INCORPORATING A MECHANISTIC BANK FAILURE ALGORITHM INTO THE CELLULAR AUTOMATON EVOLUTIONARY SLOPE AND RIVER (CAESAR) MODEL, AND APPLYING IT TO A HIGHLY EROSIVE REACH OF THE COLORADO RIVER, AUSTIN, TEXAS

THESIS

Presented to the Graduate Council of
Texas State University-San Marcos
in Partial Fulfillment
of the Requirements

for the Degree

Master of SCIENCE

by

Delbert G. Humberson, B.A.

San Marcos, Texas
May 2009

ACKNOWLEDGEMENTS

It is with great pleasure and gratitude that I offer my sincerest thanks to Dr. Joanna Curran, Dr. Mark Fonstad, and Dr. Tom Coulthard. Without their support and willingness to entertain the questions of a neophyte, this thesis would have never reached fruition. It has been a fantastic journey made possible by their steady hands on the helm.

I dedicate this thesis to my family, to whom I also offer the greatest thanks. My wife, with great generosity and care, has held up more than her fair share of duties while my time was devoted to schooling. Her love and support inspired me to return to academics, and I never would have reached this far without her. I look forward to returning the favor one day. To my son of 18 months, I would like to thank him for giving me laughter, love, and a reason to never give up. I would like to thank my parents for giving me a love of learning and the strength to explore the unknown, and to my younger brother, who has much more inside of him than he seems to know, I would like to thank for reminding me of the virtue of perseverance.

This manuscript was submitted on 13 November 2008.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

ABSTRACT

INCORPORATING A MECHANISTIC BANK FAILURE ALGORITHM INTO THE CELLULAR AUTOMATON EVOLUTIONARY SLOPE AND RIVER (CAESAR) MODEL, AND APPLYING IT TO A HIGHLY EROSIVE REACH OF THE COLORADO RIVER, AUSTIN, TEXAS

by

Delbert G. Humberson, B.A.

Texas State University-San Marcos

May 2009

SUPERVISING PROFESSOR: MARK FONSTAD

Streams are a dynamic part of the Earth's landscape. The altered hydrologic and sedimentologic regimes caused by urbanization bring about changes in channel morphology that are problematic for urban stream management. The amount of time an urban stream needs to stabilize within the new hydrologic and sedimentologic regimes depends upon local variables, but some streams remain unstable 50 years after urbanization has started. In some cases, stabilization may not be possible. The Cellular Automaton Evolutionary Slope and River (CAESAR) model, which is a cellular-automata (CA) landscape evolution model, can help stream managers understand channel evolution over long time periods. Landscape evolution models typically come in two forms, CA based models or vector based models. CA landscape evolution models are capable of routing sediment in single and multi-threaded channels, creating

channel scour and deposition, but traditionally have not been able to simulate the lateral migration of a stream. Vector based landscape evolution models, on the other hand, have traditionally been able to simulate lateral channel migration but not multi-threaded flow. Recent progress enables CAESAR to qualitatively simulate lateral channel migration with a deterministic algorithm. This study attempts to improve CAESAR's algorithm for simulating bank failure by using a mechanistic algorithm that incorporates bank slope and bank materials. The new algorithm was applied to a highly erosive reach of the Colorado River in Austin, Texas in order to assess its effectiveness. Results indicate a success, in that regional bank migration occurred as a result of cell by cell bank failure. Furthermore, the failure that occurred was qualitatively realistic. Despite the success, there is room for improvement, and future research should focus on simulating the study site with more detailed input or on incorporating more sophisticated mechanistic bank failure algorithms.

CHAPTER I

INTRODUCTION

Throughout history, people have managed streams with a partial understanding of stream behavior. Over the long-term, this management has led to modern-day issues such as ecosystem fragmentation by dams, reduced freshwater reservoir capacity due to sedimentation, poorly executed river restoration plans, and the destruction of private property through bank erosion. As urban populations continue to grow, the number of streams subject to management is increasing. In 1952, New York City was the largest city in the world with a population less than eight million. By 2004, 17 cities had a population of at least eight million, the largest of which (Shanghai) had a population exceeding 14 million (Chin 2006).

Streams' morphologies are constantly adjusting to increases and decreases in both flow and sediment supply. For example, an influx of sediment without an increase in stream flow will lead to sediment deposition onto the streambed. This deposition results in the creation of bedforms and an increase in the stream's slope, which increases the stream's capacity to move the sediment. Conversely, an increase in flow without an increase in sediment supply would result in erosion, channel enlarging, and eventually a decrease in channel slope, which lessens the stream's ability to move sediment. The

process in which a stream attempts to balance the amount of erosion and deposition taking place is often referred to as dynamic equilibrium. A significant change to a stream's hydrologic or sedimentologic regime can upset channel equilibrium, and result in significant changes in the channel's morphology until a new equilibrium is achieved. It should be noted that the term dynamic equilibrium is used cautiously. Streambeds will progressively degrade during geologic time scales, which is a direct contradiction of the term equilibrium (Wolman 1967). However, dynamic equilibrium is used in this paper to describe the relative stability that stream channels can have over shorter time scales.

Rapid urban growth alters urban channel morphology through a disruption of the stream's equilbirium. In 1967, Wolman outlined three stages of an urban stream that reflects the urbanization process,

> 1) an initial stable or equilibrium condition in which the landscape may either be primarily agricultural or dominated by forests,
> 2) a period of construction during which bare land is exposed to erosion, and
> 3) a final stage consisting of a new urban landscape dominated by streets, rooftops, gutters, and sewers. (Wolman 1967, p. 386)

Wolman's second stage is associated with an increase in sediment being delivered to the channel, which can create channel bars that focus the stream flow into a number of smaller channels within the larger channel. If these channel bars become stabilized by vegetation, they reduce the volume of water the channel conveys, which can cause increased flooding (Wolman 1967). The third stage, on the other hand, is associated with reduced sediment supplies as impervious cover increases. With more impervious cover, runoff increases and has the potential to scour any excess sediment in the stream channel deposited during the second phase. The third phase leads to progressive erosion

without accompanying deposition, which can result in channel widening or incision (Wolman 1967). Chin (2006) summarizes Wolman's stages in Figure 1.

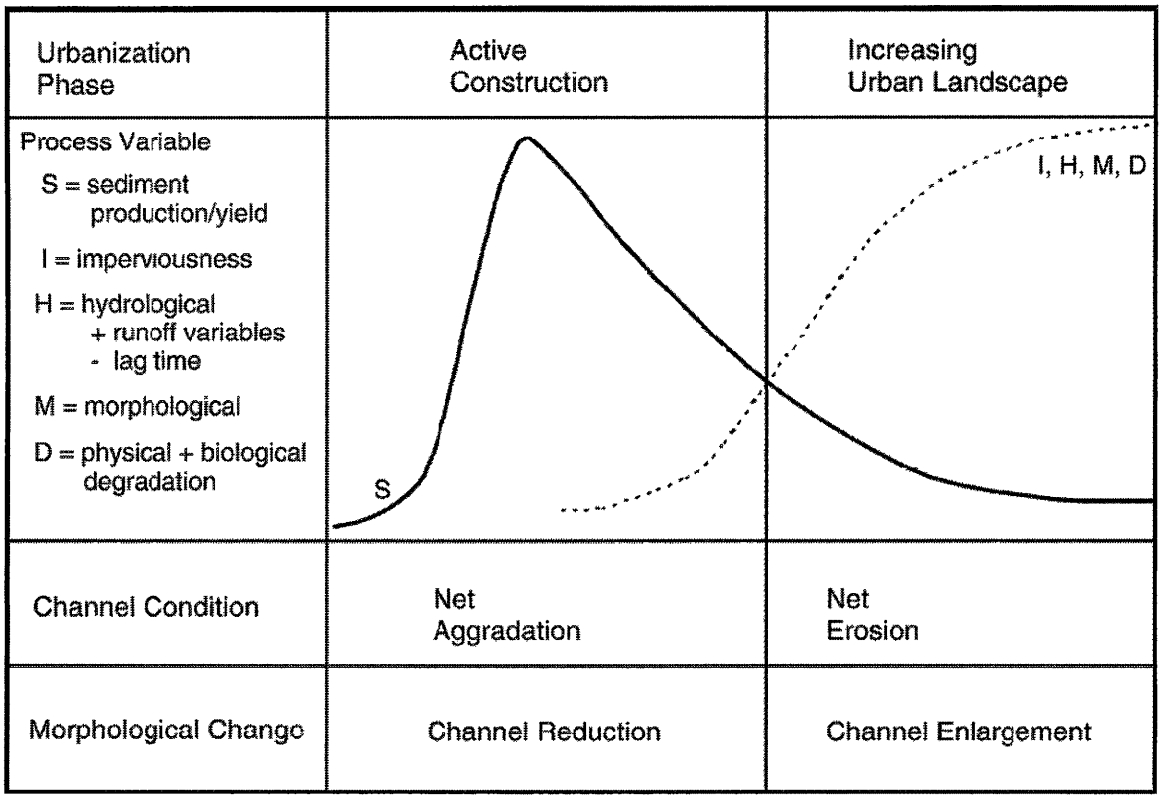


Figure 1. Graphic Representation of Urbanization Effects on Stream Channels (Chin 2006). Curves are approximate.

Wolman's three stage model was published in 1967, and analysis of over 100 studies conducted across the world since 1956 show that changes in urban stream morphology as a response to altered hydrologic and sedimentologic regimes generally agree with Wolman's model. Some of the studied streams had reached a new equilibrium, but this is dependent on several factors and may not always be possible (Chin 2006). These factors include the underlying geology, regional climate, vegetation, soil properties, ongoing construction, and urban structures like road crossings and channelization.

Understanding the evolution of urban streams is crucial to proper stream management, especially since the original channel morphology may not be sustainable in an urban environment. For example, stream bank stabilization may not be appropriate in a channel that is still enlarging as a response to urbanization (Chin 2006). However, the spatial variability and long time-scales involved with a stream's geomorphic response to urbanization present a challenge for stream managers.

Several models have been developed to aid in urban stream management. The Hydrologic Modeling System (HEC-HMS) (US Army Corps of Engineers 2006a) and the Distributed Routing Rainfall-Runoff Model (DR3M) (US Geological Survey No Date), for example, were developed to simulate channel flow response to precipitation runoff, but do not handle sediment routing or geomorphic changes. Other models, such as the River Analysis System (HEC-RAS) (US Army Corps of Engineers 2006b) and a model developed by Resource Management Associates and the Waterways Experiment Station (RMA2 WES) (US Army 2006a), were created to generate water surface profiles and stream velocities. These have simulation capabilities for sediment transport (RMA2 WES requires the use of a two-dimensional, vertically averaged sediment transport model named SED2D WES [US Army 2006b]). However, the documentation for version 4.0 of HEC-RAS states that HEC-RAS will only erode and deposit sediment within fixed lateral boundaries (US Army Corps of Engineers 2006b). As a result, the model is incapable of simulating lateral channel migration, which limits the time period of effective simulation. SED2D WES will alter the bed geometry, but assumes that changes in the bed are not significant enough to affect flow (US Army 2006b). Thus, once the channel has changed enough to significantly affect flow, the user's manual

recommends that the SED2D WES simulation be stopped, and RMA2 run again to generate new stream profiles before continuing the SED2D WES simulation. Stopping a model for every channel change capable of affecting flow is not conducive to a long-term simulation. All of these models are valuable for their designed functions, but to further our understanding of the evolution of urban streams a model capable of routing sediment and changing stream morphology over long periods of time is required. This role is particularly well suited to the Cellular Automaton Evolutionary Slope and River (CAESAR) model, which is a cellular-automata (CA) landscape evolution model developed by Dr. Coulthard of the University of Hull (Coulthard and Van De Wiel 2006b). This study proposes to investigate the applicability of CA models to urban stream management.

Landscape evolution models can further our understanding of long-term stream behavior (Nicholas 2005). Traditionally, these models have come in two flavors. Vector based models, which have had success in simulating the meandering effects of stream-bank erosion but fail to simulate braided channels, and CA models, which have succeeded in simulating braided channels but have failed to simulate meandering (Coulthard and Van De Wiel 2006a). The ability to combine both features into a single landscape evolution model, which has been called the 'holy grail' of planform modeling (Paola 2000 in Coulthard and Van De Wiel 2006a), would further our understanding of stream systems.

Recently, Coulthard and Van De Wiel (2006a) have had some success incorporating meandering behavior into CAESAR with a novel approach that uses a stream's radius of curvature and a lateral erosion constant. This study attempts to build

on Coulthard and Van De Wiel's algorithm by incorporating bank materials and bank slope. To assess the effectiveness of these modifications, the modified algorithm is applied to a highly erosive reach of the Colorado River in Austin, Texas where massive bank erosion has already caused the abandonment of a trailer park. The main objective of this study is to determine whether or not banks that are failed mechanistically on a cell by cell basis display realistic bank migration. This study simulates the lateral migration of a river's cut-bank from a planform point-of-view within the Cellular Automaton Evolutionary Slope and River (CAESAR) model by incorporating the physical mechanics of bank failure, rather than using a deterministic algorithim dependent on stream curvature and near-bank stream velocities.

CHAPTER II

MODEL BACKGROUND

Recently, cellular-automata (CA) models have been used to fill a niche between models based on computational fluid dynamics (CFD), which are often too complex for applications beyond single flood events, and long-term landscape evolution models that are of coarse resolution and meant to simulate processes spanning thousands of years (Coulthard and Van De Wiel 2006b). Practioners of CA modeling are often criticized for employing non-traditional methods, however some CA models are effective enough to influence public policy (Fonstad 2006). Since the impacts of urban development on stream channels often take several decades to fully manifest (Chin 2006), CA models can supplement traditional modeling techniques for predicting urban stream channel behavior. CAESAR has proven to be a versatile CA model, and has been applied to several areas ranging in size from a single 500m stream reach to catchments that are $500km^2$ in area. Furthermore, it has also been used to simulate periods of time ranging from individual floods to 10,000 years, and has used grids of varying cell sizes ranging from 2mx2m to 50mx50m (Coulthard and Van De Wiel 2006b).

Werner (1999) states that natural patterns, although deceptively simple (e.g. meandering river bends, dunes across a desert, or shoreline bedform patterns), are the result of complicated processes. Firstly, they are driven by nonlinear processes that

modify the local environment (Werner 1999). For example, studies show that for a given stream discharge, there is a non-linear relationship between the amount of sand on a gravel streambed and the volume of gravel mobilized (Wilcock and Crowe 2003). Secondly, they are in open systems that are driven by the exchange of energy, material, or information across their boundaries (Werner 1999), such as a stream receiving sediment from long-term hillslope erosion. These complex processes are a challenge for models, especially the coupling of processes that operate over different time-scales (Coulthard et al. 2002; Werner 1999).

CFD models, which are based on solutions to partial differential equations (Nicholas 2005), are able to simulate flow dynamics at a high level of detail. However, their computational complexity limits the areal extent of the simulations (Coulthard et al. 2000). Additionally, their reliance on boundary conditions, such as predefined sediment and stream discharges for the upstream reach boundary, limits their ability to operate over long time periods because the boundary conditions themselves change as the environment evolved (Coulthard et al. 2000). While these models are excellent for certain purposes (e.g. calculating a detailed stream surface profile during a flood), a more holistic view is needed to further our understanding of urban stream evolution.

Landscape evolution models, on the other hand, have been developed to operate over large areas and investigate the interconnections between hydrology, fluvial erosion, lithology, climate, land use, tectonics, and catchment evolution (Coulthard 2001). Many of these models represent the landscape through a grid or a series of connected nodes, and route water down the steepest slopes. Unfortunately, routing water down the steepest slope means precludes divergent flow (Coulthard 2001; Coulthard et al. 2002). As a

result, simulation of features such as mid-channel bars and braiding is not possible (Coulthard et al. 2002). Landscape evolution models also tend to rely on average erosion rates over long time steps in order to handle large time scales. This averaging leads to a loss of detail for large individual events (Coulthard et al. 2002). The loss of these details may be suitable when investigating hundreds of thousands of years of catchment evolution, but large individual events (e.g. a large pulse in sediment supply) and divergent flow often characterize evolving urban streams (Chin 2006).

The model developed by Murray and Paola (1994) was groundbreaking in that it created qualitatively realistic braiding without resorting to detailed modeling of every process in a river channel (Coulthard and Van De Wiel 2006b; Nicholas 2005). Murray and Paola (1994, 1997) used a simple CA approach that calculated sediment transport based on water discharge routed over local topography. In a CA model, the landscape is represented by a grid whose cells interact with each other according to rules based on an approximation of the governing physics (Murray and Paola 1994; Nicholas 2005). Murray and Paola (1994, 1997) created an intuitive graphic that illustrates the rules used in their model (Figure 2).

Braided rivers are extremely dynamic, and have water and sediment flow rates that are constantly fluctuating. Since the goal of Murray and Paola's model was to learn what conditions were essential to creating a braided river, using a CA model that excluded many of the fluvial processes was appropriate (Murray and Paola 1997). Although a brief description of their model is included here, it is recommended that readers consult Murray and Paola (1997) for a detailed description.

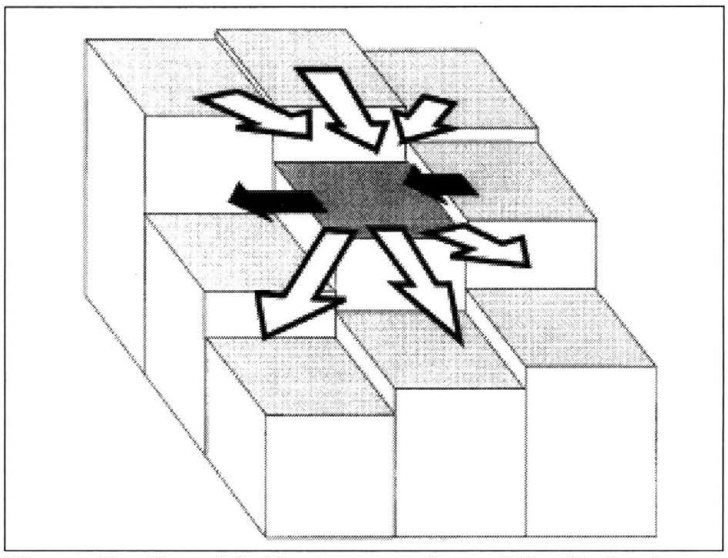


Figure 2. Graphic Representation of the Rules in the Cellular Automaton Model Developed by Murray and Paola (1994). The white arrows represent how water and sediment transportation are routed for the center cell with the magnitudes represented by arrow length. The black arrows represent lateral sediment transport.

In Murray and Paola's model, water is routed through a grid that is "typically tens of cells wide by hundreds or thousands of cells long" (Murray and Paola 1997, p. 1002). Each cell contains information regarding bed elevation, water discharge, and sediment discharge in arbitrary units. The initial condition for their model is a uniform slope with white-noise elevation perturbations (Murray and Paola 1994; Murray and Paola 1997). Each iteration of the model starts with the introduction of water at the upstream end the grid. This water is routed downstream cell by cell while moving sediment. The iteration ends when the water reaches the downstream end of the grid, and the cells' elevations are adjusted based on the net difference between the amount of sediment entering and leaving a cell (Murray and Paola 1997).

Water is routed from each cell to its three downstream neighbors based on locally determined bed slopes. The equations used ensure that all water is routed to the next row, and discharge is distributed so that areas with the steepest positive slopes receive the most water. Should the model encounter a situation where all three downstream neighbors have a higher elevation than the originating cell, the water is routed uphill in a manner that allocates the least amount of water up the steepest slopes. While the idea of water flowing uphill is counterintuitive, flow uphill has been observed in real streams when either the water slope is positive or the water momentum is high (Murray and Paola 1997).

Several sediment transport relations were tested in Murray and Paola's model, but a common factor for all the equations was that sediment transport rate was proportional to water discharge. Therefore cells with more water flowing out had more sediment transport out, and conversely areas with less water flowing out had less sediment flux out (Murray and Paola 1997). The model incorporates a component to route sediment laterally downhill, which allows the braided channels some freedom to migrate (Murray and Paola 1997).

Despite Murray and Paola's breakthrough, there remain limitations in their model. Doeschl-Wilson and Ashmore (2005) compared results from small-scale laboratory flume experiments to results derived from Murray and Paola's model. During the flume experiments, high resolution cellular digital elevation models (DEM) of the sediment bed were created, and then used as the initial input in the Murray and Paola model. Their results show that although the model is able to create braiding from a uniform slope, it fails to maintain braided river behavior that adapts and creates new

morphologies. Instead, the existing features tend to either increase in size or depth, or eventually merge together. Also, Doeschl-Wilson and Ashmore state that despite the model's qualitatively realistic braiding, it did not recreate quantitatively realistic representations of their flume results in terms of spatial, topographic, and temporal scale. The model's dependence on local bed slope, rather than on actual water surface slope, results in exaggerated discharge in areas of low elevation. Another interesting result was that using DEMs with finer cell resolution created worse results than DEMs consisting of coarser cells. According to Doeschl-Wilson and Ashmore, this is related to the model's use of simplified equations and its dependence on bed slope. With a finer resolution DEM the model is affected by local elevation extremes, whereas in a coarser resolution DEM, these extremes are smoothed out by averaging over a larger cell area. A simplified topography is better suited for the simplified equations used by Murray and Paola (Doeschl-Wilson and Ashmore 2005). Murray and Paola's model only supported flow in one main direction, making it unsuitable for simulating braided river catchment evolution (Coulthard et al. 2002).

When considering Murray and Paola's model's inability to recreate completely realistic braiding, it is important to remember that their goal was to determine the minimal conditions needed to cause braiding (Murray and Paola 1997). They found that these conditions include excessive scour, excessive deposition and unconstrained lateral flow (Murray and Paola 1994). Due to their simplified approach it is difficult to compare the results to nature, but they are consistent with three known limiting cases:

> 1) networks formed by purely erosive processes are dendritic rather than braided,
> 2) braided rivers do not form in effectively cohesive materials where there is no bedload transport, and

> 3) fully developed meandering cannot develop without some form of bank stabilization to constrain the flow laterally. (Murray and Paola 1994, p. 57)

By using abstractions of the underlying physics, Murray and Paola were able to simulate a process for which exact physical equations do not exist (Doeschl-Wilson and Ashmore 2005). With their novel approach and groundbreaking results, Murray and Paola's model validated the pursuit of fluvial simulations that did not simulate every physical processes within a river channel at a minute level (Coulthard and Van De Wiel 2006b).

Murray and Paola's (1994) work inspired the creation of CAESAR, which built on Murray and Paola's routing scheme (Coulthard and Van De Wiel 2000b). CAESAR was initially developed to simulate the effects of land cover and climate change on river catchment evolution in the UK (Coulthard and Van De Wiel 2000b), and has demonstrated the ability to show the detailed formation of channel bars, braided channels, channel avulsions, and channel change (Coulthard et al. 2000). Significant differences between CAESAR and Murray and Paola's model include the ability for flow routing in all directions within CAESAR, and each grid cell in CAESAR has local values for bed elevation, water discharge, flow depth, grain size distribution, and vegetation cover (Coulthard et al. 2000). At this point it is prudent to note that CAESAR's code is open-source, and has had many additions and improvements since its original incarnation. As a result, the following description of its processes may be out of date.

To route flow in all directions, CAESAR scans the model in four directions rather than just the one direction used by Murray and Paola (Coulthard et al. 20002). In the first scan, the model moves row by row from left to right, and proportionally routes water from a source cell to its neighbors. The model repeats this process in three successive scans, one moving from right to left, another from top to bottom, and lastly from bottom

to top. Coulthard et al. (2002) have created a graphic that illustrates this process (Figure 3). The algorithm used by Coulthard et al. (2002) combines bed elevation and flow depth to determine if the water will be routed to a cell. Unlike Murray and Paola, this algorithm does not guarantee that all water is routed downstream; instead it allows the water to become trapped and flow around obstacles (Coulthard et al. 2002).

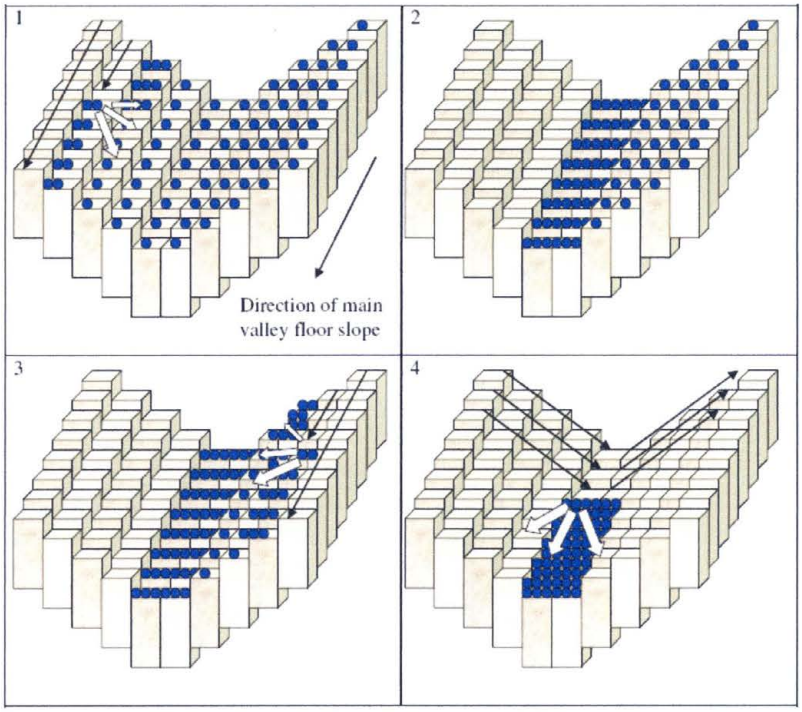


Figure 3. Graphic Representation of CAESAR's Four Part Routing Process (Coulthard et al. 2002). The three dimensional cells represent a simple valley, and the circles represent water that is being routed as the scans are performed.

In order to simulate catchment evolution over long periods of time, CAESAR captures the many feedbacks and interactions between physical processes (Coulthard and Van De Wiel 2006b). These interactions are difficult to simulate because different catchment processes have different levels of influence depending on the time scales of

interest. For example, soil creep may be negligible during a series of floods, but could be a major cause of hillslope erosion over thousands of years. Conversely, the spatial distribution of channel bars could be very important for a single flood event, but not so important when simulating over thousands of years (Coulthard et al. 2002). CAESAR handles these dichotomies by focusing on cells where processes are mostly fluvial, and only periodically evaluating hillslope processes. This procedure is accomplished by identifying which cells are wet, and then creating a buffer zone of dry cells around the wet cells. Areas within the buffer are calculated at every model iteration, and areas outside of the buffer are recalculated every 100 years of simulation time. The buffer zone is intelligent, and will grow to accommodate larger flow events (Coulthard et al. 2002).

The model uses a variable time step that is related to the volume of erosion and deposition. The variable time step limits the amount of erosion that can occur in a single model iteration to 10% of the local slope (Coulthard and Van De Wiel 2006b). As a result, during a flood where large amounts of sediment are mobilized, the time step can be as small as a fraction of a second and detailed information can be captured during the entire event (Coulthard et al. 2002). For low flows with little to no sediment erosion, the time step can increase to approximately one day (Coulthard and Van De Wiel 2006b). Regulating the amount of erosion that can occur with a variable time step also prevents computational instability (Coulthard et al. 2000; Coulthard and Van De Wiel 2006b). The processes described above allow CAESAR to capture detailed behavior for large single events, but still maintain the ability to simulate processes that are relevant for timescales ranging over thousands of years (Coulthard et al. 2002).

CAESAR consists of five main processes. The first is the hydrological model, which generates a total discharge value that represents both surface and subsurface discharges. The value of total discharge is based on rainfall and soil moisture storage (Coulthard et al. 2002). Since a cell within the buffer zone is able to receive water from cells outside the buffer zone, an algorithm that multiplies the total discharge value for a buffer cell by the number of outside cells that drain into it is used. The equations used to calculate the total discharge are derived from a modified version of TOPMODEL (Beven and Kirkby 1979 in Coulthard et al. 2002).

CAESAR's second process handles discharge routing. A threshold value is calculated for each cell in the buffer zone. This threshold determines what fraction of the total discharge (calculated in the first process) should be treated as subsurface flow, and what fraction should be treated as surface flow. Similar to Murray and Paola's (1994) model, subsurface flow is routed from a cell to downstream neighbors (between 0-3) in proportion to their local bedslopes. In order to route surface flow, depth is calculated based on the surface discharge using a rearrangement of the Manning's equation. Then, CAESAR uses a combination of flow depth and bed elevation to proportion surface discharge from the originating cell to its three downstream neighbors (Coulthard et al. 2002).

CAESAR's third process involves fluvial erosion and deposition. CAESAR represents sediment stratigraphy with 12 layers that consist of 9 different grain sizes. One layer is used to represent bedload, which is the amount of sediment mobilized by surface flow but not permanently suspended in the water column. Another layer represents the surface active layer, which is the thickness of the bed surface that interacts

and exchanges material with the bedload. The remaining 10 layers represent the subsurface. By describing the stream bed with multiple layers, the model can simulate channel surface armoring (Coulthard et al. 2002). Armoring occurs when the surface consists of coarser sediment than the subsurface, with the larger surface grains hiding the smaller grains below. Another benefit of this layered approach is that previously deposited sediment can serve as future erosion sources (Coulthard and Van De Wiel 2006). Currently, CAESAR uses the Wilcock and Crowe (2003) equation to drive fluvial erosion and deposition.

The Wilcock and Crowe sediment transport model was created for mixed sand and gravel sediments, and represents a breakthrough in calculation of bedload sediment transport. Many previous models were based on the bulk grain size distribution with the assumption that streams with similar grain size distributions would have similar transport rates in similar flows (Wilcock and Crowe 2003). However, the substrate does not interact with the flow, and is the result of historical depositional processes that are not accommodated in a sediment transport model (Wilcock and Crowe 2003). The bed surface, on the other hand, is the result of current processes and provides the grains available for transport. As a result, a transport model based on the surface grain size distribution has the ability to predict sediment transport rates in transient bed conditions such as a bed that is armoring, degrading, or aggrading (Wilcock and Crowe 2003).

Surface based transport models existed prior to Wilcock and Crowe (Parker 1990; Proffitt and Sutherland 1983). However, the Wilcock and Crowe (2003) model incorporates the nonlinear effect of sand content on gravel transport rates. Furthermore, Wilcock and Crowe's model is the result of observations of flow, transport, and surface

grain size in 48 flume runs using 5 different mixtures of gravel and sand. Wilcock and Crowe state that "this is the first relatively comprehensive record of surface-based transport observations with an unambiguous measurement of surface grain size for a range of sediment, flow, and transport rate" (Wilcock and Crowe 2003, p. 127).

Another feature of the model is the incorporation of a hiding function. While hiding functions are found in several transport models, it is important to mention its presence since it causes smaller sediments to require larger shear stresses to be mobilized (Wilcock and Crowe 2003). Essentially, the hiding function simulates the effect of smaller grains being hidden from stream flow by surrounding larger grains. The model is also easy to use since it requires the user to specify only the current shear stress and surface size distribution to predict a bedload transport rate (Wilcock and Crowe 2003).

CAESAR's fourth process incorporates slope processes, including mass movements and soil creep. Mass movement occurs in CAESAR when a local slope exceeds a user specified threshold (e.g. 45 degrees). To simulate mass movement, CAESAR shifts soil from the uphill cell to the downhill cell until the slope no longer exceeds the threshold. CAESAR then checks cells uphill of the original mass movement location to ensure that soil shifting did not destabilize uphill slopes. If an unstable slope is created, CAESAR repeats the procedure of shifting soils until all slopes are stable. Unlike mass movement, which is treated as an instantaneous transfer of material, soil creep is calculated on a monthly basis, the amount of creep that occurs is proportional to the local slope. In both slope processes, material is transferred by moving soil between the active layers of the contributing and receiving cells, which allows hillslope processes to supply sediment to the stream channel (Coulthard et al. 2002).

CAESAR's final process involves vegetation cover, which is modeled in two ways. The first method involves simulating drastic changes in land cover (e.g. deforestation), and is simulated by modifying the hydrological model. This modification affects the amount of surface runoff that is generated, which in turn will affect stream processes. The second method involves simulating a vegetated layer in the channel, which reduces the potential for bed erosion. Vegetation protects the bed surface from erosion due to turbulent bursts and sweeps in the channel flow, and this erosion-resistant vegetation is simulated by adding an extra fraction to the active layer whose resistivity is equivalent to a boulder 0.26m in diameter. This value is based on a study by Prosser (1996 in Coulthard et al. 2002). Unlike the other sediment layers, once the added vegetation layer is eroded, it can never be deposited. CAESAR also incorporates a simple vegetation growth procedure that allows vegetation to slowly accumulate on exposed surfaces for 10 years. After ten years, the model treats the surface as fully vegetated (Coulthard et al. 2002).

A recent addition to CAESAR is the ability to simulate river meandering (Coulthard and Van De Wiel 2006a). Although this ability has existed in vector based landscape evolution models, cellular automata (CA) models traditionally have not had this capability (Coulthard and Van De Wiel 2006a). Although the exact causes of meandering are still debated, Coulthard and Van De Wiel (2006a, p. 123) state that "it is now accepted that secondary circulation, i.e. flow not parallel to the main channel direction, can preferentially erode one river bank and deposit material at the toe of the other." One reason that CA models have been unable to simulate meandering is due to their simplified approach. By relying on empirically derived equations, such as the

Manning's equation, past CA models have not incorporated the details regarding secondary circulation (Coulthard and Van De Wiel 2006a).

Approaches used in vector based landscape evolution models include using regional properties such as a stream's radius of curvature or cross-channel gradient to drive meandering. This approach is not straightforward to implement in CA models, whose cells operate with only knowledge of themselves and their immediate neighbors (Coulthard and Van De Wiel 2006a). Coulthard and Van De Wiel (2006a) developed an algorithm that provides grid cells a value for radius of curvature and enables meandering. First, the algorithm determines which cells are edge cells, wet cells, and dry cells. Second, the algorithm passes a nine (3x3) cell filter over the entire grid. Whenever the center cell is an edge cell, the number of wet cells is subtracted from the number of dry cells. Third, the resultant value is assigned to the edge cell in question, and its magnitude represents the local radius of curvature while its sign indicates whether or not it is an inside or outside bend. Coulthard and Van De Wiel (2006a) use these values to drive erosion along the outside bends and deposition along the inside bends of the meanders.

This new meandering algorithm has not been quantitatively assessed. While problems exist with deposition along point bars in meander bends, it creates qualitatively realistic meandering with greater erosion rates at tighter bends (Coulthard and Van De Wiel 2006a). Coulthard and Van De Wiel (2006a) identify several areas for future algorithm development, including but not limited to 1) improvement of the routine responsible for deposition along the inside of a meander bend and 2) calibration of the model and validation of its migration rates and planform sinuosities.

In addition to its limited capability to simulate channel meandering, CAESAR has some other shortcomings. For example, it is incapable of simulating streams whose widths are smaller than the width of one grid cell. Thus it is impossible to simulate the effects of rills and gullies (Coulthard et al. 2000). The model also does not support spatial variation in vegetation cover or rainfall, which may or may not be a problem depending on the area simulated (Coulthard et al. 2000). Furthermore, it is often difficult to validate model results with field measurements. There is a shortage of reliable continuous bedload transport and erosion data, although historical water discharge is well documented in many streams (Coulthard and Van De Wiel 2006b). In order to assess Murray and Paola's model, Doeschl-Wilson and Ashmore (2005) resorted to creating DEMs of a laboratory stream to overcome the challenge of nonexistent continuous topographical data. Alternative approaches to model validation include comparison of model outputs to empirical data (Coulthard and Van De Wiel 2006b; Nicholas 2005), comparisons of model outputs to alternative modeling approaches (Nicholas 2005), and comparisons of model outputs to historical maps, aerial photographs, and topographical data (Coulthard and Van De Wiel 2006b).

Despite CAESAR's shortcomings and the difficulty associated with validating its output, it is important to remember that all models are approximations of reality. CFD models lose the ability to study large areas over long periods of time in exchange for the ability to produce extremely detailed results within a stream reach, and CAESAR loses the ability to produce detailed depth and flow outputs in exchange for the ability to simulate large areas for long periods of time (Coulthard et al. 2000). As such, it is not prudent to take the absolute values produced by CAESAR as a valid representation of

reality, however CAESAR is a valid tool for examining relative changes (Coulthard and Van De Wiel 2006b). Coulthard and Van De Wiel state that CAESAR's greatest strength is its ability to simulate patterns of erosion and deposition, and that "it can consistently demonstrate which zones or areas of a river are more likely to be eroding and incising or depositing and possibly unstable" (Coulthard and Van De Wiel 2006b, p. 117).

CHAPTER III

STUDY SITE BACKGROUND

The Colorado River in Texas can be described as a low sinuosity, coarse grained meander belt (McGowen and Garner 1970 in Sears 1978). The Colorado River's drainage area is approximately 109,603 square kilometers and it spans the states of Texas and New Mexico (Blum 1992) (Figure 4). Blum (1992) defines the upper Colorado River's boundaries as extending until the Balcones Escarpment, which is a fault-line escarpment created by echelon normal faults during the Miocene. Downstream of the escarpment is the lower Colorado River basin, which lacks any major tributaries and is characterized by a much narrower drainage basin when compared to the Upper Colorado (Blum 1992; Sears 1978). As a result of this drainage basin morphology, the lower Colorado has historically behaved as a conduit that stored and periodically delivered sediment that is predominantly from the upper Colorado to the Gulf of Mexico (Blum 1992). The bedload of the Colorado River is dominantly coarse grained, with silts and clays accounting for only 10 percent (Sears 1978).

Geomorphic analysis of the alluvial terrace and valley fill sequence of the lower Colorado River first began over 100 years ago with mapping efforts sponsored by the U.S. Geological Survey (Hill and Vaughan, 1897;1902 in Blum 1992). These studies and those that followed focused on identifying terrace surfaces, now called

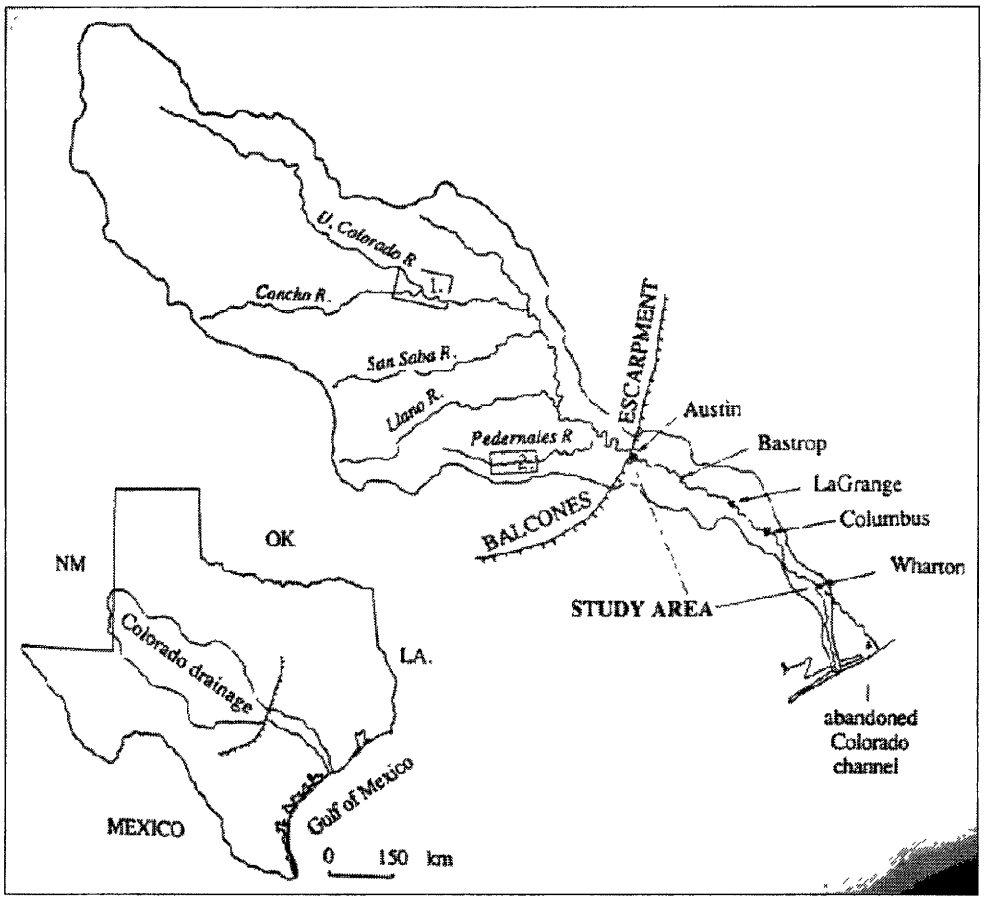


Figure 4. Colorado River Drainage Basin (Blum 1992).

morphostratigraphic units (Blum 1992). Three classifications of terrace/alluvial deposits were derived from these studies:

> 1) The "Uvalde Gravels", which consist of extensive dissected surfaces underlain by chert-dominated gravels, strongly cemented by secondary carbonates, that occur at elevations up to 250 feet above the bed of the modern Colorado River near the Balcones Escarpment;
>
> 2) The "Asylum and Capitol terraces", which consist of extensive partially dissected surfaces underlain by deeply leached, non-calcareous gravelly and sandy sediments...and occur at elevations of 200 and 150 feet above the present river channel; and
>
> 3) low terraces and the modern channel that are underlain by limestone and chert gravel, and calcareous mud derived from the carbonate rocks of the Edwards Plateau, gravel and sand from the Llano region, and reddish siliceous sand and mud transported downstream from Triassic and Permian redbeds in the upper part of the drainage. (Blum 1992, p. 86)

The third category is most relevant to this study due to the time scales involved.

The Colorado River has a history of very flashy discharges and highly destructive flooding, with 15 floods and millions of dollars in damage between 1843 and 1938 (Figure 5) (Lower Colorado River Authority 2007a).

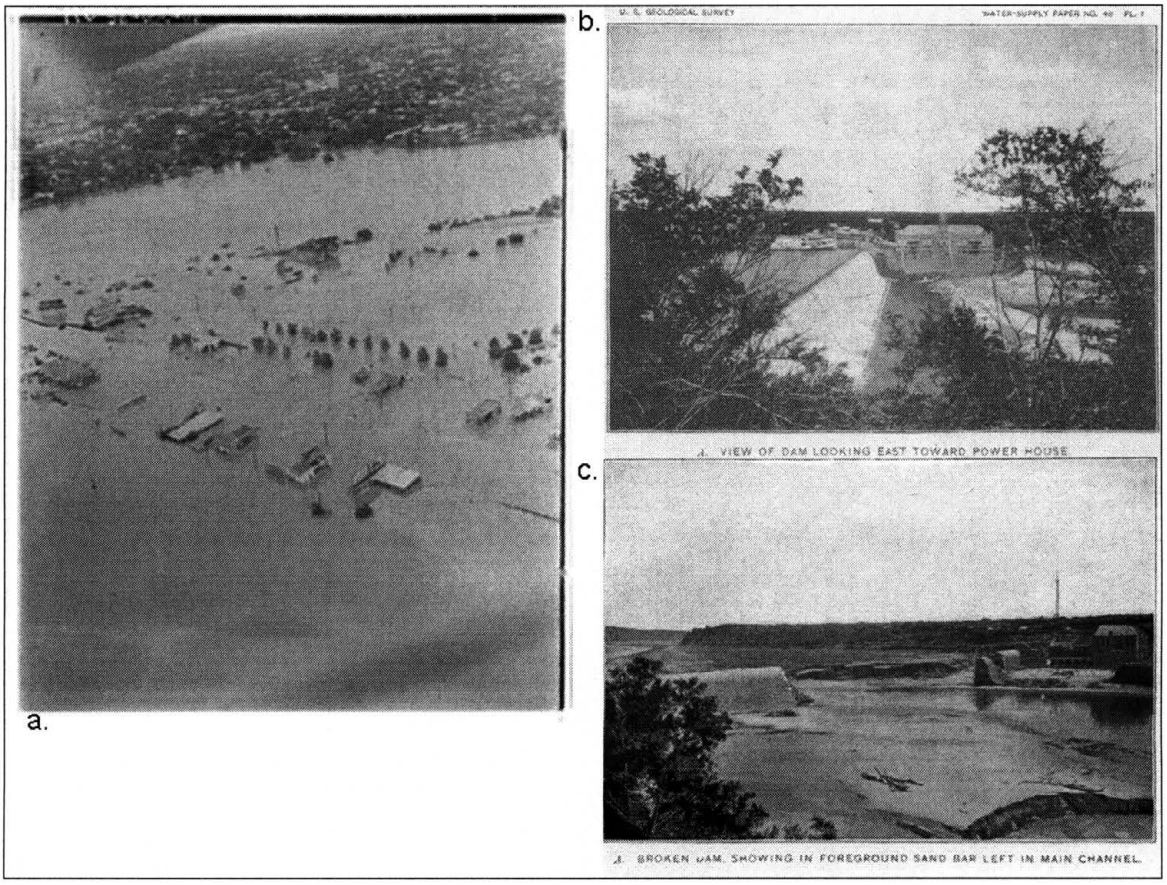


Figure 5. Various Photos Demonstrating the Historical Destructiveness of the Colorado River. Image a is from the City of Austin (2007). Images b and c are from Taylor (1900).

Figure 5a (City of Austin 2007) was taken in 1935, and looks northeast over Barton Springs Road and Riverside Drive in Austin, Texas. Figures 5b and 5c (Taylor 1900) show the Great Granite Dam, which was constructed in 1893 (City of Austin 2007), before and after the 1900 flood that destroyed it. To combat this flooding, the

Lower Colorado River Authority constructed a series of six dams, which begin in the highland lakes in the upper Colorado basin and end in Austin, Texas. These dams were completed in 1941, and in conjunction with the Longhorn Dam built by the City of Austin in 1960, they have reduced the force of major and minor floods downstream (Figure 6) (Lower Colorado River Authority 2007a; City of Austin 2007).

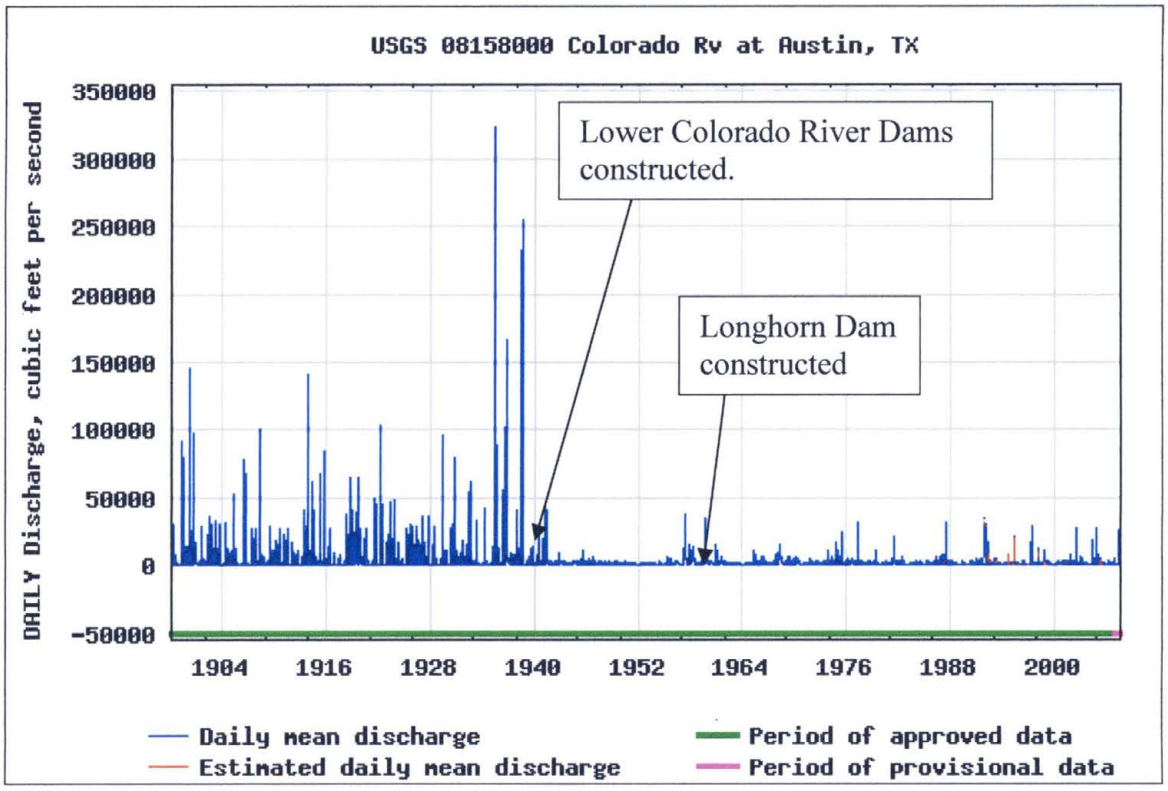


Figure 6. Historical Discharge Data for the Colorado River at USGS Gauge 08158000 in Austin, Texas (US Geological Survey 2007).

It is obvious from the hydrograph that the dams have reduced peak flows along the Colorado River by an order of magnitude, from over 300,000 cubic feet per second to approximately 30,000 cubic feet per second. What is not as obvious is how these dams prevent sediment from being transported through the lower Colorado river. Wolman's third evolutionary step for urban streams is a state in which limited sediment

input could lead to uncontrolled channel enlargement. This study focuses on a stretch of the Colorado river that begins just downstream of Longhorn dam and extends to the Highway 183 bridge, a length of approximately 1.5 miles (Figure 7).

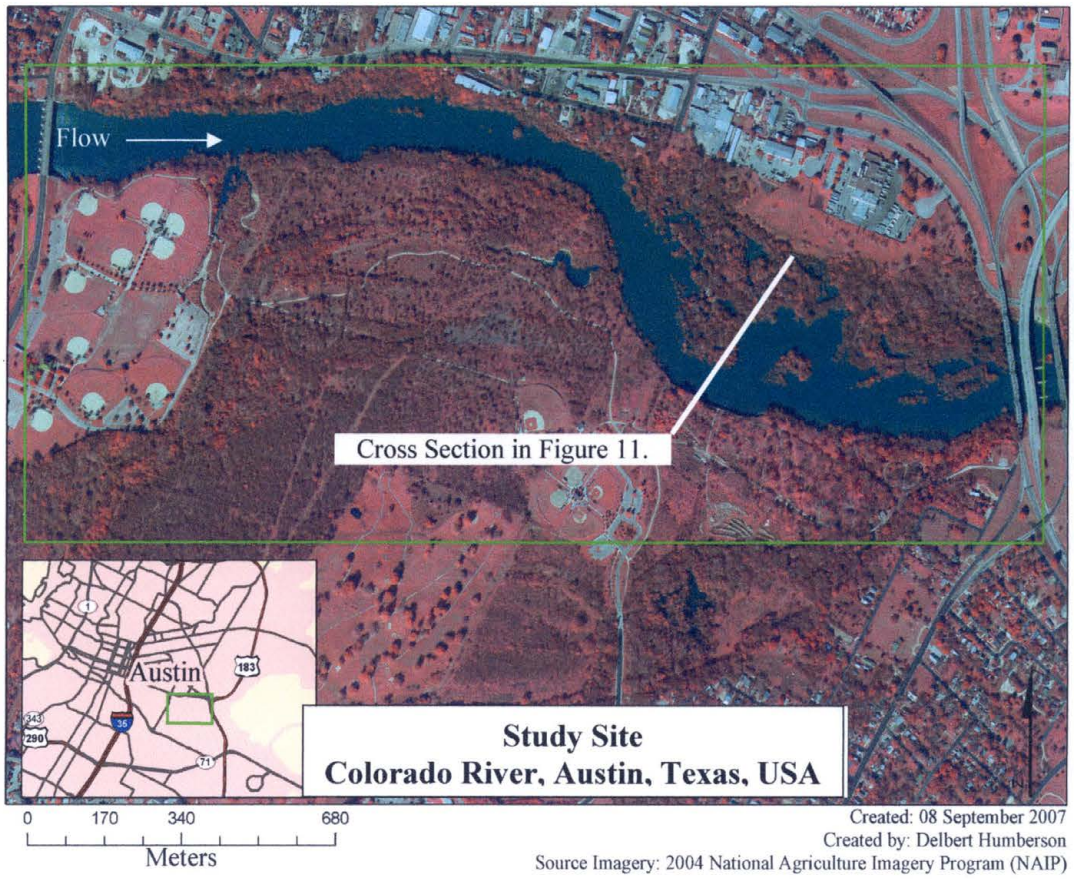


Figure 7. Study Site Location. This area is located in Southeastern Austin, Texas. The proposed study site is marked with a box.

Historical photos show the land adjacent to the stream channel was developed into a trailer park during the 1960s and 1980s, but the park was subsequently abandoned as the river eroded the banks. Figure 8 displays the bank before it was developed, and in Figure 7 the park's paved roads are all that remain (located in the southeastern portion of the study area).

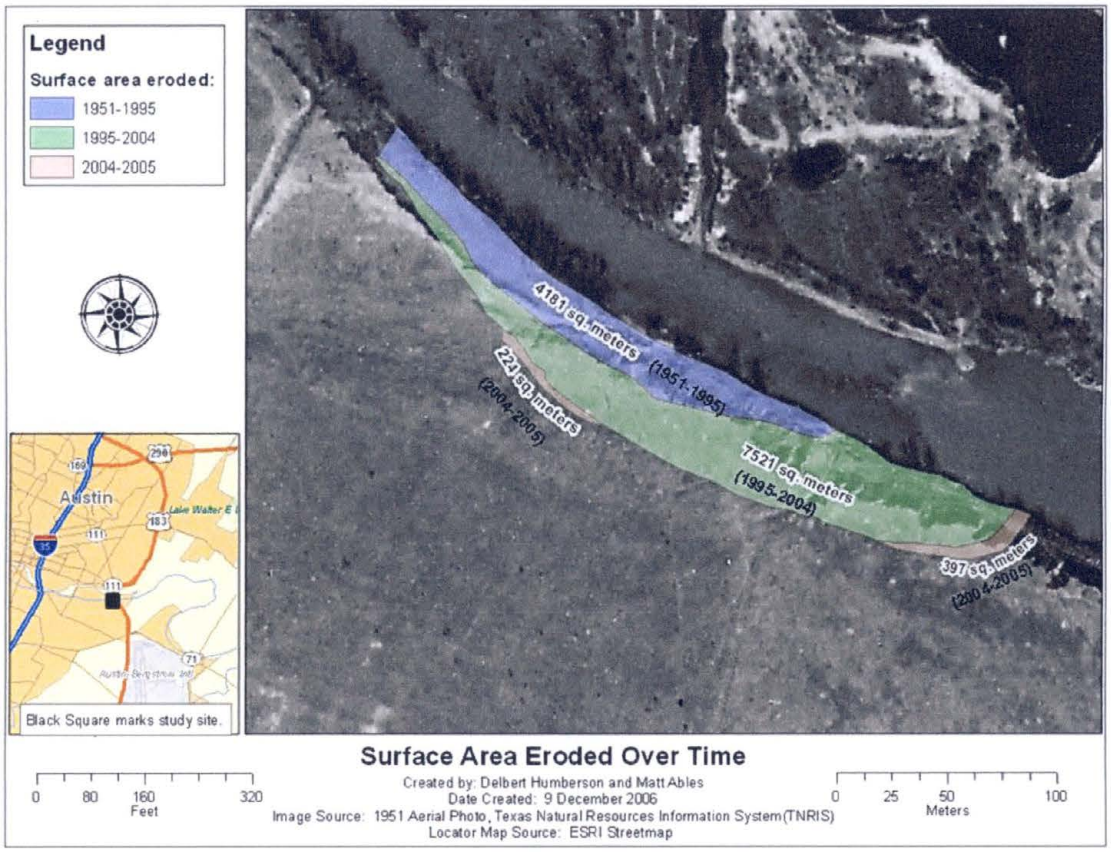


Figure 8. Cutbank Erosion Over Time. The background image was taken in 1951, and is projected in Universal Transverse Mercator (North American Datum 1983) coordinates. Areas were measured using ESRI's ArcGIS 9.0.

Analysis of historical aerial photos reveals that the channel banks near the trailer park were relatively stable during 1951-1964. Upstream erosion had started by 1980, and this erosion continued at a rapid pace through 2005 (Figure 8). The rapid retreat of the right bank may or may not be indicative of uncontrolled channel enlargement, but it does illustrate the importance of understanding the long-term behavior of urban streams.

Site visits to the stream reveal that the bed mainly consists of gravel and sand, which is consistent with the descriptions provided by Blum (1992) and Sears (1978), although some clay has been observed. Core borings taken in 1958 (See Appendix A)

indicate that the left bank at Longhorn dam consists of sandy silts, sandy clays, and silty sands. Further downstream, the river meanders to the right, so the stratigraphy for the left bank across from the abandoned trailer park may differ, especially since surficial evidence suggests the past presence of a gravel mine. The right bank near the abandoned trailer park offers an excellent view of its stratigraphy, and shows alternating layers consisting of mostly sand and gravel (Figure 9).

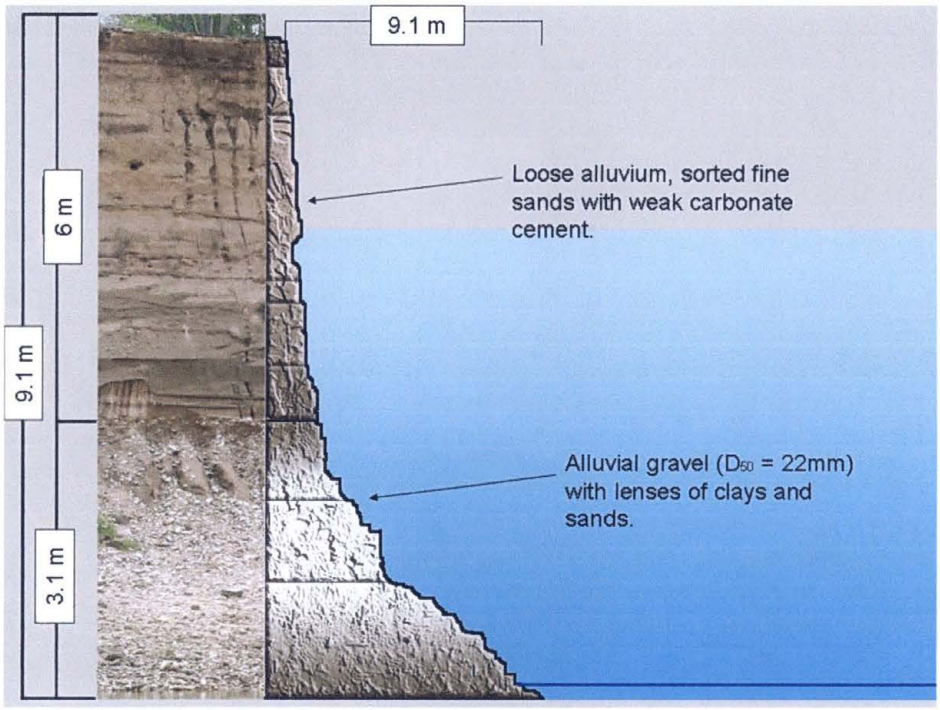


Figure 9. Stratigraphy of Cutbank Near the Abandoned Trailer Park. The left side of the image is a photo of the bank, the right side displays a rough profile.

Core borings (See Appendix A) show that these alternating layers of gravel and sand occur in the right bank at Longhorn dam, and continue inland. According to the core borings, shale underlies the channel at Longhorn dam. The study reach contains mid channel bars composed of sand and gravel with emergent vegetation. Although the

right bank is clearly identified at all flows, the left bank can change dramatically. This is due to the asymmetry within the channel (Figure 10). During low flows, the topography confines the river, but at higher flows the river spills over the left bank and into an abandoned gravel mine.

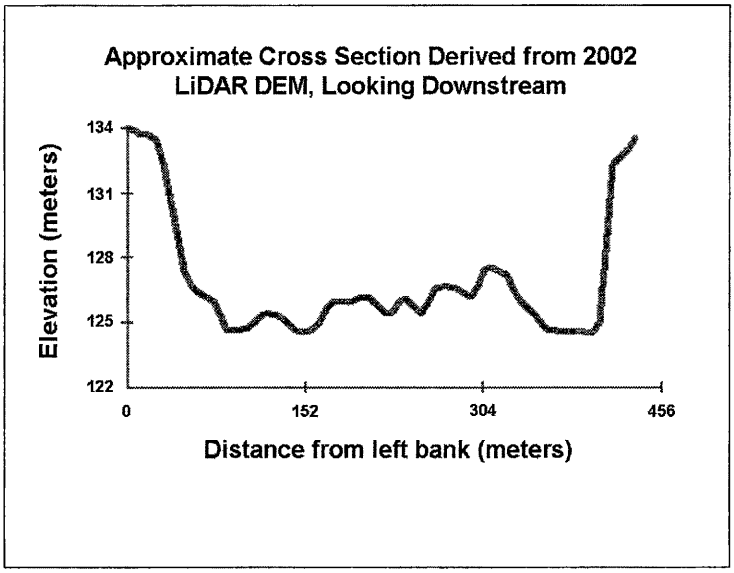


Figure 10. Cross Section of the Colorado River.
See Figure 7 for the location of this cross section.

Longhorn dam prevents bedload from being transported from upstream areas, and is the upstream boundary of the study site. This simplifies modeling the reach because bedload input to the study reach is zero. The bridge was chosen as the downstream boundary in order to avoid complexities with simulating flow around bridge piers.

CHAPTER IV

RIVERBANK FAILURE: A BRIEF OVERVIEW

Riverbank failure is a complex process that involves a combination of subaerial processes, fluvial erosion, and mass wasting (Casagli et al. 1999; Robert 2003; Wynn 2006). Subaerial processes refer to climatic influences that reduce soil strength (e.g. frost heave, soil dessication) (Wynn 2006).

Fluvial erosion occurs when flowing water entrains and transports surface particles by exerting a surface drag force on the sediment (Robert 2003; Wynn 2006). If the drag force is not sufficiently balanced by either bank strength or the sediment's gravitational force, sediment is eroded from the bank (Robert 2003). According to Wynn (2006), considerable research has been done on cohesionless soils, but results are contradictory. Shear forces are highly variable spatially, and erosion depends on bank material properties, channel curvature, and bank geometry (Robert 2003). For example, near bank velocities may be maximized at the outer banks of meander bends, an idea that is reinforced by the common observation of high bank erosion rates in these areas (Robert 2003).

As mentioned by Coulthard and Van De Wiel (2006a), secondary flow is generally accepted as the cause of meander bend migration. Secondary flow is the result

of two forces acting on the water as it enters a bend, 1) centrifugal forces and 2) pressure gradient forces (Robert 2003). The centrifugal force created by the water's momentum piles water against the outer bank of a meander bend. This piling causes the water surface to become super-elevated at the outer bend, which in turn creates a pressure gradient that dominates centrifugal forces near the bed. The overall effect is that surface water near the outside of the bend flows outward and water near the bed flows inward, resulting in a helical flow as the water moves downstream (Robert 2003). These flows not only help cause bank failure, but also help drive sediment laterally across the channel to create the point bars that are often found along the inside of meander bends (Robert 2003).

Mass wasting is the collapse of bank material as a result of the weight of the bank overpowering the shear strength of the soil (Wynn 2006). Mass wasting is a function of bank geometry, stratigraphy, bank materials and vegetation (Wynn 2006). A dominant independent parameter over shear strength is matric suction, which is the difference between air and water pressure in the porous space within soils (Casagli et al. 1999). In partially saturated soils, pore water pressure is negative and matric suction helps strengthen a bank's shear strength. Once these areas are saturated, during a flood for example, the matric suction decreases and pore water pressure becomes positive (Robert 2003). This change decreases a bank's shear strength, but as long as water remains at an elevated level, the confinement pressure the stream imposes on the bank helps prevent it from collapsing (Casagli et al. 1999). However, as the water levels decline and the matric suction remains weakened, bank failure becomes more likely (Casagli et al. 1999).

A loss of matric suction is not the only cause of mass wasting. Increases in bank height and bank angle caused by fluvial erosion of the bank toe also increases the likelihood of mass wasting (Robert 2003; Wynn 2006). Instead of bank slumping, this type of failure results in block failure.

The complexity of the processes involved in mass wasting will be a challenge to model in CAESAR, which does not currently store much of the information described above (e.g. secondary flow vectors or matric suction values). However, efforts have been made to determine bank failure or a soil's coefficient of cohesion by using simple parameters such as bank geometry, flow depth, flow duration, and stratigraphy (Eaton 2006; Simon et al. 2000). While Eaton's (2006) goal did not include using their determined coefficient of cohesion to predict bank failure, it does demonstrate that methods have been established to approximate these values from bank geometry. For this study, bank stability within CAESAR is assessed using a method proposed by Osman and Thorne (1988). Details regarding this selection are provided in the methodology section.

As Coulthard and Van De Wiel (2006b) mentioned, the absolute numbers of CAESAR's output should be treated cautiously due to the lack of quantitative assessment that is possible. However, CAESAR has been a valuable tool for qualitatively assessing a study reach, particularly when it comes to identifying areas that are prone to erosion and deposition. A goal of this research is to improve CAESAR's ability to aid in long-term urban stream management.

CHAPTER V

STUDY METHODOLOGY

This study's methodology is split into two main phases. In the first phase, a new model was programmed in C++ called CAESAR-Lite. CAESAR-Lite is based on a simplification of CAESAR's flow and sediment routing coupled with a mechanistic algorithm for bank failure adopted from Osman and Thorne (1988). CAESAR-Lite was created to speed up initial testing and debugging of the new algorithms through a much smaller model. CAESAR-Lite does not contain many of the processes included in CAESAR, but it does retain fluvial processes that include routing of stream flow, bed erosion, and sediment transport. The second phase involves incorporating the mechanistic bank-failure model into CAESAR, then applying CAESAR to the study site.

Methodology Phase 1

For input, CAESAR-Lite only requires an ASCII digital elevation model (DEM), a text file that specifies the grid location of the input flow (x/y coordinates), what the input flow is (steady flow in cubic meters per second), and a representative grain size for the entire simulated area (sediment is assumed to be of uniform size for simplicity's sake). CAESAR-Lite is an iterative model where each iteration includes the following steps:

1. Obtain input flow and route the discharge,
2. Use the routed discharge to drive bed erosion and update the topography,
3. Check the river banks' stability, and cause them to collapse if necessary, and
4. Repeat steps 1-4 (Figure 11).

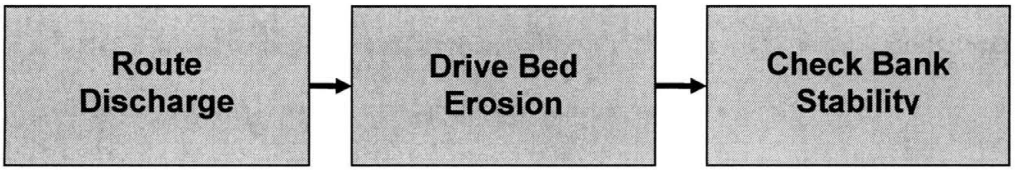


Figure 11. Flow Chart Displaying the Steps For Each of CAESAR LITE's Iterations.

These steps are discussed in more detail below.

Route Discharge

Like CAESAR's original implementation, CAESAR-Lite sweeps the DEM in four directions routing discharge as it sweeps. In any given direction, discharge is routed in proportion to slope from a source cell to the three adjacent cells in a given direction. The amount of discharge sent to a cell in the *'i'* direction is based on Equation 1:

Equation 1

$$Q_i = Q_{tot} \frac{(e+d)-e_i}{\sum_1^3 [(e+d)-e_i]}$$ (Coulthard et al. 2000)

The subscript, i, indicates the receiving cell. Q_i (m^3/s) is the amount of discharge to be routed to a cell, Q_{tot} (m^3/s) is the total discharge in the source cell, e (m) is the elevation of a grid cell and d (m) is the depth of the water in the cell. Once discharge has been routed to all of the receiving cells, depth per cell is calculated via a rearrangement of Manning's equation (Equation 2).

Equation 2

$$d = \left(\frac{Qn}{S^{0.5}} \right)^{\frac{3}{5}}$$ (Coulthard et al. 2000)

In Equation 2, d (m) is the depth for a given cell, Q (m^3/s) is the discharge available in that cell, S is the average downhill slope of the cell, and n (also called Manning's n) is a parameter that represents the roughness (friction) of the channel. It is important to note that a cell may receive different discharge values from different scans (i.e. The left-to-right scan may result in a larger discharge than the right-to-left scan or vice-versa). In these cases, for both CAESAR and CAESAR-Lite, the maximum calculated discharge is retained for use in calculating bed erosion.

Drive Bed Erosion

CAESAR is capable of tracking several different sediment layers in the stratigraphy as well as multiple grain-diameters within each layer. Since this complicates the erosion and sediment transport algorithm, CAESAR-Lite only

handles one grain size whose diameter is specified by the user. To drive bed erosion, CAESAR-Lite uses the Meyer-Peter and Müller (1948) Equation Modified by Wong and Parker (2006) is used (Equation 3).

Equation 3

$$q_b^* = 3.97\left(\tau^* - \tau_c^*\right)^{1.5}$$

In Equation 3, the left side of the equation is the dimensionless sediment transport rate per unit width of channel. Since it is dimensionless, it can be converted to a volumetric or mass value. The difference calculation on the right side of equation represents the difference between the shear stress applied to the bed of the stream and the shear stress required to move the sediment. Thus, sediment transport is a function of excess shear stress, and there is no transport when shear stress does not exceed the required amount to mobilize the bed.

A note of caution, bedload transport equations are approximate at best and Equation 3 is an empirical equation developed for sand and gravel beds. Since the transport equation does not apply to finer sediments (silts and clays), the results may not be fully representative.

Once equation 3 has been applied to every cell in the DEM, the topography is updated. If there is a net gain of sediment in a cell, the elevation value for that cell is increased by the appropriate depth. If a cell loses more sediment than it receives, the elevation value for that cell is decreased by the appropriate depth.

Check Bank Stability

Bank stability is assessed using the method proposed by Osman and Thorne (1988). Although it is an older model, the simplified geometry that it uses lends itself to adaptation within a cellular grid. Unlike the models before it, the Osman and Thorne (1988) method takes into account the effects of bed degradation and lateral erosion. Like many other models, bank stability is determined through the calculation of a factor-of-safety, *FS* (Equation 4).

Equation 4

$$FS = \frac{resisting\_force}{driving\_force}$$

However, what constitutes the factor-of-safety is dependent on the model. For the Osman and Thorne method, the resisting force is proportional to a soil's cohesion (*c'* [Pa])) and internal angle of friction (*φ'*), and the driving force is the weight (*Wt* [N]) component acting in the downslope direction (Equation 5).

Equation 5

$$FS = \frac{c' FE + N \tan \phi'}{Wt \sin \beta}$$

Essentially, Equation 5 states that the bank is no longer safe when the weight of the bank overpowers the forces holding the bank up. Thus, a ratio of

less than 1 indicates a bank that will fail. For this model, the factor-of-safety value is conservatively set at 1.3.

Figure 12 displays a bank before and after bed-scour and lateral erosion of the toe, and this geometry is used to calculate the terms in Equation 5. Representing continuous lateral erosion of the toe (Δw in Figure 12) within a grid of finite-sized grid cells is not straightforward. However, Δw (m), H' (m), H (m), and H_0 (m)(Figure 12) are crucial to the calculation of the factor-of-safety. To track all these values, CAESAR-Lite incorporates two new grids that do not exist in the original version of CAESAR. These new grids have the same resolution as the input DEM.

One grid stores the values of cell elevations before any bed scour occurs, which allows the calculation of both bank elevation prior to any bed degradation (H_0) and H'. This value remains constant for each iteration of the model until riverbank failure occurs. Upon bank failure, the new cell elevation overwrites the elevation that is currently stored.

The second grid stores the amount of continuous lateral toe erosion (Δw) that has occurred within each cell. Before the model starts, the grid cell values are initialized to equal zero. During each model iteration, the amount of toe erosion that has occurred for that timestep is calculated for each cell representing a river bank. The calculated value is then added to the stored value. With this grid, CAESAR-Lite is able to accommodate the representation of a continuous value within a discretized grid.

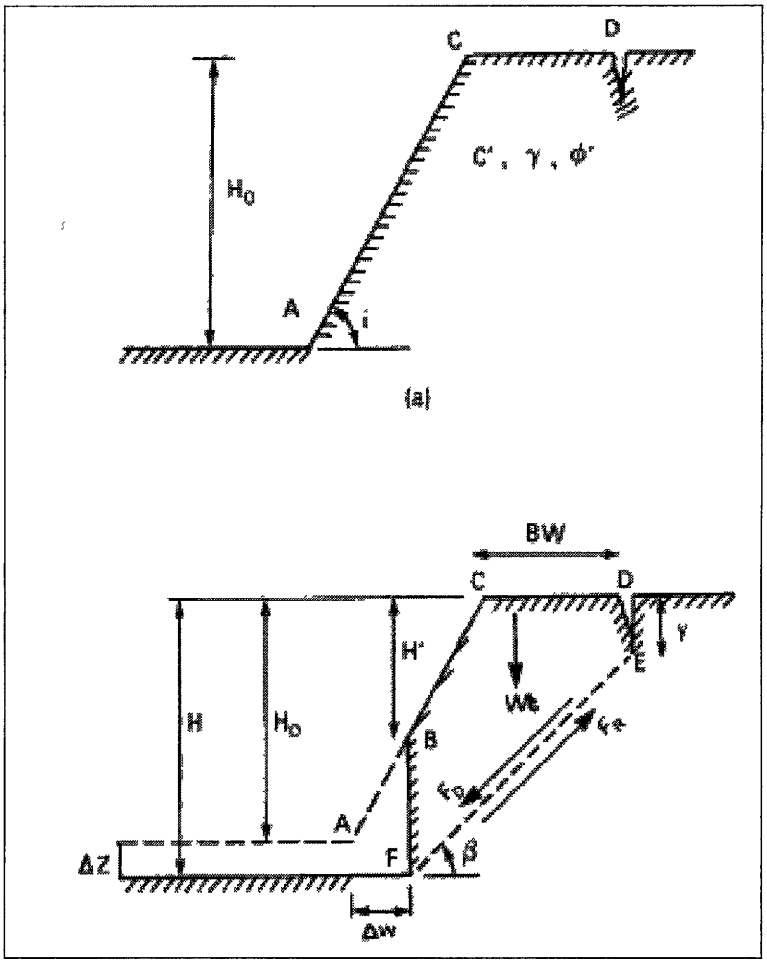


Figure 12. Bank Geometry Before and After Erosion. (Top) A view of the right riverbank before bed degradation and toe erosion. (Bottom) A view of the right riverbank once it has eroded to the point of failure. Osman and Thorne (1988).

When calculating bed and bank geometries this study makes one assumption: the difference between the elevation of the dry grid cell and the elevation of the wet grid cell is representative of the difference between the bank's elevation and bed's elevation. For more specifics regarding the original bank stability model, see Osman and Thorne (1988).

If a riverbank within the model reaches a critical state (factor-of-safety less than 1.3), a specified volume is removed from that grid-cell and deposited

into all adjacent "wet" cells. This deposit is split between the receiving cells in proportion to their slopes relative to the failed bank. Ideally, the amount of failure transferred from the bank to the stream would be calculated by the Osman and Thorne (1988) approach, but due to time constraints the eroded volume was manually specified to be the volume required to reduce the elevation value of the grid cell by 0.05 meters. This removal is performed in the same time step in which the riverbank reaches criticality, and the newly formed bank geometry is used in subsequent factor-of-safety calculations.

Radius of Curvature

Typically, bank failure occurs on the outside edges of a meander bend, and deposition occurs on the inside edge (Figure 13).

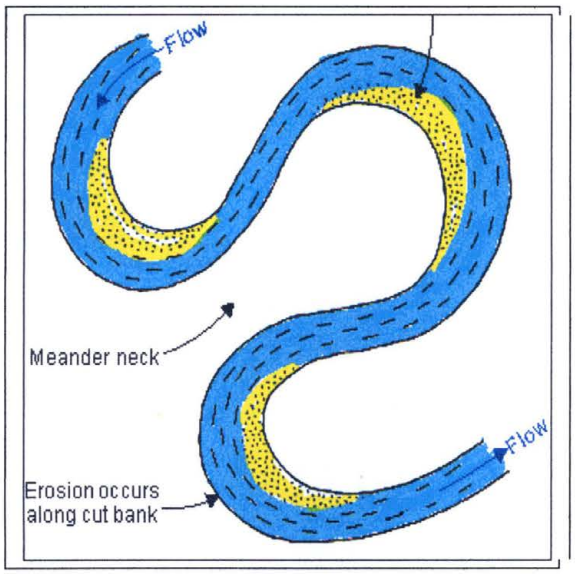


Figure 13. Typical Erosion and Deposition Zones in a Meandering River. Image retrieved from: http://www.aegweb.org

The radius of curvature for each cell plays an important role in driving this feature within CAESAR-Lite because the curvature indicates that the current cell is at the edge of a stream, and whether or not it is on the inside or the outside. The method to identify edge cells as inside or outside was adapted from the work done by Coulthard and Van De Wiel (2006a), and is illustrated in Figure 14. The Osman and Thorne (1988) approach is only applied to "outside" edges.

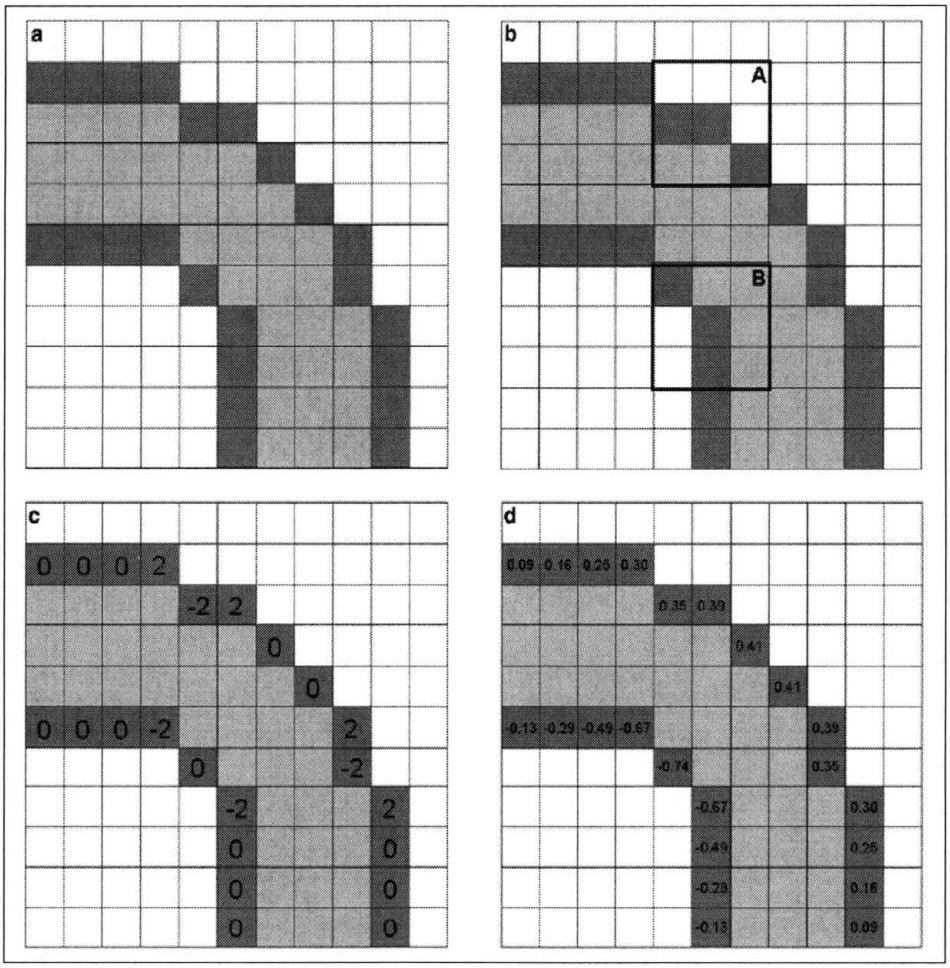


Figure 14. Steps Used to Delineate Radius of Curvature. Once edge cells have been identified (dark cells in 6a), a 3x3 filter is passed over them (figure 6b). Within the 3x3 filter, the number of wet cells is subtracted from the number of dry cells (figure 6c, edge cells are excluded from the count). A negative number indicates an inside bend, a positive number indicates an outside bend. Finally, a smoothing filter is passed over the edges (figure 6d) (Coulthard and Van De Wiel 2006a).

CAESAR-Lite was run two times. The initial topographies were of a fabricated sinuous channel in a low grade valley (Figure 15), and the input discharge was the held constant at 10 cubic meters per second. The duration for both runs was 1500 minutes, which is a little over one day. However, Run-A was specific to sand of grain size 2 millimeters, while Run-B was specific to gravel of grain size 64mm.

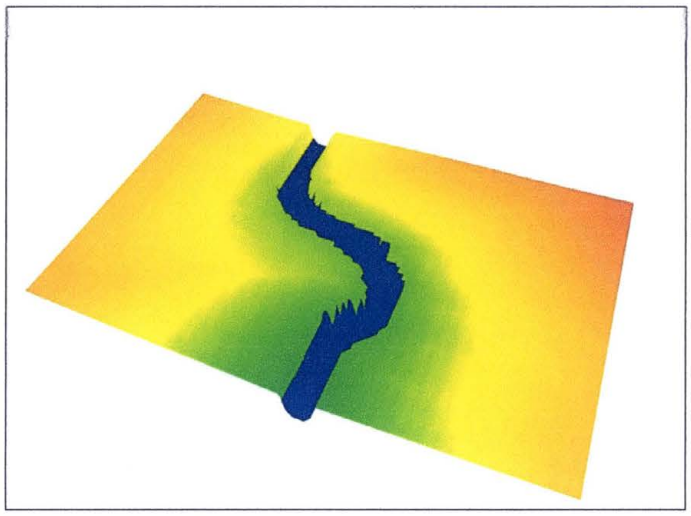


Figure 15. Initial Topography for Phase I. This river was created in ArcMap, and does not exist in reality. Flow is moving from top of the image towards the bottom.

Sand and gravel are both cohesionless and have similar angles of internal friction, however gravels are more resistant to entrainment by the stream due to a larger mass. Thus it is expected that the sand banks will migrate faster than the gravel banks since they are more easily erodible from the bed. Faster bed scour results in a more rapid steepening of the banks, which in turn decreases the factor-of-safety at a faster rate.

Results of Phase I

For this study, the primary interest is in the planform changes to the river geometry (changes viewed from a top-down perspective). Thus, the amount of erosion/migration that occurred over the duration of the model run are provided in a planform manner. Figure 16 is the original DEM used in the analysis.

Figures 17-18 display the difference in elevation after Run-A (sand) and Run-B (gravel). In Figures 17 and 18, it is clear that Run-A (the sand river) migrated further than the gravel river as expected. Flow is from left-to-right in these images. An interesting trend in both runs is the aggradation along the center of the channel.

Figure 19 displays the difference resulting from the subtraction of the elevations in Run-A from the elevations in Run-B. The majority of the reach shows more degradation of the bed occurring in the sand river, which is expected. However, there are thin strips along the outside bends of Run-B where the gravel river has experienced greater incision.

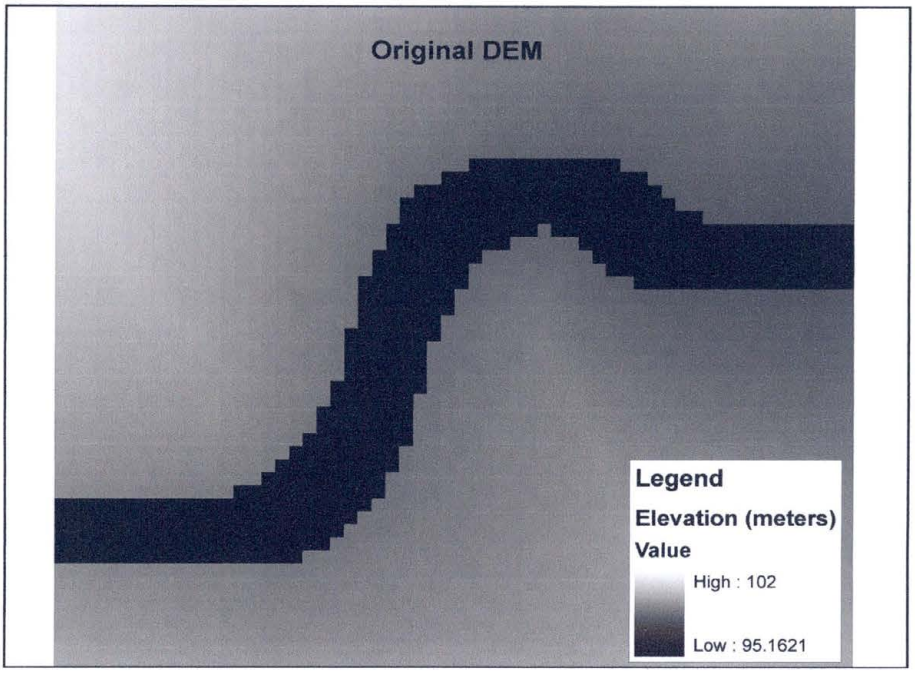


Figure 16. The Elevation Model of the River, Viewed From Above.

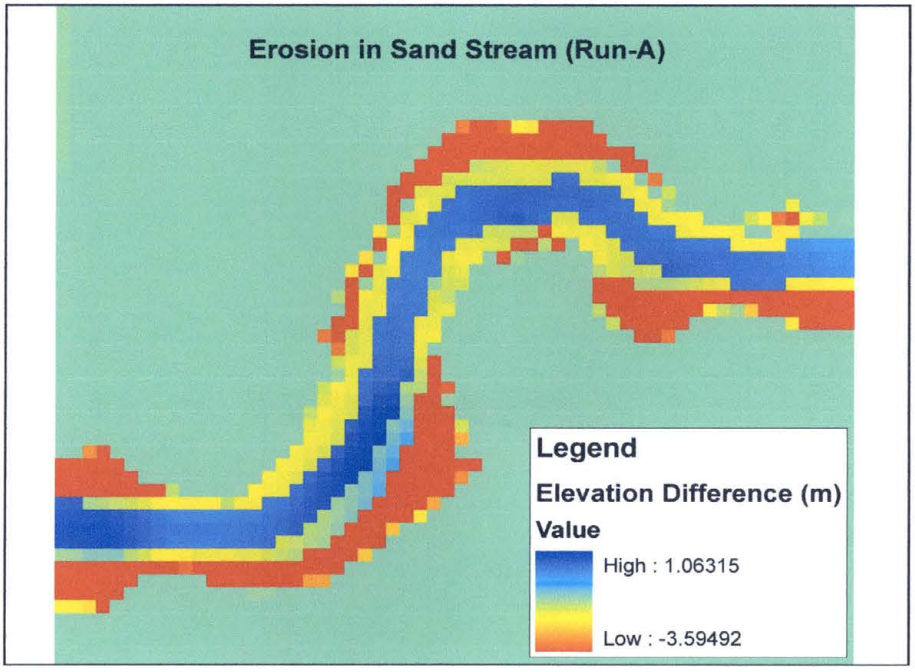


Figure 17. The Net Erosion Between the Original DEM and Run-A.

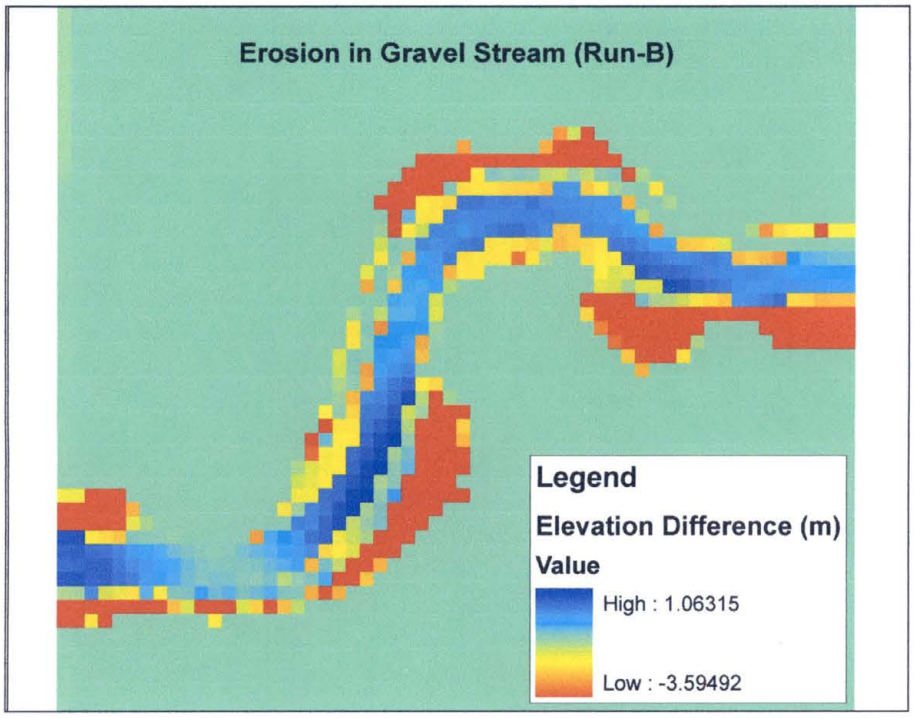


Figure 18. The Net Erosion Between the Original DEM and Run-B Using the Same Classification as Figure 17.

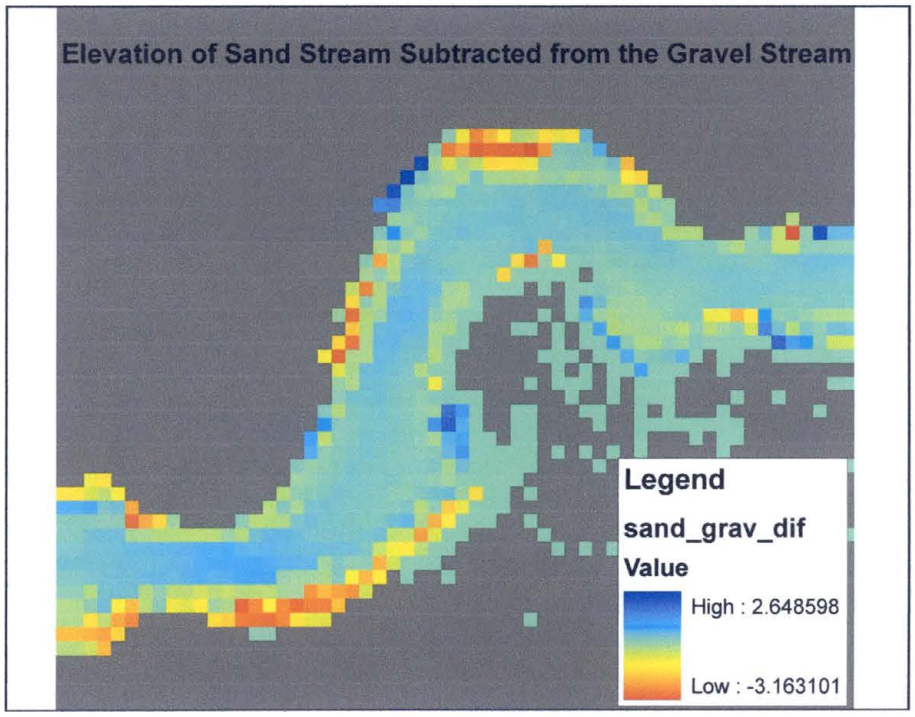


Figure 19. The Result of Subtracting the Final Elevations in Run-A From the Final Elevations in Run-B.

Conclusions for Phase I

Given the original intent of this project, the results are very encouraging. The different migration rates of the outside-edges between the sand and gravel rivers indicate that a mechanistic approach to bank failure has been successfully modeled using a CA model. Over the majority of the river, sand river migration was greater than the gravel river migration. Locations where scour in the gravel river is greater than the sand river are interesting, since the coarser gravel should be more resistant to entrainment. This anomaly could be a direct result of the gravel river's resistance to meandering. By preventing bank migration, the water is forced to remain in a narrower channel. This concentrates flow, which can lead to increased incision, especially along the thalweg. In both model-runs the center of the channel experienced a large amount of aggradation. These results may not be valid, and may be the result of the algorithm that tries to force eroded bank sediment to deposit on point-bars in the inside of the bend. However, the intent of this study was to successfully replicate lateral migration of the outer bends, and in this regard, the results are very positive.

Methodology Phase II

Due to the portability of object-oriented code, incorporating the bank-failure algorithm used in CAESAR-Lite into CAESAR was straightforward. In CAESAR-Lite, bank failure was achieved by applying one function repeatedly to all outside edge cells. As input, the function in CAESAR-Lite needs the x and y coordinates of the dry cell (bank), the x and y coordinates of the wet cell (stream bed), the elevation grid, and the discharge grid. When the function is applied to

an edge-cell it determines the factor-of-safety value adopted from Osman and Thorne (1988), then the function returns the depth of sediment to be removed from the bank if the bank is unstable.

In CAESAR, a function that used coordinates of the dry and wet cells as input, and returned a depth to be removed already existed. The main step in incorporation was to replace how the existing function in CAESAR determined removal depth with the adaption of Osman and Thorne (1988) used in CAESAR-Lite. In addition to the function, three new grids were added to CAESAR. One to track original cell elevations, another to track lateral toe erosion, and one to track the total amount of sediment removed due to the bank-failure algorithm. Finally, code was also added to export these grids to an ASCII file for final analysis.

Since CAESAR already has its own algorithms for routing discharge, routing sediment, and determining scour, all of which are more robust and capable than their counterparts in CAESAR-Lite, no other portions of CAESAR-Lite needed to be ported into CAESAR.

In order to apply CAESAR to the study site, CAESAR was run in reach mode, which required the following input:

1) topographical data,

2) bed rock elevation data,

3) grain size distributions, and

4) discharge/sediment fluxes for all inflow points.

Available LiDAR elevation data, collected in 2002 and provided by the University of Texas's Center for Space Research (Dr. Gutierrez, personal

communication, July 13, 2007), provided the initial topographical input (Figure 20). Unfortunately, the provided LiDAR data did not include detailed metadata, so it was not possible to determine the exact day the data were collected. The laser used to collect LiDAR data was unable to penetrate far below the water surface, so point channel elevation data was gathered in the field. The channel bottom could then be represented by a grid interpolated from these points within ESRI's ArcGIS 9.0 software.

The channel elevation data were gathered by collecting a continuous stream of points while walking back and forth along the channel and along the cutbank with a GPS unit (Figure 21). The GPS unit used to gather channel elevation points was a Trimble AG-132. This was chosen over a high-precision total-stations survey for a couple of reasons: 1) high precision is not necessary for CAESAR, and 2) the volume of points returned vs. time invested is very high (3,599 points for 2 days of work). This particular GPS unit is capable of sub-meter accuracy when coupled with a base-station, however when used autonomously (as it was for this study) its horizontal accuracy is within 10-15 meters depending on atmospheric conditions (Trimble, 2000). Fortunately, as Figure 21 indicates, the collected data horizontally aligned with the LiDAR image.

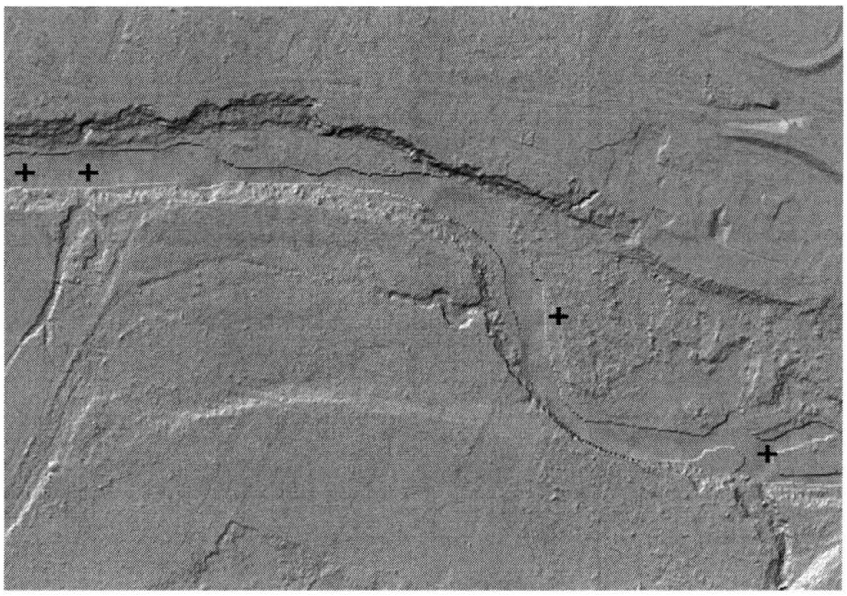


Figure 20. 2002 LiDAR Data Supplied by the University of Texas. Black crosses indicate location of Pebble Counts.

Figure 21. Data Points Used to Interpolate Channel Elevation Grid.

Vertical accuracy of the GPS unit was not nearly as good, and was not specified in the Trimble AG-132 user's manual. To verify the vertical accuracy of the GPS-collected points, the elevation data were compared to the LiDAR

elevation data outside of the channel. It is important to note that the vertical datum used in the LiDAR data and the GPS device are unknown. However, 152 out of 1,554 examined GPS points were within 0.006 – 0.497 meters of the LiDAR data, which indicates that the vertical datums are similar if not the same. However, discrepancies between the GPS and LiDAR elevation data could range as high as 11.6 meters. These extreme values are a result of data collected along the edge of an 11.6m cutbank and other steep slopes, where a small horizontal shift results in a very high vertical shift. Another possible cause of poor vertical accuracy is the thick tree-cover in certain areas, which interferes with the GPS signals.

The data collected within the channel could not be verified with the LiDAR data since LiDAR does not penetrate the water-surface. Fortunately, no tree-cover interfered with data collected in the channel, and the channel slopes are relatively small compared to the bank slopes. Thus, the GPS elevation data were used in interpolating the channel elevations. Other methods also exist to determine widespread channel elevations without intensive groundwork (Fonstad and Marcus 2005). These methods rely on stream gage data, multispectral imagery, and Manning's n, and could be used to supplement the GPS data gathered here in future studies.

The first step in creating the interpolated channel was to determine the boundaries of the interpolated surface. These boundaries were created by digitizing the channel boundaries from 2005 imagery provided by the U.S. Department of Agriculture's National Agricultural Imagery Program. Boundaries

were digitized at a scale of 1:3,500. Once the interpolation boundaries were defined, the data used for the interpolation needed to be defined. These points were gathered from the GPS data collected in the field, and point data taken from the LiDAR imagery.

Once the boundaries and data were defined, the next step was the interpolation itself. ESRI's ArcMap has a built-in tool to facilitate interpolating a gridded surface from a set of geospatial point data. To paraphrase Tobler's (1970) first law of geography, everything is related, but near things are more related than far things. This concept describes the inverse distance weighted algorithm (IDW), which is one of the various grid interpolation algorithms provided by ESRI. For the channel elevations, IDW was used with parameters defined as:

1) Power = 2
2) SearchRradius = Variable
3) Number of Points = 12
4) Maximum Distance = None

The "Power" parameter affects how much influence nearer points have on an interpolated value versus further points, and the default value of two was chosen. Having the "Search Radius" parameter defined as "variable" and the "Maximum Distance" defined as "None" means that the algorithm is not restricted distance-wise when it looks for the 12 nearest points. These 12 points are then used to interpolate an elevation value for the grid-cell in question. For more information regarding IDW, please see Philip and Watson (1982) and Watson and Philip (1985).

Due to the uneven distribution of elevation points, some areas were interpolating too high. To overcome the faulty interpolations, these areas were manually lowered by digitizing polygons with assigned subtraction values, then subtracting these values from the interpolated grid. Subtraction values were created by estimating a mean value from measured points upstream and downstream, and taking the difference between the estimated mean and the interpolated surface.

Some severe elevation changes remained in the new channel elevation grid where the subtractions took place. To overcome these severities, a smoothing filter was passed over the grid 10 times. The smoothing filter's algorithm assigned a new elevation to each grid cell by taking the mean of the grid cell's neighbors. Once the filter was applied, the new channel bed was merged into the original LiDAR using ArcMap's raster calculator.

Bedrock elevation data are based on core borings taken in 1958 (See Appendix A) by the City of Austin, and converted into a digital elevation model to be used as input for CAESAR. For this study, bedrock was given a constant elevation of 118.87 meters above sea level. It should be noted that the bedrock elevation actually varies, and is exposed immediately below Longhorn dam. However, the bedrock is not exposed further downstream in the area of concern, and setting the elevation of the bedrock at 118.872 meters above sea level guarantees that it will remain buried underneath the sediment while still providing a limit to how far the bed can erode.

Bed surface grain size distributions (Wolman 1954) of the reach were measured in four locations (Figure 22). To simplify the model, it was decided to represent the entire reach with a single grain size distribution. The grain size distribution collected on a gravel bar in the middle of the reach was used because deposits on bars are most representative of the grains that are mobilized during a high-flow event.

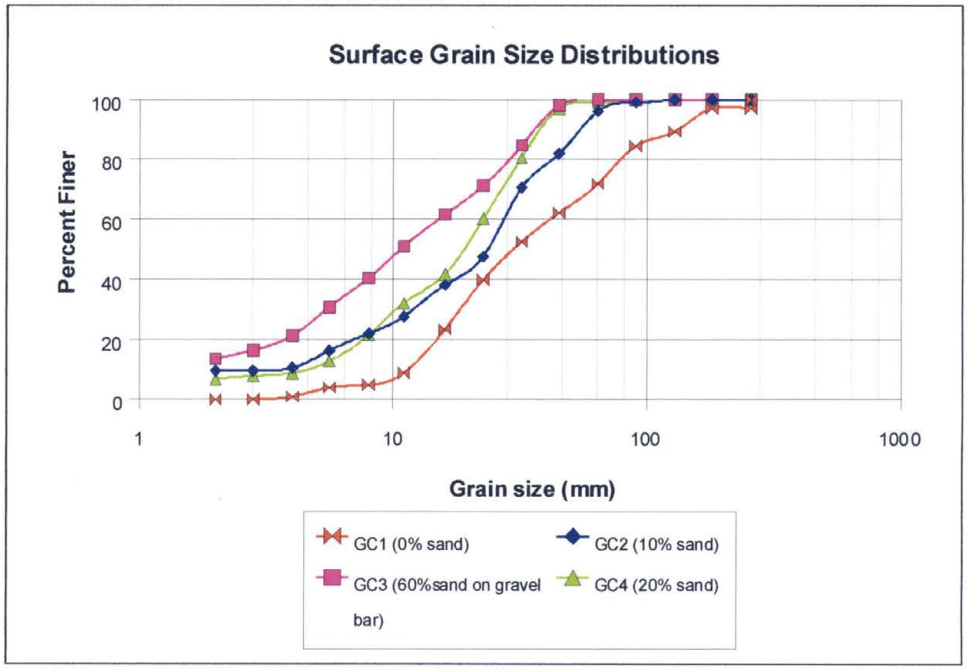


Figure 22. Results of Pebble Counts Performed at Study Site. Approximate locations of the gravel counts can be seen in Figure 20.

The discharge and sediment input locations were manually specified in the channel on the downstream side of Longhorn Dam, with the sediment input set at zero. Mean daily discharge values for the year 2002 were obtained from the U.S. Geological Survey for gauge #08158000 (Figure 23), which is located within the study site.

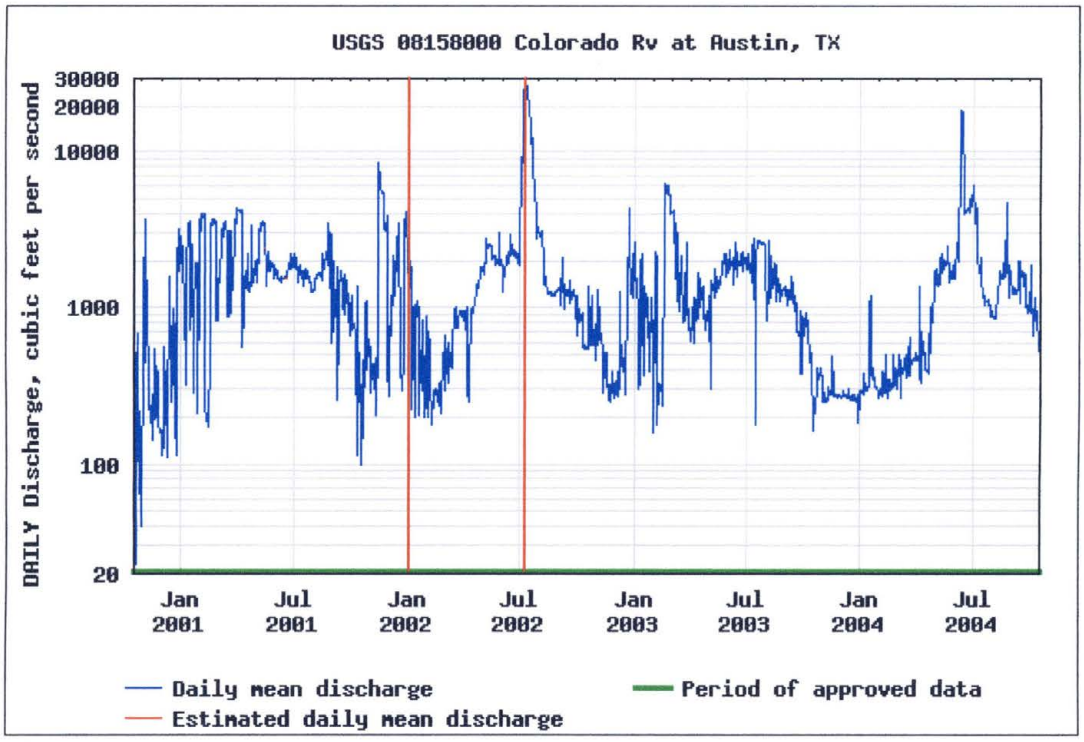


Figure 23. Hydrograph for the Study Site. Data obtained for USGS Gauge #08158000 (U.S. Geological Survey 2007), red lines indicate the simulated period.

Using these data, the model simulated the channel's behavior for approximately the first 200 days of 2002.

CHAPTER VI

RESULTS

The model's output was analyzed and compared to aerial imagery collected in 2004. The analysis included quantitative measurements of elevation change and linear bank retreat, and qualitative observations of erosional areas. After one week, the model had simulated 192 days, which captured the rising edge of a high-flow event (peaking at 753.2 cubic meters per second). During this period, bank failure led to channel widening throughout the entire reach, especially in the first meander bend (Figure 24). The bank failure algorithm was responsible for failing 362.02 million cubic meters of sediment over 5,999 different cells (Figure 25).

The banks were relatively stable until day 181, at which time the discharge began to increase due to the high-flow event with an initial mean-daily discharge of 106.18 cubic meters per second. During days 181-192 the banks rapidly migrated. The defined channel becomes lost as the higher flows fill the channel area and scour much of the bottom topography (Figure 24). Average linear bank retreat was quantified in two areas (Figure 25). These values were determined by digitizing the banks before and after the model run in ESRI's ArcMap 9.0, and then measuring the distance between the old and new banks at stations spaced approximately 15 meters apart. For each area of interest,

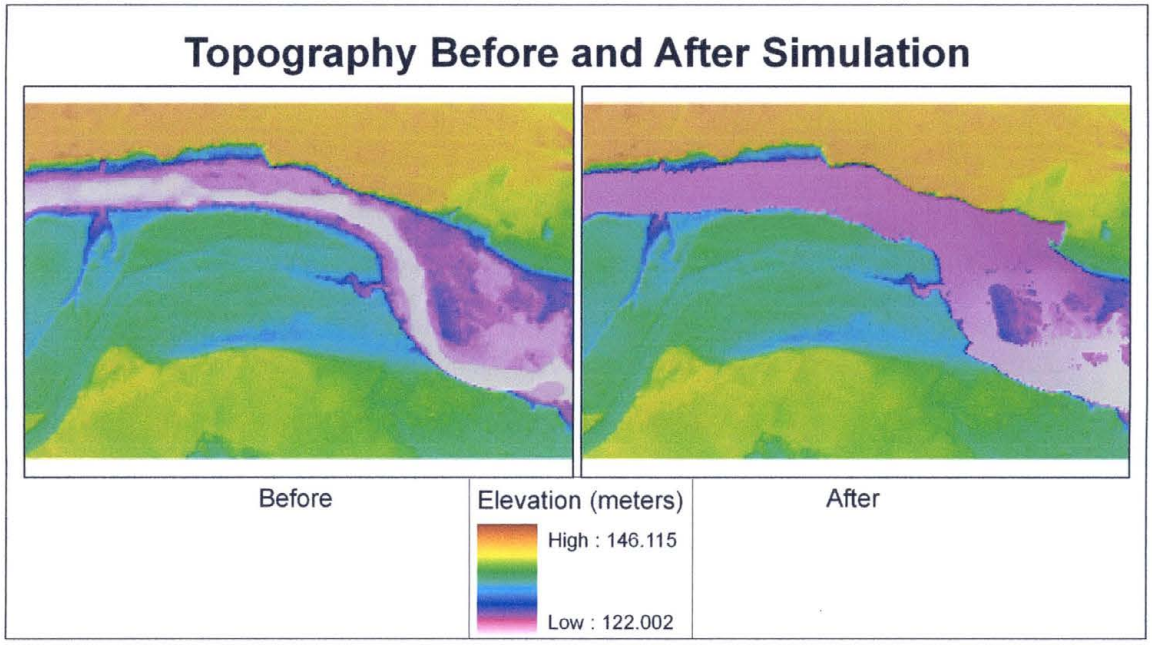


Figure 24. Topography Before and After Simulation.

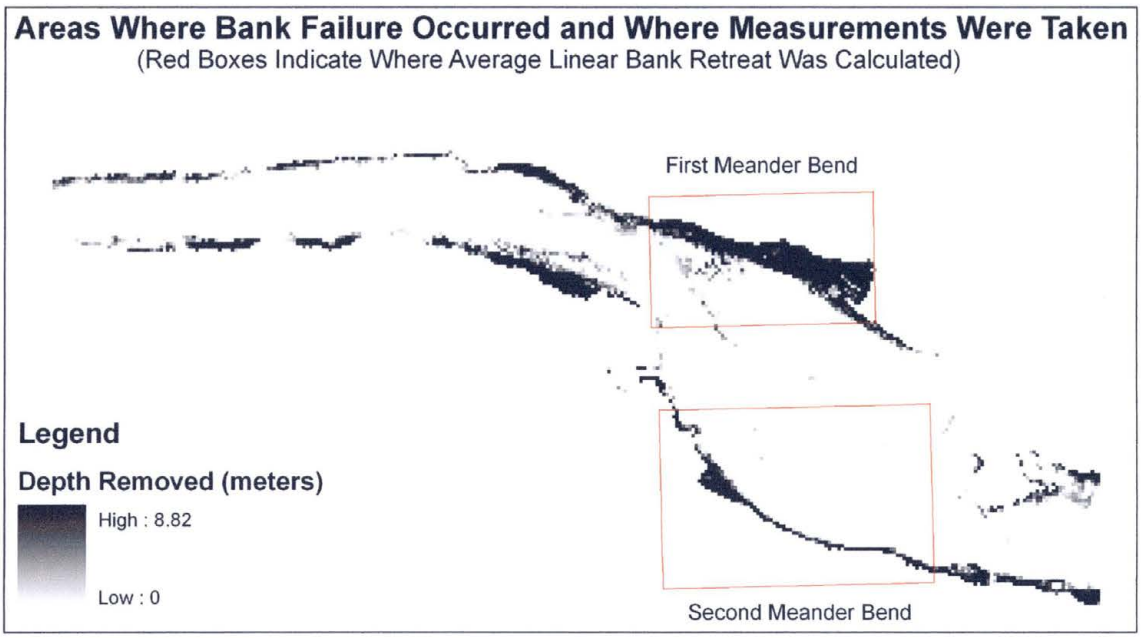


Figure 25. Bank Failure Occurrence During the Model Simulation. Red boxes indicate where average linear bank retreat was calculated.

the mean value of their stations' measurements provided the average amount of linear bank retreat over the reach (Figure 26). The first bend eroded by an average distance of

17.08 meters over a 388 meter length, while the second bend had an average retreat distance of 7.57 meters over a 352 meter length (Figure 25).

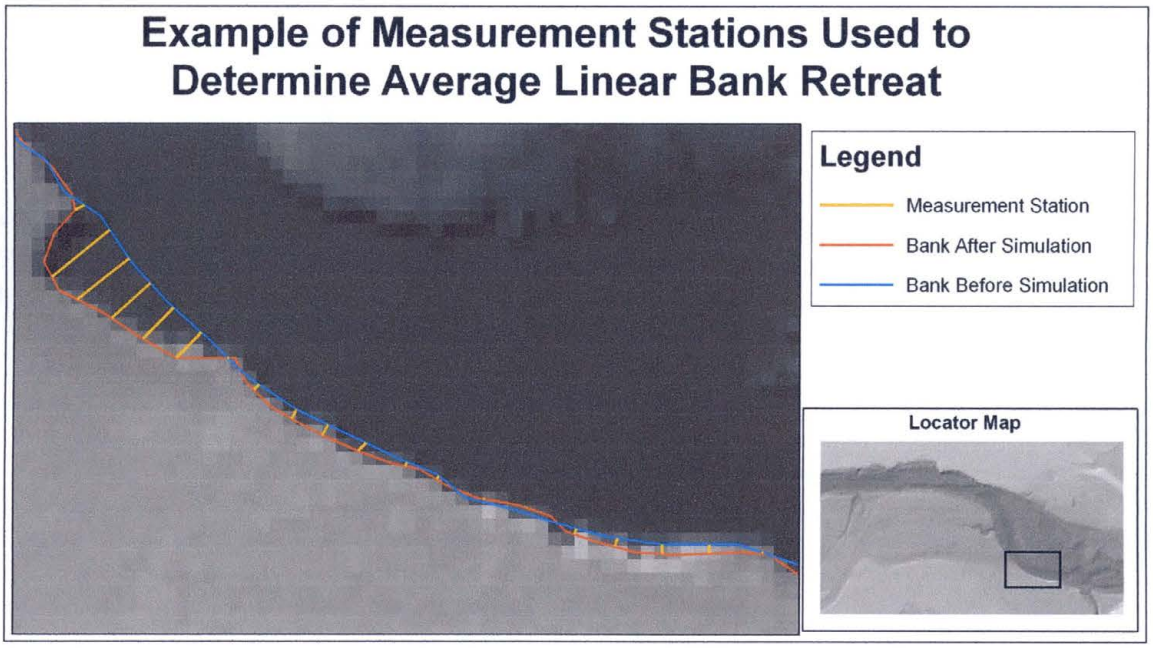


Figure 26. Measurement Stations Along the Second Meander Bend.

When compared to 2004 aerial imagery (Figure 27) collected for the U.S. Department of Agriculture's National Agriculture Imagery Program (NAIP), discrepencies between the model erosion and the actual bank erosion become apparent. The aerial photos do not confirm any noticeable migration at the first bank. The second meander, on the other hand, experienced an average failure distance of 6.61 meters which is remarkably similar to the 7.57 meters simulated by the model. This number should be treated with caution as the algorithm used a specified arbitrary number to determine the amount of sediment to be removed in the event of bank failure. Thus, the similarity between the model and the simulation may be in part due to a good estimate of unit bank erosion volume.

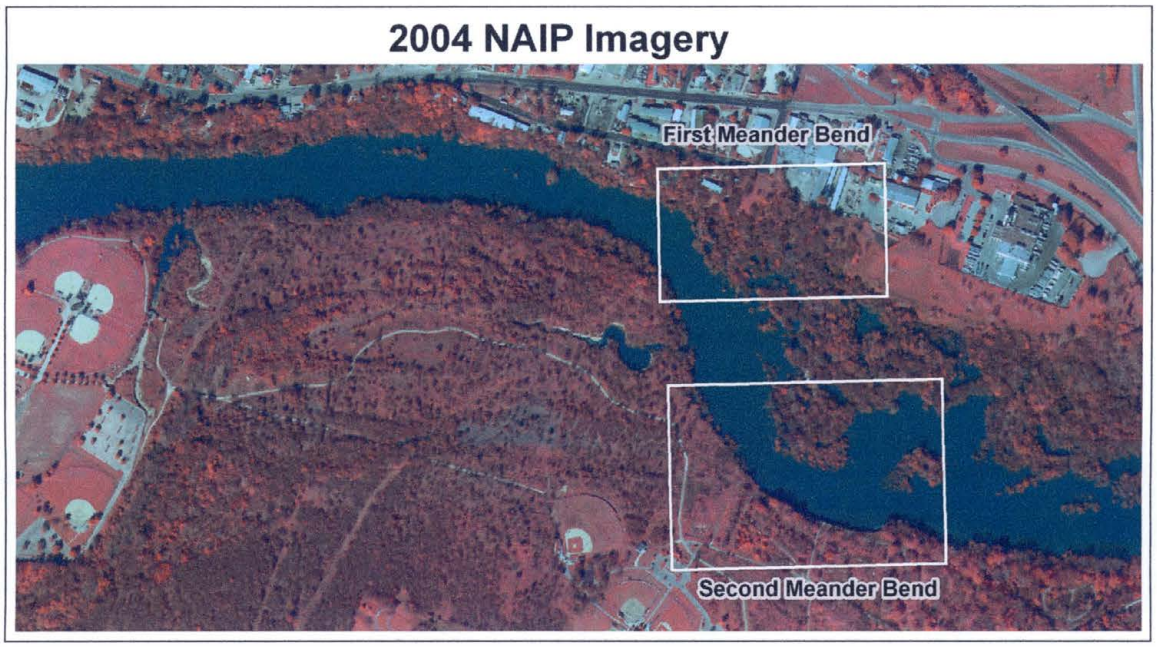


Figure 27. 2004 NAIP Imagery of the Study Site.

The primary goal of this study is to reproduce qualitatively realistic meandering by mechanistically failing banks on a cell by cell basis. In this regard, the model has been successful. Both bank stability and outward bank erosion have occurred simultaneously across the reach over the same flow regime. The banks have migrated outwards together, despite the fact that each bank cell fails on its own accord with no regard to how the neighboring cells are behaving.

A standard method for validating CA models does not currently exist (Fonstad 2006). To help quantify the success of this model and ensure that the calculated factor-of-safety values were being applied correctly, that is steeper banks should have lower factor-of-safety values, a Spearman's Rank Correlation Coefficient was calculated. This statistic was chosen due to its nonparametric nature. The inputs for this statistic were the factor-of-safety values for edge cells whose factor-of-safety was less than or equal to 1.3 during the final iteration of the model, and their corresponding slopes. The resulting

Spearman's r had a value of $r(548) = -0.53$, $p<0.05$. There is some variance in the correlation between factor-of-safety and slope, which can be partly explained by the fact that the slopes for Spearmans r were calculated in ESRI's ArcMap from the post-failure topography, and they do not take include the Δw that was stored during the model run. The significant negative correlation and resultant model behavior indicate that the factor-of-safety is calculated correctly, and that banks are reaching a critical state in appropriate manner.

CHAPTER VII

DISCUSSION

Despite the differences between the model's output and the 2004 aerial imagery, the model's behavior is highly encouraging. The banks remained very stable until high flows were introduced, as evidenced by the location of most of the bank failure shown in Figure 25. Furthermore, the cells are failing in concert even though they fail independently of each other. Both of these observations indicate that qualitatively realistic bank migration can be successfully modeled with a mechanistic approach.

During the model run, the first meander bend was the first major resistance encountered by high flows, so the massive bank migration found here is not unrealistic despite the stark difference with the aerial imagery. In reality, large trees in the low floodplain and along the bank help the river spend excess energy and stabilize the bank. However, without these trees and their effects the first bend would be highly vulnerable to bank failure. The second meander bend experienced bank failure that mimicked reality not only in linear distance, but also in pattern. Figure 26 shows that the upstream end of the bend experienced the largest amount of failure, and that the linear amount of retreat decreased further downstream. Historical photographs show that this cutbank initially migrated in this same manner, with initial bank failure occurring upstream and then progressing downstream.

While the behavior exhibited by the model is encouraging, there are some limitations to the Osman and Thorne (1988) method that this study's bank failure algorithm is adopted from. It is not able to account for vegetative influences and/or pore pressures, nor is it able to account for layering of different sediment types. These features are available in more recent models of bank failure (Langendoen et al. 2001; Van De Wiel and Darby 2007). Despite these limitations, this model is appropriate in this study for several reasons. Its simplified representation of channel geometry lends itself to adaptation to a grid-based DEM. Also, the cut-bank that is the focus of this study-site has a very uniform distribution of sediment and little vegetation, making any vegetative influence on soil cohesion negligable. Given these conditions, this study was very successful, however several shortcomings need to be addressed before it can be said that mechanistic bank failure has been completely incorporated into CAESAR.

The first and one of the most obvious concerns is unrealistic bank failure that occurred in the first meander bend as well as along the channels. Many of the areas that do not fail in reality are also vegetated with trees and shrubs, and it has already been said that vegetation could be imparting a stabilizing effect on the banks. Where the model successfully represented area of bank erosion, there was very little vegetation. Within the model simulation, these differences are not noted by CAESAR or the mechanistic bank failure algorithm because the algorithm used here focuses entirely on the channel's geometry.

Within this simulation, bank retreat in relatively straight portions of the channel occurred at a rate similar to bank retreat in the second meander bend. This similarity could be a result of the cells being completely independent of each other for the bank

failure process, so tighter bends may not migrate faster than straighter bends. Coulthard and Van de Wiel (2006a) use the radius of curvature to help tight bends migrate faster, and this model may benefit from that approach.

Another issue is the model simulating bank failure during the rising limb of the flood hydrograph. This is inconsistent with the expected connection between pore pressures and high flows. As mentioned before, as long as water levels remain elevated the confinement pressure keeps the banks stable. The banks should not fail until the water levels start to decline, but this behavior is not captured here. The banks migrated rapidly during the rising edge of a high-flow event. Under the Osman-Thorne (1988) method, this is a direct result of the increased shear-stress that is imparted on the bed and bank. The increased shear-stress leads to increased bed degradation as well as increased toe-scour, which results in a steeper and less stable bank. To improve the timing of bank erosion, a more modern bank-failure model should be used.

Necessary simplifications of the study reach characteristics may have contributed to the overestimation of bank erosion. The reach was modeled without tributary inputs, which is not correct. Areal imagery reveals that tributaries are an important supply of sediment, and that their sediment contributions lead to sand bar formation. By eliminating sediment input, shear stress generated during the flood event contribute directly to bank and bed erosion. There is no opportunity for expending shear stress to continue the transport of introduced sediment.

Another source of error due to simplification derives from the use of a predetermined value of unit bank erosion. Each time a cell's factor-of-safety value reached a critical state, a constant volume of sediment was removed. Ideally, this

volume should be dependent on local bank geometry. While these simplifications may or may not have been an issue for the short period simulated in this study, future studies may want to simulate longer periods of time with more detailed and realistic input.

Finally, during the simulation an incised channel was erased while bathymetry was homogenized into a planar bed. This behavior did not occur in reality, and is the result of the inability of current CA fluvial models to create fine-scale bedforms (Dr. Coulthard and Dr. Fonstad, personal communications, October 20, 2007). This issue illustrates the importance of choosing the correct model for the job, and for this study the emphasis is on planform modeling rather than bedforms. In this regard it may be unsuitable to use this model to evaluate flows that grossly exceed the capacity of the channel of interest, as is evidenced by the complete removal of the incised channel. However, since the interest was in the upper terraces, the removal was not a major concern.

CHAPTER VIII

CONCLUSIONS

This study's results are encouraging. Both Phase 1 and Phase 2 demonstrated that failure of the outside banks on a cell by cell basis results in regional outer bank migration in areas where banks are known to be rapidly eroding and migrating. The mechanistic approach used here means that the improved model simulates bank migration using a process based modeling approach. This is an improvement over past landscape evolution models which simulated bank migration based on the symptoms of bank erosion but not the causes. This investigation is another step towards coupling both channel meandering and braiding into a single model, which has been described as the "holy grail" of planform modeling (Paola 2000 in Coulthard and Van De Wiel 2006a).

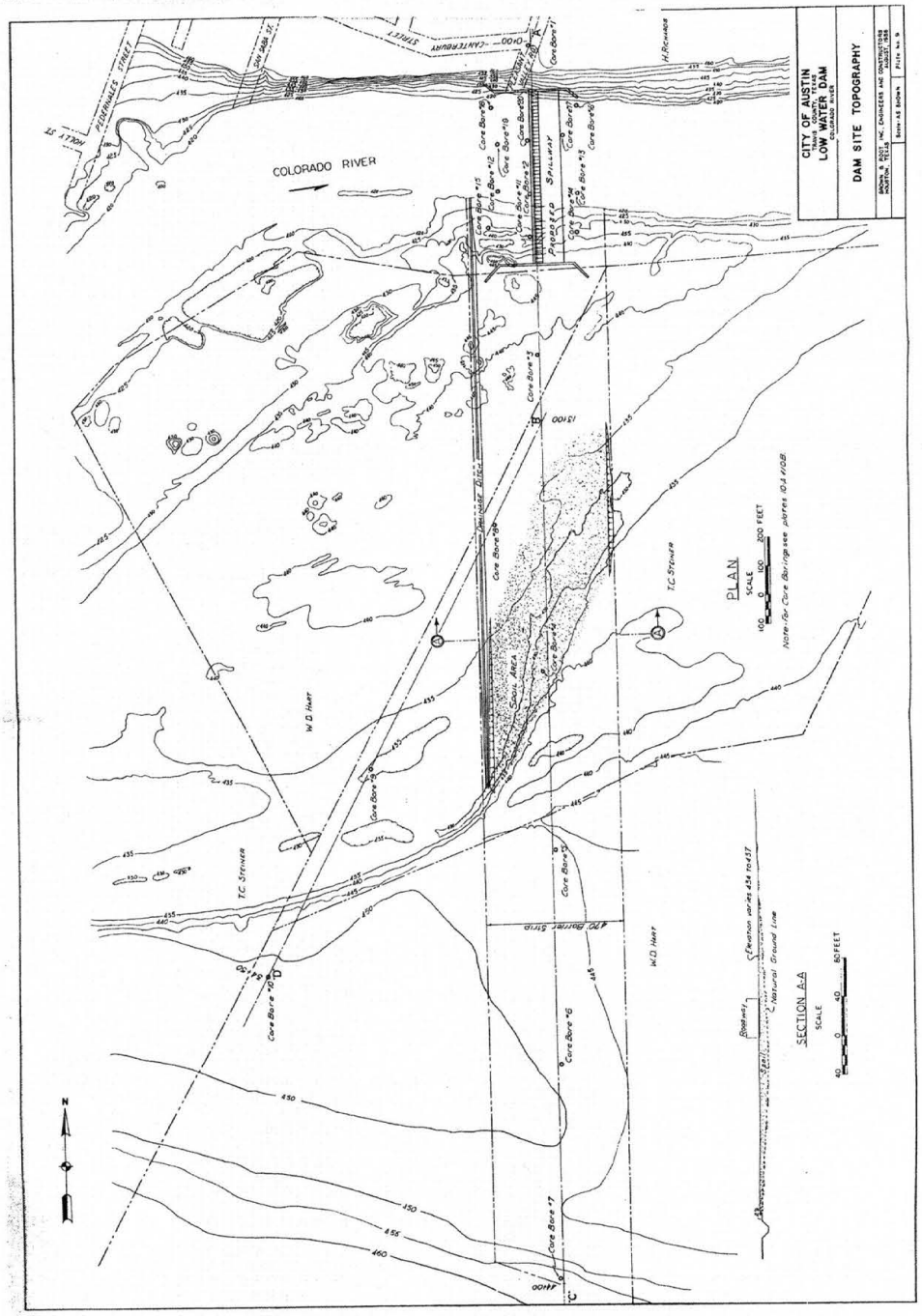
Despite these compelling results, there remain areas for future improvement. Future research should modify CAESAR to handle spatially variable vegetation cover and their effects on bank stability. Also, a bank failure adaptation of a more modern model could improve simulation results. In particular, the effects of varied sediment layers and pore water pressure should be accounted for in the next version of this model. At this and any study-site, modeling efforts can be improved by higher degrees of detail and accuracy from field derived model inputs.

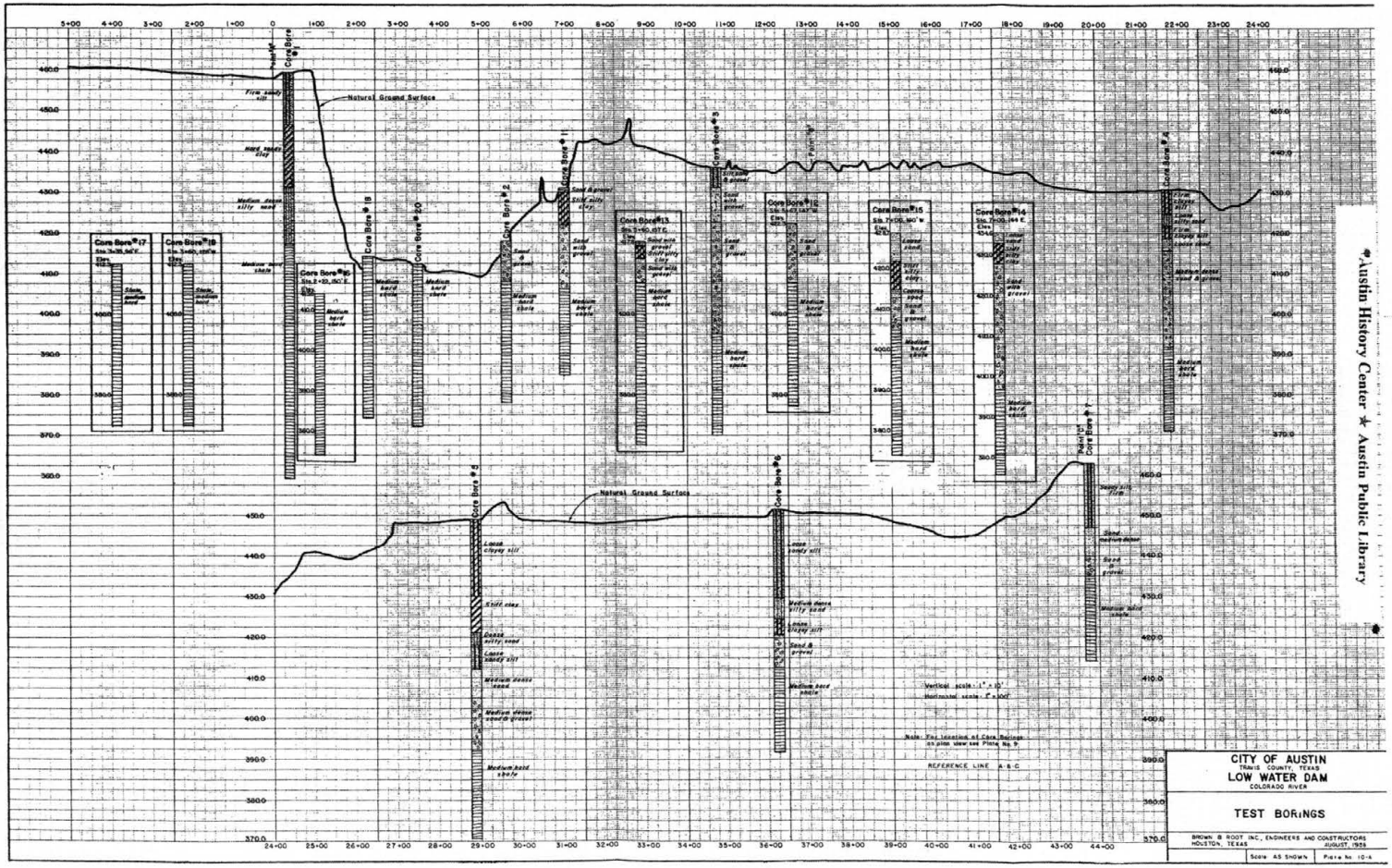
One question to consider for future research is how complicated should a model become? One of the strengths of CA modeling is its simplified approach, and complexity may not be correct path to take. For more precise bank erosion simulations, a model such as CONCEPTS (Langendoen et al. 2001) may be more appropriate. CONCEPTS is a 1-dimensional flow model, whereas CAESAR and CAESAR-Lite are 2-dimensional models, but CONCEPTS has a much more robust bank failure mechanism that addresses the shortcomings of the bank failure algorithm used here. However, CONCEPTS requires much more detailed input than CAESAR, its 1-dimensional nature prevents it from simulating braided channels, and it is not designed to simulate an entire catchment over long periods of time.

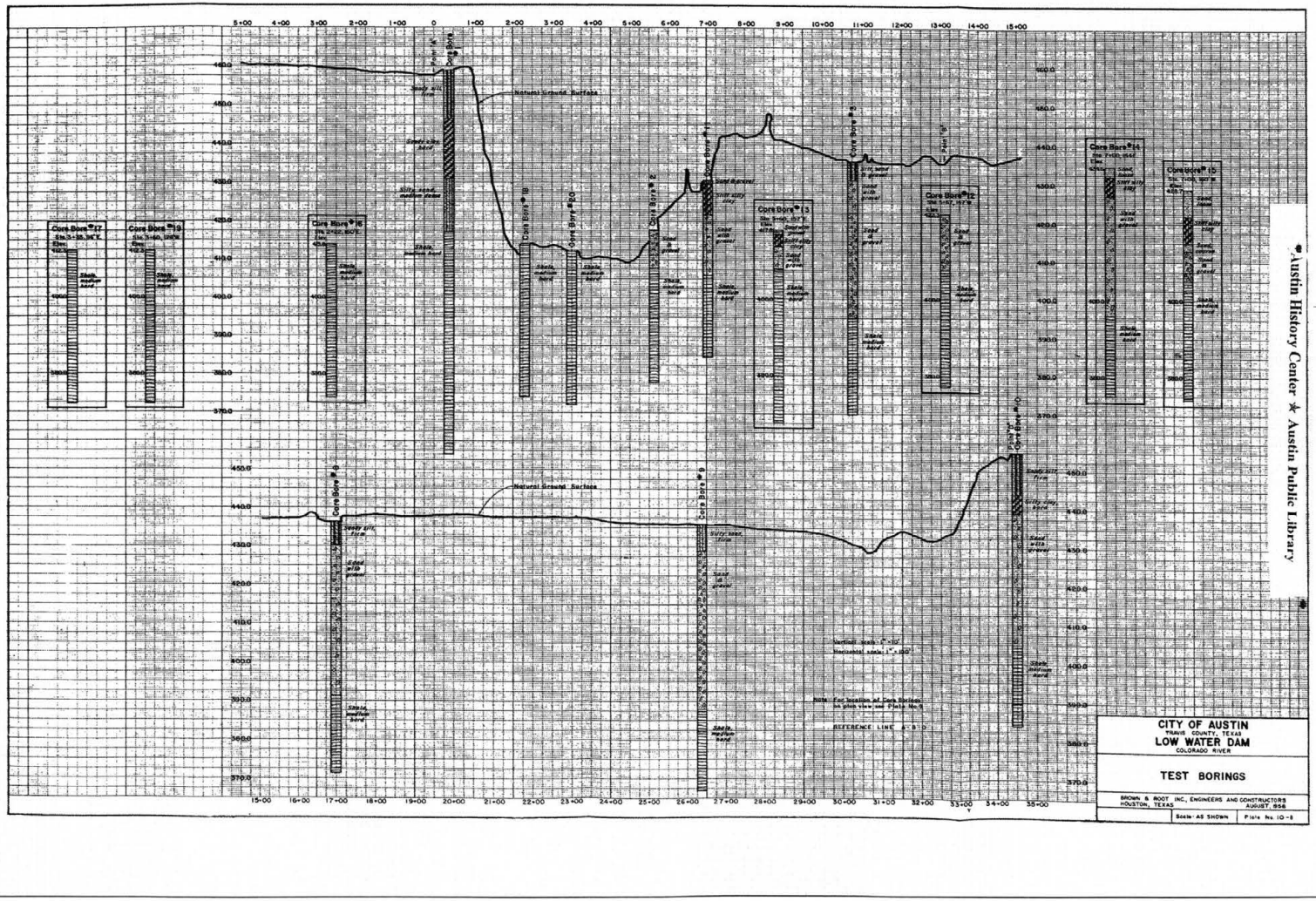
Even though CAESAR and other models of its type are not precision models, they can be used to fill the niche between the geologic time-scale models and the computational fluid dynamics models. Their value lies in their computational efficiency, which allows for hypothesis testing and simulation of large areas over large amounts of time. Incorporation of a mechanistic approach to bank failure provides urban stream-managers another tool to use in long-term channel prediction.

APPENDIX A

CORE BORINGS, CITY OF AUSTIN, 1958







APPENDIX B

C++ CODE FOR CAESAR-LITE

```
// Delbert Humberson
// Main program for CAESAR-Lite.
// to get this program to run, it must be set up in a win32 console
// due to the use of OpenGL
// dels_ca_32console.cpp : Defines the entry point for the console application.
//

#include "stdafx.h" //CAESAR-Lite header
//using namespace System;
#include <stdlib.h> //Needed for "exit" function

//Include OpenGL header files, so that we can use OpenGL
#ifdef __APPLE__
#include <OpenGL/OpenGL.h>
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

using namespace std;
elev_grid A; //create the elevation grid array from the ascii_dem.txt file
discharge_grid water; //create the discharge_grid using the in_x and in_y variables
sediment_grid sediment;
double inflow;
int t;
float xbase,ybase,zbase;//these are manipulated to allow the user to move the camera
int flag=0;  //this flag verifies the valley has filled with water, and the erosion should start

//Called when a key is pressed
void handleKeypress(unsigned char key, //The key that was pressed
                                int x, int y) {    //The current mouse coordinates
      ostringstream oss;
      string filler;
      string timestep;
      switch (key) {
            case 27: //Escape key
                  oss<<t;
```

```
        timestep = oss.str();
        water.write_file(A,"depth_"+timestep+".txt");
        sediment.write_file(A,"erosion_"+timestep+".txt");
        sediment.write_edges(A,"edges_"+timestep+".txt");
        sediment.write_delta_w(A,"delta_w"+timestep+".txt");
        A.write_file("DEM_"+timestep+".txt");
        A.write_old_elev("Old_Elev_"+timestep+".txt");
        A.write_original_elev("Original_Elev_"+timestep+".txt");
        sediment.write_FS(A, "FS_"+timestep+".txt");
        sediment.write_dH(A, "dH_"+timestep+".txt");
        water.write_flow_dir(A, "flow_dir_"+timestep+".txt");
        sediment.write_gradient(A,"gradient_"+timestep+".txt");
        exit(0); //Exit the program

case 'p': //p key
        cout<<"paused, press any key to continue";
        cin>>filler;
        break;

case 'w':
        ybase -= 1;
        break;
case 'a':
        xbase +=1;
        break;
case 'd':
        xbase-=1;
        break;
case 's':
        ybase +=1;
        break;
case 'z':
        zbase +=1;
        break;

case 'x':
        zbase -=1;
break;
```

```
	}
}

//Initializes 3D rendering
void initRendering() {
	//Makes 3D drawing work when something is in front of something else
	glEnable(GL_DEPTH_TEST);
}

//Called when the window is resized
void handleResize(int w, int h) {
	//Tell OpenGL how to convert from coordinates to pixel values
	glViewport(0, 0, w, h);

	glMatrixMode(GL_PROJECTION); //Switch to setting the camera perspective

	//Set the camera perspective
	glLoadIdentity(); //Reset the camera
	gluPerspective(90.0,//45.0                    //The camera angle
	                    (double)w / (double)h, //The width-to-height ratio
	                    1.0,                   //The near z clipping coordinate
	                    200.0);                //The far z clipping coordinate
}

void drawScene() {
	double x_coord,y_coord,z_coord;
	double depth_unit;
	depth_unit=inflow/5;  //split the max depth into 5 depths to create a shaded gradient for flow
	//Clear information from last draw
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glMatrixMode(GL_MODELVIEW); //Switch to the drawing perspective
	glLoadIdentity(); //Reset the drawing perspective


	glBegin(GL_QUADS); //Begin quadrilateral coordinates
```

```
int x,y;
int color_shift; //to shade the colors based on flow depth
double color_value;//stores the color value based on color_shift
//this loop draws the current state of the grid cells
for(x=0;x<x_max;x++){
      for(y=0;y<y_max;y++){
            if(sediment.get_dZ(x,y)>0){
                  color_shift = (int)(water.get_max_depth(x,y)/depth_unit)+1;
                  switch(color_shift){
                  case 5:
                        color_value = 0.5;
                        break;
                  case 4:
                        color_value = 0.55;
                        break;
                  case 3:
                        color_value = 0.6;
                        break;
                  case 2:
                        color_value = 0.8;
                        break;
                  case 1:
                        color_value = 0.93;
                        break;
                  default:
                        color_value = 1;
                        break;
                  }

                  glColor3f(0.0,color_value,0.0);
            }
            else if(sediment.get_dZ(x,y)<0){
                  color_shift = (int)(water.get_max_depth(x,y)/depth_unit)+1;
                  switch(color_shift){
                  case 5:
                        color_value = 0.4;
                        break;
                  case 4:
```

```
                color_value = 0.5;
                break;
        case 3:
                color_value = 0.55;
                break;
        case 2:
                color_value = 0.8;
                break;
        case 1:
                color_value = 0.93;
                break;
        default:
                color_value = 1;
                break;
        }
        glColor3f(color_value,0.0,0.0);
}


else if(sediment.get_Curvature(x,y)>0){
        glColor3f(1.0,0.55,0);
}

else if(sediment.get_Curvature(x,y)<0){
        glColor3f(1.0,1.0,0);
}

else if(sediment.get_Curvature(x,y)==0 && sediment.get_edge(x,y)){
        glColor3f(0,0,0);
}


else if(water.get_max_depth(x,y)>5*depth_unit){
        glColor3f(0.0,0.0,.50);
}
else if(water.get_max_depth(x,y)>4*depth_unit){
        glColor3f(0.0,0.0,0.80);
}
```

```
else if(water.get_max_depth(x,y)>3*depth_unit){
      glColor3f(0.2,0.2,1);
}

else if(water.get_max_depth(x,y)>2*depth_unit){
      glColor3f(0.4, 0.4, 1);
}
else if(water.get_max_depth(x,y)>depth_unit){
      glColor3f(0.67,0.67,1);
}
else if(water.get_max_depth(x,y)>0){
      glColor3f(0.8,0.8,1);
}

else{
      glColor3f(.55,.49,.42);
}
x_coord = (double)xbase+(double)0.075*(x-(x_max/2));
y_coord = (double)ybase+(double)0.075*(y-(y_max/2));
z_coord = (double)zbase+(double)-7;//-5;



glVertex3f(x_coord,y_coord, z_coord);
glVertex3f(x_coord+0.075, y_coord, z_coord );
glVertex3f(x_coord+0.075, y_coord+0.075, z_coord );
glVertex3f(x_coord, y_coord+0.075, z_coord );


glVertex3f(x_coord,y_coord, z_coord );
glVertex3f(x_coord+0.075, y_coord, z_coord );
glVertex3f(x_coord+0.075, y_coord+0.075, z_coord );
glVertex3f(x_coord, y_coord+0.075, z_coord );
```

```
            }

        }
        glEnd(); //End quadrilateral coordinates
        glFlush();
        glutSwapBuffers(); //Send the 3D scene to the screen

}

//this is called by the gluttimer function, which animates the grid
void change_x(int y){
        int x_coord, y_coord;
        string timestep;
        ostringstream oss;

        cout<<"timestep: "<<t<<" Out Q: "<<water.get_outQ()<<'\n';


        t++;
        water.route_water(inflow,A);
        if(t>=1500){
                oss<<t;

                timestep = oss.str();
                water.write_file(A,"depth_"+timestep+".txt");
                sediment.write_file(A,"erosion_"+timestep+".txt");
                sediment.write_edges(A,"edges_"+timestep+".txt");
                A.write_file("DEM_"+timestep+".txt");
                sediment.write_FS(A, "FS_"+timestep+".txt");
                sediment.write_dH(A, "dH_"+timestep+".txt");
                sediment.write_gradient(A,"gradient_"+timestep+".txt");
                exit(0); //Exit the program
        }
        else
        if(water.get_outQ()>= 0.99*inflow){
                flag = 1;
        }
```

```
	if(flag ==1){
		A.update_old_elev();
		sediment.route_sediment(water,A);
		for(x_coord=0;x_coord<x_max;x_coord++){
			for(y_coord=0;y_coord<y_max;y_coord++){
				A.update_elev(x_coord,y_coord,sediment.get_dZ(x_coord,y_coord));
			}
		}
		sediment.fail_banks(A,water);

	}


	glutPostRedisplay();
	glutTimerFunc(0.5,change_x,0);


}


int _tmain(int argc, char** argv)//int argc, _TCHAR* argv[])
{

	xbase=0.0;
	ybase=0.0;
	zbase=0.0;

	int in_x,in_y;//this will store the x,y location of water input read from the in_flow.txt file

	int t;//counter

	double D;//grainsize diameter of the substrate in the system
	string month;//this string will be used to label output file names with the month of the simulation


	//open connection to in_flow.txt
	ifstream infile("in_flow.txt");
	infile>>in_x; //read in x coordinate of input flow
```

```
        infile>>in_y; //read in y coordinate of input flow
        infile>>inflow; //read in discharge value for input flow
        infile>>D;//read in grain size diameter
        A.initialize("ascii_dem.txt"); //create the elevation grid array from the ascii_dem.txt file
        A.create_original_elev();//preserve the original values in original_elev add to c#
        water.initialize(in_x,in_y); //create the discharge_grid using the in_x and in_y variables
        sediment.initialize(D);


        water.route_water(inflow,A);

        //Initialize GLUT

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(800, 600); //Set the window size

        //Create the window
        glutCreateWindow("Delberts CA Model!");
        initRendering(); //Initialize rendering

        //Set handler functions for drawing, keypresses, and window resizes
        glutDisplayFunc(drawScene);
        glutKeyboardFunc(handleKeypress);
        glutReshapeFunc(handleResize);
        glutTimerFunc(0.5,change_x,0);
        glutMainLoop(); //Start the main loop.  glutMainLoop doesn't return.



}
// Created by: Delbert Humberson
// Description:  The header file for CAESAR-Lite.  Contains class definitions and their methods.
// To get this program to run, it must be set up in a win32 console
// project, not a normal console project, due to the use of OpenGL.
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
```

```
//

#pragma once

#ifndef _WIN32_WINNT          // Allow use of features specific to Windows XP or later.
#define _WIN32_WINNT 0x0501   // Change this to the appropriate value to target other versions of Windows.
#endif

#include <stdio.h>
#include <tchar.h>

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

// TODO: reference additional headers your program requires here
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <math.h>
#include <string>
#include <assert.h>
#include <sstream>
#include <limits>
using namespace std;

/*****************************************************************************************************
GLOBAL VARIABLES
These are global to make it easier for the user to modify them as needed */
static const int x_max =58;//175;   //number of columns in the ascii dem
static const int y_max = 91;//179;  //number of rows in the ascii dem
static const double min_Q = 0.01; //minimum flow needed to do any calculations for routing and sediment
transport (borrowed from CAESAR)
const long double       Pi = 3.14159265358979323846264338327950288419 68;
const double time_factor=1;//60 for minutes, multiply by another 60 for hours, etc.
```

```
//*****************************************************************************************************
**

//these functions test if I have a result that is infinity (inf) or not quite a number (qnan)
template<typename T>
inline bool isnan(T value)
{
return value != value;

}

// requires #include <limits>
template<typename T>
inline bool isinf(T value)
{
return std::numeric_limits<T>::has_infinity &&
value == std::numeric_limits<T>::infinity();
}

//*****************************************************************************************************
***
//CLASS DEFINITION: ELEV_GRID
//The elev_grid class is a 2-dimensional array that constains elevation values.  It gets its values from
//an ascii dem file created within arcmap.  For now, the user must manually specify the number of rows
//and columns in the static global integer variables x_max (number of columns) and y_max (number of rows).

class elev_grid{
private:
      static const int length_header = 1000;     //the maximum expected length of the first 6 lines in
                                                 //the ascii dem(ncols, nrows,...,nodata_value.  1000 is
                                                 //excessively large
      static const int num_header_lines = 6; //number of header lines found at the top of ascii dem
      string header_lines;                  //this string will store the header so that it can be used
                                            //in any output headers
      double elev[x_max][y_max];            //the 2 dimensional array that will store the elevation values
      double old_elev[x_max][y_max];        //this stores the elevations before they are updated
      double original_elev[x_max][y_max]; //this preserves the original bank heights for the osman-thorne
                                            //method
```

```
      double DX;                          //this will store the resolution of the DEM (in meters)

public:

      elev_grid(){
      }

      //the following takes the name of the file as a parameter and uses it to initilize the values
      void initialize(string name_infile){
            string in_line;               //will store the current line being read from infile

            int x,y;
            istringstream iss;            //will store the ascii dem strings one line at a time,
                                          //which will then be parsed into the elev array


            try {
                  ifstream infile(name_infile.data());

                        for(y=0; y < num_header_lines; y++){ //this for loop stores the header lines
                              getline(infile, in_line);
                              header_lines += in_line + "\n";
                              if(y==4){          //if we are in the 4th line, need to get the value for DX
                                    iss.str(in_line);
                                    iss>>in_line;//we can ignore this first read, what we want is in the
                                                 //2nd read.
                                    iss>>DX;
                                    iss.clear();

                              }

                        }

                        for(y=y_max-1; y >=0; y--){ //these nested for loops store the ascii dem values
                                                    //into the elev array
                              getline(infile,in_line); //read in a line of the ascii dem
                              iss.str(in_line);       //convert the stored string into a string stream for
```

```
                                                //input into the array

                        for(x=0; x<x_max;x++){
                              iss>>elev[x][y];  //use iss>> (similar to cin>> and fin>>)
                                                //to place in array
                        }
                  }
            //}

            infile.close();
      }


      catch (...) {
            cout << "Elevation Grid file open error\n";

      }
}



//the following function returns the elevation value for a given x,y coordinate
double get_elev(int x, int y){
      try{
            return elev[x][y]; //return the elevation value if x,y in range
      }
      catch(...){ //do this if x and y are not in range,
                  ..for example when calculating mean slope along boundary cells
            return 1000000000;  //by returning an unrealistically high elevation, we prevent
                                //negative slopes from entering the
                                          //mean slope calculation
      }

}

double get_old_elev(int x, int y){
      try{
            return old_elev[x][y];
```

```
	}
	catch(...){
		return 10000000000;
	}
}

void update_old_elev(){
	int x,y;
	for(x=0;x<x_max;x++){
		for(y=0;y<y_max;y++){
			old_elev[x][y] = elev[x][y];
		}
	}
}

//returns the original elevation
double get_original_elev(int x, int y){
	return original_elev[x][y];
}

//this creates the ascii grid storing the original values
void create_original_elev(){
	int x,y;
	for(x=0;x<x_max;x++){
		for(y=0;y<y_max;y++){
			original_elev[x][y] = elev[x][y];
		}
	}
}

//when dw resets, need to reset original elevation
void update_original_elev(int x, int y, double h){
	original_elev[x][y] = h;
}



//the following will be invoked by the sediment_grid class, and will erode/deposit as neccessary
```

```
void update_elev(int x, int y, double h){
	try{
		//test if we're at the boundaries.  Since I don't calculate bed transport there,
		//i need to manually force them.
		//this assumes that main flow is from left to right, so it always forces the right
		//boundary to be lower than
		//the cells to the left, and the left boundary to be higher than the cells to the right.
		//for the bottom and top boundaries, it forces them to be higher than the rows adjacent
		//to them,
		//assuming a valley with a left-to-right orientation
		if(x==x_max-1){
			elev[x][y]=elev[x-1][y]-0.001;
		}
		else if(x==0){
			elev[x][y]=elev[x+1][y]+0.001;
		}
		else if(y==y_max-1){
			elev[x][y]=elev[x][y-1]+0.001;
		}
		else if(y==0){
			elev[x][y]=elev[x][y+1]+0.001;
		}
		else{
			elev[x][y]+=h;
		}
	}
	catch(...){//out of bounds error
		cout<<"Out of bounds error for update_elev()";
	}
}

//the following functions write new ascii files, taking the file name as the argument
//It does not check if the file exists already!  so the programmer must take this into account
void write_file(string out_file){
	int x,y;
	ofstream outfile(out_file.data());
	outfile<<header_lines;
```

```
	for(y = y_max-1; y>=0; y--){
		for(x=0;x<x_max; x++){
		outfile<<elev[x][y]<<" ";
		if(x+1==x_max){
			outfile<<'\n';
		}

		}
	}
outfile.close();
}

void write_old_elev(string out_file){
	int x,y;
	ofstream outfile(out_file.data());
	outfile<<header_lines;


	for(y = y_max-1; y>=0; y--){
		for(x=0;x<x_max; x++){
		outfile<<old_elev[x][y]<<" ";
		if(x+1==x_max){
			outfile<<'\n';
		}

		}
	}
outfile.close();
}

void write_original_elev(string out_file){
	int x,y;
	ofstream outfile(out_file.data());
	outfile<<header_lines;


	for(y = y_max-1; y>=0; y--){
```

```
                for(x=0;x<x_max; x++){
                outfile<<original_elev[x][y]<<" ";
                if(x+1==x_max){
                      outfile<<'\n';
                }

                }
          }
      outfile.close();
      }

      //the get_DX function returns the grid resolution of a file
      double get_DX(){
            return DX;
      }

      string get_header(){
            return header_lines;
      }

};




//*****************************************************************************************
//CLASS DEFINITION: DISCHARGE GRID
//the discharge grid class will contain a grid of x_max by y_max cells, and will be responsible
//for tracking depth and Q for each cell.  It will also be used to route the water
//(using the algorithm presented in Coulthard et al. 2002)
class discharge_grid{
      private:
            double Q[x_max][y_max]; //this grid tracks Q per cell
            double outQ;//this variable stores the output discharge for a timestep
            double inQ; //this variable store the input discharge for a timestep
            int x_in, y_in; //stores location of the flow input, for this model only one cell allowed
            int flow_dir[x_max][y_max]; //stores the main direction of flow based on the maximum discharge
```

```
//this private function get_input gets the input.  It will be called by the public routing
//function.  It is assumed that the in_flow is passed in from the main program.
//for now, this will be steady flow.  By calling the input everytime, we make it easy for
//the user to modify the main program to read in different inputs to create dynamic flow
void get_input(double in_flow){
      Q[x_in][y_in]+= in_flow;
      inQ = in_flow;
}


//this private function returns the mean bed slope for a cell, used to calculate depth
//for a discharge.  It was also adapted from C# code found in CAESAR
double mean_bed_slope(elev_grid &elev, int x, int y)
{

      double slope=0;
      int slopetot=0;
      double slopemax=0;
      double DX = elev.get_DX(); //the length of a gridcell, used to calculate slope
      double root = pow(2.0*pow(DX,2),0.5); //for diagonal cells, DX = sqrt(2*(DX^2))

      if(elev.get_elev(x,y)>elev.get_elev(x,y-1))
      {
            slope=(elev.get_elev(x,y)-elev.get_elev(x,y-1))/DX;
            slopemax+=slope;
            slopetot++;
      }
      if(elev.get_elev(x,y)>elev.get_elev(x+1,y-1))
      {
            slope=(elev.get_elev(x,y)-elev.get_elev(x+1,y-1))/root;
            slopemax+=slope;
            slopetot++;
      }
      if(elev.get_elev(x,y)>elev.get_elev(x+1,y))
      {
            slope=(elev.get_elev(x,y)-elev.get_elev(x+1,y))/DX;
            slopemax+=slope;
```

```
        slopetot++;
}
if(elev.get_elev(x,y)>elev.get_elev(x+1,y+1))
{
        slope=pow((elev.get_elev(x,y)-elev.get_elev(x+1,y+1))/root,1);
        slopemax+=slope;
        slopetot++;
}
if(elev.get_elev(x,y)>elev.get_elev(x,y+1))
{
        slope=(elev.get_elev(x,y)-elev.get_elev(x,y+1))/DX;
        slopemax+=slope;
        slopetot++;
}
if(elev.get_elev(x,y)>elev.get_elev(x-1,y+1))
{
        slope=(elev.get_elev(x,y)-elev.get_elev(x-1,y+1))/root;
        slopemax+=slope;
        slopetot++;
}
if(elev.get_elev(x,y)>elev.get_elev(x-1,y))
{
        slope=(elev.get_elev(x,y)-elev.get_elev(x-1,y))/DX;
        slopemax+=slope;
        slopetot++;
}
if(elev.get_elev(x,y)>elev.get_elev(x-1,y-1))
{
        slope=(elev.get_elev(x,y)-elev.get_elev(x-1,y-1))/root;
        slopemax+=slope;
        slopetot++;
}
if(slopemax>0){slope=(slopemax/slopetot);}
else{slope = 0;}


return(slope);
```

```
}

//calc_depth calculates the depth in a cell for a given discharge.  It stores the depth in
//max_depth[x][y] if it is the largest depth to occur for the current scan (1 scan includes 4
//sweeps - lr,rl,ud,du.  it will also store the main direction of the flow based on maximum
//discharge (1 = lr, 2=rl,3=ud,4=du)
double calc_depth(elev_grid &elev, int x,int y, int maindir){
    double meanslope, current_depth;

                    assert(Q[x][y]>=min_Q);
                    meanslope = mean_bed_slope(elev,x,y);//get mean slope for current cell

                    if(meanslope>=0.005){
                           current_depth = pow((Q[x][y]*0.03)/(pow(meanslope,0.5)),0.6);
                          //convert current Q to depth to calculate erosion as well
                          //as proportion water routing.  We are assuming manning's n = 0.03.
                    }
                    else{
                          current_depth = Q[x][y]; //small slopes can lead to excessive depths,
                          //this statement is put in to prevent excessive depths from occuring.
                          //Not sure why we set d to Q, but this is suggested
                          //by Coulthard et al. (2002)
                    }




                    if(current_depth > max_depth[x][y]){
                          //store the flow_dir and maximum depth, will be used to drive erosion
                          max_depth[x][y] = current_depth;
                          flow_dir[x][y] = maindir;
                    }
        return current_depth;

    }
public:
```

```
double max_depth[x_max][y_max];
//this grid tracks the maximum calculated depth per cell during the flow routing

discharge_grid(){
}

//the following takes the location of the flow input as an argument by x,y index values
//it is assumed these will be passed through by the main program
void initialize(int loc_x_input, int loc_y_input){
      int x,y;
      x_in=loc_x_input;
      y_in=loc_y_input;

      //here we initialize the Q and depth values to zero
      for(x=0;x<x_max;x++){
            for(y=0;y<y_max;y++){
                  Q[x][y]=0;
                  max_depth[x][y]=0;
                  flow_dir[x][y]=0;
            }
      }
}



//this routine routes water, first left-to-right, then right-to-left, top-to-bottom,
//and then bottom-to-top using the algorithm described in Coultard et al. (2002)
void route_water(double inflow, elev_grid &elev){
      int x,y;
      double depth;//stores depth of water in current cell
      double total_elevation;//stores the elevation+water depth for current cell
      double slope_sum,slope1,slope2,slope3;
      //stores the slope (total_elevation/DX)for the 3 neighbor cells in a given sweep
      double Q1,Q2,Q3;  //will store the flow values to the neighbor cells
      double root = pow(2*pow((double)elev.get_DX(),2),0.5);

      get_input(inflow);//add the input flow to the grid
```

```
for(x=0;x<x_max;x++){ //zero out all the max_depth and flow_dir values to 0
      for(y=0;y<y_max;y++){
            max_depth[x][y]=0;
            flow_dir[x][y]=0;
      }
}

//for the flow routing, I only calculate it on cells that are not on the edge of the DEM
//at the end of the scanning algorithm, any flows along the
//boundary cells are removed from
//the DEM, and totalled for discharge output for the reach

//do scan for left to right, need to calculate depths for cells with flow > min_Q
for(y=1;y<y_max-1;y++){
      for(x=1;x<x_max-1;x++){
            if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                  depth = calc_depth(elev,x,y,1);//get depth for current cell
                  total_elevation = depth+elev.get_elev(x,y);

                  //calculate slopes for the neighbors, negative slopes set = 0
                  slope1 = (total_elevation - elev.get_elev(x+1,y+1))/root;
                  if(slope1<0)slope1=0;
                  slope2 = (total_elevation - elev.get_elev(x+1,y))/elev.get_DX();
                  if(slope2<0)slope2=0;
                  slope3 = (total_elevation - elev.get_elev(x+1,y-1))/root;
                  if(slope3<0)slope3=0;

                  //get sum of slopes for determining slope ratios
                  slope_sum = slope1+slope2+slope3;

                  //calculate discharges to neighbors and route the water from the
                  //source to the neighbors
                  if(slope_sum != 0){ //if all 3 slopes are negative,
                                      //water must remain trapped for a subsequent sweep
                        Q1 = Q[x][y]*(slope1/slope_sum);
                        Q2 = Q[x][y]*(slope2/slope_sum);
                        Q3 = Q[x][y]*(slope3/slope_sum);
```

```
                                    Q[x][y]-= (Q1+Q2+Q3);
                                    //subtract transferred flow from current cell
                                    Q[x+1][y+1] += Q1;
                                    Q[x+1][y] += Q2;
                                    Q[x+1][y-1] += Q3;

                              }
                        }

                  }
            }

//do scan for right to left, need to calculate depths for cells with flow > min_Q
            for(y=1;y<y_max-1;y++){
                  for(x=x_max-1;x>=1;x--){
                        if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                              depth = calc_depth(elev,x,y,2);//get depth for current cell
                              total_elevation = depth+elev.get_elev(x,y);

                              //calculate slopes for the neighbors, negative slopes set = 0
                              slope1 = (total_elevation - elev.get_elev(x-1,y+1))/root;
                              if(slope1<0)slope1=0;
                              slope2 = (total_elevation - elev.get_elev(x-1,y))/elev.get_DX();
                              if(slope2<0)slope2=0;
                              slope3 = (total_elevation - elev.get_elev(x-1,y-1))/root;
                              if(slope3<0)slope3=0;

                              //get sum of slopes for determining slope ratios
                              slope_sum = slope1+slope2+slope3;

                              //calculate discharges to neighbors and route
                              //the water from the source to the neighbors
                              if(slope_sum != 0){
                              //if all 3 slopes are negative, water must remain
                              //trapped for a subsequent sweep
                                    Q1 = Q[x][y]*(slope1/slope_sum);
```

```
                        Q2 = Q[x][y]*(slope2/slope_sum);
                        Q3 = Q[x][y]*(slope3/slope_sum);

                        //assert(Q1+Q2+Q3<= Q[x][y]);
                        Q[x][y]-= (Q1+Q2+Q3);
                        //subtract transferred flow from current cell
                        Q[x-1][y+1] += Q1;
                        Q[x-1][y] += Q2;
                        Q[x-1][y-1] += Q3;
                  }
            }

      }
}

//do scan for top to bottom, need to calculate depths for cells with flow > min_Q
for(x=1;x<x_max-1;x++){
      for(y=y_max-1;y>=1;y--){
            if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                  depth = calc_depth(elev,x,y,3);//get depth for current cell
                  total_elevation = depth+elev.get_elev(x,y);

                  //calculate slopes for the neighbors, negative slopes set = 0
                  slope1 = (total_elevation - elev.get_elev(x-1,y-1))/root;
                  if(slope1<0)slope1=0;
                  slope2 = (total_elevation - elev.get_elev(x,y-1))/elev.get_DX();
                  if(slope2<0)slope2=0;
                  slope3 = (total_elevation - elev.get_elev(x+1,y-1))/root;
                  if(slope3<0)slope3=0;

                  //get sum of slopes for determining slope ratios
                  slope_sum = slope1+slope2+slope3;

                  //calculate discharges to neighbors and route the water from the
                  //source to the neighbors
                  if(slope_sum != 0){ //if all 3 slopes are negative, water must
                                      //remain trapped for a subsequent sweep
                        Q1 = Q[x][y]*(slope1/slope_sum);
```

```
                        Q2 = Q[x][y]*(slope2/slope_sum);
                        Q3 = Q[x][y]*(slope3/slope_sum);

                        Q[x][y]-= (Q1+Q2+Q3);
                        ////subtract transferred flow from current cell
                        Q[x-1][y-1] += Q1;
                        Q[x][y-1] += Q2;
                        Q[x+1][y-1] += Q3;
                    }
                }

        }
}


//do scan for bottom to top
for(x=1;x<x_max-1;x++){
        for(y=1;y<=y_max-1;y++){
                if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                        depth = calc_depth(elev,x,y,4);//get depth for current cell
                        total_elevation = depth+elev.get_elev(x,y);

                        //calculate slopes for the neighbors, negative slopes set = 0
                        slope1 = (total_elevation - elev.get_elev(x-1,y+1))/root;
                        if(slope1<0)slope1=0;
                        slope2 = (total_elevation - elev.get_elev(x,y+1))/elev.get_DX();
                        if(slope2<0)slope2=0;
                        slope3 = (total_elevation - elev.get_elev(x+1,y+1))/root;
                        if(slope3<0)slope3=0;

                        //get sum of slopes for determining slope ratios
                        slope_sum = slope1+slope2+slope3;

                        //calculate discharges to neighbors and route the water from the
                        //source to the neighbors
                        if(slope_sum != 0){ //if all 3 slopes are negative, water must
                                            //remain trapped for a subsequent sweep
                                Q1 = Q[x][y]*(slope1/slope_sum);
```

```
                        Q2 = Q[x][y]*(slope2/slope_sum);
                        Q3 = Q[x][y]*(slope3/slope_sum);

                        //assert(Q1+Q2+Q3<= Q[x][y]);
                        Q[x][y]-= (Q1+Q2+Q3);
                        //subtract transferred flow from current cell
                        Q[x-1][y+1] += Q1;
                        Q[x][y+1] += Q2;
                        Q[x+1][y+1] += Q3;
                    }
                }

        }
}

//we route the water again, in slightly different directions,
//according to Dr. Coulthard this works better

//do scan for right to left, need to calculate depths for cells with flow > min_Q
for(y=1;y<y_max-1;y++){
        for(x=x_max-1;x>=1;x--){
                if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                        depth = calc_depth(elev,x,y,2);//get depth for current cell
                        total_elevation = depth+elev.get_elev(x,y);

                        //calculate slopes for the neighbors, negative slopes set = 0
                        slope1 = (total_elevation - elev.get_elev(x-1,y+1))/root;
                        if(slope1<0)slope1=0;
                        slope2 = (total_elevation - elev.get_elev(x-1,y))/elev.get_DX();
                        if(slope2<0)slope2=0;
                        slope3 = (total_elevation - elev.get_elev(x-1,y-1))/root;
                        if(slope3<0)slope3=0;

                        //get sum of slopes for determining slope ratios
                        slope_sum = slope1+slope2+slope3;

                        //calculate discharges to neighbors and route the water from the
                        //source to the neighbors
```

```
                        if(slope_sum != 0){ //if all 3 slopes are negative, water
                                            //must remain trapped for a subsequent sweep
                              Q1 = Q[x][y]*(slope1/slope_sum);
                              Q2 = Q[x][y]*(slope2/slope_sum);
                              Q3 = Q[x][y]*(slope3/slope_sum);


                              Q[x][y]-= (Q1+Q2+Q3);
                              //subtract transferred flow from current cell
                              Q[x-1][y+1] += Q1;
                              Q[x-1][y] += Q2;
                              Q[x-1][y-1] += Q3;
                        }
                  }

            }
      }

//do scan for left to right, need to calculate depths for cells with flow > min_Q
for(y=1;y<y_max-1;y++){
      for(x=1;x<x_max-1;x++){
            if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                  depth = calc_depth(elev,x,y,1);//get depth for current cell
                  total_elevation = depth+elev.get_elev(x,y);

                  //calculate slopes for the neighbors, negative slopes set = 0
                  slope1 = (total_elevation - elev.get_elev(x+1,y+1))/root;
                  if(slope1<0)slope1=0;
                  slope2 = (total_elevation - elev.get_elev(x+1,y))/elev.get_DX();
                  if(slope2<0)slope2=0;
                  slope3 = (total_elevation - elev.get_elev(x+1,y-1))/root;
                  if(slope3<0)slope3=0;

                  //get sum of slopes for determining slope ratios
                  slope_sum = slope1+slope2+slope3;

                  //calculate discharges to neighbors and route the water from
                  //the source to the neighbors
```

```
                        if(slope_sum != 0){ //if all 3 slopes are negative, water must remain
                                            // trapped for a subsequent sweep
                              Q1 = Q[x][y]*(slope1/slope_sum);
                              Q2 = Q[x][y]*(slope2/slope_sum);
                              Q3 = Q[x][y]*(slope3/slope_sum);


                              Q[x][y]-= (Q1+Q2+Q3);
                              //subtract transferred flow from current cell
                              Q[x+1][y+1] += Q1;
                              Q[x+1][y] += Q2;
                              Q[x+1][y-1] += Q3;

                        }
                  }

      }
}
```

```
//do scan for bottom to top
for(x=1;x<x_max-1;x++){
      for(y=1;y<=y_max-1;y++){
            if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                  depth = calc_depth(elev,x,y,4);//get depth for current cell
                  total_elevation = depth+elev.get_elev(x,y);

                  //calculate slopes for the neighbors, negative slopes set = 0
                  slope1 = (total_elevation - elev.get_elev(x-1,y+1))/root;
                  if(slope1<0)slope1=0;
                  slope2 = (total_elevation - elev.get_elev(x,y+1))/elev.get_DX();
                  if(slope2<0)slope2=0;
                  slope3 = (total_elevation - elev.get_elev(x+1,y+1))/root;
                  if(slope3<0)slope3=0;
```

```
                        //get sum of slopes for determining slope ratios
                        slope_sum = slope1+slope2+slope3;

                        //calculate discharges to neighbors and route the water from the
                        //source to the neighbors
                        if(slope_sum != 0){ //if all 3 slopes are negative, water must
                                            //remain trapped for a subsequent sweep
                              Q1 = Q[x][y]*(slope1/slope_sum);
                              Q2 = Q[x][y]*(slope2/slope_sum);
                              Q3 = Q[x][y]*(slope3/slope_sum);


                              Q[x][y]-= (Q1+Q2+Q3);
                              //subtract transferred flow from current cell
                              Q[x-1][y+1] += Q1;
                              Q[x][y+1] += Q2;
                              Q[x+1][y+1] += Q3;
                        }
                  }

            }

            //do scan for top to bottom, need to calculate depths for cells with flow > min_Q
      for(x=1;x<x_max-1;x++){
            for(y=y_max-1;y>=1;y--){
                  if(Q[x][y]>=min_Q){//test to make sure we have adequate flow in the channel
                        depth = calc_depth(elev,x,y,3);//get depth for current cell
                        total_elevation = depth+elev.get_elev(x,y);

                        //calculate slopes for the neighbors, negative slopes set = 0
                        slope1 = (total_elevation - elev.get_elev(x-1,y-1))/root;
                        if(slope1<0)slope1=0;
                        slope2 = (total_elevation - elev.get_elev(x,y-1))/elev.get_DX();
                        if(slope2<0)slope2=0;
                        slope3 = (total_elevation - elev.get_elev(x+1,y-1))/root;
                        if(slope3<0)slope3=0;
```

```
                        //get sum of slopes for determining slope ratios
                        slope_sum = slope1+slope2+slope3;

                        //calculate discharges to neighbors and route the water from the
                        //source to the neighbors
                        if(slope_sum != 0){ //if all 3 slopes are negative, water must
                                            //remain trapped for a subsequent sweep
                              Q1 = Q[x][y]*(slope1/slope_sum);
                              Q2 = Q[x][y]*(slope2/slope_sum);
                              Q3 = Q[x][y]*(slope3/slope_sum);


                              Q[x][y]-= (Q1+Q2+Q3);
                              //subtract transferred flow from current cell
                              Q[x-1][y-1] += Q1;
                              Q[x][y-1] += Q2;
                              Q[x+1][y-1] += Q3;
                        }
                  }

            }
      }
}
//now to sum up boundary flows and remove them from the grid:

outQ=0;
for(x=0;x<x_max;x++){
      outQ+=Q[x][y_max-1];
      outQ+=Q[x][0];

      Q[x][y_max-1]=0;
      Q[x][0] = 0;
}

for(y=0;y<y_max;y++){
      outQ+=Q[0][y];
      outQ+=Q[x_max-1][y];
```

```
        Q[0][y] = 0;
        Q[x_max-1][y] = 0;

}


//at this point, the water has been routed in all four directions twice,
//outQ and max_depth have been updated
}

double get_outQ(){
        return outQ;
}

double get_max_depth(int x, int y){
        return max_depth[x][y];
}

//returns the main direxction of flow (values 1-4)
int get_flow_dir(int x, int y){
        return flow_dir[x][y];
}

//write_file outputs and ascii dem of water depths per timestep
void write_file(elev_grid &elev, string out_file){
int x,y;
ofstream outfile(out_file.data());
ofstream outfile2("tau.txt");
outfile<<elev.get_header();
outfile2<<elev.get_header();

for(y = y_max-1; y>=0; y--){
        for(x=0;x<x_max; x++){
        outfile<<max_depth[x][y]<<" ";
        if(x>0&&x<x_max-1&&y>0&&y<y_max-1){outfile2<<get_tau(elev,x,y)<<" ";}
        else{
                outfile2<<"0 ";
        }
```

```
      if(x+1==x_max){
            outfile<<'\n';
            outfile2<<'\n';
      }

      }
}
outfile2.close();
outfile.close();
}

//write_file outputs and ascii dem of water depths per timestep
void write_flow_dir(elev_grid &elev, string out_file){
int x,y;
ofstream outfile(out_file.data());

outfile<<elev.get_header();


for(y = y_max-1; y>=0; y--){
      for(x=0;x<x_max; x++){
      if(flow_dir[x][y] ==1) outfile<<"> ";
      else if(flow_dir[x][y]==2) outfile<<"< ";
      else if(flow_dir[x][y]==3) outfile<<"v ";
      else if(flow_dir[x][y]==4) outfile<<"^ ";
      else outfile<<"  ";

      if(x+1==x_max){
            outfile<<'\n';

      }

      }
}
outfile.close();
}

//get_tau returns the shear stress imparted by the water onto the stream bed
```

```
            //we use the formula Tau = rho*g*h*S, where h (depth) is substituted for hydraulic
            //radius (wetted perimeter/x-sectional area).  This is usually only valid for streams
            //with width to depth ratios exceeding 20:1, but it's the easiest way to approximate tau
            //with this cellular representation.  It is also the method used in CAESAR
            double get_tau(elev_grid &elev, int x, int y){
                  double rho = 1000; //density of water kg per cubic meter
                  double s = 2.65; //specific gravity of quartz
                  double g = 9.81;  //gravitational acceleration in meters per second^2
                  if(max_depth[x][y]>0){
                        return rho*g*max_depth[x][y]*mean_bed_slope(elev,x,y);
                  }
                  else{
                        return 0;
                  }
            }


};


//*************************************************************************************************
//CLASS DEFINITION: SEDIMENT GRID
//The class sediment_grid calculates erosion per grid cell based on the grain diameter
//and local slopes at a specific grid cell.
//for now, we use a basic meyer-peter muller formula (1948) modified by Wong (2003), but this can be
//easily updated to something more robust such as the wilcock-crowe methods.
//Also, we assume monosize sediment, but this can be easily updated to account for varying proportions of
//grain sizes.

class sediment_grid{
      private:
            double temp_gradient[x_max][y_max];
            bool is_edge[x_max][y_max];//this grid tracks whether or not a cell is an edge cell,
                                       //that is a dry cell with a wet cell as a manhattan neighbor
            double radius_curve[x_max][y_max];//tracks how curvy the bank is at an edge cell
            double temp_radius[x_max][y_max]; //used when smoothing the values in radius_curve
            int adjacent_cell[x_max][y_max][2];//this 3d array will store the location of the cell
```

```
                                        //with the steepest slope attached to an edge cell
double dZ[x_max][y_max]; //this grid stores the elevation difference after erosion takes place
double delta_Qb[x_max][y_max]; //this grid will store the change in Qb after erosion has
                               //been calculated for a particular timestep

double total_delta_w[x_max][y_max]; //this grid stores how much the change in width within
                                    //a grid cell is, once it is DX, it should be reset to 0
double dH[x_max][y_max]; //this will store any bank failure volumes
double FS_grid[x_max][y_max]; //this will store the factor of safety for a an edge cell

double S;//used for calulating critical  shear stress

double D;//diameter of sediment (meters
double dimensionless_crit_tau, crit_tau;  //the critical shear stress, or the stress at which
                                          //the grains begin to move in units of Pa
double out_Qb;  //tracks the amount of sediment removed from the system for a given timestep
double lateral_gradient[x_max][y_max];



double calc_Qb(discharge_grid &discharge, elev_grid &elev,  int x, int y){
      double qb, dimensionless_qb;
      //qb is the volumetric transport rate for bedload per unit width,
                                    //dimensionless_qb is the same as above, but dimensionless
      double s = 2.65;              //specific gravity of quartz
      double rho=1000;              //density of water in kg per cubic meter
      double g = 9.81;  //gravitational acceleration due to gravity in meters per second^2
      double tau=discharge.get_tau(elev,x,y); //get shear stress at a particular cell
                                              //from the discharge grid
      double dimensionless_tau = tau/((s-1)*rho*g*D);//convert shear stress into dimensionless
                                                     //shear stress

      if(dimensionless_tau>dimensionless_crit_tau){
            dimensionless_qb = 3.97*pow((dimensionless_tau-
dimensionless_crit_tau),1.5);//calculate transport via Meyer-Peter Muller method mofidified by Wong (2003)
            qb = dimensionless_qb*pow((s-1)*g*pow(D,3),0.5);
            //calculate actual volumetric transport rate per unit width
```

```
                return qb*elev.get_DX(); //convert to actual volumetric transport rate per second

        }
        else{
                return 0;
        }
}

//the function get_ttl_slope sums up the slopes of all downslope neighbors
//this is used to route bedload proportionately from a source cell to receiving cells in
//the route_sediment function
double get_ttl_slope(elev_grid &elev, int x, int y){
        int x2, y2;
        double root = pow((double)2.0*pow((double)elev.get_DX(),2.0),0.5);//distance between
                                                                          //diagonal cells
        double total_slope = 0.0; //this will track the sum of the slopes of all
                                  //downslope neighbors

        assert(x>=1 && x<x_max-1);//make sure that we're not working with one of the
                                  //boundary cells
        assert(y>=1 && y<y_max-1);//make sure that we're not working with one of the
                                  //boundary cells


        for(x2=x-1;x2<=x+1;x2++){//this loop checks for downstream neighbors and adds their
                                 //slopes
                for(y2=y-1;y2<=y+1;y2++){
                        if(elev.get_elev(x,y)>elev.get_elev(x2,y2)){
                                if(x2!=x && y2!=y){
                                        total_slope += (elev.get_elev(x,y)-
elev.get_elev(x2,y2))/root;//for diagonal neighbors
                                }
                                else{
                                        total_slope += (elev.get_elev(x,y)-
elev.get_elev(x2,y2))/elev.get_DX();//for manhattan neighbors
                                }
                        }
```

```
                }

        }
        return total_slope;
    }

public:
    sediment_grid(){
    }

    void initialize(double grain_diameter){
        int x,y;
        double S;//S is a doubleing point number calculated from grain size, and used to
                 //derive dimensionless critical
                 //shear stress (also called shield's number) based on the shield's curve
                 //whose limit is 0.045
        const double rho = 1000; //density of water in kilograms per cubic meter
        const double g = 9.81; //gravitational acceleration in meters per second^2
        const double s = 2.65; //specific gravity of quartz.
                               //Many non-cohesive sediments can be quantified with this value

        D = grain_diameter;//get the grain size and then calculate the crit_tau
        S = ((pow(D,1.5))/(9.98*pow(10.0,-7.0)))*pow((s-1)*9.81,0.5);
        assert(S>=0);
        dimensionless_crit_tau = (0.105*pow(S,-0.3))+(0.045*exp(-35*pow(S,-0.59)));
        assert(dimensionless_crit_tau>=0);
        crit_tau = dimensionless_crit_tau*(s-1)*rho*g*D;



        //initialize the delta_Qb values to zero
        for(x=0;x<x_max;x++){
            for(y=0;y<y_max;y++){
                delta_Qb[x][y] = 0;
                dZ[x][y] = 0;
                is_edge[x][y]=false;
                radius_curve[x][y]=0;
                temp_radius[x][y]=0;
```

```
                total_delta_w[x][y]=0;
                FS_grid[x][y] = 0;
                dH[x][y]=0;
                //cumulative_delta_Qb[x][y] = 0;
            }
        }
}


//route_sediment checks the grid for all cells with a discharge above the defined
//minimum discharge.
//then it calculates the mobilized sediment per grid cell, and routes it proportionally to all
//downslope neighbors.
void route_sediment(discharge_grid &discharge, elev_grid &elev){
        int x,y;
        int x2, y2;
        double temp_dZ;
        double Qb; //tracks the change in sediment for a cell
        double total_S;//average downstream slope
        double temp_S;//stores slope between the pair of cells currently under investigation
        double current_elev; //stores the elevation of the current cell
        double root = pow((double)2.0*pow((double)elev.get_DX(),2.0),0.5);//distance between
        //diagonal cells
        for(x=0;x<x_max;x++){ //these nested for-loops reset dZ, is_edge, and radius_curve and
                              //delta_Qb for this timestep to zero
            for(y=0;y<y_max;y++){
                delta_Qb[x][y] = 0.0;
                dZ[x][y]=0.0;
                is_edge[x][y]=false;
                radius_curve[x][y]=0;
                temp_radius[x][y]=0;
                FS_grid[x][y]=9999;
                dH[x][y]=0;
            }
        }

        out_Qb = 0; //initialize sediment removed from system to 0
        for(x=1;x<x_max-1;x++){//we will only route sediment within grid cells that are
```

```
                    //not along the border of the dem
for(y=1;y<y_max-1;y++){


      if(discharge.max_depth[x][y]>0){ //test if Q exceeds minimum Q
                                       //(max_depth =0 if min_Q is not exceeded)

            //assign edge cells
            for(x2=x-1;x2<=x+1;x2+=2){
                  if(is_edge[x2][y]!=true && discharge.max_depth[x2][y]==0 && x2!=0
                  && x2!=x_max-1 && y != 0 && y != y_max-1){
                              is_edge[x2][y] = 1;
                  }
            }
            for(y2=y-1;y2<=y+1;y2+=2){
                        if(is_edge[x][y2]!=true && discharge.max_depth[x][y2]==0 &&
                        x!=0 && x!=x_max-1 && y2 != 0 && y2 != y_max-1){
                              is_edge[x][y2] = 1;

                        }
                  }


            Qb = calc_Qb(discharge, elev, x, y);
            delta_Qb[x][y] -= Qb;
            total_S = get_ttl_slope(elev,x,y); //get sum of downslopes
            current_elev = elev.get_elev(x,y);

            for(x2=x-1;x2<=x+1;x2++){
                        for(y2=y-1;y2<=y+1;y2++){

                              if(elev.get_elev(x2,y2)<current_elev){
                                    if((x2!=x) && (y2!=y)){
                                          temp_S = (current_elev-
                                          elev.get_elev(x2,y2))/root;
                                          //if diagonal neighbors
                                    }
```

```
                                    else{
                                          temp_S = (current_elev-
                                          elev.get_elev(x2,y2))/elev.get_DX();
                                          //if manhattan neighbors
                                    }
                                    if(total_S<=0){
                                          temp_S=0;
                                          total_S=1;
                                    }



                                    if(temp_S>total_S){
                                    //make sure that temp_s is less than total_S
                                          cout<<"WARNING temp_S > Total_S: temp_S,
                                          total_S: "<<temp_S<<","<<total_S;
                                    cin>>temp_S;
                              }
                              assert(temp_S < total_S);//|
                              delta_Qb[x2][y2]+= Qb*(temp_S/total_S);

                              }
                        }

                  }

            }
      }


//this final loop updates the elevations of the grid, and determines the sediment discharge
//leaving the grid
//default is to set time factor to minutes (=60), but we can make this variable during
//the erosion process below
      for(x=0;x<x_max;x++){
            for(y=0;y<y_max;y++){
                  double temp_sum=0; //for tracking edge cells
```

```
                        if (is_edge[x][y]){//calculate the difference between wet and dry cells to
                                           //determine if it outside or inside edge
                              for(x2=x-1;x2<=x+1;x2++){ //cycle through adjacent neighbors
                                    for(y2=y-1;y2<=y+1;y2++){
                                          if(discharge.get_max_depth(x2,y2)==0 &&
!(is_edge[x2][y2])){
                                                temp_sum+=1.0;//add 1 if the adjacent cell is dry,
                                                            //not the same cell as x,y, and is not
                                                            //an edge cell
                                          }
                                          else if(discharge.get_max_depth(x2,y2)>0){
                                          //if adjacent cell is wet, it must not be an edge cell,
                                          //so subtract
                                                temp_sum-=1.0;
                                          }
                                    }
                              }
                              radius_curve[x][y] = temp_sum;//assign value to radius_curve, if
                                                          //positive it's an outside bend and
                                                          //vice-versa
                              temp_radius[x][y] = temp_sum;

                        }

                        if (x==x_max-1 || y==y_max-1||x==0||y==0){ //remove sediment from the
                                                                  //boundary cells
                              out_Qb+=delta_Qb[x][y];
                              delta_Qb[x][y]=0;
                        }
                        else{

                                    temp_dZ = (delta_Qb[x][y]*time_factor)/(pow(elev.get_DX(),2));
                                    //convert volumetric bedload per second to change in elevation
                                    //per second

                                    if(sqrt(pow(temp_dZ,2.0))< 0.0000000000001){
                                    //test absolute value
```

```
                        temp_dZ = 0; //threshold to prevent calculation errors
                    }
                    dZ[x][y] = temp_dZ;

                }
            }
        }

//smooth the edge cells with multiple passes of the filter

//Pass #1
for(x=1;x<=x_max-1;x++){
        for(y=1;y<=y_max;y++){
                if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
        }
}

for(x=1;x<=x_max-1;x++){
        for(y=1;y<=y_max;y++){
                radius_curve[x][y]=temp_radius[x][y];
        }
}

//Pass #2
for(x=1;x<=x_max-1;x++){
        for(y=1;y<=y_max;y++){
                if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
        }
}

for(x=1;x<=x_max-1;x++){
        for(y=1;y<=y_max;y++){
                radius_curve[x][y]=temp_radius[x][y];
        }
}

//Pass #3
```

```
for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
      }
}

for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            radius_curve[x][y]=temp_radius[x][y];
      }
}

//Pass #4
for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
      }
}

for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            radius_curve[x][y]=temp_radius[x][y];
      }
}

//Pass #5
for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
      }
}

for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            radius_curve[x][y]=temp_radius[x][y];
      }
}
```

```
//Pass #6
for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
      }
}

for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            radius_curve[x][y]=temp_radius[x][y];
      }
}

//Pass #7
for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
      }
}

for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            radius_curve[x][y]=temp_radius[x][y];
      }
}

//Pass #8
for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
      }
}

for(x=1;x<=x_max-1;x++){
      for(y=1;y<=y_max;y++){
            radius_curve[x][y]=temp_radius[x][y];
```

```
		}
	}

	//Pass #9
	for(x=1;x<=x_max-1;x++){
		for(y=1;y<=y_max;y++){
			if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
		}
	}

	for(x=1;x<=x_max-1;x++){
		for(y=1;y<=y_max;y++){
			radius_curve[x][y]=temp_radius[x][y];
		}
	}

	//Pass #10
	for(x=1;x<=x_max-1;x++){
		for(y=1;y<=y_max;y++){
			if(is_edge[x][y]) temp_radius[x][y]=smooth_curvature(x,y);
		}
	}

	for(x=1;x<=x_max-1;x++){
		for(y=1;y<=y_max;y++){
			radius_curve[x][y]=temp_radius[x][y];
		}
	}



}


//this returns the dZ value for an x,y coordinate
double get_dZ(int x, int y){
	return dZ[x][y];
```

```
}


double smooth_curvature(int x, int y){
	int x2, y2;
	double count = 0.0;
	double sum = 0.0;
	for(x2=x-1;x2<=x+1;x2++){
		for(y2=y-1;y2<=y+1;y2++){
			if(is_edge[x2][y2]){
				sum+=radius_curve[x2][y2];
				count+=1.0;
			}

		}
	}
	return sum/count;

}

//this creates a grid with gradients from outside to inside bends.
//we multiply by 1000 to give us enough variance for the smoothers
void create_lateral_gradient(elev_grid &elev, discharge_grid &discharge){
	int x,y,x2,y2;


	for(x=0;x<x_max;x++){
		for(y=0;y<y_max;y++){
			lateral_gradient[x][y]=temp_radius[x][y]*1000;
			temp_gradient[x][y]=0;

		}
	}
	//write_gradient(elev,"initialgradient.txt");
	//pass a smoothing filter over it until we no longer have a wet cell without
	//a gradient value
```

```
int counter = 0; //becomes false when we have no grids with water without a
                 //gradient value

while(counter<3){
      //run the smoother once
      for(x=1;x<x_max-1;x++){
            for(y=1;y<y_max-1;y++){

                  if(discharge.get_max_depth(x,y)>0){//if cell is wet
                        for(x2=x-1;x2<=x+1;x2++){
                        //check neighbors for radius of curvature
                              for(y2=y-1;y2<=y+1;y2++){
                                    temp_gradient[x][y]+=lateral_gradient[x2][y2];
                              }
                        }
                        temp_gradient[x][y]= temp_gradient[x][y]/9;
                        //sum the curvatures and divide by 9
                  }
            }



      }
      //assign temp values into lateral gradient
      for(x2=0;x2<x_max;x2++){
            for(y2=0;y2<y_max;y2++){
                  lateral_gradient[x2][y2]=temp_gradient[x2][y2];

            }
      }


      counter++;
}

}
```

```
void create_pointbars(elev_grid &elev,discharge_grid &discharge, double h, int in_x, int in_y){
      bool filler = true;
      int x = in_x;
      int y = in_y;
      int x2, y2;
      int goto_x,goto_y;
      double min_gradient=999999999999.0;
      int flag = 0;

      if(x==0 || x==x_max ||y==0 ||y==y_max){
      cout<<"edge cell error!";
      exit(0);
      }

      while(filler){
            if(x>=x_max || y>=y_max||x<0||y<0){
            //    cout<<"Error out of bounds during point bar creation";
                  exit(0);
            }
      //cout<<"entered filler";
      for(x2=x-1;x2<x+1;x2++){ //check neighbors
                  for(y2=y-1;y2<y+1;y2++){
                        if(x2!=x && y2!=y){

                              if(discharge.get_max_depth(x2,y2)>0 && lateral_gradient[x2][y2] <
min_gradient){
                              min_gradient=lateral_gradient[x2][y2];
                              goto_x = x2;
                              goto_y = y2;
                        //    cout<<"else statemetn"<<'\n';
                              }
                              else
if(lateral_gradient[x][y]<0&&discharge.get_max_depth(x2,y2)==0){
                                    elev.update_elev(x,y,h);
                              filler = false;
```

```
                        //    cout<<"false\n";
                              flag =1;
                              }

                        }

                  }
      }
            //cout<<"before if\n";
            if(flag==0 && (min_gradient >= lateral_gradient[x][y])){
                  elev.update_elev(x,y,h);
                  filler=false;

                  //cout<<"false\n";
            }
            //cout<<"gotos x,y: "<<goto_x<<','<<goto_y<<"\n";

            x=goto_x;
            y=goto_y;

      }

      //cout<<"done making point bars";


}

void write_gradient(elev_grid &elev, string out_file){
int x,y;
ofstream outfile(out_file.data());
outfile<<elev.get_header();


for(y = y_max-1; y>=0; y--){
      for(x=0;x<x_max; x++){
      outfile<<lateral_gradient[x][y]<<" ";
      if(x+1==x_max){
            outfile<<'\n';
```

```
	}

	}
}

outfile.close();
}
//write_file outputs and ascii dem of eroded sediment per timestep
void write_file(elev_grid &elev, string out_file){
int x,y;
ofstream outfile(out_file.data());
outfile<<elev.get_header();


for(y = y_max-1; y>=0; y--){
	for(x=0;x<x_max; x++){
	outfile<<dZ[x][y]<<" ";
	if(x+1==x_max){
		outfile<<'\n';
	}

	}
}

outfile.close();
}

void write_edges(elev_grid &elev, string out_file){
	int x,y;
ofstream outfile(out_file.data());
outfile<<elev.get_header();


for(y = y_max-1; y>=0; y--){
	for(x=0;x<x_max; x++){
	outfile<<temp_radius[x][y]<<" ";
	if(x+1==x_max){
		outfile<<'\n';
```

```
	}

	}
}

outfile.close();


}

void write_delta_w(elev_grid &elev, string out_file){
	int x,y;
ofstream outfile(out_file.data());
outfile<<elev.get_header();


for(y = y_max-1; y>=0; y--){
	for(x=0;x<x_max; x++){
	outfile<<total_delta_w[x][y]<<" ";
	if(x+1==x_max){
		outfile<<'\n';
	}

	}
}

outfile.close();


}

void write_FS(elev_grid &elev, string out_file){
	int x,y;
ofstream outfile(out_file.data());
outfile<<elev.get_header();


for(y = y_max-1; y>=0; y--){
```

```
        for(x=0;x<x_max; x++){
        outfile<<FS_grid[x][y]<<" ";
        if(x+1==x_max){
              outfile<<'\n';
        }

        }
}
outfile.close();
}

void write_dH(elev_grid &elev, string out_file){
      int x,y;
ofstream outfile(out_file.data());
outfile<<elev.get_header();


for(y = y_max-1; y>=0; y--){
      for(x=0;x<x_max; x++){
      outfile<<dH[x][y]<<" ";
      if(x+1==x_max){
            outfile<<'\n';
      }

      }
}


outfile.close();
}

//bank_erosion_osman_thorne determines lateral erosion with a method adapted from
//Osman and Thorne (1988), Riverbank Stability Analysis. I: Theory, Journal of Hydraulic
// Engineering 114: 2 (134-150)

// this is a simplified method, adapted for use for non-cohesives.
//critical shear stresses determined by D50
//The parameters x&y represent dry edge cell, x2,y2 represent wet edge cell
```

```
//this returns the amount of sediment to be removed from the edge cell
double bank_erosion_osman_thorne(int x, int y, int x2, int y2, elev_grid &elev, discharge_grid
                                 &discharge){

    double delta_w;          //the amount of lateral erosion to occur for a given elevation change
    double dw;                          //used to calculate delta_w per Osman&Thorne 1988
    double dimensionless_crit_tau;      //shields number, based on the curve approaching 0.045
    double crit_tau;                    //critical tau based on shields number

    double s=2.65;                      //specific gravity of quartz
    double rho = 1000;                  //density of water in kilograms per meter
    double g = 9.81;                    //acceleration of gravity in SI units (m per sec^2)
    double R;                           //initial rate of soil erosion per Osman&Thorne 1988
    double dB;                          //overall lateral migration rate of bank base in meters per
                                        //second
    double gamma = rho * g;             //unit weight of water in N per cubic meter
    double H;                           //height of bank (elev of dry cell - elev of wet cell
    double H0;                          //old height of bank (old elev of dry cell - old elev of wet
                                        //cell)
    double Hprime;                      //height of bank at a distance delta_w from center of wet cell
                                        //in the direction of the dry cell
    double cprime;                      //coefficient of cohesion for a soil
    double c;                           //value derived from FS and cprime
    double phiprime;                    //internal angle of friction (radians)
    double phi;                         //value derived from phiprime and FS
    double FS;                          //Factor of safety, ratio of driving forces to resisting forces
    double K;                           //determined by value of tension_crack_length and H
    double tension_crack_length=0;      //length of tension cracks before bank failure
                                        //(here we assume 0)
    double beta;                        //failure plane angle (radians)
    double i;                           //previous bank angle before erosion (radians)
    double H_ratio_m;                   //H/H'm per Osman and Thorne 1988
    double H_ratio_c;                   //H/H'c per Osman and Thorne 1988
    double lambda_1;                    //lambda 1,2, and 3 are used to calculate H_ratio_c
    double lambda_2;
    double lambda_3;
    double Wt;                          //weight of the failure block
    double BW;                          //width of the failure block
```

```
double VB;                          //volume of the failure block per unit length of channel

//many of the lines below that involve calculating values could be aggregated, but
//I like the readability of the calculations split up.

//this function makes the assumption that if the calculated crit_tau is greater than
//actual tau, no lateral migration takes place.  This might be an oversimplification,
//because it does not consider hiding effects.
//The idea here is that we are determining if the bank is getting oversteepened by
//toe scour, which results in either bed degradation or lateral bank movement or both.


double tau=discharge.get_tau(elev,x2,y2);
//get shear stress at a particular cell from the discharge grid



//lets assign some soil properties based on the D50.  I understand a poorly sorted soil
//can have the same D50 as a well sorted, but for simplicity's sake we assume the D50
//is representative
//we use a simple look up table, derived from Andrew Simon, Andrea Curini, Robert Thomas
//and Eddy Langendoen
//bank stability and toe erosion model (http://www.ars.usda.gov/Research/docs.htm?docid=5044)
if (D >= .0000625) //if gravel or sand
{
    phiprime = (36 * Pi) / 180; //convert degrees to radians, trig functions use radians
    cprime = 0; //units of Pa
}
else if (0.0000039 <= D && D < 0.0625)//if silt
{
    phiprime = (25 * Pi) / 180;//convert degrees to radians, trig functions use radians
    cprime = 5000; //units of Pa
}
else
{//if clay  (units here are for stiff clay)
    phiprime = (10 * Pi) / 180;//convert degrees to radians, trig functions use radians
    cprime = 15000;//units of Pa
```

```
}
// cout<<"assigned soil properties\n";


//for Osman-thorne's relations, need to convert tau from pascal to dynes per cm^2
      double crit_tau2 = crit_tau;
crit_tau2 *= 10;
      tau*=10;

//calculate inital R
R = (223 * pow(10.0, -4.0)) * crit_tau2 * exp(-0.13 * crit_tau2);

//need to convert R from units of gm/cm^2*min to KN/m^2*min.
//in their paper they use a conversion factor of 0.098066 to go from old R to new R.
//I also divide the old R to go from units per minute to units
//per second

R *= (0.098066/60);
dB = R / gamma;

      if(tau > crit_tau2){
            dw = dB * ((tau - crit_tau2) / crit_tau2);
      }
      else{
            dw=0;
      }
      if(isnan(dw)){
            cout<<"dw isnan"<<'\n';
            string checker;
            cin>>checker;
      }
      if(isnan(total_delta_w[x][y])){
            cout<<"total_delta_q[x][y] isnan\n";
            string checker;
            cin>>checker;
      }
delta_w = dw * time_factor; //gives me my lateral movement for current timestep
      total_delta_w[x][y] += delta_w;//need to add this in C#
```

```
      if(total_delta_w[x][y] >= elev.get_DX()){
      //this condition creates instabilities, so just remove cell at this point
            total_delta_w[x][y]=0;
            elev.update_original_elev(x,y,elev.get_elev(x,y));
            string checker;
            cin>>checker;
            return elev.get_elev(x,y)-elev.get_elev(x2,y2);
            //this makes the cell even with the bed
      }

      //next lines define i, H, H0, and Hprime
      H = elev.get_original_elev(x,y) - elev.get_elev(x2,y2);
      H0= elev.get_original_elev(x,y) - elev.get_old_elev(x2,y2);

      //this means too much deposition has occurred
      if((H<=0)||(H0<=0)){
            return 0;
      }

//the if-statement below represents H' = Tan(alpha)*(Dx-delta_w), but for diagonal neighbors
//DX becomes sqrt(2DX^2).  Also, alpha represents angle of pre-erosion bank slope with
//respect to horizontal, so tan(alpha) is H0/DX.
//Failure slope angle is also determined in this step
//Beta is determined by assuming FS = 1, and then using a relationship that relates Beta
//to phi, where phi = phi_prime/FS, so phi = phi_prime (per Osman and Thorne 1988)

      if(x2!=x && y2!=y)
{       //test if they are not manhattan neighbors, since diagonals affect dX

   Hprime = (H0/(pow(2*pow(elev.get_DX(),2.0),0.5)))*(pow(2.0*pow(elev.get_DX(),2.0),0.5)-
            total_delta_w[x][y]);
   i = atan(H0 / (pow(2 * pow(elev.get_DX()-total_delta_w[x][y], 2.0), 0.5)));


}
else
```

```
{
            Hprime = (H0/elev.get_DX())*(elev.get_DX()-total_delta_w[x][y]);
            i = atan(H0 /(elev.get_DX()-total_delta_w[x][y]));


}

//calculate K and beta
K = tension_crack_length / H;

beta = 0.5 * ((atan(pow(H / Hprime, 2) * (1 - pow(K, 2)) * tan(i))) + phiprime);

//use beta to calculate FS, Wt, c, and phi
Wt=(gamma/2)*(((pow(H,2)-pow(tension_crack_length,2))/tan(beta))-(pow(Hprime,2)/tan(i)));

FS = ((((H-tension_crack_length)*cprime)/sin(beta))+(Wt*cos(beta)*tan(phiprime)))
      /((gamma/2)*(((pow(H,2)-pow(tension_crack_length,2))/tan(beta)) -
      (pow(Hprime,2)/tan(i)))*sin(beta));


      FS_grid[x][y] = FS; //store factor of safety in grid


      if(isnan(FS) || isinf(FS)){
      //this is for debugging purposes, need to make sure FS is a real number
            cout<<"Error-FS value invalid: "<<FS<<"\nPress enter to exit program\n";

            cout<<"H0: "<<H0<<'\n';
            cout<<"DX: "<<elev.get_DX()<<'\n';
            cout<<"total_delta_w[x][y]: "<<total_delta_w[x][y]<<'\n';
            cout<<"FS: "<<FS<<'\n'<<"VB: "<<VB<<'\n'<<"dH: "<<dH[x][y]<<'\n';
            cout<<"H: "<<H<<'\n';
            cout<<"beta: "<<beta<<'\n';
            cout<<"Hprime: "<<Hprime<<'\n';
            cout<<"i: "<<i<<'\n';
            cout<<"tau: "<<tau<<'\n';
            cout<<"crit_tau2: "<<crit_tau2<<'\n';
            cout<<"cprime: "<<cprime<<'\n';
```

```
            cout<<"phiprime: "<<phiprime<<'\n';
            cout<<"WT: "<<Wt<<'\n';
            string checker;
            cin>>checker;
            write_FS(elev,"error_FS.txt");

            exit(0);
      }
      if(FS>=1.3){
            return 0;}
      else{

            return .005;
            }
      }
```

```
void fail_banks(elev_grid &elev, discharge_grid &discharge){
      int x,y,x2,y2;
      double total_slope =0;
      double DX = elev.get_DX();
      double root = pow(2.0*pow(DX,2),0.5);

      //go through all the edge cells and see if they fail


      create_lateral_gradient(elev, discharge);
      //create the lateral gradient to make point bars


      for(x=0;x<x_max;x++){
            for(y=0;y<y_max;y++){
```

```
double temp_slope=0; //for tracking steepest slope
double max_slope = 0;
int temp_x,temp_y;
double h,temp_h; //depth of sediment to move from bank to bed
h=0.0;
if (is_edge[x][y] && temp_radius[x][y]>0){
      for(x2=x-1;x2<=x+1;x2++){
      ///cycle through adjacent neighbors to find steepest slope
            for(y2=y-1;y2<=y+1;y2++){
                  if(discharge.get_max_depth(x2,y2)>0){
                  //if adjacent cell is wet, it must not be an edge cell,
                  so subtract
                        //calculate slope to x2,y2
                        if (x2!=x && y2!=y){
                              temp_slope = (elev.get_elev(x,y)
                              -elev.get_elev(x2,y2))/root;
                        }
                        else{
                              temp_slope = (elev.get_elev(x,y)-
                              elev.get_elev(x2,y2))/DX;
                        }

                        //calculate total slope, and determine which
                        //neighbor has the most volume removed

                        total_slope+=temp_slope;
                        //store total slope so that we can proportion the
                        //failed sediment
                        temp_h =
                        bank_erosion_osman_thorne(x,y,x2,y2,elev,discharge)*
                        (1+temp_radius[x][y]);

                        if(temp_h > h){
                              h=temp_h;
                              temp_x=x2;
                              temp_y=y2;
                        }
```

```
            }


      }
}


elev.update_elev(x,y,-1*h);

for(x2=x-1;x2<=x+1;x2++){ //cycle through adjacent neighbors to find
                          //those of lower elevation
                          //we want to fail the adjacent edge cells
                          //as well, so we fail them too
      for(y2=y-1;y2<=y+1;y2++){


            if(discharge.get_max_depth(x2,y2)>0){
            //if adjacent cell is wet
                  //calculate slope to x2,y2
                  if (x2!=x && y2!=y){
                        temp_slope = (elev.get_elev(x,y)-
                        elev.get_elev(x2,y2))/root;
                  }
                  else{
                        temp_slope = (elev.get_elev(x,y)-
                        elev.get_elev(x2,y2))/DX;
                  }

                  //route sediment to wet cells below failed bank
                  // proportionately based on slope/total slopt

                  assert(temp_slope >= total_slope && total_slope>0);

                  //route the point bars
```

```
create_pointbars(elev,discharge,h*(temp_slope/total_slope),x2,y2);
                                                          /
                                                    }
                                              }


                                        }
                                  }



                              }
                        }
                  }


                  //returns whether or not it is an edge cell
                  bool get_edge(int x, int y){
                        return is_edge[x][y];
                  }

                  //the function get_outQb returns the amount of bedload removed from the grid
                  double get_outQb(){
                        return out_Qb;
                  }

                  //get_Qb gets the bedload in cubic meters per second for cell x,y
                  double get_Qb(int x, int y){
                        return delta_Qb[x][y];
                  }

                  //returns the radius of curvature at cell x,y
                  double get_Curvature(int x, int y){
                        return temp_radius[x][y];
                  }
};
```

APPENDIX C

CODE ADDED TO CAESAR

```
//Delbert's global arrays
  double[,] original_elev; //the original elevation for a grid cell. If a bank fails, the resulting elevation is assigned to the
                           //original_elev grid cell.
  double[,] total_delta_w; //the amount of lateral toe erosion within a grid cell. grid cells are discrete, so I needed a way to
                           //remember continuous lateral toe erosion.
  double[,] ot_removal;  //the amount of sediment removed from a bank as a result of my algorithm.
  //end Delbert
```

****

Next is the function that is called to determine if a bank should fail. It is adapted from Osman and Thorne (1988).

****

```
  //bank_erosion_osman_thorne determines lateral erosion with a method adapted from
  //Osman and Thorne (1988), Riverbank Stability Analysis. I: Theory, Journal of Hydraulic Engineering 114: 2
   //(134-150)
  //Added by Delbert Humberson 14 November, 2007
  // this is a simplified method, adapted for use for non-cohesives.  critical shear stresses determined by D50
  //The parameters x&y represent dry edge cell, x2,y2 represent wet edge cell
  double bank_erosion_osman_thorne(int x, int y, int x2, int y2){

    double delta_w;           //the amount of lateral erosion to occur for a given elevation change
    double dw;                //used to calculate delta_w per Osman&Thorne 1988
    double dimensionless_crit_tau;  //shields number, based on the curve approaching 0.045
    double S;                 //used in determining shield's number, represents S*

    double s=2.65;            //specific gravity of quartz
    double rho = 1000;        //density of water in kilograms per meter
    double g = 9.81;          //acceleration of gravity in SI units (m per sec^2)
    double R;                 //initial rate of soil erosion per Osman&Thorne 1988
    double dB;                //overall lateral migration rate of bank base in meters per second
    double gamma = rho * g;       //unit weight of water in N per cubic meter
    double H;                 //height of bank (elev of dry cell - elev of wet cell
```

```
double H0;              //height of bank (old elev of dry cell - original elev of wet cell)
double Hprime;          //height of bank at a distance delta_w from center of wet cell
                        //in the direction of the dry cell
double crit_tau;        //critical tau, point at which sediment moves
double tau2;            //will be used to store tau in dynes per second
double cprime;          //coefficient of cohesion for a soil
double c;               //value derived from FS and cprime
double phiprime;        //internal angle of friction (radians)
double phi;             //value derived from phiprime and FS
double FS;              //Factor of safety, ratio of driving forces to resisting forces
double K;               //determined by value of tension_crack_length and H
double tension_crack_length=0;  //length of tension cracks before bank failure (here we assume 0)
double beta;            //failure plane angle (radians)
double i;               //previous bank angle before erosion (radians)
double D = d50(index[x2,y2]);   //the D50 of the wetcell
double Wt;              //weight of the failure block
double VB;              //volume of the failure block per unit length of channel


//this function makes the assumption that if the calculated crit_tau is greater than actual tau,
//no lateral migration takes place. This might be an oversimplification, because it does not consider
//hiding effects. The idea here is that we are determining if the bank is getting oversteepened by
//toe scour, which results in either bed degradation or lateral bank movement or both.
//Tau is retrieved by: Tau[x2,y2]
//here we calculate the dimensionless crit tau as well as the crit tau.
S = ((Math.Pow(D,1.5))/(9.98*Math.Pow(10,-7)))*Math.Pow((s-1)*9.81,0.5);
dimensionless_crit_tau = (0.105*Math.Pow(S,-0.3))+(0.045*Math.Exp(-35*Math.Pow(S,-0.59)));
crit_tau = dimensionless_crit_tau*(s-1)*rho*g*D;
```

```
//lets assign some soil properties based on the D50.  I understand a poorly sorted soil
//can have the same D50 as a well sorted, but for simplicity's sake we assume the D50 is representative
//we use a simple look up table, derived from Andrew Simon, Andrea Curini, Robert Thomas and
// Eddy Langendoen
//bank stability and toe erosion model (http://www.ars.usda.gov/Research/docs.htm?docid=5044)

if (D >= .0000625) //if gravel or sand
{
   phiprime = (36 * Math.PI) / 180; //convert degrees to radians, trig functions use radians
   cprime = 0; //units of Pa
}
else if (0.0000039 <= D && D < 0.0625)//if silt
{
   phiprime = (25 * Math.PI) / 180;//convert degrees to radians, trig functions use radians
   cprime = 5000; //units of Pa
}
else
{//if clay  (units here are for stiff clay)
   phiprime = (10 * Math.PI) / 180;//convert degrees to radians, trig functions use radians
   cprime = 15000;//units of Pa
}



//for Osman-thorne's relations, need to convert tau from pascal to dynes per cm^2

crit_tau *= 10;
tau2 = Tau[x2,y2]*10;

//calculate inital R
R = (223 * Math.Pow(10.0, -4.0)) * crit_tau * Math.Exp(-0.13 * crit_tau);
```

```
//need to convert R from units of gm/cm^2*min to KN/m^2*min.  in their paper they use a conversion factor
//of 0.098066 to go from old R to new R.  I also divide the old R to go from units per minute to units
//per second


R *= (0.098066/60);
dB = R / gamma;
if(tau2 > crit_tau){
        dw = dB * ((tau2 - crit_tau) / crit_tau);
}
else{

        dw=0;
}

delta_w = dw * time_factor; //gives me my lateral movement for current timestep
total_delta_w[x,y] += delta_w;



//now we test to make sure that we did not exceed the width "DX" of a cell with out total_delta_w
if(total_delta_w[x,y] >= DX){
        total_delta_w[x,y]=0;
        original_elev[x,y] = elev[x2,y2];
        //if we do exceed DX, then we want to reset total_w to zero, update the original_elev of the
         //dry cell to that of the wet cell, then return the difference in height between the wet and dry.
         //Essentially this makes it so the dry cell has failed and is the same height as the wet cell.

        return elev[x,y]-elev[x2,y2]; //this makes the cell even with the bed
}
```

```
//next lines define i, H, H0, and Hprime
H = elev[x,y] - elev[x2,y2];
H0 = elev[x,y]- original_elev[x2,y2];

//this means too much deposition has occurred
if((H<=0)||(H0<=0) || H<H0){
        return 0;
}


//the if-statement below represents H' = Tan(alpha)*(Dx-delta_w), but for diagonal neighbors
//DX becomes sqrt(2DX^2).  Also, alpha represents angle of pre-erosion bank slope with respect to
//horizontal, so tan(alpha) is H0/DX.  Failure slope angle is also determined in this step
//Beta is determined by assuming FS = 1, and then using a relationship that relates Beta to phi,
//where phi = phi_prime/FS, so phi = phi_prime (per Osman and Thorne 1988)


if(x2!=x && y2!=y)
{   //test if they are not manhattan neighbors, since diagonals affect dX
        Hprime = (H0/(Math.Pow(2*Math.Pow(DX,2.0),0.5)))*(Math.Pow(2.0*Math.Pow(DX,2.0),0.5)
        -total_delta_w[x,y]);
        i = Math.Atan(H0 / (Math.Pow(2 * Math.Pow(DX-total_delta_w[x,y], 2.0), 0.5)));
}
else
{
        Hprime = (H0/DX)*(DX-total_delta_w[x,y]);//change in C#
        i = Math.Atan(H0 /(DX-total_delta_w[x,y]));
 }
```

```
        //calculate K and beta
        K = tension_crack_length / H;
        beta = 0.5 * ((Math.Atan(Math.Pow(H / Hprime, 2) * (1 - Math.Pow(K, 2)) * Math.Tan(i))) + phiprime);

        //use beta to calculate FS, Wt, c, and phi
        Wt=(gamma/2)*(((Math.Pow(H,2)-Math.Pow(tension_crack_length,2))/Math.Tan(beta)) –
        (Math.Pow(Hprime,2)/Math.Tan(i)));

        FS = ((((H-tension_crack_length)*cprime)/Math.Sin(beta)) + (Wt*Math.Cos(beta)*Math.Tan(phiprime)))
                /((gamma/2)*(((Math.Pow(H,2)-Math.Pow(tension_crack_length,2))/Math.Tan(beta)) -
                (Math.Pow(Hprime,2)/Math.Tan(i)))*Math.Sin(beta));

        if(FS>=1.3){
                return 0;
        }
        else{

                return .001;
        }
}
```

****

Below is code for saving the amount of sediment removed from banks due to osman-thorne into a grid. "menuItem12" is a checkbox that is checked if you want to save the final elevations after a model run.

****

```
if (menuItem12.Checked == true)
        {
          save_data(3, 0); // save elevations
          save_data(18, 0); //save ot_removal ...added by Delbert
```

```
}
```

****

Next we have the code that initializes the values for my global arrays (original_elev, total_delta_w, and ot_removal). Included are snippets of CAESAR's source code.

****

```
void load_data(){
        int x,y=1,z,xcounter,x1=0,y1=0,n;
        String input;
        double tttt=0.00;

        // initialize total_delta_w and ot_removal to 0 -- Added by Delbert 29 July 2008
        int x_count, y_count;
        for (x_count = 0; x_count < xmax; x_count++)
        {
           for (y_count = 0; y_count < ymax; y_count++)
           {
              total_delta_w[x_count, y_count] = 0;
              ot_removal[x_count, y_count] = 0;
           }

        }
        //end Delbert's addition

//CAESAR code removed here for readability purposes

for (x = 0; x<=(lineArray.Length-1); x++){
        if(lineArray[x]!=""&&xcounter<=xmax){
                        tttt=double.Parse(lineArray[x]);
                        elev[xcounter,y]=tttt;
                        init_elevs[xcounter,y]=elev[xcounter,y];
```

```
                    original_elev[xcounter, y] = tttt; //added by Delbert, initialize original elev
                    xcounter++;
                    }
        }
y++;
```

****

The following one-liner is some text that is spit out on the message-bar when loading the data in CAESAR. It alerts the user that they are working with the modified CAESAR rather than the original version.

****

```
this.InfoStatusPanel.Text="Initialising Delbert's Version..";
```

****

Below is the code that creates the instances of the newly declared global arrays.

****

```
 old_elev = new double[xmax+2, ymax+2];//added by Delbert Humberson 14 November 2007
 original_elev = new double[xmax+2, ymax+2];//added by Delbert 29 July 2008
  total_delta_w = new double[xmax, ymax];//added by delbert 29 July 2008
  ot_removal = new double[xmax+2, ymax+2];//added by Delbert 23 Sep 2008
```

LITERATURE CITED

Beven, K.J., and M.J. Kirkby. 1979. A physically based variable contributing-area model of catchment hydrology. *Hydrological Science Bulletin* 24, no. 1: 43-69.

Blum, Michael D. *Modern Depositional Environments and Recent Alluvial History of the Lower Colorado River, Gulf Coastal Plain of Texas*. Ph.D. diss., University of Texas, 1992.

Casagli, Nicola, Massimo Rinaldi, Alessandro Gargini, and Andrea Curini. Pore Water Pressure and Streambank Stability: Results From a Monitoring Site on the Sieve River, Italy. *Earth Surface Processes and Landforms* 24: 1095-1114.

Chin, Anne. 2006. Urban transformation of river landscapes in a global context. *Geomorphology* 79: 460-487.

City of Austin. 2007. *City of Austin - Seaholm District History*. http://www.ci.austin.tx.us/seaholm/history.htm, accessed september 2 2007, last updated date not available.

Coulthard, Tom. 2001. Landscape evolution models: a software review. *Hydrological Processes* 15: 165-173.

Coulthard, Tom, M.J. Kirkby, and M.G. Macklin. 2000. Modelling geomorphic response to environmental change in an upland catchment. *Hydrological Processes* 14: 2031-2045.

Coulthard, Tom, M.J. Macklin, and M.J. Kirkby. 2002. A cellular model of holocene upland river basin and alluvial fan evolution. *Earth Surface Processes and Landforms* 27: 269-288.

Coulthard, Tom, and Marco Van De Wiel. 2006a. A cellular model of river meandering. *Earth Surface Processes and Landforms* 31: 123-132.

Coulthard, Tom, and Marco Van De Wiel, 2006b. The Cellular Automaton Evolutionary Slope and River model (CAESAR). In *Accounting for Sediment in Rivers.*, N. Wallerstein, ed. University of Nottingham, UK: Flood risk management research consortium, pp. 101-119.

Doeschl-Wilson, Andrea B., and Peter E. Ashmore. 2005. Assessing a numerical cellular braided-stream model with a physical model. *Earth Surface Processes and Landforms* 30: 519-540.

Eaton, Brett C. 2006. Bank Stability Analysis for Regime Models of Vegetated Gravel Bed Rivers. *Earth Surface Processes and Landforms* 31: 1438-1444.

Fonstad, Mark A. 2006. Cellular Automata as Analysis and Synthesis Engines at the Geomorphology-Ecology Interface. *Geomorphology* 77, no. 3-4: 217-234.

Fonstad, Mark A. and Andrew W. Marcus. 2005. Remote Sensing of Stream Depths with Hydraulically Assisted Bathymetry (HAB) Models. *Geomorphology* 72, no. 4: 320-339.

Hill, R.T., and T.W. Vaughn. 1897. *Geology of the Edwards Plateau and Rio Grande Plain Adjacent to Austin and San Antonio, Texas, with Reference to the Occurrence of Groundwaters.* Annual Report 18, United States Geological Survey. Washington, D.C.

Hill, R.T., and T.W. Vaughn. 1902. *Description of the Austin Quadrangle.* USGS Atlas Folio 76, United States Geological Survey. Washington, D.C.

Langendoen, E.J., A. Simon, and R.E. Thomas. 2001. *CONCEPTS- A process-based modeling tool to evaluate stream-corridor restoration designs*. Proceeding 2001 Wetland Engineering & River Restoration Conference, Reno, Nevada, D.F. Hayes, ed., ASCE, Reston, VA, on CDROM.(259KB).

Lower Colorado River Authority. 2007a. *INDEX-LCRA dams form the Highland Lakes.* http://www.lcra.org/water/dams/index.html, accessed 2 September 2007, last updated 20 July 2007.

Lower Colorado River Authority, 2007b. *Dams and lakes.* http://www.lcra.org/about/overview/maps/dams_lakes.html, accessed 2 September 2007, last updated 7 May 2007.

McGowen, J.H., and L.E. Garner. 1970. Physiographic features and stratification types of coarse-grained point pars: Modern and ancient examples. *Sedimentology* 14: 77-111.

Meyer-Peter, E. and R. Müller. 1948. Formulas for bedload transport. In: *Proc. IAHR* (2nd meeting, Stockholm, Sweden), 39-64.

Murray, Brad A., and Chris Paola. 1994. A cellular model of braided rivers. *Nature* 371: 54-57.

Murray, Brad A., and Chris Paola. 1997. Properties of a cellular braided-stream model. *Earth Surface Processes and Landforms* 22: 1001-1025.

Nicholas, Andrew. 2005. Cellular modeling in fluvial geomorphology. *Earth Surface Processes and Landforms* 30: 645-649.

Osman, Akode, and Colin Thorne. 1988. Riverbank Stability Analysis: I: Theory. *Journal of Hydraulic Engineering* 114, no. 2: 134-150.

Paola, C. 2000. Modelling stream braiding over a range of scales. In *Gravel Bed Rivers 5*. Hydrological Society Inc.: Wellington, New Zealand; 11–38.

Parker, G. 1990. Surface-based bedload transport relation for gravel rivers. *Journal of Hydraulic Research* 28, no. 4: 417-436.

Philip, G.M., and D.F. Watson. 1982. A Precise Method for Determining Contoured Surfaces. *Australian Petroleum Exploration Association Journal* 22: 205-212.

Proffitt, G.T., and A.J. Sutherland. 1983. Transport of nonuniform sediments. *Journal of Hydraulic Research* 21, no. 1: 33-43.

Prosser, I.P. 1996. Thresholds of channel initiation in historical and Holocene times, south-eastern Australia. In *Advances in Hillslope Processes,* Volume 2. M.G. Anderson, and S.M. Brooks (eds). Chichester: John Wiley and Sons, pp. 687-708.

Robert, André. 2003. *River Processes, An Introduction to Fluvial Dynamics*. London: Arnold.

Sears, Robert Bonner. *Selective Modification of Sand Size Alluvium, Colorado River, Texas*. Masters thesis, University of Texas, 1978.

Simon, A., A. Curini, S.E. Darby, and E.J. Langendoen. 2000. Bank and near-bank processes in an incised channel. *Geomorphology* 35: 193-217.

Taylor, Thomas U. 1900. *The Austin Dam*. Department of the Interior Water-Supply and Irrigation Papers of the United States Geological Surey No. 40, Washington Government Printing Office.

Tobler, W. R. 1970. A computer movie simulating urban growth in the Detroit region. *Economic Geography* 46: 234–240.

Trimble. 2000. *AgGPS 124/132 Operation Manual.* Trimble Navigation Limited. http://trl.trimble.com/docushare/dsweb/Get/Document-9665/Ag124_132%20Rev%20C1.pdf, accessed 10 October 2008.

US Army. 2006a. *Users Guide To RMA2 WES Version 4.5*. Mississippi: US Army Engineer Research and Development Center Waterways Experiment Station Coastal and Hydraulics Laboratory. http://chl.erdc.usace.army.mil/Media/3/2/7/RMA2_v45_Users_Guide_01-20-006.pdf, accessed 31 March 2007.

US Army. 2006b. *Users Guide To SED2D WES Version 4.5*. Mississippi: US Army Engineer Research and Development Center Waterways Experiment Station Coastal and Hydraulics Laboratory. tp://chl.erdc.usace.army.mil/Media/3/3/1/SED2D_Users_Guide_Draft_1-20-006.pdf, accessed 31 March 2007.

US Army Corps of Engineers. 2006a. *Hydrologic Modeling System HEC-HMS: User's Manual.* California: US Army Corps of Engineers Institute for Water Resources Hydrologic Engineering Center. http://www.hec.usace.army.mil/software/hec-hms/documentation/HEC-HMS_Users_Manual_3.1.0.pdf, accessed 31 March 2007.

US Army Corps of Engineers. 2006b. *HEC-RAS River Analysis System: User's Manual.* California: US Army Corps of Engineers Institute for Water Resources Hydrologic Engineering Center. http://www.hec.usace.army.mil/software/hec-ras/documents/HEC-RAS_v4.0_Users_Manual.pdf, accessed 31 March 2007.

US Geological Survey. No Date. *Summary of DR3M.* http://water.usgs.gov/cgibin/man_wrdapp?dr3m, accessed 31 March 2007, last updated date not available.

US Geological Survey, 2007. *USGS Real-Time Water Data for USGS 08158000 Colorado Rv at Austin, TX,* National Water Information System: Web Interface. http://waterdata.usgs.gov/nwis/dv?cb_00060=on&format=gif_default&begin_date=1898-03-01&end_date=2007-09-01&site_no=08158000&referred_module=sw, accessed on 2 September 2007.

Van De Wiel, Marco J., Tom J. Coulthard, Mark G. Macklin, and John Lewin. 2007. Embedding reach-sclae fluvial dynamics within the CAESAR cellular automaton landscape evolution model. *Geomorphology* 90: 283-301.

Van De Wiel, Marco, and Stephen Darby. 2007. A new model to analyse the impact of woody riparian vegetation on the geotechnical stability of riverbanks. *Earth Surface Processes and Landforms* (in press).

Watson, D.F., and G.M. Philip. 1985. A Refinement of Inverse Distance Weighted Interpolation. *Geoprocessing* 2: 315-327.

Werner, B.T. 1999. Complexity in Natural Landform Patterns. *Science* 284: 102-104.

Wilcock, Peter R., and Joanna C. Crowe. 2003. Surface-based Transport Model for Mixed-Size Sediment. *Journal of Hydraulic Engineering* 129, no. 2: 120-129.

Wolman, M.G. 1954. A method of sampling coarse river-bed material: *Transactions of the American Geophysical Union* 35: 951-956.

Wolman, M.G. 1967. A cycle of sedimentation and erosion in urban river channels. *Geografiska Annaler* 49A: 385-395.

Wong, Miguel, and Gary Parker. 2006. Reanalysis and Correction of Bed-Load Relation of Meyer-Peter and Müller Using Their Own Database. *Journal of Hydraulic Engineering* 132, no. 11: 1159-1168.

Wynn, Tess. 2006. Streambank Retreat: A Primer. *Watershed Update* 4, no. 1: 1-13.

VITA

Delbert Geronimo Humberson was born in Hopewell, Virginia, on November 27, 1979, the son of Erlinda Omabtang and Robert Humberson. After graduating from Jakarta International School, Jakarta, Indonesia, in 1997, he entered the University of Texas at Austin in 1998. During his time there, he worked in several jobs, including serving as a Geographer for the U.S. Geological Survey focused on Geographic Information Systems. Delbert received a Bachelor of Arts with special honors in Geography in 2005. Upon receiving his undergraduate degree, Delbert continued his work at the U.S. Geological Survey and entered the Graduate College of Texas State University-San Marcos in the Fall of 2005. Currently, Delbert is working as a Hydrologist with the U.S. International Boundary and Water Commission in El Paso, Texas.

Permanent Address: 3216 Lorne Road

El Paso, Texas 79925

This thesis was typed by Delbert. G. Humberson.