' ' << j;

//                declare a pointer pointing to the class element, and build
the object of the class
                  element *peventproduct=new
element(i*NumberofColumn+j,nodeelevation[i][j],nodeelevation[i][j+1]

      ,nodeelevation[i+1][j],nodeelevation[i+1][j+1],Size);

//                conduct the function eventproduct which represent the fire
occurs in the cell
                  peventproduct-
>eventproduct(vegetationsize,vegetationdensity,
                        FireImpactDepth,FrictionAngle,KineticFrictionAngle);
```

```
//                update the informations of mass and velocity on the cell
                  for(int k=0;k<4;k++)
                  {
                        massout[i][j][k]=peventproduct-
>massflowout[k].gettotalmass();
                        xoutvelocity[i][j][k]=peventproduct-
>massflowout[k].getxvelocity();
                        youtvelocity[i][j][k]=peventproduct-
>massflowout[k].getyvelocity();

                        massvariation[i][j]-=peventproduct-
>massflowout[k].gettotalmass();
                        massloss[i][j]-=peventproduct-
>massflowout[k].gettotalmass();
                        elementmassout[i][j]-=peventproduct-
>massflowout[k].gettotalmass();
                  }

//                clear the object and memory
                  delete peventproduct;

//                if there is no mass flowing out the cell, go to next fire
impact cell
                  if(massout[i][j][0]==0 && massout[i][j][1]==0 &&
massout[i][j][2]==0 && massout[i][j][3]==0)
                              continue;

//                if there is mass flowing out, continue calculate the
information of other cells
                  double massmove=1;

//                start a new loop, if there is mass flux exists the loop
will continue
                  while(massmove!=0)
                  {
                        //set that there is no mass flux
                        massmove=0;

//                      start a loop which will calculate dry ravel movement
of each cell of the domain
                        for(int x=0;x<NumberofRow;x++)
                        {
                              for(int y=0;y<NumberofColumn;y++)
                              {
//                                  set the initial value of array massin,
xinvelocity,and yinvelocity
                                    if(x==0)    //for the case that the cell
is on the lefe boundary
                                    {
                                          massin[x][y][0]=0;
                                          xinvelocity[x][y][0]=0;
                                          yinvelocity[x][y][0]=0;
                                    }
                                    else  // for the case that the cell is
not on the boundart
                                    {
```

```
                        massin[x][y][0]=massout[x-1][y][3];

      xinvelocity[x][y][0]=xoutvelocity[x-1][y][3];

      yinvelocity[x][y][0]=youtvelocity[x-1][y][3];
                                    }

                                    if(x==NumberofRow-1)//for the case that
the cell is on the right boundary
                                    {
                                          massin[x][y][3]=0;
                                          xinvelocity[x][y][3]=0;
                                          yinvelocity[x][y][3]=0;
                                    }
                                    else // for the case that the cell is not
on the boundart
                                    {
                                          massin[x][y][3]=massout[x+1][y][0];

      xinvelocity[x][y][3]=xoutvelocity[x+1][y][0];

      yinvelocity[x][y][3]=youtvelocity[x+1][y][0];
                                    }

                                    if(y==0)//for the case that the cell is
on the upper boundary
                                    {
                                          massin[x][y][1]=0;
                                          xinvelocity[x][y][1]=0;
                                          yinvelocity[x][y][1]=0;
                                    }
                                    else // for the case that the cell is not
on the boundart
                                    {
                                          massin[x][y][1]=massout[x][y-1][2];

      xinvelocity[x][y][1]=xoutvelocity[x][y-1][2];

      yinvelocity[x][y][1]=youtvelocity[x][y-1][2];
                                    }

                                    if(y==NumberofColumn-1)//for the case
that the cell is on the lower boundary
                                    {
                                          massin[x][y][2]=0;

                        xinvelocity[x][y][2]=0;

                        yinvelocity[x][y][2]=0;
                                    }
                                    else // for the case that the cell is not
on the boundart
                                    {
                        massin[x][y][2]=massout[x][y+1][1];

                        xinvelocity[x][y][2]=xoutvelocity[x][y+1][1];
```

```
                              yinvelocity[x][y][2]=youtvelocity[x][y+1][1];
                                    }

//                                  update the mass change
                                    for(int k=0;k<4;k++)
                                    {

      massvariation[x][y]+=massin[x][y][k];

      elementmassin[x][y]+=massin[x][y][k];
                                    }
                              }
                        }

//                      set mass, velocity of out-flow to be zero
                        for(int m=0;m<NumberofRow;m++)
                        {
                              for(int n=0;n<NumberofColumn;n++)
                              {
                                    for(int k=0;k<4;k++)
                                    {
                                          massout[m][n][k]=0;
                                    xoutvelocity[m][n][k]=0;
                                          youtvelocity[m][n][k]=0;
                                    }
                              }
                        }

//                      update the mass change and velocity change
                        for(int m=0;m<NumberofRow;m++)
                        {
                              for(int n=0;n<NumberofColumn;n++)
                              {
//                                  if there is not mass flow in the cell
                                    if(massin[m][n][0]==0 &&
massin[m][n][1]==0 && massin[m][n][2]==0 && massin[m][n][3]==0)
                                    {
                                          for(int k=0;k<4;k++)
                                          {
                                                massout[m][n][k]=0;
                                                xoutvelocity[m][n][k]=0;
                                                youtvelocity[m][n][k]=0;
                                          }
                                          continue;
                                    }

//                                  initial the new object of class element
                                    element * peventoccur=new
element(m*NumberofColumn+n,nodeelevation[m][n],

      nodeelevation[m][n+1],nodeelevation[m+1][n],nodeelevation[m+1][n+1],Siz
e);

//                                  declare the total mass flowing in the
cell
```

```
                                    double
totalmass[4]={massin[m][n][0],massin[m][n][1],massin[m][n][2],massin[m][n][3]
};
//                                  declare the velocity matrix of mass flux
                                    double
veolociy[4][2]={xinvelocity[m][n][0],yinvelocity[m][n][0],

xinvelocity[m][n][1],yinvelocity[m][n][1],

xinvelocity[m][n][2],yinvelocity[m][n][2],

xinvelocity[m][n][3],yinvelocity[m][n][3]};

//                                  set the in-flow mass and velocity
                                    peventoccur-
>setmassflowin(totalmass,veolociy);

//                                  call the function eventoccur to calculate
the mass flux changes
                            peventoccur-
>eventoccur(MinMassOut,KineticFrictionAngle);

//                                  update the massout, x- and y- velocity,
and mass variation in each element
                                    for(int k=0;k<4;k++)
                                    {
                              massmove+=fabs(peventoccur-
>massflowout[k].gettotalmass());

                                          massout[m][n][k]=peventoccur-
>massflowout[k].gettotalmass();

                                          xoutvelocity[m][n][k]=peventoccur-
>massflowout[k].getxvelocity();
                                          youtvelocity[m][n][k]=peventoccur-
>massflowout[k].getyvelocity();

                                          massvariation[m][n]-
=massout[m][n][k];
                                          elementmassout[m][n]-
=massout[m][n][k];
                                    }

//                                  clear the object "peventoccur"
                                    delete peventoccur;
                              }
                        }

//                      reset massin, x- and y- velocity for next calculation
cycle
                        for(int m=0;m<NumberofRow;m++)
                        {
                              for(int n=0;n<NumberofColumn;n++)
                              {
                                    for(int k=0;k<4;k++)
                                    {
                                          massin[m][n][k]=0;
```

```
                                    xinvelocity[m][n][k]=0;
                                          yinvelocity[m][n][k]=0;
                                    }
                              }
                        }
                  }

//                reste mass and velocity matrix for next calculation
                  for(int m=0;m<NumberofRow;m++)
                  {
                        for(int n=0;n<NumberofColumn;n++)
                        {
                              for(int k=0;k<4;k++)
                              {
                                    massin[m][n][k]=0;
                                    massout[m][n][k]=0;
                              xinvelocity[m][n][k]=0;
                                    yinvelocity[m][n][k]=0;
                                    xoutvelocity[m][n][k]=0;
                                    youtvelocity[m][n][k]=0;
                              }
                        }
                  }

//    output the results

//    output mass flowing out from each element
      pointoutput << "Mass flow out from each cell" << endl;
      for(int i=0;i<NumberofRow;i++)
      {
            for(int j=0;j<NumberofColumn;j++)
            {
                  pointoutput << elementmassout[i][j] << "\t";
            }
            pointoutput << endl;
      }

//    output mass flowing into the element during the whole process
      pointoutput << "Mass flow into each cell" << endl;
      for(int i=0;i<NumberofRow;i++)
      {
            for(int j=0;j<NumberofColumn;j++)
            {
                  pointoutput << elementmassin[i][j] << "\t";
            }
            pointoutput << endl;
      }

//    output the mass lossed from each element
      pointoutput << "Mass loss from each cell" << endl;
      for(int i=0;i<NumberofRow;i++)
      {

            for(int j=0;j<NumberofColumn;j++)
            {
                  pointoutput << massloss[i][j] << "\t";
                  averageproduction+=massloss[i][j];
```

```
			}
			pointoutput << endl;
		}
		averageproduction=averageproduction/(NumberofRow*NumberofColumn);

//		output the mass variation of each element
		pointoutput << "Mass variation for each cell" << endl;
		for(int i=0;i<NumberofRow;i++)
		{
			for(int j=0;j<NumberofColumn;j++)
			{
				pointoutput << massvariation[i][j] << "\t";
				averagevariation+=massvariation[i][j];
			}
			pointoutput << endl;
		}
		averagevariation=averagevariation/(NumberofRow*NumberofColumn);

//		output mass deposition in each element
		pointoutput << "Mass deposition for each cell" << endl;
		for(int i=0;i<NumberofRow;i++)
		{
			for(int j=0;j<NumberofColumn;j++)
			{
				pointoutput << (massvariation[i][j]-massloss[i][j]) <<
"\t";
				averagedeposition+=(massvariation[i][j]-massloss[i][j]);
			}
			pointoutput << endl;
		}
		averagedeposition=averagedeposition/(NumberofRow*NumberofColumn);

//		output the average value of dry ravel production, mass variation of dry
ravel in each element,
//		and the dry ravel deposition in each element.
		pointoutput << averageproduction << '\t' << averagevariation << '\t' <<
averagedeposition << endl;

		pointoutput.close();

// clear the objects, close the output file, and release the memory
		for (int i = 0; i < NumberofRow+1; i++)
			delete [] nodeelevation[i];
		delete [] nodeelevation;

		for(int i=0;i<NumberofRow;i++)
		{
			delete [] massvariation[i];
			delete [] massloss[i];
			delete [] elementmassin[i];
			delete [] elementmassout[i];

			for(int j=0;j<NumberofColumn;j++)
			{
				delete [] massin[i][j];
				delete [] massout[i][j];
				delete [] xinvelocity[i][j];
```

```
                delete [] yinvelocity[i][j];
                delete [] xoutvelocity[i][j];
                delete [] youtvelocity[i][j];
            }

            delete [] massin[i];
            delete [] massout[i];
            delete [] xinvelocity[i];
            delete [] yinvelocity[i];
            delete [] xoutvelocity[i];
            delete [] youtvelocity[i];
        }

        delete [] massvariation;
        delete [] massloss;
        delete [] elementmassin;
        delete [] elementmassout;
        delete [] massin;
        delete [] massout;
        delete [] xinvelocity;
        delete [] yinvelocity;
        delete [] xoutvelocity;
        delete [] youtvelocity;

        return 0;
}
```

B. Statistical Analysis

ANOVA codes

Watershed 0507 Production

```
DATA A;
INFILE 'c:\stat\prod507.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

Watershed 0508 Production

```
DATA A;
INFILE 'c:\stat\prod508.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

Watershed 0542 Production

```
DATA A;
INFILE 'c:\stat\prod542.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

Watershed 0560 Production

```
DATA A;
INFILE 'c:\stat\prod560.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

Watershed 0507 Deposition

```
DATA A;
INFILE 'c:\stat\dep507.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

Watershed 0508 Deposition

```
DATA A;
INFILE 'c:\stat\dep508.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

Watershed 0542 Deposition

```
DATA A;
INFILE 'c:\stat\dep542.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

Watershed 0560 Deposition

```
DATA A;
INFILE 'c:\stat\dep560.prn';
INPUT PROD METHOD $ WATSHD;
PROC GLM;
CLASS METHOD WATSHD;
MODEL PROD = METHOD;
MEANS METHOD/TUKEY ALPHA=0.01;
MEANS METHOD/TUKEY;
MEANS METHOD/TUKEY ALPHA=0.1;
RUN;
```

ANOVA Results

Watershed 0507 Production

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	507

Number of observations 50

Dependent Variable: PROD

Sum of Source	DF	Squares	Mean Square	F Value	Pr > F
Model	1	0.7200	0.7200	0.00	0.9949
Error	48	845766.0000	17620.1250		
Corrected Total	49	845766.7200			

R-Square	Coeff Var	Root MSE	PROD Mean
0.000001	79.08771	132.7408	167.8400

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	0.72000000	0.72000000	0.00	0.9949

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	0.72000000	0.72000000	0.00	0.9949

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	48
Error Mean Square	17620.13
Critical Value of Studentized Range	3.79322
Minimum Significant Difference	100.7

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	167.96	25	OBS
A			
A	167.72	25	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	48
Error Mean Square	17620.13
Critical Value of Studentized Range	2.84352
Minimum Significant Difference	75.49

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	167.96	25	OBS
A			
A	167.72	25	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.1
Error Degrees of Freedom	48
Error Mean Square	17620.13
Critical Value of Studentized Range	2.37195
Minimum Significant Difference	62.971

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	167.96	25	OBS
A			
A	167.72	25	PRED

Watershed 0508 Production

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	508

Number of observations 50

Dependent Variable: PROD

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	0.180	0.180	0.00	0.9979
Error	48	1284614.640	26762.805		
Corrected Total	49	1284614.820			

R-Square	Coeff Var	Root MSE	PROD Mean
0.000000	69.63200	163.5934	234.9400

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	0.18000000	0.18000000	0.00	0.9979

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	0.18000000	0.18000000	0.00	0.9979

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	48
Error Mean Square	26762.81
Critical Value of Studentized Range	3.79322
Minimum Significant Difference	124.11

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	235.00	25	PRED
A			
A	234.88	25	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	48
Error Mean Square	26762.81
Critical Value of Studentized Range	2.84352
Minimum Significant Difference	93.036

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	235.00	25	PRED
A			
A	234.88	25	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.1
Error Degrees of Freedom	48
Error Mean Square	26762.81
Critical Value of Studentized Range	2.37195
Minimum Significant Difference	77.607

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	235.00	25	PRED
A			
A	234.88	25	OBS

Watershed 0542 Production

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	542

Number of observations 50

Dependent Variable: PROD

Sum of Source	DF	Squares	Mean Square	F Value	Pr > F
Model	1	1.280	1.280	0.00	0.9980
Error	48	9894376.720	206132.848		
Corrected Total	49	9894378.000			

R-Square	Coeff Var	Root MSE	PROD Mean
0.000000	105.8812	454.0186	428.8000

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	1.28000000	1.28000000	0.00	0.9980

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	1.28000000	1.28000000	0.00	0.9980

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	48
Error Mean Square	206132.8
Critical Value of Studentized Range	3.79322
Minimum Significant Difference	344.44

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	429.0	25	OBS
A			
A	428.6	25	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	48
Error Mean Square	206132.8
Critical Value of Studentized Range	2.84352
Minimum Significant Difference	258.2

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	429.0	25	OBS
A			
A	428.6	25	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.1
Error Degrees of Freedom	48
Error Mean Square	206132.8
Critical Value of Studentized Range	2.37195
Minimum Significant Dfference	215.38

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	429.0	25	OBS
A			
A	428.6	25	PRED

Watershed 0560 Production

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	560

Number of observations 50

Dependent Variable: PROD

Sum of Source	DF	Squares	Mean Square	F Value	Pr > F
Model	1	0.080	0.080	0.00	0.9987
Error	48	1344919.040	28019.147		
Corrected Total	49	1344919.120			

R-Square	Coeff Var	Root MSE	PROD Mean
0.000000	97.18370	167.3892	172.2400

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	0.08000000	0.08000000	0.00	0.9987

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	0.08000000	0.08000000	0.00	0.9987

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	48
Error Mean Square	28019.15
Critical Value of Studentized Range	3.79322
Minimum Significant Difference	126.99

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	172.28	25	PRED
A			
A	172.20	25	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	48
Error Mean Square	28019.15
Critical Value of Studentized Range	2.84352
Minimum Significant Difference	95.195

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	172.28	25	PRED
A			
A	172.20	25	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

lpha	0.1
Error Degrees of Freedom	48
Error Mean Square	28019.15
Critical Value of Studentized Range	2.37195
Minimum Significant Difference	79.408

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	172.28	25	PRED
A			
A	172.20	25	OBS

Watershed 0507 Deposition

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	507

Number of observations 100

Dependent Variable: PROD

Sum of Source	DF	Squares	Mean Square	F Value	Pr > F
Model	1	1382505.64	1382505.64	3.23	0.0754
Error	98	41964973.60	428214.02		
Corrected Total	99	43347479.24			

R-Square	Coeff Var	Root MSE	PROD Mean
0.031894	179.9034	654.3806	363.7400

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	1382505.640	1382505.640	3.23	0.0754

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	1382505.640	1382505.640	3.23	0.0754

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	98
Error Mean Square	428214
Critical Value of Studentized Range	3.71512
Minimum Significant Difference	343.81

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	481.3	50	PRED
A			
A	246.2	50	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	98
Error Mean Square	428214
Critical Value of Studentized Range	2.80646
Minimum Significant Difference	259.72

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	481.3	50	PRED
A			
A	246.2	50	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.1
Error Degrees of Freedom	98
Error Mean Square	428214
Critical Value of Studentized Range	2.34837
Minimum Significant Difference	217.33

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	481.3	50	PRED
B	246.2	50	OBS

Watershed 0508 Deposition

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	508

Number of observations 100

dependent Variable: PROD

Sum of Source	DF	Squares	Mean Square	F Value	Pr > F
Model	1	809078202	809078202	1.99	0.1614
Error	98	39822270778	406349702		
Corrected Total	99	40631348981			

R-Square	Coeff Var	Root MSE	PROD Mean
0.019913	621.4870	20158.12	3243.530

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	809078202.5	809078202.5	1.99	0.1614

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	809078202.5	809078202.5	1.99	0.1614

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	98
Error Mean Square	4.0635E8
Critical Value of Studentized Range	3.71512
Minimum Significant Difference	10591

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	6088	50	OBS
A			
A	399	50	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	98
Error Mean Square	4.0635E8
Critical Value of Studentized Range	2.80646
Minimum Significant Difference	8000.6

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	6088	50	OBS
A			
A	399	50	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.1
Error Degrees of Freedom	98
Error Mean Square	4.0635E8
Critical Value of Studentized Range	2.34837
Minimum Significant Difference	6694.7

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	6088	50	OBS
A			
A	399	50	PRED

Watershed 0542 Deposition

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	542

Number of observations 100

Dependent Variable: PROD

Sum of Source	DF	Squares	Mean Square	F Value	Pr > F
Model	1	188964.09	188964.09	0.21	0.6490
Error	98	88817839.70	906304.49		
Corrected Total	99	89006803.79			

R-Square	Coeff Var	Root MSE	PROD Mean
0.002123	177.4828	952.0003	536.3900

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	188964.0900	188964.0900	0.21	0.6490

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	188964.0900	188964.0900	0.21	0.6490

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	98
Error Mean Square	906304.5
Critical Value of Studentized Range	3.71512
Minimum Significant Difference	500.18

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	579.9	50	PRED
A			
A	492.9	50	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	98
Error Mean Square	906304.5
Critical Value of Studentized Range	2.80646
Minimum Significant Difference	377.84

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	579.9	50	PRED
A			
A	492.9	50	OBS

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.1
Error Degrees of Freedom	98
Error Mean Square	906304.5
Critical Value of Studentized Range	2.34837
Minimum Significant Difference	316.17

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	579.9	50	PRED
A			
A	492.9	50	OBS

Watershed 0560 Deposition

The GLM Procedure

Class Level Information

Class	Levels	Values
METHOD	2	OBS PRED
WATSHD	1	560

Number of observations 100

Dependent Variable: PROD

Sum of Source	DF	Squares	Mean Square	F Value	Pr > F
Model	1	77896.81	77896.81	0.18	0.6705
Error	98	41927197.30	427828.54		
Corrected Total	99	42005094.11			

R-Square	Coeff Var	Root MSE	PROD Mean
0.001854	184.9425	654.0860	353.6700

Source	DF	Type I SS	Mean Square	F Value	Pr > F
METHOD	1	77896.81000	77896.81000	0.18	0.6705

Source	DF	Type III SS	Mean Square	F Value	Pr > F
METHOD	1	77896.81000	77896.81000	0.18	0.6705

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.01
Error Degrees of Freedom	98
Error Mean Square	427828.5
Critical Value of Studentized Range	3.71512
Minimum Significant Difference	343.66

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	381.6	50	OBS
A			
A	325.8	50	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	98
Error Mean Square	427828.5
Critical Value of Studentized Range	2.80646
Minimum Significant Difference	259.6

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	381.6	50	OBS
A			
A	325.8	50	PRED

Tukey's Studentized Range (HSD) Test for PROD

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.1
Error Degrees of Freedom	98
Error Mean Square	427828.5
Critical Value of Studentized Range	2.34837
Minimum Significant Difference	217.23

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	METHOD
A	381.6	50	OBS
A			
A	325.8	50	PRED