COASTLINE EVOLUTION MODELING: IMPLICATIONS OF MEGA-NOURISHMENTS AND GROIN FIELDS

A Dissertation

by

ANDREW EDWARD WHITLEY

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Timothy Dellapenna
Co-Chair of Committee,	Jens Figlus
Committee Members,	Wesley Highfield
	Juan Horrillo
Head of Department,	Shari Yvon-Lewis

December 2022

Major Subject: Oceanography

ABSTRACT

As many coastal areas suffer from chronic erosion, innovative solutions beyond traditional local nourishment must be explored. One such solution is mega-nourishment (MN) where a large sediment volume is deposited in a single location and redistributed via natural processes thereby feeding adjacent beaches. However, only one MN, the Dutch Sand Motor, has been built to date, and potential MN coastline interactions with engineered features such as groins are unknown. To investigate this, one-line numerical modeling approaches of MN evolution are presented using the Coastline Evolution Model (CEM) and GenCade. CEM is modified to allow for a much more detailed parameterization and higher resolution operation to address MNs specifically. Both models are parameterized for the Sand Motor to explore the feasibility of the MN approach. The models are then used to identify the implications of combining a MN with a groin field (GF). This is accomplished by re-writing CEM in MATLAB and adding highly robust groin-simulating algorithms.

Both models are able to reproduce the morphological patterns observed at the Sand Motor. GenCade produces mean measured-modeled differences on the order of 50 m showing relatively accurate absolute shoreline positions. CEM captures the 400-meter-feature tip migration due to its wave shadowing algorithm. CEM groin algorithms are validated using measured shorelines from Galveston Island. The idealized MN-GF interaction scenarios are modeled using similar baseline conditions. Results also show that a MN placed adjacent to a GF results in sediment feeding of beaches on both sides of the MN and on the far side of the GF on multidecadal time scales. MN feeding rates are highly dependent on the offshore wave climate, with climates rich in high angle waves slowing MN diffusion and feeding rates. Model results

also indicate that shoreline advance can be maximized in areas of erosional hot spots near a GF by building a MN on top of a groin field or directly downdrift of it. The advances made to CEM allow its use in future MN research, which should include the examination of different MN/GF parameters as well as temporal variability in forcings.

DEDICATION

This dissertation is dedicated to my wonderful wife, Veronica, and our incredible children, Kateri and Arthur. This work would not have been possible without their love and support.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Timothy Dellapenna for his support throughout this dissertation. I would also like to thank my co-chair and primary research advisor Dr. Jens Figlus for all his time and guidance while producing this research. Many thanks to my committee members Dr. Juan Horrillo and Dr. Wesley Highfield for their support.

I would like to thank my co-authors Dr. Antonios Valsamidis and Dr. Dominic Reeve for their help in producing the peer-reviewed journal articles that make up this dissertation.
I would like to thank Bruce Ebersole and Wesley Highfield for their help with the background theory and research approach during the NSF-PIRE fellowship. Many thanks to Matthieu de Schipper, Sierd de Vries, Juan Horrillo, Bas Huisman, Arjen Luijendijk**,** and Kathelijne Wijnberg for their helpful discussions through the course of this research. Thanks to Yan Ding, Sung-Chan Kim, Rusty Permenter, and Richard Styles at USACE for their support with GenCade. I would also like to thank Andrew Ashton, Kenneth Ells, and Brad Murray for their helpful discussions with CEM. Many thanks to my colleagues and the administrators at San Jacinto College who supported me through much of this work.

Special thanks to my family: Kateri Joy Safranski; Arthur Joseph Safranski; John C. Whitley, III; Helen S. Whitley; and Yianni Whitley for their love and support. Above all, I would like to thank my wife Veronica Nicole Brumbaugh Whitley for always being there for me and never giving up on me.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by Professor Jens Figlus, Ph.D. (co-advisor and primary research advisor) of the Department of Ocean Engineering and Professor Timothy Dellapenna, Ph.D. (advisor) of the Department of Oceanography. This work was also supervised by a dissertation committee consisting of Professor Juan Horrillo, Ph.D. of the Department of Ocean Engineering and Professor Wesley Highfield, Ph.D. of the Department of Marine and Coastal Environmental Science.

Co-authors on Chapter 3 of this dissertation include Dr. Jens Figlus, Dr. Antonios, Valsamidis, and Dr. Dominic Reeve. Dr. Valsamidis was heavily involved in co-writing Section 3.2.1 and in developing the theory behind the groin algorithms added to CEM in Chapter 4. Co-authors on Chapter 4 include Dr. Jens Figlus and Dr. Antonios Valsamidis. All other work in this dissertation was completed independently by the student.

Funding Sources

Graduate study was supported by a Graduate Assistantship (Teaching) from Texas A&M University – Galveston Campus from Fall 2014 to Spring 2017. Much of the Chapter 3 work was completed under the 2017 NSF-PIRE Coastal Flood Risk Reduction Program Fellowship. This work was supported by the National Science Foundation under Grant No. OISE-1545837 under Dr. Jens Figlus and the MORPHINE project, EPSRC Grant No. EP/N007484/1, under Dr. Dominic Reeve and Dr. Antonios Valsamidis. This work was supported from June 2017 to August 2017 by the Texas Institute of Oceanography Graduate Research Grant. For the Spring

2017 semester and from Fall 2019 to present, the student was supported by San Jacinto College where he was appointed as a Professor of Physics in the Department of Physical Sciences (South Campus).

NOMENCLATURE

2D	Two (2) Dimensional
3D	Three (3) Dimensional
BC	Boundary Conditions
CEM	Coastline Evolution Model (Ashton *et al.*, 2001)
CERC	Coastal Engineering Research Center (USACE)
Delft3D	Deltares 3D hydrodynamic/morphodynamic software (Lesser *et al.*, 2004)
DHI	Delft Hydraulics Institute
DOI	Digital Object Identifier System
ERDC	Engineering Research and Development Center (USACE)
GenCade	GENESIS + CasCade Shoreline Model (Frey *et al.*, 2012)
GENESIS	Genesis shoreline model (Hanson and Kraus, 1989)
GF	Groin Field
GLO	[Texas] General Land Office
LITPACK	Littoral processes package for MIKE (DHI, 2009)
LST	Longshore Sediment Transport
MATLAB	MATrix LABoratory (programming language and software)
MIKE	DHI water modeling software (DHI, 2009)
MLW	Mean Low Water
MN	Mega-nourishment
NDBC	National Data Buoy Center
NOAA	National Oceanic and Atmospheric Administration

NSF	National Science Foundation
OISE	Office of International Science and Engineering (NSF)
ONELINE	Queen's University coastline model (Dabees and Kamphuis, 1999)
PIRE	Partnerships for International Research and Education
Q2D-morpho	Quasi-2D morphology model (van den Berg *et al.*, 2012)
PDF	Probability Distribution Function
RMS	Root Mean Square
RMSD	Root Mean Square Difference
SM	Sand Motor (*i.e.,* the Delftland Sand Engine or *Zandmotor*)
SMS	Surface-water Modeling System (Aquaveo, 2017)
TGLO	Texas General Land Office
UNIBEST	Deltares Coastline Model (Deltares, 2011)
U.S.	United States
USACE	United States Army Corps of Engineers

Table N.1. List of variables

A	Offshore wave climate asymmetry
A_W	Conversion factor for bypassing
B	Fractional amount of sediment bypassing a groin
$C_{g,b}$	Breaking wave group celerity
D_{50}	Median sediment grain diameter
D_B	Berm elevation
D_C	Depth of closure
D_G	Water depth at groin tip
D_{LT}	Depth of longshore transport
d_g	Longshore distance between wave breaking point and groin
E	Deep water wave energy contribution to shoreline diffusivity
E_{nor}	Normalized E
E_{tot}	Cumulative E
F	Fractional amount of sediment in CEM cell
g	Acceleration due to gravity
H	Wave height
$H_{0,rms}$	Offshore root-mean-square wave height
H_0	Offshore wave height (either RMS or significant can be used)
$H_{0,s}$	Offshore significant wave height
H_b	Breaking wave height
H_b'	Breaking wave height (due to refraction/diffraction)
H_i	Wave height at groin tip
K	Empirical proportionality coefficient (CERC Equation)
K_2	Proportionality constant (GenCade LST equation)
k_2	Proportionality constant
K_d	Diffraction coefficient
L_g	Cross-shore length of groin
n	Sediment porosity
P	Fractional groin permeability

Table N.1. Continued

p_f	Fractional probability of offshore wave angle
Q_{in}	Sediment flux into CEM cell
Q_o	Amplitude of LST rate
Q_{out}	Sediment flux out of CEM cell
Q_s	Volumetric longshore sediment flux
S_s	Slope of shelf
S_{sf}	Slope of shoreface
t	Time
T	Wave period
U	Offshore wave climate highness
W	Model cell width
x	Alongshore position of coastline
y	Cross-shore position of coastline
z_{max}	Water depth where refraction begins
Dt	Model time step
Dx_{tip}	Longshore tip migration
Dy_{max}	Change in max. cross-shore shoreline extent (*i.e.,* tip retreat)
α_1	Non-dimensional parameter (GenCade)
α_2	Non-dimensional parameter (GenCade)
α_b	Breaking wave angle (relative to shoreline)
α_{bd}	Breaking wave angle due to diffraction
α_s	Mean wave direction at groin tip
β	Average bathymetric slope
γ_b	Wave breaking index
ε	Diffusion coefficient
θ	Shoreline angle
ρ	Water density
ρ_s	Sediment density
ϕ_o	Offshore wave angle of incidence (relative to x-axis)
ϕ_b	Breaking wave angle (relative to x-axis)

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. EXECUTIVE SUMMARY

Considering the large amount of human infrastructure on the coast and that much of the world's shores suffer from chronic erosion, innovative coastal defenses beyond traditional nourishment must be explored. One solution pioneered by the Dutch is the use of mega-nourishment (MN), where a large volume of sediment is deposited at a single nourishment site. This sediment is then redistributed via natural processes, thus feeding nearby beaches. Since many developed beaches have been fortified with groin fields (GF), robust predictive capabilities on interactions between MNs and groins are needed.

At the time of writing, only one MN has been built, the Dutch Sand Motor. This MN was not built in the vicinity of any groins, and as such, there is no field data on how a MN and a GF may interact. Furthermore, all field data on MNs is for a single site in the Netherlands, and as such, there is no field data on how a MN may evolve in other areas of the world. As a result, numerical modeling is necessary to fill these knowledge gaps.

One-line models are often the preferred method of numerical modeling in engineering applications given their simplicity and relatively low data requirements. Limited one-line modeling has been performed in MNs to this point, and thus, the capabilities of specific one-line models to simulate MN evolution is not well understood. Therefore, two one-line numerical modeling methods using the Coastline Evolution Model (CEM) and GENESIS + Cascade (GenCade) are presented to evaluate their

capabilities in modeling MN coastline evolution. The models are then used to investigate the implications of building a MN on a coast with a GF.

CEM and GenCade have been used to simulate large-scale coastal evolution, and GenCade is the current U.S. industry standard. Since CEM was designed as a theoretical model, it is highly modified here to explore specific study sites and operate at higher spatial resolution (25 m). As CEM's original version written in the programming language C does not have the ability to simulate groins, a new CEM version coded in MATLAB is presented here. The new version includes the mentioned modifications as well as robust groin simulating algorithms.

Both models are found capable of reproducing the morphological patterns observed at the SM and are therefore suitable for MN simulation. GenCade produces a low measured-modeled difference over the entire MN area (on the order of 50 m) showing relatively accurate predictions of absolute shoreline position. CEM measured-modeled differences are typically on the order of 5 m greater than GenCade, and it can capture the 400-meter feature tip migration due to its wave shadowing algorithm. The best results from both models can be obtained for higher K values than those commonly suggested in the literature.

The groin simulating algorithms added to CEM are validated using measured shoreline evolution data from Galveston Island, Texas, including a GF comprised of 13 (out of 15) groins. Measured-modeled differences in Galveston over 2 years of evolution (2014 to 2016) are on the order of 12-16 m for CEM while GenCade's differences are on the order of 30 m.

The idealized MN-GF interaction scenarios are modeled using similar baseline conditions to that of Galveston. Model results show that a MN placed adjacent to a GF results in sediment feeding of beaches on both sides of the MN and on the far side of the GF on multidecadal time scales. MN feeding rates are highly dependent on the offshore wave climate, with climates rich in high angle waves slowing MN diffusion and feeding rates. Model results also indicate that shoreline advance can be maximized in areas of erosional hot spots (downdrift of a GF) by building a MN on top of a groin field or directly downdrift of it.

Future research on MN-GF interaction should include the examination of different MN-GF configurations including, but not limited to, ranges of MN-GF area ratios, groin lengths and spacing, and MN aspect parameters. It is also recommended that the effects of temporal variations in offshore wave climate and K be examined for the MN-GF situation. Furthermore, hypothetical MN scenarios for specific study sites around the world should be explored. The incorporation of additional processes in CEM and/or GenCade such as cross-shore processes, sediment sources/sinks, and offshore advection may further improve the accuracy of the models.

2. INTRODUCTION

Considering that 70% of the world's coastlines are eroding (Davison *et al.*, 1992) with roughly 23-40% of the world's population living within 100 km of the coastline (Small and Nicholls, 2003), it is clear that a large amount of infrastructure is vulnerable to retreating coastlines. Continuing development of coastal regions combined with sea level rise increases the damage risk from shoreline retreat (Gornitz *et al.*, 1994). As a result, coastal defense schemes aiming to maintain the coastline position have become common in populated coastal regions. Hard coastal defenses, such as groins, jetties, breakwaters, dams, dykes, and seawalls, are very costly, permanently alter the natural environment, and may come with unforeseen long-term effects on coastal evolution (van Slobbe *et al.*, 2013). Soft defense schemes generally involve beach nourishment, which is less costly and more environmentally sustainable. However, nourished sediment is often transported out of the region requiring repetitive nourishments (Peterson and Bishop, 2005).

The problems of chronic erosion and sediment scarcity call for innovative shore defense concepts beyond traditional approaches. An innovative technical solution is mega-nourishment (MN), a relatively new concept that relies on natural forces to disperse a large sediment volume at a single nourishment site. An effective MN acts as a feeder beach, providing a sediment source for nearby shores. This could be potentially more cost-effective than recurring smaller nourishments as all of the nourishment would occur at a single site at one point in time rather than nourishing at multiple sites along a beach multiple times. A pilot MN, known as the Delftland Sand Motor (SM), or

Zandmotor in Dutch, was constructed in 2011 on the Dutch coast to examine the feasibility of MNs feeding larger stretches of the coast. MN could be applied to beaches around the world where natural threats to the coast can be very dangerous and destructive. However, the SM has not evolved entirely as expected raising issues related to the local ecosystem, recreational and navigational safety, ground water, and potential changes to nearshore processes (Stive *et al.*, 2013). This necessitates additional modeling efforts to improve the predictive capabilities of potential MN projects.

One-dimensional (one-line) coastline models can be used to predict shoreline morphology without considering the full complexity of the coastal system (Zacharioudaki and Reeve, 2008). These do not have large data requirements to evaluate, have shown to be remarkably robust despite their simplicity, and have remained the preferred method of evaluating potential shoreline change in many engineering applications (Thomas and Frey, 2013). However, most analytical solutions to these models cannot incorporate time-varying input data or account for longshore variations in wave breaking. To mitigate shortcomings in analytical one-line models, numerical one-line models have been developed that can account for time-varying input data, highly oblique wave angles, and complex boundary conditions. Commonly used numerical coastline models in the U.S. include GENESIS (Hanson and Kraus, 1989) and GenCade (Frey *et al*., 2012). These models can account for longshore variations in wave conditions, wave refraction and diffraction, and the effects of man-made structures such as groins and seawalls. GENESIS typically operates on domains ranging from 1 to 30 km over time periods up to 20 years, making it unsuitable for region-wide simulations

(Gravens *et al.*, 1991). GenCade couples GENESIS with Cascade (Larson *et al.*, 2003), a regional-scale coastline model, allowing GenCade to simulate coastlines on spatial scales up to hundreds of meters and time scales up to millennia. GenCade has become the industry standard in one-line numerical modeling for engineering projects in the U.S. Several other one-line models developed in other countries are also available. These include ONELINE (Dabees and Kamphuis, 1999), LITPACK (DHI, 2009) and UNIBEST (Deltares, 2011).

Given the processes included and the spatial/temporal scales involved, GenCade is a suitable numerical model for MN simulation, and it is used in this dissertation. However, there does appear to be a disconnect between the models used in the engineering community for practical applications (*e.g.*, GenCade) and those used in the scientific community for theoretical studies (Syvitski *et al.*, 2009). Theoretical models are often oversimplified while industrial software packages, though more complex, are not easily modifiable to incorporate more physical processes that may not have been adequately investigated in the past. In an effort to bridge that gap and expand the number of model options available in coastal research, a one-line model designed for theoretical studies is also used here in the examination of MN evolution: the Coastline Evolution Model (CEM). While GenCade is commonly used in the coastal engineering community, CEM has been used primarily in theoretical scientific studies to explore shoreline morphology on very large temporal (thousands of years) and spatial (hundreds of kilometers) scales (Ashton *et al.*, 2001). Unlike GenCade, CEM is capable of representing an offshore wave climate stochastically rather than using a time series,

allowing small potential changes in wave climate to be readily examined (*e.g.*, Moore *et al*., 2013). CEM also has the advantage that it is a self-contained open-source model capable of running without overhead unlike other models that require additional software (*e.g.,* GenCade project work often requires the Surface-Water Modeling System). Considering the scales on which CEM operates, it should be ideal for large coastline formations such as MNs that evolve on large spatial and temporal scales.

While some one-line modeling work has been applied to MNs such as the semi-analytical methods presented in Valsamidis *et al*. (2017) and a probabilistic shoreline approach by Kroon *et al.*, (2020), the accuracy of specific one-line models in the reproduction of observed MN morphology (such as CEM or GenCade) is not well understood. Furthermore, many coastal areas suffering from chronic erosion have been reinforced with groin fields in an effort to mitigate sediment loss. While a MN could potentially be advantageous at such sites, the interactions between a MN and a groin field are unknown. One-line modeling of MN coastline interactions with a groin field could shed light on erosion and deposition patterns that could result from a physical MN built near a groin field.

The one-line models presented here are primarily driven by wave-induced longshore sediment transport (LST; see the Literature Review in Chapter 2 for details). When waves approach the coast and break at an oblique angle, they induce a nearshore water flow in the longshore direction (Komar and Inman, 1970). The wave action also suspends sediment particles on the seabed. The particles are suspended in the nearshore flow resulting in a longshore sediment flux (Q_S). When simulating a protruding

shoreline feature such as a Gaussian-shaped MN in a one-line model, changes to the coastline are caused by longshore sediment flux gradients $\left(\frac{\partial Q_S}{\partial x}\right)$ as positive fluxes (more sediment leaving than going in) cause erosion and negative fluxes cause accretion (Figure 2.1). For instances when waves primarily approach from low angles (less than 45 degrees), this has the effect of causing the feature tip to retreat while the flanks extend (Falqués, 2003). When simulating groin fields, shores updrift of the groin field experience accretion due to the negative sediment flux as less sediment is able to be transported due to groin inhibition (Basco, 2002). However, LST inhibition from the groins also causes sediment starvation downdrift of the groin field. Despite the general trends expected in modeling nourishments and groin fields individually, the interactions of a large-scale nourishment coastline with a nearby groin field are not well understood. Furthermore, the performance of GenCade and CEM in simulating accurate MN evolution is untested prior to this work.

The research presented in the dissertation consists of two research projects. The first project examines the feasibility of one-line numerical coastline modeling methodologies in the application of mega-nourishment (MN) simulation. This requires significant modification of CEM to operate at higher spatial and temporal resolutions than usual and allow for additional topographic and wave refraction parameters to be user-specified that are not present in the default version of the model. The second project examines the interactions between MN coastline evolution and groin fields. While both MNs and groin fields have the same purpose of delaying the impacts of shoreline erosion, they have opposite mechanisms by which this is achieved. In light of

the more recent shift from only hard to soft or hard/soft combinations of erosion risk reduction schemes, the necessity to assess the issues of MN/groin field combinations becomes important. In order to use one-line models (CEM, in particular) to evaluate this, additional algorithms to properly model groin behavior (including LST inhibition, groin bypassing, and wave diffraction from groins) were developed and added to CEM in this project. GenCade already contained this functionality.

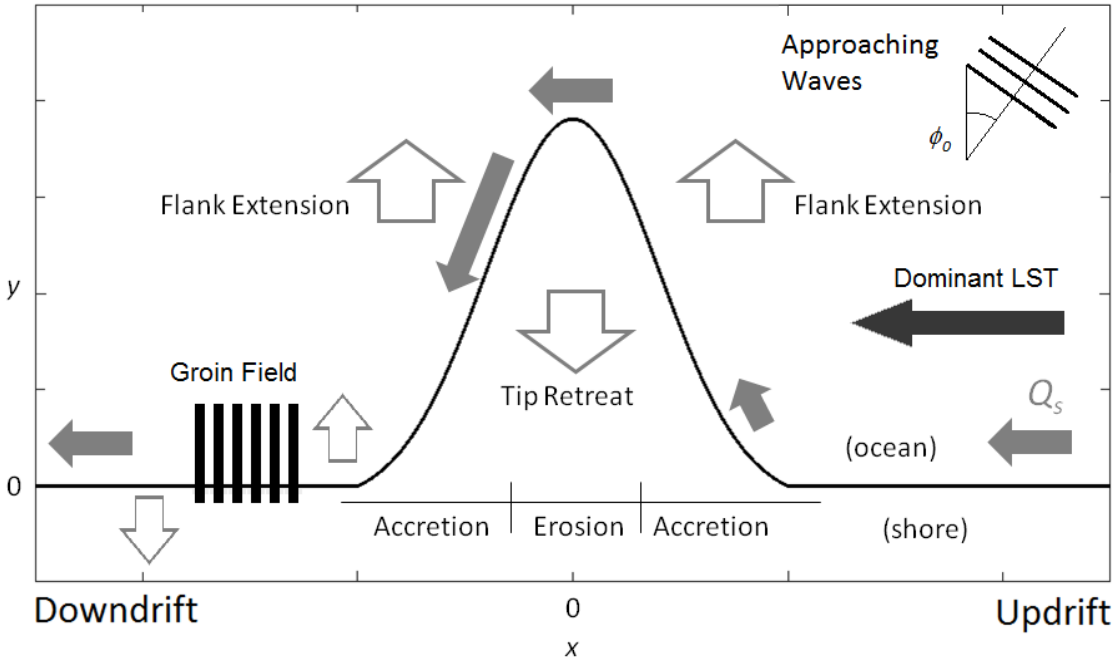

Figure 2.1. A schematic diagram of coastline changes for a simulation of a Gaussian-shaped coastline with a groin field downdrift of the feature. Here dominant LST is downdrift and offshore waves are predominantly low angle ($|\phi_0| < 45^o$). Sediment fluxes (Q_s) are indicated by gray arrows (scaling indicates magnitude) while shoreline change is indicated by white ones. Beach areas where sediment influxes are greater than outfluxes (positive flux gradient) experience accretion (coastline extension) while the opposite gives rise to erosion (coastline retreat). Under most circumstances, sediment accumulates updrift of groin fields due to LST inhibition while sediment starvation causes erosion downdrift of the field.

The dissertation first presents a literature review (Chapter 3) of relevant coastal risk reduction measures, the state-of-the-art in coastline modeling, and the MN concept. The details of the first research project in evaluating the feasibility of using CEM and GenCade in MN simulation is next included in the dissertation (Chapter 4). This is followed by the investigation of MN shoreline evolution in combination with groin fields (Chapter 5). Both research projects have accompanying articles that have been published in or submitted to peer-reviewed journals. Chapter 6 presents overall conclusions for the dissertation.

2.1. Research Questions

2.1.1. One-Line Modeling of Mega-Nourishment Evolution

The first research project in the course of the dissertation examines one-line numerical modeling methods in the exploration of mega-nourishment (MN) evolution. As limited amounts of one-line modeling have been applied to MNs, the performance of specific models such as CEM and GenCade in the evaluation of MN evolution is not well understood. Furthermore, an appropriate *K* value in the CERC Transport Equation is up for debate as a MN model parameter (see Section 3.4). Through the course of this project, the following research questions (RQs) are addressed:

RQ1. Much one-line modeling has been performed with traditional nourishments and natural coastlines, but relatively little one-line modeling has been done with

MNs. Can MN evolution be reasonably simulated using one-line models (specifically CEM and GenCade)?

It is hypothesized that the models are capable of reasonably reproducing observed MN evolution given proper calibration. CEM has been used to show the evolution of large-scale features in the literature, and the scales at which GenCade operate should make it ideal for MN simulation.

RQ2. The *K* value in the CERC Transport Equation is used to relate available wave power to longshore sediment fluxes. Are commonly used *K* values given in the literature adequate for MN simulation in a one-line model (specifically CEM and GenCade)?

While a number of methods presented in the literature derive *K* from field data to determine a relationship between immersed weight transport and wave power, Rosati *et al*. (2002) use a linear regression over a very large data set of field observations to obtain the value of $K = 0.92$, which is one of the more recent determinations. It is therefore hypothesized that this value should be adequate to simulate a MN.

RQ3. What are the major differences between the shorelines produced by CEM compared to those produced by GenCade when modeling MN evolution?

CEM's wave transformation technique includes a wave shadowing algorithm while GenCade's default wave transformation (used here) does not. It is hypothesized that these wave shadowing algorithms in CEM can lead to areas of sediment starvation that result in asymmetric evolution of a modeled MN under an asymmetric wave climate (as observed at the SM).

2.1.2. Numerical Modeling of Mega-Nourishment Shoreline Interactions with a Groin Field

The second research project explores the interactions between MN coastlines and groin fields. As no mega-nourishment thus-far built is near a groin field, it is unknown how the presence of a groin field affects MN evolution. It is important to ascertain these effects as many beaches that would greatly benefit from a MN are already reinforced with groin fields (*e.g.*, Galveston Beach). Therefore, the following research questions are addressed numerically in this project:

RQ4. Many coastlines that would benefit from a MN have been reinforced by groin fields. However, MNs are designed to feed adjacent beaches via LST while groins are designed to inhibit LST. What are the implications of

combining these two features on the same coastal system?

It is expected that as a MN diffuses, a nearby groin field will inhibit LST across it, thus reducing the diffusion rate of the MN. However, with such a large supply of sediment added from the MN, it is hypothesized that beaches on the side of the groin field opposite the MN will be fed as there are a number of mechanisms for sediment to be transported past the groins, specifically bypassing and through-passing (due to groin permeability).

RQ5. Different coasts around the world can experience vastly different wave climates. What effects do wave climate (specifically, the probability distribution of offshore wave angles) have on shoreline evolution in a coupled MN-groin field system?

An asymmetric wave climate should be accompanied by a dominant direction of LST. When a MN is placed updrift of a groin field, it is expected that this should result in augmented accumulation of sediment updrift of the groin field. Under normal circumstances (with no MN), this should be accompanied by sediment starvation downdrift of the field. However, with such a large sediment supply from the MN, it is hypothesized that the overall rate of erosion updrift of the groin field will be diminished from that without a MN present. Consequently, if a MN is placed downdrift of a groin field, it is still expected that the MN should diffuse in both directions longshore. This

should significantly reduce the erosion signal normally present on the downdrift side of the groin field. Furthermore, it is hypothesized that a higher probability of high-angle ($|\phi_0| > 45^0$) incident waves in the wave climate should reduce the rate of MN diffusion.

RQ6. Beaches with groin fields where a dominant direction of littoral drift is present often suffer from an erosional hotspot downdrift of the groin field due to sediment starvation. Can this erosion be mitigated through mega-nourishment, and if so, under what conditions is the mitigation maximized?

It is hypothesized that building the mega-nourishment downdrift of the GF will maximize the mitigation of the erosion as the MN will provide a new source of sediment for the erosional hotspot.

3. LITERATURE REVIEW

3.1. Coastal Protection

In early U.S. history, the coastal zones of the United States were mostly uninhabited (Galgano, 2004). However, the development on barrier beaches increased drastically in the late 20th century. Since that time, coastal erosion has become a significant land management issue as financial losses from erosion and flood damage approach economically and politically unsustainable levels. Roughly 70% of beaches worldwide and 90% of U.S. beaches are reported to be eroding (Bird, 1985; Leatherman, 1988). Strategies to combat this problem involve the landward relocation of structures, raising infrastructure to account for projected flooding, and protection of the shore through engineering projects (Galgano, 2004).

While natural defenses such as dunes and beaches are a first line defense system against storm surge, flooding, and erosion (Bridges *et al*., 2015), the implementation of artificial defenses has become common in populated coastal regions (van Slobbe *et al*., 2013). Artificial coastal defense methods typically aim to maintain the shoreline front and are classified as either hard or soft defense schemes. Hard coastal defense systems include hardened structures such as groins (see Groin Fields section below), jetties, breakwaters, levees, dikes, and seawalls (Wijnberg, 2002; Dean and Dalrymple, 2004; van Slobbe *et al*., 2013). These have different design purposes. Seawalls are built as a defense against erosion where eroding beyond the seawall would result in infrastructure damage (Kamphuis, 2010). Levees and dikes protect against flooding from high water

levels while breakwaters protect beaches through water wave attenuation (Bridges *et al.*, 2015). Jetties have the purpose of maintaining a navigational inlet from infilling while groins stabilize the shoreline through LST inhibition (Bridges *et al.*, 2015; Galgano, 2004). While many of these hardened structures provide necessary protection during high-energy events, the implementation of hard coastal defenses is a very costly venture that permanently alters the natural environment (Valsamidis *et al.*, 2017). It has also been hypothesized that hard engineering structures may come with unforeseen effects on coastal evolution over long time scales, thus alternative strategies are desirable (van Slobbe *et al*., 2013).

Soft coastal defense schemes generally involve beach nourishment (see Beach Nourishment below), the replenishment of sediment on beaches to replace eroded sediment mass (Charlier and de Meyer, 1995; Dean and Dalrymple, 2004). This is less costly than hardened structures and more sustainable with respect to the natural environment. However, sediment placed during beach nourishment is often transported out of the area through natural processes. Therefore, scheduled recurring nourishments over time are required to maintain a baseline shore position. Furthermore, issues have also been raised concerning undesirable impacts of repetitive nourishment with respect to the local ecology (Peterson and Bishop, 2005; Speybroeck *et al*., 2006). Therefore, alternatives to frequent sediment replenishment are quite desirable.

3.2. Groin Fields

Groins are one of the oldest and most popular hard coastal defense schemes. As described in Basco (2002), these are structures built perpendicular to the coastline to control LST by inhibiting transport longshore. Groins are typically built on continuous beaches with the intent of anchoring the beach or extending the lifetime of a beach fill. The goal of groin placement on a continuous beach is to maintain a minimum dry beach width, which can reduce storm damage and minimize local coastline erosion in the vicinity of the groin. Typically, groin fields consisting of multiple groins are used to anchor a stretch of beach (Basco and Pope, 2004).

While groins do diminish LST rate, sediment is still capable of moving past groins by several mechanisms (Basco, 2002). Most groins are permeable, allowing some sediment to pass through them (called through-passing). Sediment can also over-pass by moving over the top of submerged groins. Sediment transport behind groins on the beach is also possible, called shore-passing or flanking (Basco and Pope, 2004). Bypassing of groins is also possible through hydrodynamic sediment transport as water flows around the tip of the groin. The amount of sediment bypassed is dependent on the ratio of groin length to the water depth at the groin tip and the water depth at which LST occurs. Bypassing can be accentuated when nearshore sand bars are present as they act as a conduit for maintaining LST regime seaward of the groin tips. When numerically modeling transport across groins, groin permeability, over-passing, and shore-passing are often all combined into a single term while bypassing is calculated separately (see

Methods for details). It is important to account for these effects in numerical models as they can greatly influence coastline morphology.

Groins also have the effect of water wave diffraction, particularly on waves that approach the coast at oblique angles (Basco, 2002). Waves diffract around the tip of groins diminishing their wave heights and angles as they approach the shore and break. This in turn diminishes the wave induced LST magnitude. Effects of wave diffraction from groins should be accounted when determining wave induced LST patterns near groin fields (see Methods for details on how this is modeled).

When a predominantly unidirectional LST is present, the use of groins often results in a sawtooth-like coastline shape as sediment accumulates on the groin side opposite the dominant direction of littoral drift. This is often accompanied by erosion on the other side of the groin as sediment transported away from the groin cannot be replenished due to the groin's LST inhibition (*i.e.*, sediment starvation). When a bidirectional LST is present, this erosion can occur on both sides of the groin resulting in a wave-like coastline form (Kamphuis, 2010). Longer impermeable groins may also have the effect of generating local rip currents that can jet material offshore accentuating this erosion, though the conditions required for this are up for debate (Basco and Pope, 2004).

The effects groin fields have on coastline evolution have been observed for decades (*e.g.*, Basco and Pope, 2004 and Galgano, 2004), and much numerical modeling has been done on the subject (see One-line Modeling with Groins below). However, both observations and models have focused on the coastline evolution of a natural

(generally straight) coastline with groins or that of traditional beach fill (of much smaller scale than a MN) in close proximity to groins. The long-term (decadal or greater) effects a groin field has on the evolution of larger coastline features (such as a MN) is unknown, and this knowledge is particularly relevant if MNs are to be applied to beaches where groin fields are already present.

3.3. Beach Nourishment (Beach Fill)

Beach nourishment (also known as beach fill) is the artificial placement of sediment on the coast with the intent of providing improved protection of upland infrastructure from storms (Gravens *et al*., 2002). During high energy events (*i.e.*, storms), the beach berm and dunes act as a protective buffer between rising water (and associated wave energy) and nearshore structures. As the erosion of these beach features reduces this protection, putting coastal infrastructure at risk, beach nourishment has become common in the U.S. as a means of flood and storm protection.

Reinforcement of the beach berm is typically the primary feature in nourishment projects (Gravens *et al.*, 2002). The goal is to widen the berm seaward, which creates a buffer for dissipating storm wave energy. This has the added effect of increasing the beach area, which can be used for recreation. Berm reinforcement is usually accomplished by transporting sediment onto the beach and placing it with earth-moving equipment. Sediment can also be placed in artificial nearshore sandbars during the nourishment processes. These subaqueous sandbars can act as wave breakers to dissipate additional wave energy.

As typical nourishments involve sediment placement along a finite length of the shoreline, some nourishments can be built to feed other beaches (Gravens *et al.*, 2002). Known as feeder beaches, these nourishments involve the placement of material on the updrift end of an area intended to receive the beach fill. Once the sediment is in place, natural forces redistribute the material to the rest of the project area. These tend to work best in areas with unusually high loss rates, where the net LST direction is predictable, and where strong natural LST is present.

Currently, frequent nourishment is required at many sites, usually once every 3 to 10 years (Cooke *et al*., 2012). As a result, large quantities of sand are required to supply beach nourishment projects, often in areas that already have large annual sand deficits (de Schipper *et al*., 2016). Furthermore, it remains questionable if recurring nourishments have detrimental effects on fauna and natural evolution of nourishment sites (Peterson and Bishop, 2005; Speybroeck *et al*., 2006).

It is highly desirable to have an understanding of how beach fill projects will evolve after placement and how long placed sediment can be expected to remain in the region. Many engineering projects utilize one-line coastline models as tools to predict possible beach evolution after a nourishment. The one-line model is explained below.

3.4. The One-Line Model

One-line coastline models can be used to predict shoreline morphology without considering the full complexity of the coastal system (Zacharioudaki and Reeve, 2008). These do not have large data requirements to evaluate, have shown to be remarkably

robust despite their simplicity, and have remained the preferred method of evaluating potential shoreline change in many engineering applications (Reeve, 2006; Thomas and Frey, 2013). The primary premise in the one-line model is that any cross-shore profile along the beach is in equilibrium (Valsamidis and Reeve, 2017). In other words, the seabed contours are parallel with the coast, and these adjust accordingly with cross-shore changes in the shoreline. In this case, the assessment of only one bathymetric contour is required, and, for practical reasons, the shoreline is chosen (Larson *et al.*, 1987). This premise also assumes that shoreline change is primarily driven by wave-induced LST. It is further assumed that sediment transported is spread evenly across the shoreface out to the depth beyond which wave action no longer influences LST. This depth is called the depth of closure (D_C). Moreover, the combination of the continuity of mass equation and a LST formula can be used for the formation of the mathematical expression of the one-line model. Assuming that LST takes place up to D_C but no further seaward, changes in cross-shore shoreline position (y) over time (t) can be calculated via (Larson *et al.*, 1997):

$$\frac{dy}{dt} = -\frac{1}{(D_C+D_B)}\frac{dQ_S}{dx}. \tag{3.1}$$

Here x is the alongshore position and D_B is the berm elevation corresponding to the part of the beach above sea level that contributes to LST, and Q_s is the LST rate. The assumptions in the one-line model generally provide good estimation of shoreline change under normal wave conditions but may not work as well under high energy events or with systems having highly variable seabed contours. For example, Goff *et al.*

(2010) report that during Hurricane Ike, sediment was transported off the shoreface to depths 4 times greater than the depth of closure and may never naturally recover to the beach. Therefore, the conservation of mass assumption in the one-line model may not be valid under high energy conditions.

A variety of sediment transport equations to calculate Q_s are shown in the literature, but the most prevalent is the CERC equation described in Komar and Imman (1970); Rosati *et al.* (2002); and Valsamidis *et al.* (2013). This equation describes a relationship between breaking wave energy and littoral transport. When waves break at an oblique angle relative to the shoreline, the wave energy generates longshore currents that carry beach sediment via suspended load, bed load, and saltation load sediment transport. The magnitude of volumetric sediment flux (Q_s) is dependent on the wave energy (quantified by breaking wave height H_b) and the direction of propagation at wave break with more oblique waves generating a higher flux due to a larger longshore vector component. The CERC equation states:

$$Q_s = K\left(\frac{\rho\sqrt{g}}{16{\gamma_b}^{\frac{1}{2}}(\rho_s-\rho)(1-n)}\right){H_b}^{\frac{5}{2}}\sin(2\alpha_b), \tag{3.2}$$

where ρ is the water density, g is the acceleration due to gravity, ρ_s is the sediment density, K is the empirical proportionality coefficient, n is the sediment porosity, and γ_b is the breaker index (ratio of H_b to the water depth at breaking). The breaking wave angle relative to the shoreline (α_b) is expressed as

$$\alpha_b = \phi_b - \theta\,, \tag{3.3}$$

where ϕ_b is the breaking wave angle and θ is the shoreline angle, both relative to the *x*-axis (Figure 3.1). Assuming wave refraction over straight and parallel contours, $\phi_b$is related to the offshore wave angle (ϕ_o) and θ according to the following relationship (*e.g.*, Reeve, 2006):

$$\phi_b = \phi_o - \arctan(\theta). \tag{3.4}$$

The shoreline angle θ is identical to the shoreline gradient $\frac{\partial y}{\partial x}$, and thus (3.3) can be written:

$$\phi_b = \phi_o - \arctan\left(\frac{\partial y}{\partial x}\right). \tag{3.5}$$

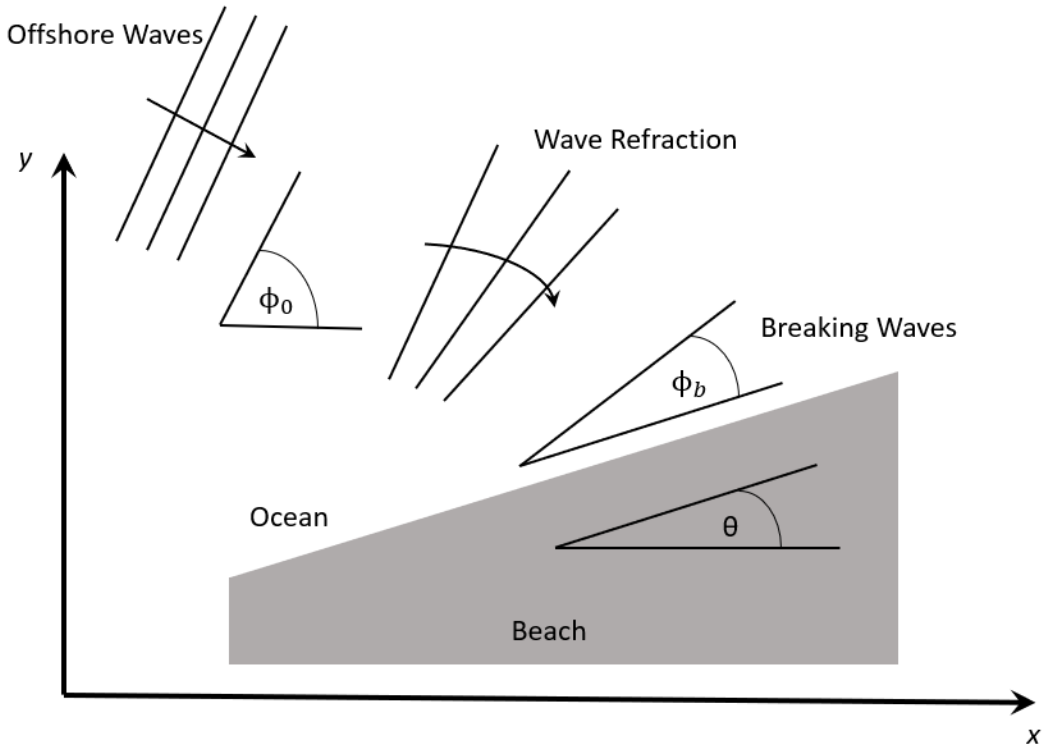

Figure 3.1. A schematic of wave and shore angles. ϕ_o is the offshore wave angle relative to the longshore (*x*) axis, ϕ_b is the breaking wave angle relative to the shoreline, and θ is the shoreline relative to the *x*-axis. Offshore waves undergo refraction and shoaling to the breaking point.

Note that the assumption of wave refraction over straight and parallel contours neglects how refraction over non-shore parallel contours may result in longshore changes in wave height and wave angle. This phenomenon may become more important as shoreline features extend cross-shore and is thus a limitation of the model (Ashton and Murray, 2006a; Whitley, 2014).

The proportionality coefficient K in (3.2) is used to relate breaking wave energy to the volumetric sediment transport rate Q_s. It is generally determined via a relationship between the immersed sediment weight transport rate (I_l) with the longshore component of wave energy flux (P_l). However, an appropriate value for K has been subject to debate in the literature (Smith *et al.*, 2009). Komar and Inman (1970), the first to introduce the CERC equation, use $K = 0.77$. Del Valle *et al*. (1993) also present field data from fluvial transport rates and present K as a function of median grain size (D_{50}) in mm:

$$K = 1.4e^{(-2.5\ D_{50})}. \tag{3.6}$$

For reference, (3.6) yields $K = 0.69$ for $D_{50} = 281$ microns (used at the Sand Engine). Rosati *et al.* (2002) show a linear regression of field data relating the immersed weight transport rate to the potential longshore sediment transport rate (Figure 3.2), which yields $K = 0.92$ when using root mean square (RMS) wave heights in (3.2). However, none of the aforementioned methods for K determination are applied in situations where a large protruding feature (such as a MN) is added to the coastline. It is possible that the K values presented in the literature are inadequate in evaluating the evolution of such a large feature. As such, the determination of an appropriate K value in one-line modeling

of MNs is one of the goals of this study. This determination is based on the ability of one-line models to reproduce observed morphology at the Sand Engine.

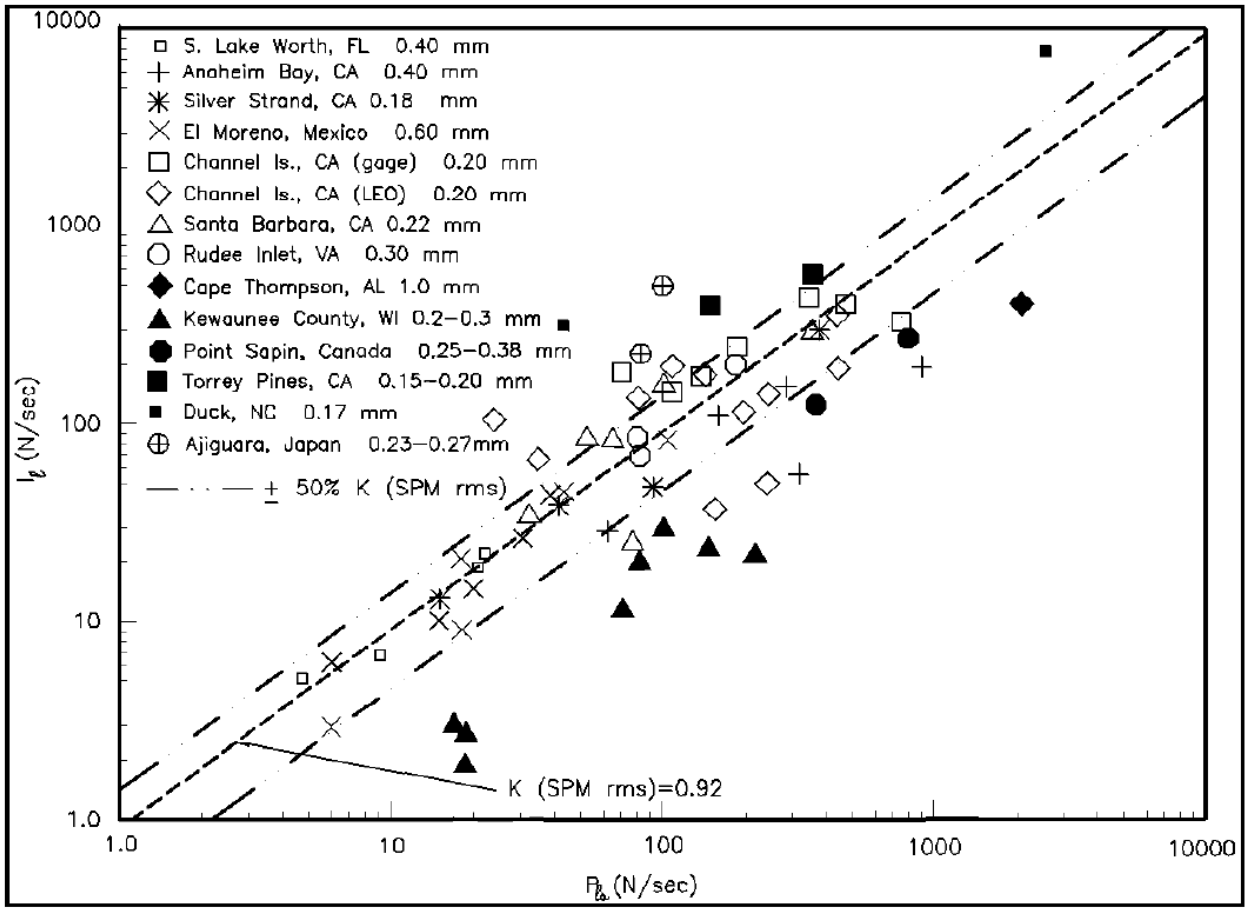

Figure 3.2. Field data from various study sites relating immersed sediment weight transport rate (I_l) with the longshore component of wave energy flux (P_l). A linear regression of this data is used to yield the *K* value of 0.92, one of the most commonly used values. The figure is from Rosati *et al.* (2002).

Both analytical and numerical solutions to the one-line model exist. Analytical solutions incorporate the assumption of uniform and constant wave characteristics, smooth plan beach profile, and small angles of wave propagation. Based on these premises, the combination of (3.1) and (3.5) yields (Pelnard-Considère, 1956):

$$\frac{\partial y}{\partial t} = \varepsilon \frac{\partial^2 y}{\partial x^2}. \tag{3.7}$$

This is a general diffusion equation describing the shoreline evolution in time where ε is the diffusion coefficient (or alongshore diffusivity) given by

$$\varepsilon = -\frac{1}{(D_C + D_B)} \frac{\partial Q_s}{\partial \theta} = \frac{2Q_o}{D_C + D_B}. \tag{3.8}$$

Q_o is the amplitude of the longshore transport rate (*e.g.*, Larson *et al.*, 1997):

$$Q_o = \left(\frac{\rho K \sqrt{g/\gamma_b}}{16(\rho_s - \rho)(1-n)} \right) H_b{}^{\frac{5}{2}}. \tag{3.9}$$

Computational versions of the one-line model are not subject to the aforementioned restrictions. These solve the system of (3.1), (3.2), and (3.3), stepping forward in time (*e.g.,* Zacharioudaki and Reeve, 2008). They can also incorporate time-varying and potentially spatially varying wave data, high values of wave angle propagation with respect to the local shoreline orientation, and complicated initial (*e.g.,* discontinuous or non-smooth coastline) and boundary (periodic, moving, pinned, *etc.*) conditions.

3.5. One-line Modeling with Groins

Since the applications of one-line models include modeling the evolution of potential beach engineering projects, much work has been done to include beach reinforcement structures such as groins into these models. Analytical solutions to the one-line model that included groins were among the first to be published (*e.g.,* LeMéhauté and Brebner, 1960; Bakker, 1969; and Bodge and Kraus, 1991) included

LST inhibition and bypassing. Larson *et al.* (1997) included solutions for time varying wave conditions at a single groin as well as incorporating wave diffraction. More modern semi-analytical one-line models, requiring the use of a computer to solve, often employ integral or Laplace transform techniques for coastline response near a groin (*e.g.*, Reeve, 2006; Valsamidis and Reeve, 2017). The most recent work in this area (Valsamidis and Reeve, 2020) couples various analytical techniques to allow for specific boundary conditions that mimic LST inhibition, bypassing, and groin permeability.

Numerical one-line models such as GENESIS (Hanson and Kraus, 1989) and GenCade (Frey *et al.*, 2012) simulate groins by mathematically incorporating the physical processes that accompany groins. These processes include LST inhibition (along with the accompanying groin permeability), groin bypassing, and wave diffraction. The framework of these numerical models also allows for complex groin field configurations that are difficult to model analytically. However, some numerical one-line models, such as CEM (Ashton and Murray, 2006a), were not originally designed with groin simulation mechanics incorporated. For CEM to be used to simulate coastline interactions with groin fields, the physical processes governing the influence of groins on shoreline evolution must be integrated into the model framework. One of the goals of this research is to include groin simulation mechanics into CEM with the intent of using that model to evaluate MN coastline interactions with groin fields (see Methods on how this is accomplished).

3.6. Mega-nourishment

Given the before-mentioned issues arising from frequent repetitive nourishment projects, innovative solutions beyond traditional soft coastal defenses are needed. The concept of mega-nourishment (MN) involves nourishing with the intent of feeding adjacent beaches (Stive *et al.*, 2013). This concept is similar to that of a feeder beach but on a much larger scale. While traditional nourishments place sediment volumes on the order of up to 5-10 million m^3 over a multi-kilometer stretch of coast (often in multiple phases), MNs involve the placement of an extremely large volume of sediment (on the order of 17-21 million m^3) at a single nourishment site (Pilkey and Clayton, 1989; Davison *et al.*, 1992). MN sediment is redistributed to adjacent beaches via natural processes such as wind, waves, and tides, which greatly increases the longshore impact of a nourishment to many kilometers of coastline (de Schipper *et al.*, 2016). This could potentially be more cost-effective than recurring smaller nourishments. The peninsula constructed in a MN is intended to have ecological and recreational benefits as well by greatly increasing beach area with a large cross-shore extent (Mulder and Tonnon, 2011). However, it should be noted that the sudden addition of such a large sediment mass on the coastline (thus creating an instability) raises issues related to the local ecosystem, recreational and navigational safety, ground water, and potential changes to nearshore processes (Stive *et al.*, 2013).

A pilot MN, known as the Delftland Sand Engine (SM), or *Zandmotor* in Dutch, was constructed in 2011 on the Dutch coast to examine the feasibility of MNs feeding larger stretches of the coast. The SM was built on the coast of Holland between

Scheveningen and Rotterdam. This region is characterized as a sandy, inlet-free, microtidal, wave-dominated coast. The SM was constructed as a hook-like peninsula containing 17 million m^3 of the total 21.5 million m^3 of sediment in the project (Figure 3.3). Upon completion in August 2011, the peninsula extended ~1 km offshore with a longshore footprint of ~2.3 km (de Schipper *et al.*, 2014). Following its construction, topographic surveys were taken nearly monthly from Aug. 2011 to Sept. 2016, described in de Schipper *et al*. (2016). The SM exhibited an initial rapid redistribution of sediment in the first 1.5 years, changing the coastline shape from a hook to a Gaussian form (de Schipper *et al.*, 2014). Erosion on the peninsula tip during this time was considerable (1.6 million m^3 loss) with a 15% reduction in cross-shore extent. Concurrently, shores adjacent to the SM accreted roughly 1.1 million m^3 increasing the peninsula's longshore footprint 60%. These effects are indicative of diffusion of the nourishment due to longshore sediment transport (LST), which occurred almost symmetrically in the first 1.5 years of evolution. There were seasonal differences in the geomorphic behavior, but considering volumetric changes during stormy months are comparable to those calculated for an entire year, wave action due to adverse meteorological forcing is likely a significant driver in the SM geomorphology.

However, the SM has not evolved entirely as expected raising issues related to the local ecosystem, recreational and navigational safety, ground water, and potential changes to nearshore processes (Stive *et al.*, 2013). This necessitates additional modeling efforts to improve the predictive capabilities of potential MN projects. Furthermore, the influence a groin field has on MN evolution is also not understood. As

many highly eroding coasts where a MN would be beneficial are reinforced with groin fields, further numerical modeling of MN-groin field interactions is necessary to fill this knowledge gap.

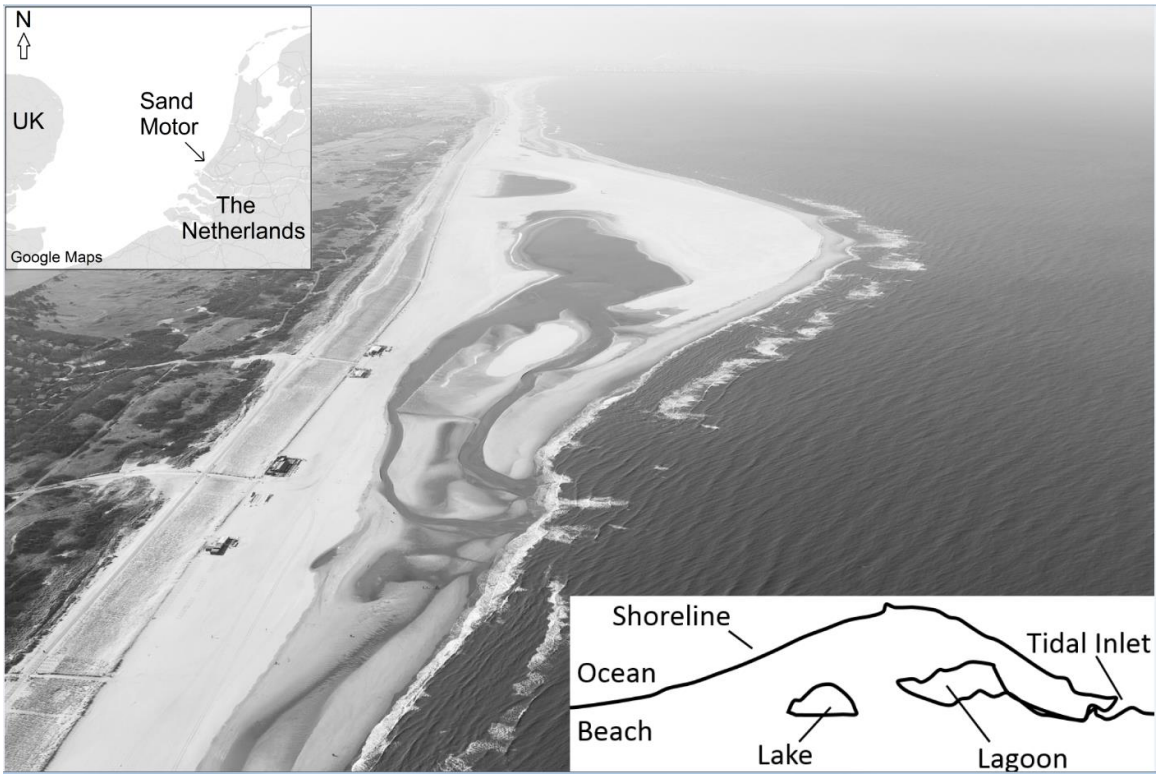

Figure 3.3. Aerial photo of the Sand Engine dated March 28, 2013. The coastline profile seen here is very similar to the one used as the initial coastline in the first project (March 1, 2013). The tidal inlet, lagoon, and lake are neglected in model simulations. Photo courtesy of the Dutch Ministry of Infrastructure and the Environment. The top left map insert is courtesy of Google Maps.

Some modeling work applied to MNs have been through process-based numerical models such as Q2D-morfo and Delft3D. Q2D-morfo is a quasi 2D nonlinear morphodynamic model that simulates wave-induced longshore sediment transport and incorporates cross-shore transport heuristically (van den Berg *et al.*, 2012). Q2D-morfo

has been used to simulate potential behavior of the SM and calculate its rate of coastline diffusion (Arriaga *et al*., 2017). Delft3D is a 3D morphology model capable of simulating hydrodynamics, sediment transport, and the resultant morphology from tidal, wind, and wave-driven currents (Lesser *et al.*, 2004). Delft3D was used in assessing pre-construction design options for the SM (Mulder and Tonnon, 2011) as well as in predicting its morphodynamics after a one-year hindcast simulation (Luijendijk *et al.*, 2017). Delft3D has been shown to work well in theoretical, laboratory, and real-life situations. However, it requires considerable computational and data resources to operate. The Deltares community reports that Delft3D typically runs about 1 month (simulation time) per day (real time) using a 1000-km grid on a high-end PC, and it would thus take a several months to run a multi-decadal simulation assuming time compression utilities are not used.

One-line models are often used as an alternative to process-based numerical models where computational resources are a limiting factor or when relative simplicity in data input/output (compared to complex 3D models) is desired. Valsamidis *et al.* (2017) present analytical and semi-analytical one-line models for rectangular and Gaussian-shaped MNs respectively that incorporate wave forcings and longshore advection parameters. However, these models cannot incorporate time-varying input data, cannot account for longshore variations in wave breaking, and must be validated using field data (difficult under current circumstances). As such, additional one-line numerical modeling methodologies for MNs are necessary if MN technology is to be applied on a larger scale.

4. ONE-LINE MODELING OF MEGA-NOURISHMENT EVOLUTION[1]

4.1. Introduction

Considering that 24% of the world's sandy coasts are eroding (Luijendijk *et al.*, 2018) with roughly 40% of the world's population living within in coastal areas (Mentaschi *et al.*, 2018), it is clear that coastal erosion poses a large societal threat. Continuing development of coastal regions combined with sea level rise increase the damage risk from shoreline retreat (Gornitz *et al.*, 1994). As a result, coastal protection schemes have become common in populated coastal regions. Hard coastal defenses, such as groins, jetties, breakwaters, dams, dykes, and seawalls, provide protection during high-energy events. However, these are often expensive and permanently alter the natural environment (*e.g.*, Brown and McLachlan, 2002; Griggs *et al.*, 1994; Williams *et al*., 2018). Soft defense schemes generally involve beach nourishment, which gives the appearance of a natural beach. However, nourished sediment is often transported out of the region requiring repetitive nourishments, which can also be costly (*e.g.*, Davison *et al.*, 1992; Peterson and Bishop, 2005).

The need for coastal defense calls for innovative shore protection concepts beyond traditional approaches. An innovative technical solution is mega-nourishment (MN), a relatively new concept that relies on natural forces to disperse a large sediment volume at a single nourishment site. An effective MN acts as a feeder beach, providing

[1] Source: Adapted from Whitley *et al.* (2021). Reproduced with permission from the Coastal Education and Research Foundation.

a sediment source for nearby shorelines. This approach could be applied to beaches around the world and be potentially more cost-effective than recurring smaller nourishments. A pilot MN, known as the Delftland Sand Motor (SM), or *Zandmotor* in Dutch, was constructed in 2011 on the Dutch coast to examine the feasibility of MNs feeding larger stretches of the coast. However, the SM has not evolved entirely as expected raising issues related to the local ecosystem, recreational and navigational safety, ground water, and potential changes to nearshore processes (Stive *et al.*, 2013). This has necessitated additional modeling efforts to improve the predictive capabilities of potential MN projects. Some MN modeling work has been performed using complex 3D models (*e.g.*, Luijendijk *et al*., 2017) such as Delft3D (Lesser *et al*., 2004). While 3D models can certainly provide detailed insights and comparison with complex processes, they may not automatically provide the more accurate results on specific elements, such as coastline evolution, than simple process-based models that run with much reduced overhead.

One-dimensional (one-line) coastline models are used to predict the change in horizontal position of a single coastal contour (*e.g.*, the shoreline) over time without considering the full complexity of the coastal system (Zacharioudaki and Reeve, 2008). One-line models do not have large data requirements, are remarkably robust, and are the preferred method of evaluating potential shoreline change for many engineering applications (Thomas and Frey, 2013). The ability of two different one-line models, GenCade and the Coastal Evolution Model (CEM), to predict the measured shoreline

evolution of the Dutch Sand Motor (SM), is assessed here as well as the validity of commonly used empirical coefficients employed in one-line model formulations.

4.1.1. Study Area

The SM was built on the coast of Holland between Scheveningen and Rotterdam. This region is characterized as a sandy, inlet-free, microtidal, wave-dominated coast. The SM was constructed as a hook-like peninsula containing 17 million m^3 of sediment (Figure 3.3). Upon completion in August 2011, the peninsula extended ~1 km offshore with a longshore extent of ~2.3 km (de Schipper *et al.*, 2014). Following its construction, topographic surveys were taken nearly monthly from Aug. 2011 to Sept. 2016 (de Zeeuw *et al.*, 2017), described in de Schipper *et al*. (2016). The SM exhibited an initial rapid redistribution of sediment in the first 1.5 years, changing the coastline shape from a hook to a Gaussian form (de Schipper *et al.*, 2014). Erosion on the peninsula tip during this time was considerable (1.6 million m^3 loss) with a 15% reduction in the feature's maximum cross-shore extent. Concurrently, shores adjacent to the SM accreted roughly 1.1 million m^3 increasing the peninsula's longshore reach by 60% of its original extent (de Schipper *et al.*, 2014). These effects are indicative of diffusion of the nourishment due to longshore sediment transport (LST), which occurred almost symmetrically in the first 1.5 years of evolution. There were seasonal differences in the geomorphic behavior with large sediment volume changes calculated on the order of 500 m^3 per meter alongshore in stormy months (Dec. 2013 – Feb. 2014) and little change (< 100 m^3/m) during calm ones (July 2013 – Aug. 2013) (de Vries *et al.*, 2014).

Considering volumetric changes during stormy months are comparable to those calculated for an entire year, wave action due to adverse meteorological forcing is likely a significant driver in the SM geomorphology.

4.2. Methods

4.2.1. One-Line Models

One-line model theory is based on the Pelnard-Considère (1956) equation (3.7), discussed in detail in Section 3.4. Many solutions to (3.7) are available from the literature on heat and solute diffusion (*e.g.*, Carslaw and Jaeger, 1959; Crank, 1975) and with appropriate specification of initial and boundary conditions may be transferred to the coastal domain. This is often done with the assumption that the wave conditions are constant in time and space so that the diffusion coefficient is a constant (*e.g.*, Grijm, 1961; LeMéhauté and Soldate, 1977; Walton and Chiu, 1979; Wind, 1990). In this form, the equation is most conveniently solved using Laplace transform techniques. A summary of the most common solutions and their applications can be found in Larson *et al.* (1987; 1997) and Komar (1983).

Bakker (1969) added to the one-line model theory by modeling beach movement near a groin and including a second line to account for the beach slope. This two-line modeling method was extended by Bakker *et al.* (1971) with a simplified representation of diffraction in the lee of the groin. The addition of more lines is an innovation that allows changes in beach slope to be modeled, but this means the strict assumption of profile equilibrium must be relaxed. The cross-shore exchange of sediment between

lines is usually formulated with a relaxation towards an equilibrium profile (Bakker *et al.*, 1971). Multiple line models may support unstable behavior under certain situations (Reeve and Valsamidis, 2014).

Numerical one-line models can account for time-varying input data, highly oblique wave angles, and complex boundary conditions. Commonly used numerical coastline models include GENESIS (Hanson and Kraus, 1989), GenCade (Frey *et al.*, 2012), UNIBEST (Deltares, 2011), LITPACK (DHI, 2009), and CEM (Ashton *et al.*, 2001). Some of these models account for longshore variations in wave conditions, wave refraction and diffraction, and the effects of man-made structures such as groins and seawalls. One-line models tend to produce more accurate results after calibration of inherent empirical coefficients with available field data. Usually, these field data originate from traditional beach nourishments of limited spatial extent. The performance of one-line models regarding MN evolution has not been tested extensively due to scarcity of measured data.

An analytical treatment of large-angle waves and spatial variation in wave conditions is not available. As such, computational methods are required. Furthermore, computational methods permit additional features such as nearshore wave transformation and interaction with structures to be included.

4.2.2. Coastline Evolution Model (CEM) Background

Based on one-line model theory, CEM has been used to explore large spatial (hundreds of km) and temporal (millennia) scales of coastline change (*e.g.*, Ells and

Murray, 2012; Slott *et al.*, 2010; Valvo *et al.*, 2006; Whitley, 2018; Williams *et al.*, 2013). It was originally utilized to show that highly oblique incident waves can give rise to anti-diffusional coastline instabilities (Falqués, 2003; Falqués and Calvete, 2005). By default, CEM represents an offshore wave climate stochastically rather than using a time series, allowing small potential changes in wave climate to be readily examined (*e.g.*, Moore *et al.*, 2013). CEM also has the advantage that it is a self-contained open-source model capable of running without overhead unlike other models that require additional software. For example, GenCade project work often requires the Surface-Water Modeling System (Aquaveo, 2017).

The version of CEM used in this chapter (coded in C as opposed to the MATLAB version used in Chapter 5) is a relatively efficient model taking only a couple of minutes per model year to simulate a 10 km-long coastline. The code for the C version can be found in Appendix A. CEM is designed as a theoretical model to explore overall patterns in large-scale coastline morphology, not to explore specific study sites. However, a major goal of this study is to test CEM capabilities when its domain and primary forcing operate at a higher resolution than usual to better capture smaller scale features.

4.2.3. CEM Framework

CEM is described in detail in Ashton and Murray (2006a; 2006b), though the major points are covered here. It is a cellular model that accounts for shoreline position (y) via the amount of sediment in each beach cell. The cellular domain numerically

represents a plan view of the local geography, each number signifying the composition of surface sediment with 1 indicating a subaerial cell, 0 indicating a subaqueous cell, and fractional quantities indicating a coastal cell. Unlike many other one-line models, (3.1) is not computed directly. Instead, shoreline adjustment is calculated via influxes (Q_{in}) and outfluxes (Q_{out}) of sediment in each beach cell. Positive sediment flux gradients give rise to accretion (coastline extension) and vice versa. The fractional value of sediment in each shoreline cell (F) is adjusted each time step via

$$\Delta F = \frac{(Q_{in} - Q_{out})}{(D_C + D_B)W^2}, \tag{4.1}$$

where W is the cell width (m). Here, a much higher spatial resolution is used (W = 25 m) than traditionally (W = 100 m or 1 km) to better capture smaller-scale features.

Sediment fluxes are calculated via (3.2) for all beach cells every time step. By default, CEM calculates Q_s using fixed values for K (0.69), ρ (1020 kg/m^3), ρ_s (2650 kg/m^3), γ_b (0.5), and n (0.4). Here, CEM is modified so that these values are user-specified based on study site characteristics (see model parameterization). CEM is only capable of using periodic boundary conditions (BCs) where sediment exiting one boundary enters the other. Domains are thus set to have boundaries over 5 km away from the nourishment bases to mitigate effects from BCs. No model coastline change occurs near the boundaries in the 3.5-year simulations of the SM.

As breaking wave data in the nearshore is scarce and quite complex, CEM is driven by offshore wave information numerically transformed into breaking wave height (H_b) and angle (ϕ_b). The wave transformation process (details in Ashton and Murray, 2006a and Whitley, 2014) handles wave refraction and shoaling over local bathymetry

assuming that bathymetric contours are straight and parallel to the coastline. Wave angle changes due to refraction are calculated using Snell's Law, re-written so that the wave angle relative to a fixed geographic axis (ϕ) is

$$\phi = \arcsin\left(\frac{C}{C_0}\sin\phi_0\right), \tag{4.2}$$

where C is the wave celerity, C_o is the incident (offshore) wave celerity, and ϕ_o is the incident (offshore) wave angle. Wave height changes due to shoaling are calculated by

$$H = H_0\sqrt{\frac{C_0}{2C_n}}\sqrt{\frac{\cos\phi_0}{\cos\phi}}, \tag{4.3}$$

where the term C_n is the group velocity and H_0 is the incident (offshore) wave height.

Equations (4.2) and (4.3) are solved iteratively over a user-specified refraction step size (traditionally 0.2 m), beginning with a pre-defined starting water depth (see below), until

$$H > \gamma_b h, \tag{4.4}$$

where H is the wave height, h is the water depth, and γ_b is the coefficient for breaking threshold at which time the wave will break. The wave angle at breaking ($\phi = \phi_b$) and the wave height at breaking ($H = H_b$) are then imported into the CERC equation (3.2) to calculate Q_s.

Since Q_s is dependent upon breaking wave information, model-produced coastline behavior is extremely sensitive to the offshore wave climate input. While many coastal models are forced with a time series of wave information, CEM uses a probability distribution function (PDF) to determine offshore wave characteristics for

any given time step. PDF theory is described in detail in Ashton and Murray (2006b). The default PDF of CEM determines the fraction of offshore waves approaching the coast from a high incident angle ($\phi_o > 45^o$ relative to the cross-shore [x-] axis) and the fraction of waves approaching from the left of the regional shoreline trend throughout the simulation. Traditionally, the PDF is controlled by two variables that control the fraction of high-angle incident waves (U, or highness) and the fraction of left-approaching waves (A, or asymmetry).

According to Ashton *et al.* (2001) and Ashton and Murray (2006a), high-angle incident waves give rise to an anti-diffusional instability in the model shoreline causing features such as sand waves, flying spits, and capes to accrete and increase in cross-shore extension. Consequently, low-angle incident waves cause such features to diffuse leading to a straighter coastline. Therefore, an offshore wave climate rich in low-angle waves ($U < 0.5$) should cause protruding features in the shoreline to diffuse while once rich in high-angle waves should cause the anti-diffusion of features in the model. Furthermore, migration of shoreline features alongshore is also dependent upon the symmetry of the wave climate. Waves approaching from the left side of the domain produce LST to the right of the domain resulting in feature migration to the right (and vice versa). Therefore, asymmetric wave climates ($A \neq 0.5$) yield the migration of any protruding shoreline features with the overall direction of migration dependent on whether the wave climate is richer with left-approaching waves ($A > 0.5$) or right-approaching waves ($A < 0.5$). On the other hand, symmetric wave climates ($A = 0.5$)

produce a nearly even number of left- versus right-approaching waves, canceling out any migratory behavior in the shoreline features.

PDFs for CEM can be used to simulate the offshore wave climate of a physical system as done by Ashton and Murray (2006a), Ashton and Murray (2006b), Ashton *et al.* (2001), Ells and Murray (2012), Moore *et al.* (2013), Slott *et al.* (2006), and Williams *et al.* (2013). PDFs are based on the offshore wave energy (E) contribution to sediment flux (and therefore shoreline diffusivity) and the angle of incidence relative to the shoreline (ϕ_o - θ). Assuming that waves are refracted and shoaled over straight and shore-parallel bathymetric contours, the CERC equation (3.2) can be transformed into a deep-water formula

$$Q_s = k_2 H_0^{\frac{12}{5}} T^{\frac{1}{5}} \sin(\phi_0 - \theta) \cos^{\frac{6}{5}}(\phi_0 - \theta), \tag{4.5}$$

where the proportionality constant k_2 = 0.32 $m^{3/5}$ $s^{-6/5}$. If the longshore diffusivity ε in the Pelnard-Considère (1956) equation (3.7) is given by

$$\varepsilon = -\frac{1}{(D_C + D_B)} \frac{\partial Q_s}{\partial \theta}, \tag{4.6}$$

inserting (4.5) into (4.6) yields

$$\varepsilon = \frac{k_2}{D_C + D_B} H_0^{\frac{12}{5}} T^{\frac{1}{5}} \left\{ \cos^{\frac{1}{5}}(\phi_0 - \theta) \left[\left(\frac{6}{5}\right) \sin^2(\phi_0 - \theta) - \cos^2(\phi_0 - \theta) \right] \right\}. \tag{4.7}$$

Since k_2, D_C, and D_B do not vary as wave characteristics change, wave contributions to ε in (4.7) can be split into two parts: one dependent on wave angle (ϕ_0 relative to θ) and one representing the contribution H_0 and T have on potential LST. The latter is given by

$$E = H_0^{\frac{12}{5}} T^{\frac{1}{5}}, \tag{4.8}$$

where E is conceptually similar to offshore "wave energy" or offshore wave height contribution to Q_s based on (4.5).

In order to create a PDF that represents the natural wave climate off the coast of a specific study site, wave energy approaching the shore from each direction offshore of the site is normalized according to equation (3.6) such that the values of T and H_o are fixed to average values representative of the local wave climate for the duration of the simulation (Slott *et al.*, 2006) and wave energy approaching the coast is distributed by probabilities for angles of incidence. By default, CEM uses a 4-bin PDF with each bin having a 45-degree width spanning a spectrum of 180 degrees. Offshore wave energy generated from tropical cyclones and other high-energy events are captured in the offshore wave data and are thus incorporated into the PDF, though nearshore waves generated during these events are not.

For a wave data set including sampled significant H_0, average T, and ϕ_0, E is calculated for every ϕ_0 data point in the time series according to (4.8). Each data point is then binned into a respective ϕ_0-range bin, and E is summed for every wave bin. The cumulative E (E_{tot}) is then found for the entire data set, representing the total amount of deepwater "wave energy" over the data's time period. The summed E in every wave bin is then divided by E_{tot} to yield a normalized E (E_{nor}) for every wave bin. E_{nor} effectively represents the fraction of cumulative offshore wave energy approaching the shore from each direction. If H_0 and T are held to fixed (effective average) values in a simulation, E_{nor} can also be used to represent the probability of a wave originating from a ϕ_0-range. This is used in the models' wave PDFs to determine the probability of ϕ_0 each time step.

Traditionally, CEM has been forced with a 4-bin PDF with each bin having a ϕ_o-range of 45^o covering all possible 180^o of offshore incident wave direction. To simulate actual directional wave spectra more accurately, the PDF bin sizes are reduced to 15^o creating 12-bin PDF's that cover the same ϕ_0 range. This allows for a higher resolution of the incident wave direction, which may be significant considering model sensitivity to offshore wave forcing.

This study incorporates several new approaches to CEM (C version; see Appendix A). First, CEM is being used at a much higher resolution than what has traditionally been done. Previous studies have used CEM at 100 m (Ashton and Murray, 2006a; Ashton and Murray, 2006b; Ashton *et al.*, 2001; Whitley, 2014) or 1 km (Slott *et al.*, 2010; Williams *et al.*, 2013) cellular resolution. Here, a 25 m cell width model is presented to better capture smaller scale features (such as a mega-nourishment) that larger cell sizes cannot. This particular cell width is chosen as a compromise between a spatial higher resolution and computational time (which greatly increases with spatial resolution).

To verify that running at higher resolution does not affect overall model results, two different CEM simulations are run using an initial profile similar to the Sand Engine (dated March 1, 2013) with the exact same topographic parameters and forcing conditions. One has a 25-m cell size and the other has a 100-m cell size. Both models are run for ~3.5 years of simulation time. Both simulations yielded coastlines nearly indistinguishable in shape. The feature shape and position (due to rightward migration) between the two coastlines are almost identical. There is a 3% difference in the aspect

ratios of the feature with the 100-m resolution model producing a feature extending 614 m offshore and the 25-m resolution model producing a feature extending 634 m. It is concluded that changing the model resolution does not significantly affect model results.

4.2.4. GENSIS + CasCade (GenCade)

GenCade, described in detail in Frey *et al.* (2012), is a one-line numerical model developed by the U.S. Army Corps of Engineers (USACE) that combines the regional-scale calculations of the Cascade model (Larson *et al.*, 2003) with the project-scale calculations of GENESIS. GenCade is thus capable of simulating coastal change on a variety of temporal and spatial scales. GenCade is designed specifically for planning and engineering design studies, and its interface is incorporated into the Surface-water Modeling System (SMS). GenCade directly calculates shoreline change via (4.1) except that $\frac{dQ_s}{dx}$ can include a source/sink term (though none are used here). Therefore, the aforementioned assumptions of numerical one-line models apply to GenCade.

LST rates are calculated via a modified version of the CERC equation (4.2),

$$Q_s = {H_b}^2 C_{g,b} \left(a_1 \sin 2\alpha_b - a_2 \cos \alpha_b \frac{\partial H_b}{\partial x} \right), \tag{4.9}$$

where $C_{g,b}$ is the breaking wave group celerity. The non-dimensional parameters a_1 and a_2 are given by

$$a_1 = \frac{K}{16(\rho_s - \rho)(1-n)1.416^{5/2}}, \tag{4.10}$$

$$a_2 = \frac{K_2}{8(\rho_s - \rho)(1-n)\tan\beta 1.416^{5/2}}, \tag{4.11}$$

where β is the average bottom slope of the shoreline out to D_c and K_2 is a proportionality coefficient. Unlike CEM, GenCade uses fixed values for n (0.4), ρ_s (2650 kg/m^3), and ρ (1000 kg/m^3) with ρ being the only value differing between CEM and GenCade (see Section 4.2.5 for model parameterization). The second term in (4.9) accounts for LST produced by diffraction of coastal structures such as breakwaters. As such, variations in K_2 only affect coastline evolution in areas influenced by wave diffraction near structures. As no structures are used here and a direct comparison with CEM is desired, K_2 is assigned a value of zero.

GenCade is driven by offshore wave time series. Wave data includes significant H_0, average T, and ϕ_0 (relative to shore normal) over a sampling interval. Wave data can be physical, hindcast, or numerically modeled. In order to perform a direct comparison with CEM, modeled wave data from a PDF are used here for GenCade so that both models effectively have the same wave climate. H_0 and T are set to fixed values for every data point in the time series (as in CEM) while ϕ_0 is determined by the same PDF as its CEM counterpart. This not only allows for a fair comparison of GenCade with CEM, but it also allows for temporal scales up to centuries of MN evolution (with future prediction if desired).

While GenCade can be coupled to an external wave model, it also has an internal wave model capable of transforming deep water wave information to breaking. This accounts for shoaling and refraction over bathymetric contours and has several options available. An offshore contour can be specified by the user, though none are used here as CEM does not have that capability. If an offshore contour is not used, GenCade

assumes that bathymetric contours are straight and parallel with the coastline, as done in CEM. As this can create numerical instabilities in regions where the coastline changes abruptly (such as sawtooth formations near groins), GenCade has an algorithm that can smooth the offshore contours. As CEM's wave algorithms assume that offshore contours are perfectly straight and parallel with the coastline, the GenCade smoothing algorithm is not used here. With this setup, both models end up using nearly identical wave transformation techniques. The only major difference is that CEM allows for a user-specified breaker index (γ_b) while GenCade uses a fixed value of $\gamma_b = 0.78$.

GenCade uses a right-handed coordinate system where the coastline is represented by a vector containing *y* values for every alongshore position at a specified resolution. Though GenCade is capable of using variable grid spacing, a fixed 25-m-resolution is used here. Multiple BC options are available, though fixed (or "pinned") BCs are used here where the cross-shore shoreline position at the boundaries do not change. As with CEM, boundaries are a large distance (see above) from the MN bases to mitigate BC effects.

While CEM takes 1.6 min per model year to run a 10-km-long coastline using a 0.2-day time step, GenCade takes roughly 10 seconds per model year with a 1-hour time step to run the same coastline on the same machine. To compare the results of CEM with GenCade, the same parameters (when possible), forcing conditions, and initial conditions are used for both models (see Table 4.1 in Section 4.2.5).

4.2.5. Model Parameterization

In order to evaluate LST equations (4.2) and (4.9), values from Table 4.1 are used. While all of the below-mentioned values can be user-specified in the version of CEM used here, ρ, ρ_s, n, and γ_b cannot be user-specified in GenCade. To allow for a fair comparison between CEM and GenCade, the same variables are used in both models. ρ = 1000 kg/m^3 is used for the approximate density of surface water (GenCade default). ρ_s = 2,650 kg/m^3 to represent the density of quartz-type sand. A porosity value of n = 0.4 is applied as Rosati *et al.* (2002) recommend this value for most situations. The default value of breaker index (γ_b) used in GenCade (γ_b= 0.78) is also applied to CEM. Topographic parameters including D_C, D_B, the shoreface slope (S_{sf}), and the slope of the shelf (S_s) are determined from topographic data described in the introduction. Of these, only D_C and D_B can be user-specified in GenCade as it assumes that S_{sf} and S_s follow an equilibrium profile.

As an appropriate value for K has been subject to debate (Smith *et al.*, 2009), several approaches are evaluated in the determination of K, as described in Mil-Homens *et al.* (2013). Historically, Komar and Inman (1970) use K = 0.77 while Rosati *et al.* (2002) recommend K = 0.96 when using root mean square (RMS) wave heights. Del Valle *et al.* (1993) also present K as a function of median grain size (D_{50}) in mm:

$$K = 1.4e^{(-2.5\, D_{50})}, \tag{4.12}$$

which yields K= 0.69 for D_{50} = 281 microns. However, none of the above methods of K determination are applied to a coastal situation where an extremely large perturbation in the coastline is suddenly constructed (such as a MN). As such, enlarging K may help account for shoreline adjustments that result from such a large shoreline instability.

Following a sensitivity analysis, it is found that enlarged *K* values ranging from 0.96 to 2.00 produce favorable results. Model results using this range of *K* values are therefore presented here and compared with the model results produced using the commonly used *K* values (*i.e.*, 0.69, 0.77, and 0.92). All of the above methods are implemented into CEM and GenCade SM simulations to ascertain which models produce coastline morphology that closely matches that observed.

Table 4.1. Table of parameters used in model simulations for the Sand Motor (SM). The same parameters are used for both CEM and GenCade. Note that GenCade does not use the parameters S_{sf} or S_s.

Parameter	Value	Description
	0.69	Empirical coefficient (del Valle *et al.*, 1993)
	0.77	Empirical coefficient (Komar and Inman, 1970)
K	0.92	Empirical coefficient (Rosati *et al.*, 2002)
	0.96 to 1.80	Empirical coefficient (Range of arbitrary high values)
ρ_s (kg/m^3)	2650	Sediment density
ρ (kg/m^3)	1000	Seawater density
n	0.4	Sediment porosity
γ_b	0.78	Breaker index
D_C (m)	8.22	Depth of closure
D_B (m)	1.90	Berm elevation
S_{sf}	0.0177	Slope of the shoreface
S_s	0.0052	Slope of the shelf
T (s)	5.0	Average offshore wave period
$H_{0,rms}$ (m)	0.85	Average offshore root mean square wave height
z_{max} (m)	15.0	Water depth at wave station
D_{50} (μm)	281	Median sediment grain size
Δt (hours)	1.0	Time step size

Model time step sizes are chosen based on model stability. It is found that a time step of 1 hour works well with all GenCade simulations. This time step is also used for CEM so that a fair comparison of the models' performance can be made.

The PDF used to simulate the wave climate at the SM is created from Meetpost Noordwijk (MPN) wave station data (1985 to 1991) provided by Wijnberg (1995). MPN is located ~5 km offshore from the Holland coast where the water depth (z_{max}) is 18 m (Wijnberg, 2002). MPN is chosen over other wave stations in the area because its proximity to the coast minimizes the uncertainty involved in wave refraction assumptions. The station's annual mean wave period is 5.0 s, which is used for T in (4.4), while the station's annual mean significant H_0 ($H_{0,s}$) is 1.2 m. It should be noted that while station wave data contains $H_{0,S}$, RMS wave height ($H_{0,rms}$) is required for the K values used. $H_{0,s}$ is converted into $H_{0,rms}$ via

$$H_{0,rms} = \frac{\sqrt{2}}{2} H_{0,s}, \tag{4.13}$$

such that the annual mean $H_{0,rms}$ = 0.85 m, which is used for H_0 in (4.12). The SM PDF (Figure 3.1) indicates a 74% probability of $\phi_0 < 0^o$ and a 42% probability of a high-angle ($> |45^o|$ relative to x-normal) incident wave. The wave climate at the SM is highly oblique due to energetic winter storms that mainly approach from the south-west (highest probability, -75 to -60 degrees relative to shore normal) and the north (lower probability, ~30 to 45 degrees relative to shore normal) (de Schipper *et al.*, 2016).

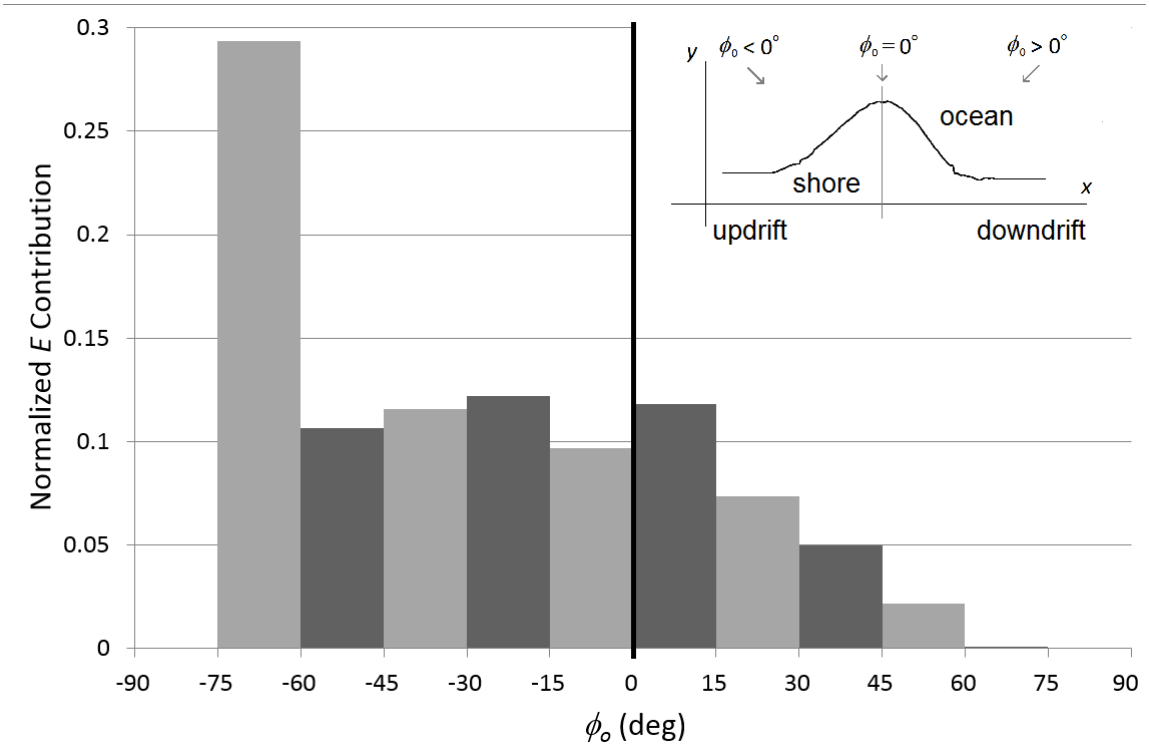

Figure 4.1. **Probability distribution function (PDF) representing the offshore wave climate for the Sand Motor (SM). Wave data is from the Meetpost Noordwijk (MPN) wave station. The ordinate indicates the normalized fractional wave energy contribution to shoreline diffusivity (*E*), and the abscissa indicates the offshore directional bin (ϕ_0-range) from which the wave energy originates. The top right insert shows the orientation of ϕ_0 with respect to the shore normal.**

A shoreline similar to the one surveyed at the SM on March 1, 2013 is used for the initial model coastline in SM simulations. The small tidal channel on the downdrift side of the feature is neglected, thus offering a single continuous convex coastline (desirable to avoid unnecessary model complexities). Model results after 3.5 years of simulation are compared with the measured coastline dated 7 September 2016 (~3.5 years after the initial coastline date). For the domain extremities where survey data is not available (beyond *x* = -2.150 km and +2.300 km), Google Earth imagery is used to estimate the coastline out to the point where the coastline is nearly straight for both the

measured initial and final coastlines. It is assumed that a perfectly straight coastline exists beyond to the boundaries.

4.3. Results

Before CEM and GenCade simulations are presented for the Sand Motor using several different *K* values each. CEM and GenCade model results for simulations of the SM both show diffusion of the nourishment feature (Figure 4.2). Coastline evolution during the 3.5 years simulated is confined between $x = \pm 4.000$ km with no change in coastline beyond. The amount of tip retreat (Δy_{max}), or change in maximum cross-shore extent, is used as a measure of diffusion rate. An increase in *K* value leads to a greater Δy_{max} and thus a greater diffusion rate (Table 4.2). A sensitivity analysis on possible K values was conducted and results based on overall shoreline mean RMSD and cross-shore tip retreat Δy_{max} are shown in Figure 4.3. Of the commonly used *K* values (0.69, 0.77, and 0.92), 0.92 produces the closest Δy_{max} to that observed in both models (13 m away for CEM and 74 m away for GenCade). However, increasing *K* beyond 0.92 can produce tip retreats within 2 m of that observed. Using $K = 1.10$ in CEM produces a Δy_{max} less than 5 m away from that observed, and using $K = 2.00$ in GenCade produces a Δy_{max} less than 2 m away from that observed (Figure 4.2). It should be noted that using $K > 1.80$ causes numerical instability in CEM due to the larger amounts of Q_s. This can be alleviated using smaller time steps and a larger spatial resolution but was not deemed necessary for this study.

The root mean square difference (RMSD) between measured and modeled cross-shore coastline position is calculated for every alongshore position in the surveyed domain (x = -2.150 km to +2.300 km). The mean RMSD is used to show how each model reproduces the measured coastline in this region. Overall, GenCade models produced lower mean RMSDs than CEM, regardless of K (Figure 4.3). Of the commonly used K values, 0.92 produces the lowest mean RMSD in both models. As with Δy_{max}, increasing K above the commonly used values can produce a lower mean RMSD in both models. Minimum mean RMSDs can be produced using K = 1.10 with CEM and K = 1.60 with GenCade (Figure 4.3).

An analysis of the survey initial and final coastlines indicates a downdrift shift in the alongshore position of the feature tip (Δx_{tip}) on the order of 400 m. While tip migration analyses of the modeled coastlines are limited to the 25 m resolution alongshore, CEM simulations show positive (downdrift) Δx_{tip} values of similar order to that observed. CEM Δx_{tip} values range from 325 to 425 m, typically with increasing K values producing greater migration magnitudes. Of all the K values simulated in CEM, K = 1.20, 1.30, and 1.40 produce the closest Δx_{tip} (400 m downdrift) to that observed with K = 0.92 producing a Δx_{tip} within 25 m of that observed. GenCade simulations using K < 1.30 show very little migration, typically on the order of 0 to 25 m downdrift. Using K > 1.30 in GenCade produces an updrift (negative) Δx_{tip} with Δx_{tip} magnitude increasing with K.

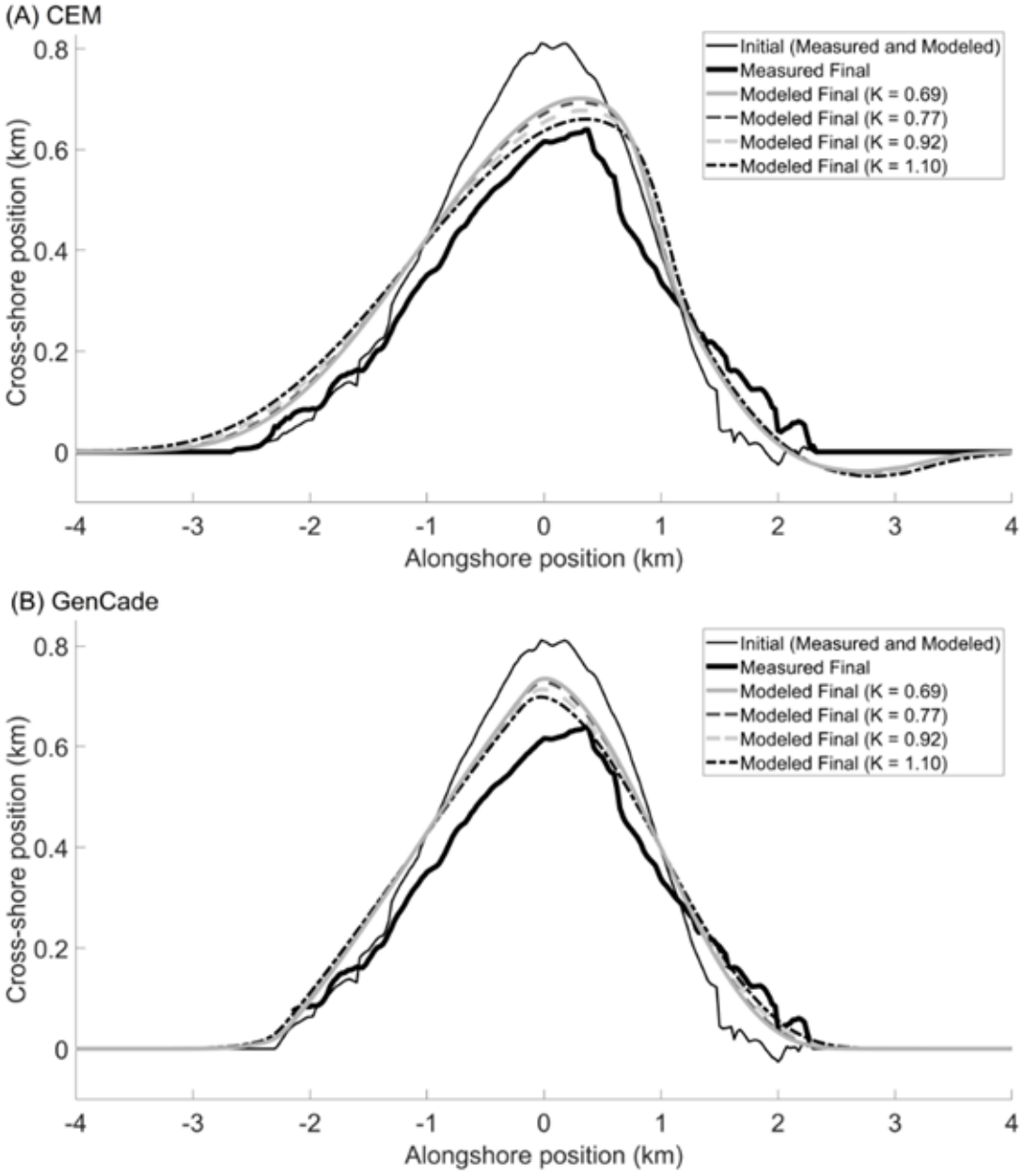

Figure 4.2. Initial and final coastlines both measured and modeled for the Sand Motor (SM) for 3.5 years of evolution. The initial coastline is the same for both measured and modeled. (A) shows model results from CEM while (B) shows model results from GenCade. Various *K* values from (2) are used in the models with the most commonly used values (0.69, 0.77, and 0.92) and a representative higher value (1.10) shown here. Note that cross-shore scale is stretched to aid in visualization of modeled shoreline differences.

Table 4.2. Metrics used in analysis of measured and modeled shorelines at the Sand Motor.

					Mean Flank Slopes	
Model	***K***	**Mean RMSD (m)**	**Δy_{max} (m)**	**Δx_{tip} (m)**	**Left (deg)**	**Right (deg)**
CEM	0.69	59.9	134	325	9.80	-17.9
CEM	0.77	58.5	143	350	9.62	-17.9
CEM	0.92	55.7	158	375	9.34	-17.2
CEM	1.10	52.6	176	375	9.09	-16.6
CEM	1.20	52.3	185	400	8.91	-16.4
CEM	1.30	52.8	193	400	8.79	-16.3
GenCade	0.69	63.1	76	25	13.5	-13.0
GenCade	0.77	60.7	84	25	12.9	-12.6
GenCade	0.92	56.6	98	25	12.1	-12.1
GenCade	1.10	52.5	112	0	10.6	-11.0
GenCade	1.60	47.6	147	-50	10.2	10.2
GenCade	2.00	49.0	170	-75	9.9	-9.9
Survey			171	400	14.6	-16.0

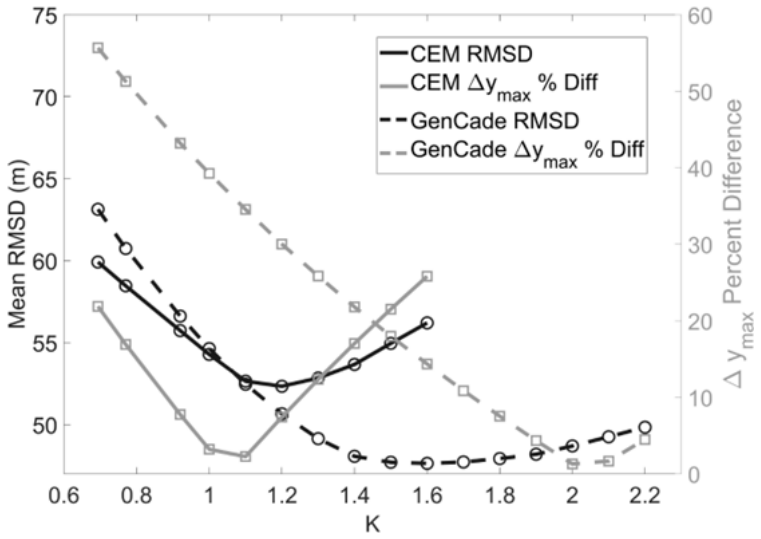

Figure 4.3. Sensitivity analysis showing the influence of *K* on mean RMSD between modeled and measured shoreline position after 3.5 years (left axis) and the percent difference between modeled and measured tip retreat (Δy_{max} percent difference; right axis). The minimization of the shown metrics indicates model performance closest to that observed at the Sand Motor for that metric.

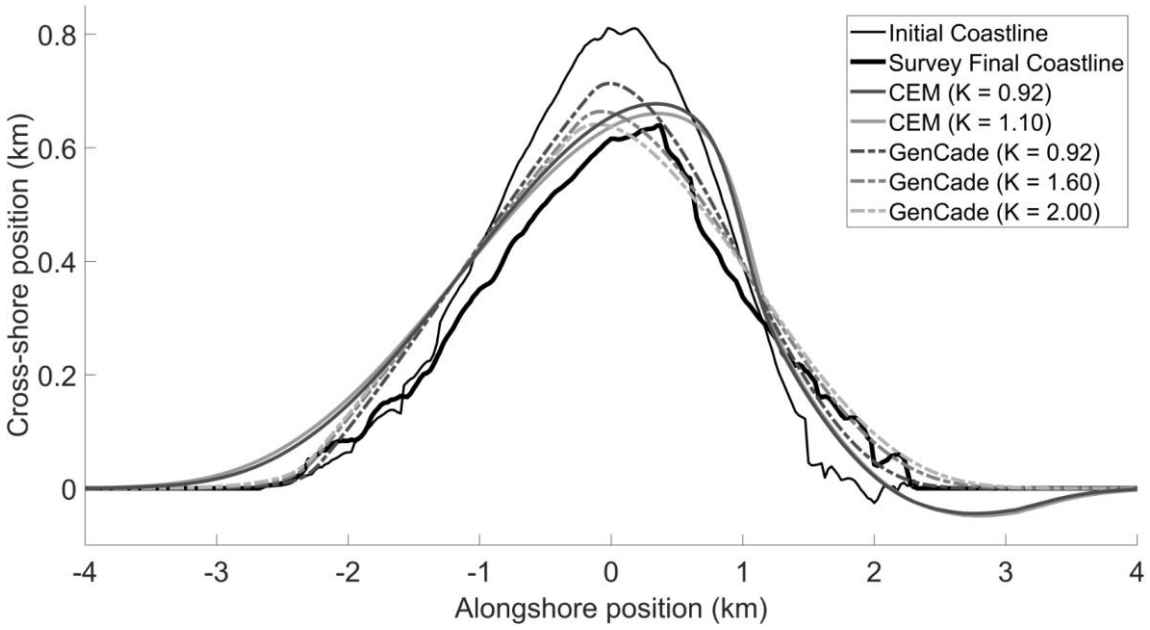

Figure 4.4. Measured and simulated shoreline positions (initial to final after 3.5 years) for CEM and GenCade. The model results with the best-performing metrics are shown. Note that cross-shore scale is stretched to aid in visualization of modeled shoreline differences.

The slope of the peninsula flanks, measured in degrees relative to the *x*-axis, is used as an indicator of feature symmetry. A mean updrift flank angle for each coastline is averaged alongshore from the minimum cross-shore extent (updrift of the peninsula) to the tip and vice versa for the downdrift flank. The initial coastline shows some asymmetry having a mean updrift flank angle of 19.9^{o} while the mean downdrift angle is -23.7^{o}. Survey results show that as diffusion occurs, the mean flank slope angles decrease to 14.6^{o} and -16.0^{o} respectively, and the formation remains asymmetric with a steeper downdrift flank slope. Similarly, both models show a decrease in mean flank slopes on both sides as the feature diffuses. However, CEM retains formation asymmetry in the final coastline with a consistently steeper downdrift flank slope in all

simulations. GenCade, on the other hand, diffuses the formation more or less symmetrically with a 0.0- to 0.5-degree difference between mean updrift and downdrift flank slopes (Figure 4.4).

4.4. Discussion

Of the commonly used *K* values presented in the literature, models using *K* = 0.92 produced morphology that most closely matched that observed. However, that *K* determination is based on field data from a variety of nourishment projects of much smaller scale than the SM (see Rosati *et al.*, 2002 for details). MNs have different coastal dynamics than traditional ones since a large intrusive feature is introduced into the system, which can cause coastline instability. For the SM, it is shown here that increasing *K* to 1.10 or higher significantly improved both models' ability to reproduce observed tip retreats and produce lower RMSD values. Therefore, using a higher-than-usual *K* value may be a better fit when one-line models are used in applications with such a large-scale coastline instability. However, as there is no single *K* value that maximizes model performance across all metrics, an appropriate value for *K* in a one-line model simulation of a MN should be a compromise between best values for different metrics. The variability of optimal *K* value as a function of evolution time has been investigated, but no clear trend based on the given dataset has been found. This may partly be due to the fact that forcing conditions change for different time segments.

A notable distinction between CEM and GenCade behavior is that while GenCade diffuses the SM near-symmetrically with little alongshore migration, CEM

shows an asymmetric feature with downdrift migration of its tip. Though not as drastic as the model, downdrift migration of the tip and a gain in feature asymmetry is also observed at the SM between March 2013 and September 2016. Longshore migration of the feature tip is expected in the physical SM and in CEM considering the high probability incident waves approaching from updrift (Ashton and Murray, 2006a; López-Ruiz *et al.*, 2014). While symmetric diffusion with a wave climate rich in relatively low incident wave angles ($\phi_0 < 45^o$) is expected for GenCade using its internal wave model, CEM has a wave shadowing algorithm where large coastline formations (such as MNs) effectively block incoming waves beyond a certain ϕ_0 along a shadow zone (see Ashton and Murray, 2006a for details). Wave shadowing negates wave energy (and thus LST) in the shadowed region, and sediment transported into a shadow zone is not transported out. In an asymmetric wave climate with predominantly downdrift propagating waves (such as at the SM), wave shadows downdrift of a large formation become common in the simulation. This has the effect of migrating large features towards the shadow zone (quantified by Δx_{tip}) and can cause erosion just downdrift of the shadow from sediment starvation. The effects of this erosion can be seen in all CEM final coastlines between x = 2 and 4 km (Figure 4.3), though such effects are not observed at the SM. Erosion here is absent in GenCade simulations as they do not have wave shadowing. Wave shadowing also has the effect of producing features with steeper downdrift flank slopes, as is seen in the CEM SM simulations, while GenCade simulations produce near symmetric features. Despite the advantages wave shadowing offers, it neglects wind waves that may be generated nearshore, which are likely to occur at the SM during

stormy seasons. This could further explain discrepancies between measured-CEM modeled coastlines on the downdrift flank where shadow zones in CEM are common.

Of the measured-modeled differences, the region with highest difference is x = 0.5 to 1.2 km for CEM. This is possibly the result of neglecting the lagoon feature present on the downdrift flank of the physical SM. The entrance to the lagoon lies near y = 1.24 km initially in March 2013. SM survey data from March 2013 to July 2016 show that the lagoon loses roughly one third of its water surface area during this time, indicating infilling of the lagoon. Neglecting this sediment sink could attribute to CEM's over-prediction of accretion in this region.

Despite their ability to reproduce observed morphology with reasonable accuracy, CEM and GenCade do suffer from a number of limitations. Neither directly accounts for cross-shore processes such as Aeolian transport, which has been shown to transfer material at the SM from the low-lying beaches to the dunes (Hoonhout and de Vries, 2017; Roest, 2017), and those transporting material offshore during storms. It is possible that sediment lost offshore during these events recovers over long (5+ years) time scales (Morton *et al.*, 1994), but this limitation should not be discounted when interpreting the results.

Both models in their forms here assume that waves refract over straight and shore-parallel bathymetric contours and that these remain constant during shoreline adjustment. Wijnberg (2002) reports alongshore variations in the bathymetric slope in the SM vicinity along with subaqueous terraces that likely affect wave refraction. Given the models' sensitivity to wave forcing conditions, changes in breaking wave behavior

due to refraction and shoaling can significantly affect coastline morphology. Taking these limitations into account, shoreline extension estimations resulting from a modeled MN project on long time scales are likely to be overestimated. Furthermore, due to the stochastic nature of a PDF, neither model here captures temporally varying trends in ϕ_o that are present in the SM's seasonal bi-modal wave climate. While some discrepancies between the modeled-measured results over a 3.5-year time period may be attributed to this variability, it is assumed that these effects average out on long time scales. This issue can be alleviated in GenCade through the use of a wave time series, though this was not done here in order to perform a direct GenCade-CEM comparison.

Further research on this subject should incorporate the evaluation of additional one-line models such as LITPACK and UNIBEST in the application of MN simulation. Furthermore, CEM's effectiveness in modeling MNs may be improved by including additional physics such as offshore currents and cross-shore processes that may contribute to the dynamics in MN evolution. The addition of sources/sinks and sea level rise (*e.g.,* Ratliff *et al.,* 2018) could also improve CEM's accuracy in examining specific study sites. CEM computational time can be improved with a variable spatial resolution as done in GenCade or with options that do not force a wave transformation for every alongshore cell. CEM accuracy can likely be improved with a more robust wave transformation technique such as SWAN (*e.g.,* Limber *et al.,* 2016) or STWAVE (*i.e.,* Smith *et al.*, 2001).

For future applications GenCade with MNs, the use of a robust wave model such as CMS-Wave (Connell and Permenter, 2013) is recommended, which can potentially

account for wave shadowing. The inclusion of an offshore contour would make long-term simulations more accurate as GenCade tends to smooth coastlines toward a straight line if an offshore contour is not specified (as seen in Figure 4.2). Using the built-in source/sink features could also mitigate the effects of physical sediment sinks on shoreline evolution. The inclusion of cross-shore transport processes into GenCade simulations could also greatly improve its accuracy. GenCade is also capable of including tidal currents and regional morphological trends, which could improve model accuracy if data for those is available.

Theoretical studies in CEM are fairly straightforward to set up. However, adapting CEM for a specific study site can be a challenge as there is no graphical user interface and little documentation exists. Ideally, it is recommended that both models be used for one-line modeling studies of MNs. While neither model can perfectly show how a MN will evolve in a physical setting, analyzing results from both should be able to show users the general morphologic trends. GenCade is likely to produce a closer picture of the absolute coastline position across the domain while CEM seems to better capture tip migration and retreat dynamics. Future work with these models on MNs should include a more detailed study of MNs in other regions of the world.

4.5. Conclusions

Mega-nourishment is an innovative solution addressing the problem of chronic coastal erosion. Through mega-nourishment, a large volume of sediment placed in a single location is redistributed through natural processes to feed nearby beaches. The

evolution over 3.5 years of mega-nourishment coastlines at the Dutch Sand Motor is explored here using GenCade and a modified version of the Coastline Evolution Model (CEM). Both models can show coastline diffusion at rates similar to that observed based on the rate of shoreline retreat at the nourishment tip. The value of the empirical constant *K* in the sediment transport (CERC) equation affects this rate with higher *K* values yielding faster diffusion. Of the *K* values commonly used in modeling traditional nourishments, $K = 0.92$ (suggested by USACE) gives a low RMS difference between modeled and measured coastlines across the entire domain. In modeling an observed mega-nourishment, however, increasing *K* to 1.20 in CEM and 1.60 in GenCade can minimize mean RMS differences along the entire domain. Using higher *K* values than commonly used also produces feature tip retreat within 4 m of that observed using $K = 1.10$ and $K = 2.00$ for CEM and GenCade, respectively. Given the size of a mega-nourishment relative to traditional nourishments, it is likely that a higher-than-usual *K* value may be a better fit when simulating mega-nourishments in a one-line model. In evaluating model performance, GenCade is able to produce coastlines with an average RMS difference between modeled and measured coastlines on the order of 50 m, thus making it a useful tool for predicting absolute coastline position. Thus, processes such as wave-shadowing in one-line models may not be necessary when predicting absolute coastline position is the goal. However, CEM is able to capture longshore migration of the feature tip on a similar scale to that observed due to its wave shadowing algorithm. CEM also produces an asymmetric feature. While surveys of the SM coastline show some asymmetry, it is not as pronounced as that modeled with CEM. In the form

presented, GenCade tends to diffuse the formation symmetrically without much migration as a simple wave transformation technique (without wave shadowing) is used. As such, wave-shadowing allows a feature's migration to be captured and shows that a feature's symmetry may change.

5. NUMERICAL MODELING OF MEGA-NOURISHMENT SHORELINE INTERACTIONS WITH A GROIN FIELD[2]

5.1. Introduction

Shore erosion is a significant threat given that 24% of the world's sandy coasts are actively eroding (Luijendijk *et al*., 2018) and approximately 40% of the world population lives on or near the coast (Mentaschi *et al.*, 2018). Human investment in coastal regions continues to grow as population centers develop with the continual threat of sea level rise and coastline retreat (Gornitz et al., 1994). As a result, shore protection schemes to reduce the risk of erosion and flooding have become common in developed coastal areas. These can take the form of hard defenses such as groins, jetties, seawalls, *etc*. While hard defenses are able to stabilize beaches and inlets and protect against storm surge, they are expensive and permanently alter the natural beach (*e.g.,* Brown and McLachlan, 2002; Griggs, *et al.,* 1994; Williams *et al*., 2018). Conversely, soft defense schemes (*e.g*., beach nourishments) make use of added sand placement, resulting in a natural beach appearance. However, costly repetitive nourishment is required as nourished sediment is often transported out of the area over time (*e.g*., Davison *et al.*, 1992; Peterson and Bishop, 2005).

Given the necessity of protecting coastal infrastructure, innovation is needed to maintain a modern and efficient defense. One such innovative solution is the use of

[2] Note that this chapter has been submitted for publication in the peer-reviewed journal *Coastal Engineering* and is under review at the time of writing.

mega-nourishment (MN), a large-scale soft defense where a sizable nourishment volume (millions of m^3) is deposited at a single site rather than spreading out construction activity and beach fill material over many smaller projects along the coast. The MN sediment is redistributed via natural processes to adjacent beaches, effectively acting as a feeder beach to nourish a long stretch of coast. This could potentially be more cost effective than repetitive traditional nourishments. Given that MNs are relatively new technology, a limited amount of information is available on their evolution. Construction was completed in 2011 on a pilot MN (17 million m^3 sediment volume) called the Delftland Sand Engine (*Zandmotor* in Dutch), which was built to examine the feasibility of MNs feeding large stretches of coastline (de Schipper *et al.*, 2016). To date, this is the only physical MN constructed at such a large scale.

However, many systems that could most benefit from the shoreline advance and sediment feeding ability provided by MN construction (actively eroding coasts with highly developed infrastructure) are reinforced by groin fields (*e.g.*, Galveston Island, Texas; Miami Beach, Florida). Groins are one of the oldest and most popular hard coastal engineering schemes. As described in Basco (2002), these are structures built perpendicular to the coastline to control longshore sediment transport (LST) through its inhibition. Groins are typically built on continuous beaches with the intent of anchoring the beach or extending the lifetime of a beach fill. The goal of groin placement on a continuous beach is to maintain a minimum dry beach width, which can reduce storm damage and minimize local coastline erosion in the vicinity of the groin. Typically, groin fields (GFs) consisting of multiple groins are used to anchor a stretch of beach (Basco

and Pope, 2004). Coastline morphology inside and near GFs is driven by a number of forcings, primarily wave induced LST under normal meteorological conditions (Kamphuis, 2010). Since groins inhibit LST, this often results in a sawtooth-shaped coastline when there is a single dominant direction of littoral drift as sediment accumulates updrift of groins. However, this also results in sediment starvation (coupled with erosion) downdrift of groins as the replenishment of sediment transported downdrift is inhibited by the groins. According to Kamphius (2010), littoral transport in both directions can result in sediment starvation on both sides of the groins due to LST inhibition. Cross-shore processes also contribute to GF shoreline morphology as the presence of groins can induce rip currents that transport material offshore (Basco and Pope, 2004). Furthermore, high-energy events can produce shore-normal wave action within groin compartments that can transport material off the beach landward and/or seaward, resulting in additional coastline retreat (Harter and Figlus, 2017).

The coupling of a MN and a GF is an interesting combination as MNs and GFs seemingly have opposite goals and effects on shore evolution. Groins anchor beaches by inhibiting LST while MNs feed beaches via natural LST. As there is no GF in the vicinity of the only MN yet constructed, there is no physical data on how these two features would interact. Numerical modeling efforts provide an opportunity to further investigate the coastline interactions between a MN and a GF. While no numerical modeling studies have been published that investigate how a coupled MN-GF system may evolve, much modeling has been performed on the Sand Engine, specifically using three-dimensional (3D) models (*e.g.*, Luijendijk *et al*., 2017) such as Delft3D (Lesser *et*

al., 2004). While 3D models are capable of providing detailed insights to complex processes, they do not necessarily provide more accurate results on coastline evolution than less complex processed-based models that operate with reduced resources such as one-line models. One-line models are coastline models that are used to predict the cross-shore movement of a single contour, in this case, the shoreline. This is done without rendering all of the complex physics of the system (Zacharioudaki and Reeve, 2008). One-line models are often the preferred method of evaluating coastline change in engineering applications as they are remarkably robust despite not having large data or computational requirements (Thomas and Frey, 2013). Some commonly used one-line models include the Generalized Model of Simulating Shoreline Change (GENESIS; Hanson and Kraus, 1989), Uniform Beach Sediment Transport (UNIBEST; Deltares, 2011), Littoral Processes and Coastline Kinetics (LITPACK; DHI, 2009), GENESIS + Cascade (GenCade; Frey *et al.*, 2012), and the Coastline Evolution Model (CEM; Ashton *et al.*, 2001). One-line models have been applied directly to the Sand Engine (*e.g.*, Tonnon *et al.*, 2018) as well as to theoretical studies involving MNs (*e.g.*,Valsamidis *et al.*, 2017; Valsamidis *et al.*, 2018). Specifically, CEM, designed to simulate shoreline evolution on large spatial and temporal scales, has been used to simulate MNs (Stevens *et al.*, 2013; Brown *et al.*, 2016). GenCade, the current U.S. Army Corps of Engineers (USACE) standard, has also been applied to the study of MNs (Whitley *et al.*, 2021). CEM and GenCade are applied here to assess the implications of combining a MN and a GF on a coastal system. To make this assessment, hypothetical numerical simulations are explored with a GF and a MN on an otherwise perfectly

straight coastline. A groin field with seven groins is used in these simulations with cross-shore groin lengths of 100 m and longshore groin spacing of 400 m. The groin lengths and spacing are similar to what is found at Galveston Island (on average), though the number of groins is less (7 compared to 15 at Galveston). The results of these hypothetical simulations may therefore present a rough idea on how a MN may evolve on Galveston Island given similar forcing conditions. This furthermore gives a MN beach area to GF area (longshore extent times groin length) ratio of about 8:1. As both models are driven by offshore wave climates, the effects of wave climate conditions are also assessed on a coupled MN-GF system. Lastly, as an erosional hotspot is often present downdrift of a GF, the conditions under which this erosion can be mitigated through strategic MN placement are evaluated. Note that even though Galveston Island serves as a baseline testbed, the goal is not to model a physical system specifically but rather to explore hypothetical MN-GF systems and examine how they theoretically should evolve.

5.2. Methods

5.2.1. The Coastline Evolution Model (CEM)

CEM is described in detail in Ashton and Murray (2006a; 2006b) and Section 4.2.3 of this dissertation. For this study, CEM is updated to run in MATLAB and novel groin simulating algorithms are added (see Section 4.2.1.1 below). The MATLAB version runs slower than the original C version but is still relatively efficient taking about 25 minutes per model year to simulate a 25 km-long coastline. The model code

for the MATLAB version of CEM can be found in Appendix B. CEM was originally designed as a large-scale theoretical model, not intended for specific case-studies (Ashton and Murray, 2006a). To better explore MN evolution, Whitley *et al.* (2021) adjusted CEM so that K, ρ, ρ_s, γ_b, and n in (3.2); the breaker index γ_b; and the depth at which to begin wave refraction can be user-specified. Those adjustments are used here.

Given the complexity and scarcity of nearshore wave data, CEM is driven by deep water wave information. This information is numerically transformed into breaking wave height (H_b) and angle (ϕ_b) accounting for refraction and shoaling (described in detail in Ashton and Murray, 2006a; Whitley, 2014; and Section 4.2.3). The transformation assumes that local bathymetric contours are straight and parallel with the coastline. Note that the construction of a MN may disturb these contours in the near term. However, as the MN diffuses and the bathymetry evolves to a natural equilibrium profile, the validity of this assumption should be strengthened.

CEM offshore wave information is determined by a probability distribution function (PDF). See Ashton and Murray (2006b) or Section 4.2.3 for details on PDF theory. For artificial wave climates used in theoretical and hypothetical CEM simulations, PDFs are controlled by two fractional variables: wave climate asymmetry (A) and highness (U) (Figure 5.1 insert). The parameter A controls the probability of negative incident wave angles relative to shore normal (*e.g.*, $A = 0.30$ indicates 30% of offshore waves are negative), and U controls the probability of high-angle ($|\phi_0| > 45^o$) waves relative to shore normal (*e.g.*, $U = 0.4$ indicates that 40% of the waves are high-angle). It should be noted that U also controls the diffusivity of protruding coastline

features such as a MN (diffusion is diagramed in Figure 5.1) as high-angle waves lead to anti-diffusion of features (Falqués, 2003; Falqués and Calvete, 2005). Simulations of MNs by CEM and GenCade performed in this study confirmed this and showed that PDFs with $U > 0.4$ lead to anti-diffusion of the MN and, in some cases, model instability. As MNs need to diffuse to be effective, only diffusive wave climates are presented here, and thus, only values of $U \leq 0.4$ are used.

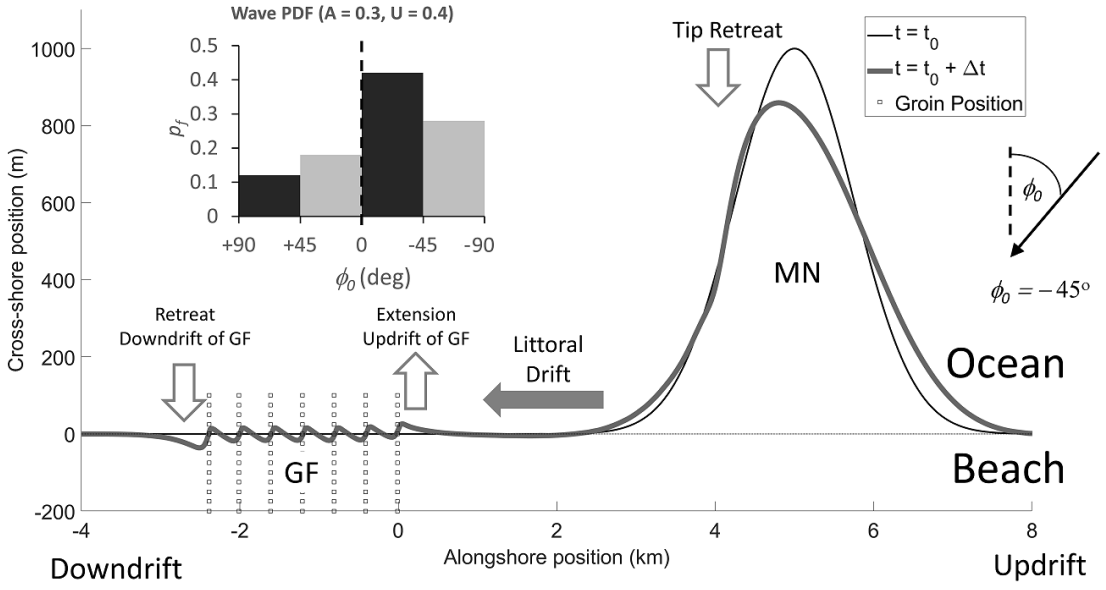

Figure 5.1. Schematic of hypothetical coastline evolution (t_0 and $t_0 + \Delta t$, respectively) including a MN and adjacent GF. The wave climate probability distribution function (PDF, top left insert) shows the fractional probability of occurrence (p_f) of offshore wave angle (ϕ_0). Hollow gray arrows indicate shoreline change of note.

5.2.1.1. CEM Groin Implementation

As CEM is to be used to explore MN coastline interactions with groin fields, groin simulation algorithms are added to CEM. Groin longshore location and cross-

shore length can be user-specified with groins occupying the right boundary of the specified cell. The presence of a groin in the model inhibits LST, though a fractional amount of sediment can be transported across via bypassing and groin permeability, handled in the same manner as in GENESIS (Hanson and Kraus, 1989) and GenCade (Frey *et al*., 2012). Groin permeability (*P*), a quantification of the sum of through-passing, over-passing, and shore-passing (Basco and Pope, 2004), is a user-specified fractional value. The fractional amount of bypassed material (*B*) is dependent on the water depth at the groin tip (D_G) and the depth of longshore transport (D_{LT}):

$$B = 1 - \frac{D_G}{D_{LT}}. \tag{5.1}$$

If $D_{LT} \leq D_G$, $B = 0$. D_G is calculated via an equilibrium profile and D_{LT} is calculated via

$$D_{LT} = \frac{A_W}{\gamma_b} \tag{5.2}$$

where A_W is the conversion factor 1.27 used when calculating depth of longshore transport for bypassing (Hanson and Kraus, 1989). Thus, the total fractional amount of sediment passing over, around, or through a groin (*F*) is given by

$$F = P(1 - B) + B. \tag{5.3}$$

A combined formulation for wave refraction and diffraction near groins is also simulated using a method described in Kamphuis (2010). Simulated groins cast a wave shadow region determined geometrically by the wave angle at the groin tip and the length of the groin. Breaking wave height due to refraction and diffraction (H_b') is determined by

$$H_b' = K_d H_b, \tag{5.4}$$

where H_b is the breaking wave height without diffraction and K_d is the diffraction coefficient. The formulation for K_d, as shown in Kamphuis (2010), is based on directional spreading of waves described in Goda (2000). Examining the Rayleigh distribution of wave heights, the wave height (H) along the shadow line is expressed as $0.71H_i$, where H_i is the wave height at the groin tip. Thus, the diffraction coefficient K_d along the shadow line is 0.71. A regression analysis of the relationship between wave energy reaching the shore and the diffracted wave angle yields the following:

$$\begin{aligned} K_d &= 0.71 - 0.0093\,\alpha_s + 0.000025\,{\alpha_s}^2 \text{ for } 0° \geq \alpha_s \geq -90° \\ K_d &= 0.71 + 0.37 \sin \alpha_s \text{ for } 40° \geq \alpha_s \geq 0° \\ K_d &= 0.83 + 0.17 \sin \alpha_s \text{ for } 90° \geq \alpha_s \geq 40°, \end{aligned} \tag{5.5}$$

where α_s is the mean wave direction at the groin tip (determined from the refracted mean offshore wave direction). The breaking wave angle due to diffraction is determined by

$$\alpha_{bd} = \alpha_b {K_d}^{0.375} \left[\frac{2d_g}{L_g\{\tan \alpha_s + \tan(0.88\alpha_b)\}}\right], \tag{5.6}$$

where α_{bd} is the breaking wave angle due to diffraction, α_b is the breaking wave angle without diffraction, d_g is the longshore distance between the breaking point (cell boundary) and the groin, L_g is the cross-shore length of the groin, and α_s is the mean wave direction at the groin tip.

5.2.2. GENESIS + CasCade (GenCade)

GenCade, described in detail in Frey *et al.* (2012) and Section 4.2.4, is a one-line model developed by the USACE that combines the regional-scale calculations of the Cascade model (Larson *et al.*, 2003) with the project-scale calculations of GENESIS (Hanson and Kraus, 1989) and is currently the U.S. industry standard one-line model.

GenCade is driven by an offshore wave data time series. Wave data include H_0, T, and ϕ_0 (relative to shore normal) over a sampling interval. Wave data can be physical, hindcast, or numerically modeled. While GenCade can be coupled to an external wave model, it also has an internal wave model capable of transforming deep water wave information to breaking. This accounts for shoaling and refraction over bathymetric contours. Since GenCade results are evaluated in conjunction with those from CEM and a fair comparison is desired, GenCade time series are generated by the same PDFs used with the simulation's CEM counterpart, effectively giving both models near-identical offshore wave climates. Multiple boundary condition (BC) options are available, though fixed (or "pinned") BCs are used here where the cross-shore shoreline position at the boundaries do not change. Here, boundaries are a large distance (7 or more km) from the MN bases and GF terminal groins to mitigate BC effects.

Groin simulation is handled in a similar manner as in CEM described above. Bypassing of groins is calculated according to (5.1), (5.2), and (5.3). Changes in wave height due to diffraction from groins is calculated by (5.4), however the diffraction coefficient K_d is handled differently. The values used for K_d depend on the wave angle. K_d values corresponding to every possible wave angle have been digitized in GenCade

according to curves presented in Kraus (1984), which are based on theory in Goda *et al*. (1978). GenCade is also extremely efficient taking about 10 seconds per model year to run a 10-km-long coastline with a 1-hour time step.

5.2.3. Model Validation for Galveston Island, Texas

To validate the newly incorporated groin algorithms, CEM simulations are parameterized for Galveston Island, Texas, which has a relatively straight coastline fortified with a GF. These simulations are compared with measured results of Galveston's shoreline as well as those produced by GenCade whose groin algorithms are well-vetted. Shoreline data for Galveston Island is available from 2014 to 2020. However, data from 2017 to 2020 is considered unreliable. Hurricane Harvey (2017) produced rapid changes in the shoreline, and while LST processes generated from tropical cyclone-induced offshore waves can be captured in CEM and GenCade, waves generated nearshore and cross-shore processes are not simulated in these models. Furthermore, cross-shore processes have been shown to be transport material offshore during tropical cyclones on Galveston Island (Harter and Figlus, 2017). While these changes often normalize on the Upper Texas Coast within 5 years (Morton *et al.*, 1994), there is not enough time in the data range to capture this. Also, data from 2018 on is produced by a different company (Aptim, 2018; 2019; 2020) compared to the pre-2018 data (Atkins, 2014; 2015; 2016; 2017). As the two use different geo-reference points, data from one company do not align properly with data from the other. Furthermore, much of the Aptim data have very low longshore resolution (one transect every ~100 m

or more), which would produce very low resolution shorelines. As such, only data from June 2014 to June 2016 are considered.

Mean low water (MLW) contours from June 2014, 2015, and 2016 are utilized in model validation (see Table 5.1 for parameter values). Topographic and groin aspect parameters are determined from a combination of Aktins (2014, 2015, 2016), NOAA (2006), and Google Earth imagery. Wave data is obtained from NOAA NDBC Station 42035 (Galveston, TX), producing an approximate PDF of $A = 0.38$, $U = 0.24$[3]. The June 2014 MLW contour is used as the initial shoreline for all simulations. As no data are available for groin permeability (P) at Galveston, a variety of P values are assessed. As the groins are made of concrete with minimal through-passing, P is likely close to zero. Furthermore, values of $P > 0.3$ give unrealistic shoreline results with minimal groin effects (defined as the distance between the maximum accretion updrift of a groin to the maximum erosion downdrift of a groin). As such, only values of $P \leq 0.3$ are discussed.

Model performance is assessed by comparing modeled coastlines to those observed at Galveston. While a large extent of the Galveston coastline is modeled (from the Bolivar Roads jetty to roughly 1 km west of the western edge of the Seawall), the area of interest is the shoreline surrounding the first twelve groins (from the west) of the GF to assess the viability of the novel groin algorithms in CEM (Figure 5.2). The

[3] Note that a higher resolution 12-bin wave PDF is used for Galveston simulations as opposed to the traditional 4-bin PDF used in all theoretical CEM and GenCade simulations. See Whitley *et al.* (2021) for details.

coastline here, while containing groin effects in the groin compartments, is relatively straight on average (θ values are low), thus avoiding complexities arising from large Q_s computations in (3) that can result from an unrealistically large α_b ($\alpha_b = \phi_b - \theta$). The longshore ($x$) axis chosen for all Galveston simulations and analyses is a line running eastward to westward on a bearing of 233.7 degrees relative to true North.

Table 5.1. Parameters used in model validation (CEM and GenCade) for Galveston Island, Texas.

Parameter	Value	Description
K	0.92	Empirical coefficient (Rosati *et al.*, 2002)
ρ_s (kg/m^3)	2650	Sediment density
ρ (kg/m^3)	1000	Seawater density (fixed in GenCade)
n	0.4	Sediment porosity
γ_b	0.54	Breaker index (Battjes and Stive, 1985)
D_C (m)	8.00	Depth of closure
D_B (m)	1.31	Berm elevation
S_{sf}	0.015	Slope of the shoreface
S_s	0.00051	Slope of the shelf
T (s)	4.16	Average offshore wave period
$H_{0,rms}$ (m)	0.71	Average offshore root mean square wave height
z_{max} (m)	15.8	Water depth where refraction begins
D_{50} (μm)	132	Median sediment grain diameter (Lisle and Comer, 2011)
Δt (hours)	1.0	Time step

Models are evaluated after 1 year (comparable to the June 2015 MLW contour) and 2 years (comparable to the June 2016 MLW contour) of evolution. Model performance is evaluated via modeled-measured shoreline position root mean square difference (RMSD). Shoreline RMSD is calculated for every groin compartment with

low values corresponding to low cross-shore differences in measured-modeled coastlines. The lowest overall mean, lowest maximum, and lowest minimum RMSD over all the groin compartments are examined. Note that there was a nourishment (~481,000 m^3) at Babe's Beach (near x = 4.5 km in Figure 5.3) just west of the GF from Sept. 2015 to Nov. 2015 (TGLO, 2015; Atkins, 2018; Elko *et al*., 2021). This explains

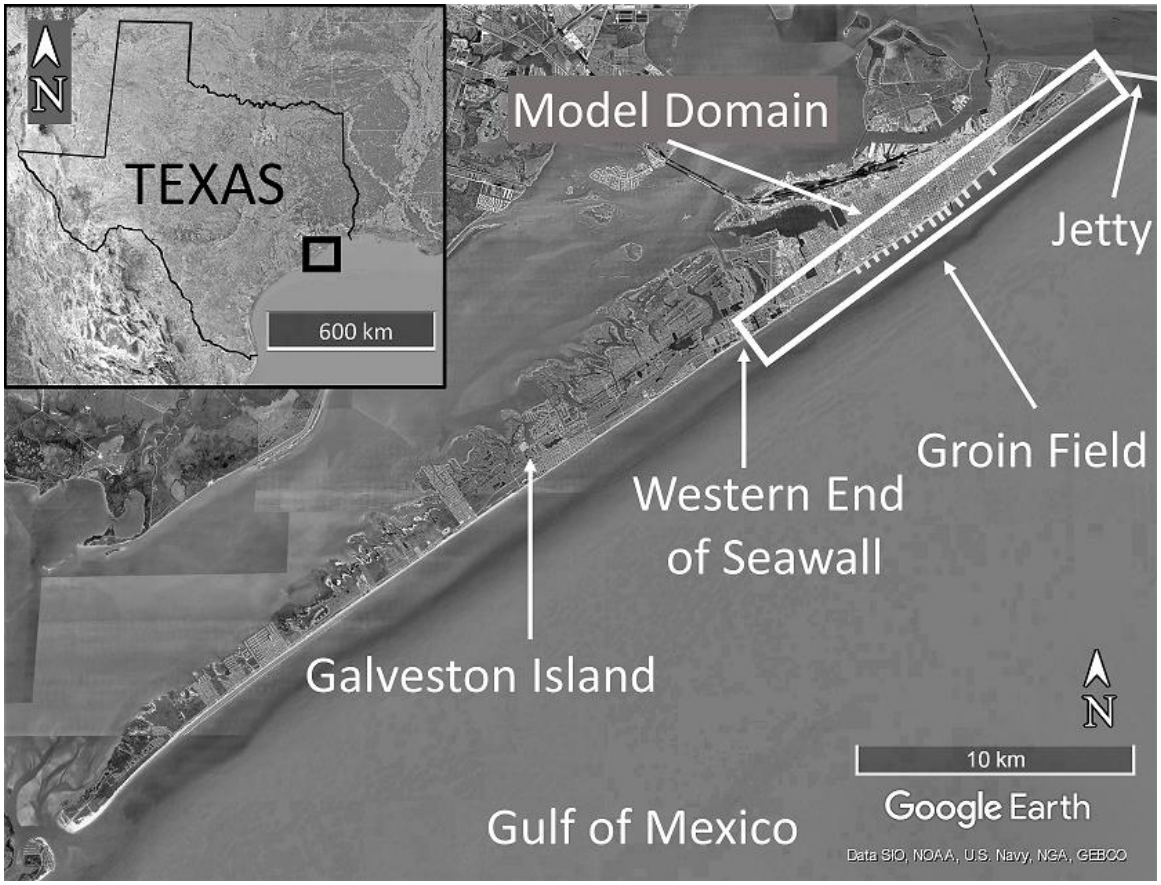

Figure 5.2. Map of Galveston Island, Texas. Comparisons between simulated and measured shoreline position evolution on Galveston Island between the Jetty (upper right) and the western end of the seawall during the period of 2014 – 2016 are used for model validation. The groins have been accentuated in this image for better visibility (image courtesy of Google Earth).

the large amount of cross-shore extension from 2015 to 2016 in measured coastlines. It should be noted that the Galveston coast experienced no tropical cyclones from 2014 – 2016, though northerly cold fronts frequently pass over the area during winter months that generate considerable wind shear stresses (Dellapenna *et al.*, 2006).

In general, both models produce approximately the same coastline shape (Figure 5.3). Neither model captures the U-shape of the Galveston coastline in the groin compartments very well. Note that a partial U-shape can by synthetically generated in CEM if the wave PDF asymmetry is reversed halfway through the simulation, though this was not produced in GenCade. This suggests that U-shape coastlines within groin compartments may be the result of a time dependence in the offshore angle of incidence, which can be implemented by changing the forcing PDF over time, even though this has not been implemented as part of the scope of this work. Groin effects are present with accretion on the west and erosion on the east due to the asymmetry of the incident wave PDF ($A \sim 0.38$). CEM typically underpredicts the low point within the groin compartments but can predict the high point within 0 to 30 m (depending on P). GenCade tends to greatly overpredict the magnitude of the groin effects (both high and low points) with high points 30 to 50 m away and low points 0 to 50 m from observed positions. Note that this low point always occurs next to the groin on the west while the observed coastlines tend to have erosion occur in the middle of the groin compartment. The low point in CEM is typically close to the west side groin but not directly adjacent to it due to reduced LST from wave diffraction.

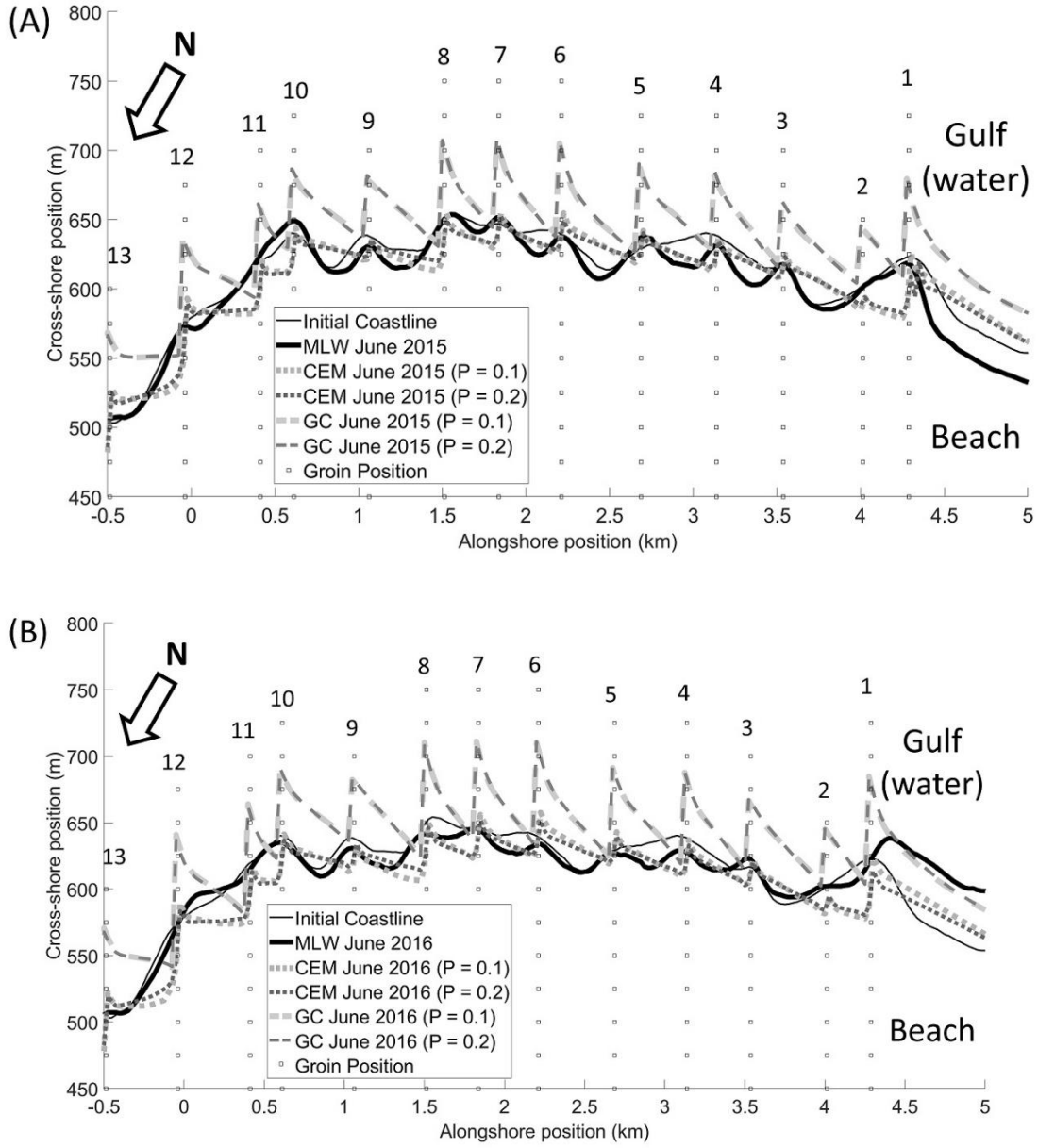

Figure 5.3. Coastline evolution of CEM and GenCade simulations for the Galveston Island GF from June 2014 – June 2015 (A) and June 2014 to June 2016 (B). Groin numbers are identified above each groin tip.

CEM mean RMSDs for the groin compartments range from 12.6 to 15.7 m for 1 year of evolution and 11.8 to 16.2 m for 2 years (Table 5.2). Minimum RMSDs of all the groin compartments per simulation range from 5.6 m to 6.2 m for 1 year and 3.6 m to

9.5 m for 2 years. GenCade mean RMSDs are 29.7 m for 1 year and range from 30.4 m to 30.8 m for 2 years. Minimum RMSDs of all the groin compartments per simulation are on the order of 17.7 m for 1 year and range from 18.6 m to 20.9 m for 2 years. Overall, CEM results produce a lower RMSD than GenCade when evaluating the mean, maximum, and minimum RMSDs for each groin compartment.

Table 5.2. Root mean square difference (RMSD) between modeled-measured coastlines for Galveston Island for 1 year (ending 2015) and 2 years (ending 2016) of evolution.

			Groin Field RMSD (m)		
Model	**End Year**	**P**	**Mean**	**Maximum**	**Minimum**
CEM	2015	0.0	15.7	30.5	6.2
CEM	2015	0.1	14.1	25.4	5.6
CEM	2015	0.2	13.1	26.9	5.7
CEM	2015	0.3	12.6	27.5	5.8
GenCade	2015	0.0	29.7	35.3	17.7
GenCade	2015	0.1	29.7	36.3	17.7
GenCade	2015	0.2	29.7	37.6	17.5
GenCade	2015	0.3	29.7	37.6	17.8
CEM	2016	0.0	16.2	28.4	9.5
CEM	2016	0.1	13.8	22.9	6.8
CEM	2016	0.2	12.4	22.2	5.1
CEM	2016	0.3	11.8	23.5	3.6
GenCade	2016	0.0	30.8	38.6	20.9
GenCade	2016	0.1	30.8	38.5	20.3
GenCade	2016	0.2	30.6	39.0	19.5
GenCade	2016	0.3	30.4	40.0	18.6

The difference between the modeled-measured beach area inside each groin compartment is also evaluated (Table 5.2). Differences in CEM-measured groin compartment beach areas range from -9876 m^2 to 7328 m^2 with negative differences corresponding to CEM showing less beach area within the groin compartment than

measured (and vice versa). GenCade-measured area differences are only positive ranging from 1141 m^2 to 17902 m^2. In general, GenCade produces much greater beach areas within the groin compartments than both the measured areas and those produced by CEM.

GenCade is the U.S. industry standard and is the primary one-line model used by the USACE. A valid model should produce comparable results to GenCade. CEM produces overall coastline forms similar to GenCade. RMSD values and modeled-measured beach areas for CEM are typically of lower magnitude than that of GenCade, and measured-modeled beach areas within the groin compartments are also lower for CEM on average. When modeling a GF, CEM should be able to produce results at least as well as the U.S. industry standard.

5.2.4. Research Approach

To assess the effects of placing a MN in proximity to a GF, baseline scenarios of a MN and a GF each alone on an otherwise straight coastline are performed. Having an otherwise straight coast allows for analysis of the effects of the MN or GF since without them no shoreline change should occur on a straight coast. This is done for a variety of wave climates (varying *A* and *U*), which should indicate the effects each variable has on a MN and a GF alone. These results are compared to coupled MN-GF simulations where the effects of varying *A* and *U* are evaluated. In all these theoretical scenarios, parameterization is the same as in Table 5.1 except for D_B = 1.00 m, H_0 = 1.00 m, T = 8.00 s, and S_S. A very small value of S_S (10^{-99}) is used to remove complications from

modeling shelf dynamics (*i.e.*, no sediment is deposited on the shelf when accreting to a water depth beyond the depth of closure; see Ashton and Murray, 2006a for details). Groin permeability ranging from $P = 0.0$ to $P = 0.3$ are evaluated.

The initial conditions for all simulations with an MN use a nourishment with a cross-shore length of 1 km and a longshore footprint of 5 km. This gives the nourishment a total volume of ~17.5 million m^3. While the initial shape of the peninsula is perfectly Gaussian instead of a hook shape, the initial parameters of the hypothetical MN have a similar volume and aspect parameters as those of the Sand Engine upon completion. All simulations with a GF use a seven-groin GF, with all groins evenly spaced (400 m) and having a cross-shore length of 100 m from the initial coastline. With a shoreface slope (S_{sf}) of 0.015, this puts the groin tips in a water depth of ~1.14 m. The distance between groins, groin lengths, and water depth at the groin tip here are all similar to those in the Galveston simulations (on average), though there are fewer groins (7 versus 15 in Galveston), and the coastline is perfectly straight. With these groin and MN parameters, the ratio of the beach area of the MN with the area within the groin field (the longshore GF extent times the cross-shore length of groins) is about 8:1. Since periodic boundary conditions (CEM default) are not possible in GenCade and a fair comparison is desired between CEM and GenCade, pinned boundary conditions are used for GenCade and periodic boundary conditions in CEM are disabled. Instead, a straight coastline is used initially beyond the boundaries of the domain (extending out half the domain distance both directions along the *x*-axis) of CEM. Furthermore, any features (a MN or GF) are placed at least several kilometers away from the boundaries to

minimize boundary condition effects.

5.3. Results

5.3.1. Baseline Scenario Results (MN-Only and GF-Only)

5.3.1.1. Mega-Nourishment Only (MN-Only) Results

Simulations of a MN on an otherwise straight coastline are presented for CEM and GenCade with varying A and U in the wave PDF. As all MN-only initial coastlines are symmetrical (Figure 5.4), only results of $A \geq 0.5$ are presented as their counterparts (*e.g.*, $A = 0.3$ is a counterpart to $A = 0.7$) are mirror-images of each other. A symmetric wave climate ($A = 0.5$) leads to symmetric diffusion of the MN, feeding either side equally. When A is increased in either model, the peninsula shape skews downdrift, and the tip migrates slightly downdrift as more sediment accumulates on the updrift flank. In this manner, the MN also acts similarly to a groin in that it inhibits LST (Duo *et al*., 2015). Furthermore, CEM has wave shadowing algorithms that create a shadow zone downdrift of the MN where no wave action (and thus no LST) takes place (see Ashton and Murray, 2006a for details). This results in sediment starvation (and shoreline retreat) downdrift of the MN early in the simulation (as also seen in Whitley *et al*., 2021), though a net shoreline extension results in this area on long time scales (after the

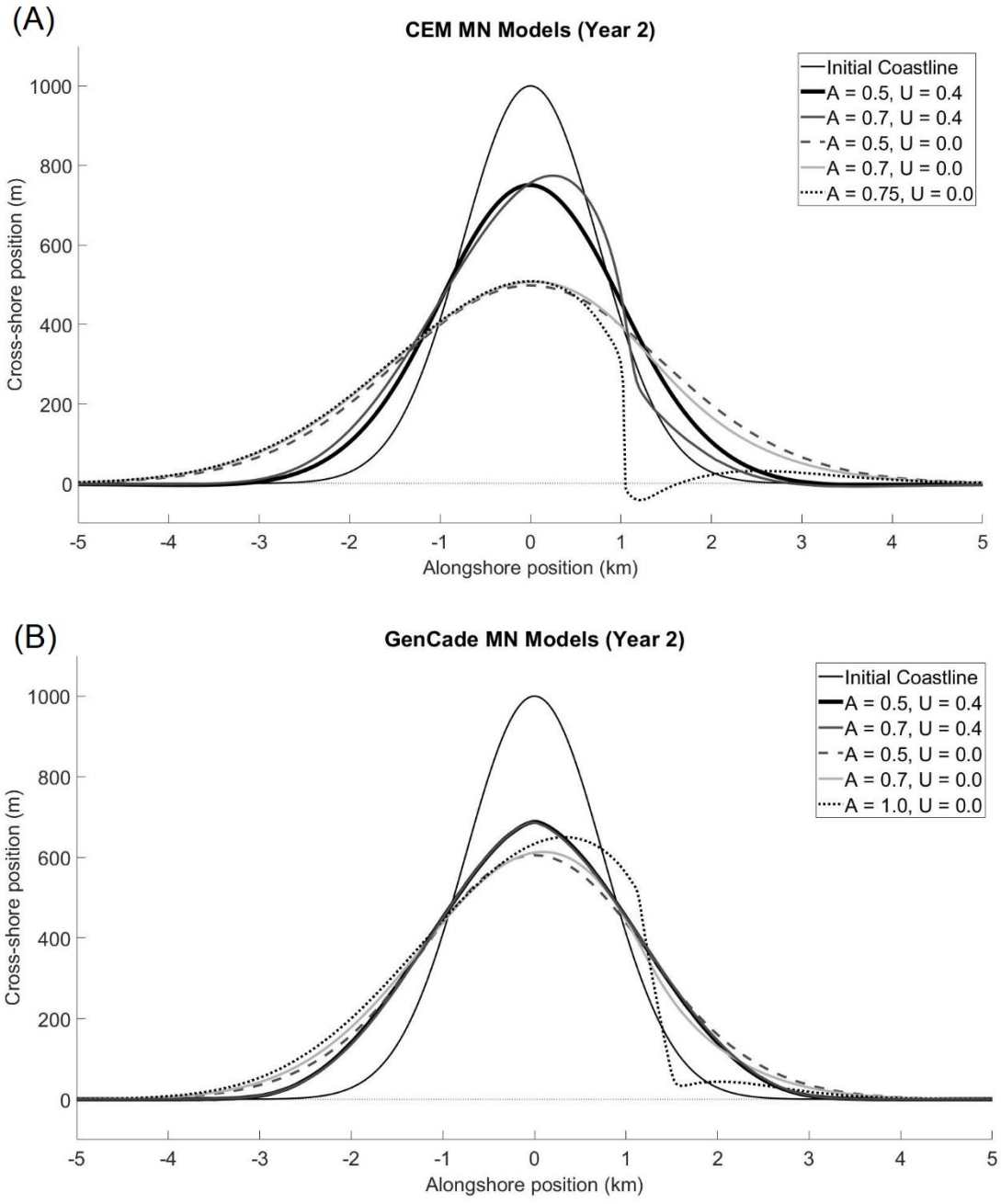

Figure 5.4. MN-only model results for CEM (A) and GenCade (B) after 2 years of evolution for varying wave climates. Year 2 is shown here because all the effects of wave climate on model evolution are visible. Results for $A > 0.75$ are unstable in CEM. The extremely high shoreline angle downdrift of the MN for high A is also visible at year 2. As time goes on, the MN diffuses (becomes near parallel with the x-axis), and the effects of wave climate asymmetry are less prominent.

MN diffuses). Asymmetric wave climates in CEM also result in the MN feeding a larger stretch of coast downdrift (*e.g.*, cross-shore extension is noticeable 2 km further downdrift at year 25 when $A = 0.7$ as opposed to when $A = 0.3$), though this is less apparent in GenCade.

When A is increased beyond 0.7, sediment starvation effects downdrift of the MN become very apparent in CEM. The downdrift flank of the MN produces a shoreline angle near perpendicular to the x-axis in early years of the simulation (*e.g.*, year 2 for $A = 0.75$, $U = 0.0$ in Figure 5.4A) coupled with shoreline retreat downdrift of the flank. A similar coastline shape is produced in GenCade when $A = 1.0$, $U = 0.0$ (Figure 5.4B). This shape is consistent with MN coastlines produced by Q2DMorfo (van den Berg *et al.*, 2014) under a highly asymmetric climate ($A = 1.0$) shown in Arriaga *et al.* (2020). However, this shape persists for up to 50 years in Q2DMorfo where diffusion eventually fills in the MN flank, resulting in a flatter coastline in just a few model years, for both CEM and GenCade. Erosional hotspots downdrift of the MN in CEM (arising from wave shadowing) are also infilled over time in most simulations. On long time scales (decade+), $A \geq 0.7$ can result in numerical instabilities in CEM. As such, only A values ranging from 0.3 to 0.7 are presented for MN-GF scenarios below.

For $U \leq 0.4$, all simulations diffuse the MN with U primarily controlling the rate of diffusion, evaluated by MN tip (greatest cross-shore extent) retreat and volume change within the initial MN longshore footprint (± 2.5 km from $x = 0$ km, the MN center, in Figure 5.4). As the Sand Engine has a project life of 25 years, both of these are evaluated at the 25-year mark in the model. In general, increasing U decreases the

MN diffusion rate. This is expected as increasing the probability of high-angle waves increases the amount of anti-diffusion taking place (even though the majority of wave action results in diffusion). Varying *A* has no effect on diffusion rate when $U = 0$. However, when $U = 0.4$, varying *A* can have a minor effect on diffusion rate with higher asymmetry ($A = 0.3$ or 0.7) in the PDF leading to a slightly diminished tip retreat (~15 m less) and volume loss (~2% less) than when $A = 0.5$. This is the result of wave shadowing on the downdrift MN flank preventing wave action from transporting sediment off of the MN, which only occurs under higher wave angles. This is also in contrast to symmetric wave climates ($A = 0.5$) where any wave shadowing occurs on both sides evenly (and thus transport off the MN is symmetric).

On average for $U = 0.0$, tip retreats are 847.5 ± 0.3 m (mean ± standard deviation) for all CEM simulations ($0.3 \leq A \leq 0.7$) and 804.5 ± 0.3 m for GenCade (same *A* range). Increasing *U* to 0.4, the average CEM tip retreat drops to 670.8 ± 8.3 m while GenCade's drops to 678.4 ± 0.3 m. Volume losses within the initial MN longshore footprint also decrease with a *U* increase. The 25-year volume loss for $U = 0.0$ is 61.1 ± 0.1% for CEM and 51.43 ± 0.04% for GenCade. When *U* is increased to 0.4, the volume losses decrease to 27.0 ± 1.1% for CEM and 28.81 ± 0.02% for GenCade. For highly asymmetric wave climates (*e.g.*, $A = 0.3$ or 0.7) in CEM, increasing *U* can also lead to a more skewed peninsula shape as it takes longer for the MN to diffuse. In GenCade, the overall shape remains relatively Gaussian regardless of *U*.

5.3.1.2. Groin Field (GF-Only) Results

Simulations of a groin field (GF) with seven evenly spaced groins on an otherwise straight coastline are presented for CEM and GenCade with varying *A* and *U* in the wave PDF (Figure 5.5). For a symmetric wave climate (*A* = 0.5) in CEM, small groin effects (≤ 2 m) develop within the GF, but there is no clear buildup or erosion on either side of the GF. GenCade results for *A* ranging from 0.4 to 0.6 are unreliable as the resultant coastline is highly dependent on the sign (direction) of the first wave angle in the time series (*i.e.*, symmetric and near-symmetric wave climates produce results expected in an asymmetric wave climate). Under these conditions (assuming $\phi_0 \neq 0$), groin effects develop immediately in GenCade and with great enough magnitude that the

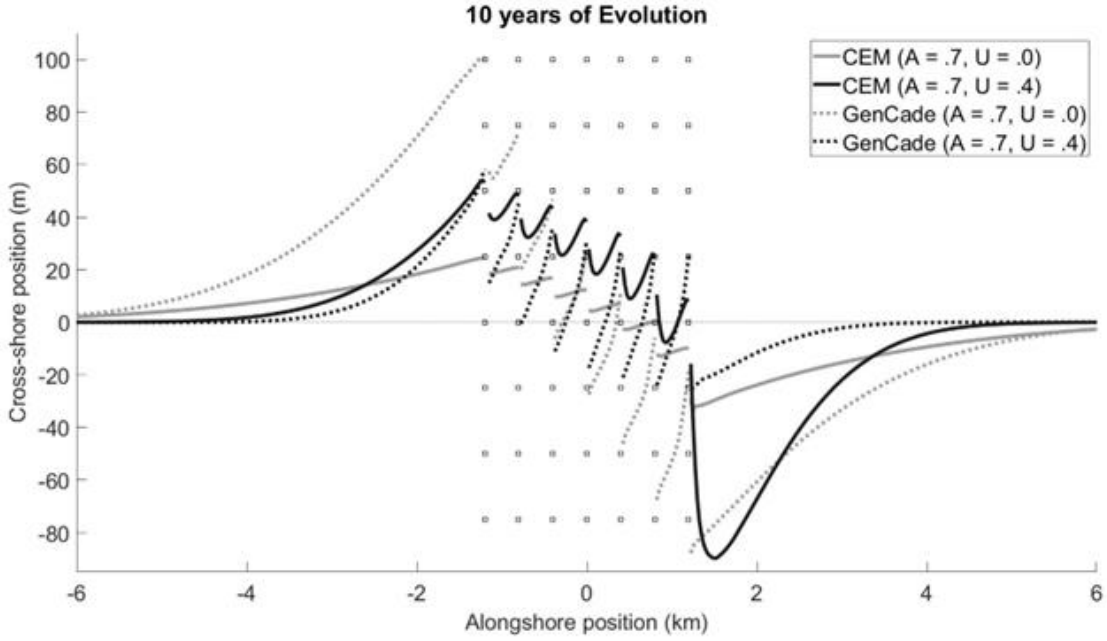

Figure 5.5. Results of CEM and GenCade simulations of a GF under an asymmetric wave climate (*A* = 0.7). Extreme values of *U* (0.0 and 0.4) are indicated. Results from both models follow the same form, though GenCade produces effects of greater magnitude.

shoreline angle (θ) becomes a dominant driver in Q_s calculations in (5.3) where $\alpha_b = \phi_b - \theta$. This results in the initial groin effect orientation persisting throughout the simulation, which is considered unrealistic. Note that this is true when H_0 and T are held constant and there is a 40% to 60% likelihood of a positive versus a negative wave angle.

For an asymmetric wave climate in CEM, there are clear areas of shoreline extension (accretion) and retreat (erosion) updrift and downdrift of the GF, respectively. In general, the greater the A, the greater accretion updrift, erosion downdrift, and magnitude of groin effects within the GF. The amount of extension and retreat both grow over time. This and the overall shoreline form are consistent with those presented in Kraus *et al*. (1994) when a net littoral drift is present (as with an asymmetric wave climate). The rate of accretion updrift of the GF tends to slow over time due to an increased amount of bypassing as the shoreline gets closer to the groin tip. However, the amount of erosion downdrift of GF tends to increase at a steady rate. GenCade asymmetric PDFs produce similar results to their CEM counterparts, though the magnitude of groin effects, updrift accretion, and downdrift erosion are much larger.

In general, increasing U in GF-only CEM simulations increases the magnitude of groin effects, updrift accretion, and downdrift erosion. There does not appear to be a correlation between altering U in GenCade and the relative size of groin effects inside the GF, though the accretion/erosion signals updrift/downdrift of the GF decrease as U increases. Increasing groin permeability (P) in CEM decreases all the above as more

sediment is allowed to through-pass the groins, while the groin effect magnitude in GenCade does not significantly change with P for $0.0 \leq P \leq 0.3$.

5.3.2. Coupled MN-GF Scenario Results

Coupled MN-GF scenarios have a MN and a GF on an otherwise straight coastline (Figures 5.6 and 5.7). A variety of wave climates are examined and a groin permeability value *(P)* of 0 is used unless otherwise noted. Simulations are performed in both CEM and GenCade, however both models have their limitations. CEM can become unstable under $A > 0.7$ or < 0.3, so only stable A ranges are shown. GenCade also becomes unstable if the shoreline extends past a groin tip as the current version of GenCade (v1.1r8) is unable to simulate structures being buried over the course of a simulation (personal communication with USACE ERDC personnel).

For the simulations presented below, the MN is directly adjacent to the GF, making the nourishment as close as possible to the GF without the initial Gaussian shoreline intruding upon the groins. While the position of the MN (centered at $x = 2.5$ km) relative to the GF (terminal groins at $x = -2.4$ km and 0 km) does not change, A in the wave climate effectively changes the updrift/downdrift orientation of the two. While $A = 0.5$ indicates a symmetric wave climate, $A = 0.3$ indicates that the MN is updrift of the GF, and $A = 0.7$ puts the MN downdrift of the GF. Wave PDF ranges of $A = 0.3$ to 0.7 and $U = 0.0$ to 0.4 (the stable extremes in all models) are presented.

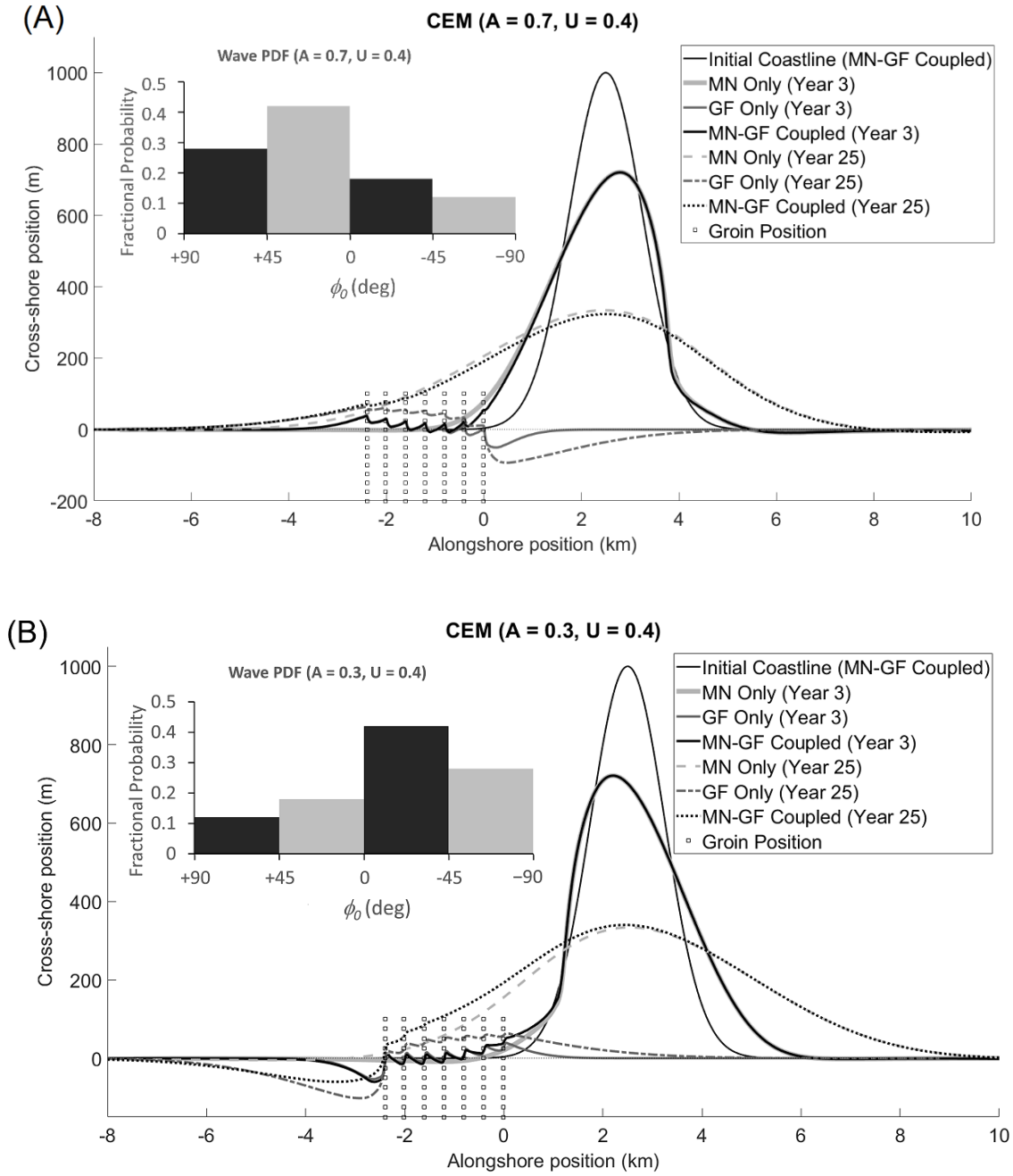

Figure 5.6. Results of coupled MN-GF simulations compared to MN-only and GF-only simulations in CEM under highly asymmetric wave conditions and with highness. These results show the conditions with the largest magnitude of accretion updrift of a GF, erosion downdrift of a GF, and groin effects within a GF. *A* = 0.7 (A) conditions indicate that the MN is downdrift of the GF while *A* = 0.3 (B) indicates that the MN is updrift of the GF.

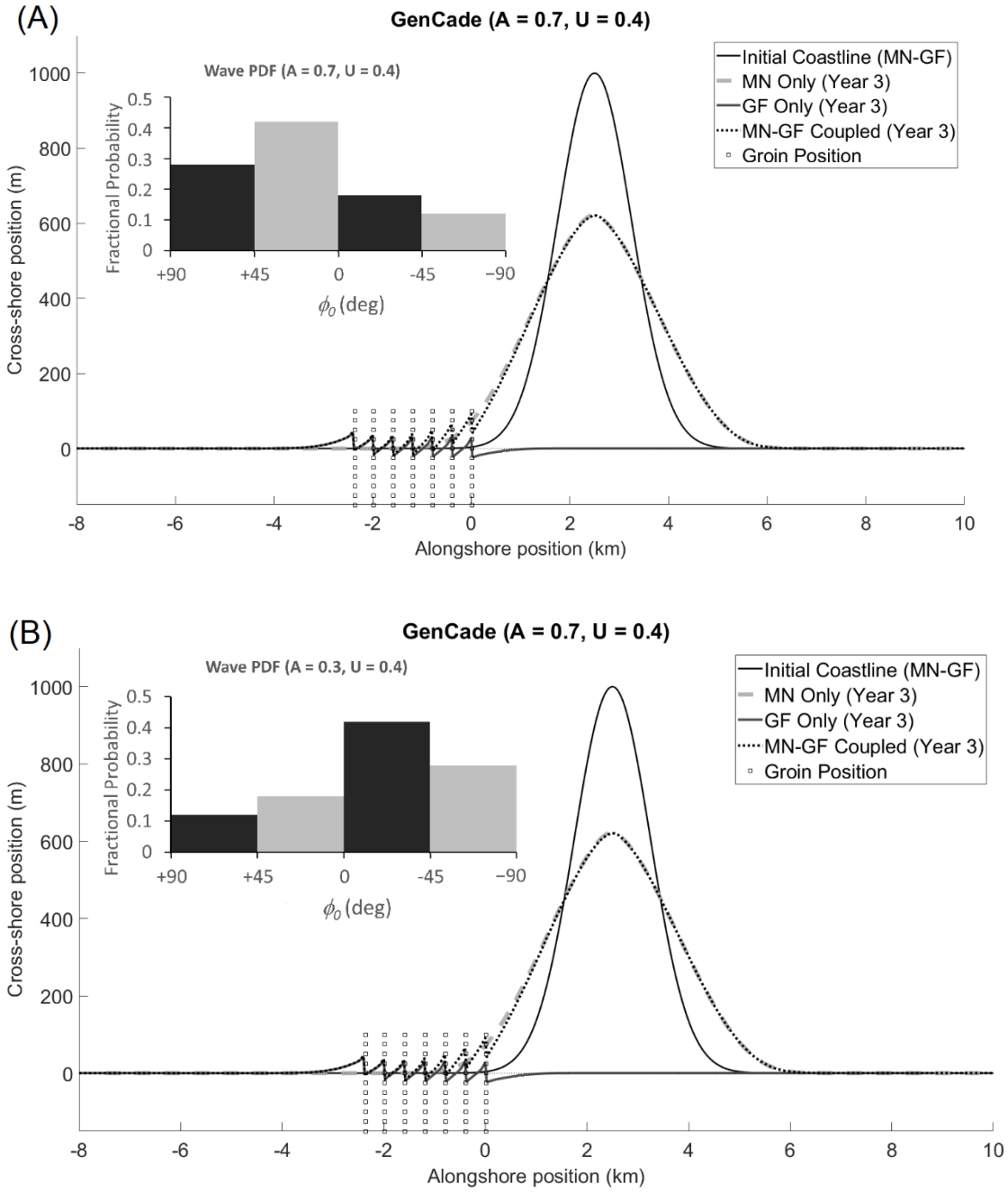

Figure 5.7. Results of coupled MN-GF simulations compared to MN-only and GF-only simulations in GenCade under highly asymmetric wave conditions and with highness. These results show the conditions with the largest magnitude of accretion updrift of a GF, erosion downdrift of a GF, and groin effects within a GF. *A* = 0.7 (A) conditions indicate that the MN is downdrift of the GF while *A* = 0.3 (B) indicates that the MN is updrift of the GF.

To investigate the influence of a GF on MN evolution, coupled MN-GF simulation results are compared with MN-only results. For symmetric wave climates (A = 0.5), the presence of a GF has no effect on MN evolution regardless of U and at all times scales in CEM. Under asymmetric wave PDFs (A = 0.3 or 0.7) with low U (0.0), very minor differences between MN-only and MN-GF are noticeable at year 3 in both CEM and GenCade. Groin effects begin to develop within the GF, as well as accretion updrift of the GF for A = 0.7 and erosion downdrift of the GF for A = 0.3 (both near x = -2.5 km), none of which are present in MN-only simulations. However, on longer time scales (25 to 75 years), there is no difference between MN-only and MN-GF simulations for an 8:1 MN to GF area ratio. This suggests that when U is low, the presence of a GF does not have any effect on MN evolution on multi-decadal time scales. This is the result of rapid diffusion of the MN which leads to the groins being completely covered by sediment by year 25, after which the MN evolves as if no GF was present. Under asymmetric (A = 0.3 or 0.7) wave climates with high U (0.4) there is also very little difference between MN-only and MN-GF early in both CEM and GenCade simulations (year 3, Figures 5.6 and 5.7) as there has not been much time for the groins to influence shoreline evolution. However, there are considerable differences between MN-only and MN-GF at year 25. When the MN is downdrift of the GF (A = .7, Figure 5.6A), the MN-GF area of the mega-nourishment tip and updrift flank (x = -1.5 to 4.5 km) have slightly less shoreline extension (up to 12 m less) than MN-only results. This is offset by accretion updrift of the GF (near x = -2.5 km). However, on longer time scales (year 75), sediment from the MN has buried the groins in MN-GF. Once this occurs, the

groins no longer influence shoreline evolution thus yielding a coastline identical to MN-only. When the MN is updrift of the GF (A = 0.3, Figure 5.6B), the MN-only shoreline at year 25 has less cross-shore extent (up to ~42 m less) than MN-GF from x = -1.8 to 3.6 km. Since the littoral drift points in the negative x direction in this wave PDF, sediment builds up between the MN and GF (x = 0 to 1.8 km) as diffusing MN sediment travelling downdrift is inhibited by the groin at x = 0 km. Note that this sediment buildup is offset by shoreline retreat downdrift of the GF (near x = -2.5 km). By year 75, most of the groins have been buried by sediment. However, the downdrift terminal groin at x = -2.5 km is still exposed and can influence shoreline evolution. As a result, there is slightly less (up to 17 m) shoreline extension in MN-GF than for MN-only downdrift of x = -2.4 km. However, this is offset by slightly more (under 17 m) shoreline extension in the region x = -2.4 to 6.0 km.

Similarly to MN-Only simulations, MN diffusion rates here are evaluated using MN tip retreat and volume loss at the 25-year mark (with CEM only as GenCade becomes unstable prior to year 25). The parameter U predominately controls the MN diffusion rate with higher U values leading to slower diffusion rates. As with the MN-only scenarios, A has no effect on MN diffusion rate when U = 0, but higher asymmetry can lead to slightly diminished diffusion rates when U = 0.4 due to wave shadowing. Furthermore, coupling a MN with a GF does not affect the diffusion rate for all A when U = 0 as these coupled MN-GF simulations have the same tip retreats and volume losses as their MN-only counterparts (same wave PDF among counterparts). However, coupling a MN and GF under high asymmetry (0.3 or 0.7) and high U (0.4) can change

tip retreats from -6.2 m (A = 0.7) to +10.5 m (A = 0.3) relative to their MN-only counterparts. This is not observed with low U (0.0) as the MN diffuses so rapidly that the groins are buried at the 25-year mark. However, under U = 0.4, the diffusion is slow enough that the influence from the GF is observable at year 25. This influence can take the form of sediment accumulation updrift of the GF (A = 0.7, near x = -2.5 km in Figure 5.6A), which is offset by a diminished shoreline extent (-6.2 m) of the MN tip (x = ~2.5 km) relative to MN-only. GF influence on shoreline evolution can also take the form of erosion downdrift of the GF (A = 0.3, near x = -2.5 km in Figure 5.6B), which is offset by increased shoreline extent (+10.5 m) in the MN tip (x = ~2.5 km) relative to MN-only.

An investigation of the erosional hotspot commonly found downdrift of a GF is also performed under a coupled MN-GF system. This hotspot does not develop under symmetric (A = 0.5) wave conditions, regardless of U. When the MN is placed downdrift of the GF ($A > 0.5$, $U \leq 0.4$), the erosional hotspot does not develop at all in either CEM or GenCade (Figure 5.6A and 5.7A, respectively). This is the result of the MN being placed in the region where the hotspot usually develops (x = 0 to 2 km). As the MN diffuses, sediment is trapped between the MN and GF preventing any shoreline retreat. However, the erosional hotspot does develop in the region of $x < -2.4$ km if the MN is updrift of the GF ($A < 0.5$) in both CEM (Figure 5.6B) and GenCade (Figure 5.7B). The hotspot is observable as early as year 3 in these simulations for all U. However, on longer time scales (25+ years), MN-GF shoreline evolutions differ with varying U. For U = 0.0 (A = 0.3) at year 25 the hotspot is no longer present as it has

been infilled with diffused sediment. At this point (and forward in time) there is no difference between MN-GF and MN-only. For higher *U* (0.4) the erosional hotspot persists up to 25 years. However, the magnitude of shoreline retreat in MN-GF is as much as 42 m less than that for GF-only at year 25 (Figure 5.6B). Sediment is able to travel past the groins through bypassing, even though impermeable groins are used (*P* = 0) in these scenarios. At year 75, the MN has diffused to the point where most of the GF is covered, and the erosional hotspot has filled in. Here, there is little difference between MN-GF and MN-only, though there is slightly less shoreline extension downdrift of the GF in MN-GF.

As with the erosional hotspot, areas of accretion updrift of the GF are not present in MN-GF simulations where *A* = 0.5 as there is no discernable sediment accumulation on either side of the GF. When the MN is downdrift of the GF (*A* = 0.7), the area updrift of the GF (*x* = -3.7 to -2.4 km) is exactly the same at year 3 for MN-GF and GF-only for all *U* in CEM and GenCade (Figure 5.6A and 5.7A, respectively) as sediment from the MN has not had time to reach this area. On longer time scales (25 years) in CEM, MN-GF has more accretion updrift in $x < -2.4$ km than GF-only, though the amount of accretion is dependent on *U*. When *U* = 0.0, MN-GF has at most 65 m more shoreline extension than GF-only. It should be noted that the MN-GF shoreline under this wave PDF is exactly the same as MN-only from year 25 to year 75. Under *U* = 0.4, there is at most 8 m more shoreline extension for $x < -2.4$ km in MN-GF than GF-only at year 25. On even longer time scales (year 75), the groins are buried, and the MN-GF shoreline is identical to MN-only. When the MN is updrift of the GF (*A* = 0.3), there is more

accretion updrift of the GF (x = 0 to 1 km) in MN-GF than GF-only for all U at year 3 in both CEM and GenCade (Figure 5.6B and 5.7B, respectively). The MN tip (at x = 2.5 km) is not far from this location (x = 0 to 1 km), and the region is being directly fed by MN diffusion as the dominant littoral drift is in the negative x direction. As a result, there is more accretion in this region as early as year 3, and the amount increases over time as the MN diffuses.

For simulations where the MN is built on top of the GF, the MN center is at x = -1.2 km, the same longshore position as the center groin in the GF. As GenCade is unstable from these initial conditions, only CEM MN-GF simulations are presented. Within a 25-year period, no simulations produce a MN diffusion rate (regardless of U) where the MN shoreline retreats to a point where interactions between the MN and GF are possible. The groins are buried (the shoreline extends past the groin tips) for all situations examined within a 25-year period. Within this timeframe, the MN-GF simulations produce the same shoreline as MN-only simulations of the same PDF. It takes about 50 years for U = 0 simulations and over 75 years for U = 0.4 simulations to diffuse the MN to a point where the shoreline retreats past the groin tips. Even when the shoreline does retreat past the groin tips, the GF has negligible effects on shoreline evolution since the distance between the shoreline and groin tips remains very small, leading to a great amount of bypassing. This results in a shoreline identical to that of MN-only.

The effects of varying groin permeability P are tested using A = 0.7, U = 0.4 as shoreline morphological effects (*e.g.*, groin effects, updrift/downdrift accretion/erosion)

are very prevalent under those conditions. *P* values ranging from 0.0 to 0.3 are examined (see Section 5.2.4). In general, the lower the *P* value, the larger the magnitude of the groin effects. Increasing *P* also reduces the amount of sediment accumulation updrift of a GF as well as decreases the amount of erosion downdrift of a GF.

5.4. Discussion

Model results show that a MN can feed a long stretch of coastline, even when LST is inhibited by an impermeable GF ($P = 0$) for a MN:GF area ratio of ~8:1. This suggests that processes such as bypassing may play an important role in how a coupled MN-GF beach may evolve. Other key factors affecting shoreline evolution of this system include the relative position of the MN and GF (MN updrift, downdrift, or on top of the GF; quantified here by *A*) and the probability of offshore high-angle waves (*U*), which controls the diffusion (and thus feeding) rate of the MN.

CEM results show that the erosion downdrift of a MN can arise in a few years due to wave shadowing and sediment starvation. The placement of a GF downdrift of a MN does not mitigate this effect. However, if a GF is updrift of a MN, LST inhibition from the GF creates sediment buildup between the MN and GF resulting in no shoreline retreat and greater longshore progradation than for MN-only or GF-only scenarios. Note that this shoreline retreat is also mitigated as the MN diffuses, even if a GF is not present. This effect does not occur in GenCade as the version used here does not implement a wave shadowing algorithm.

One of the goals here is to determine the conditions under which adding a MN maximizes shoreline extension in an erosional hotspot downdrift of a GF. This is evaluated at a 25-year time scale. The largest erosion signals of the models evaluated occur under wave climates with high asymmetry (A = 0.3 or 0.7) and high U (0.4). For comparison, the wave climate at Galveston Island is A = 0.38, U = 0.24 while the wave climate at the Sand Engine is A = 0.62, U = 0.34 (Wijnberg, 1995; 2002). The largest difference between the MN-GF shoreline and GF-only in the erosion hotspot occurs in simulations with MN on top of the GF with an asymmetric wave climate (A = 0.3 or 0.7; these are mirror images) and high U (0.4). GF-only simulations under these wave conditions show ~92 m of shoreline retreat in the erosion hotspot, while MN-GF shows 243 m of shoreline extension in the same area. The second largest difference occurs when the MN is built adjacent to the GF with a wave PDF of A = 0.7, U = 0.4. Here, GF-only has the same shoreline retreat as above while MN-GF has ~219 m of shoreline extension in the erosional hotspot area.

Several model limitations should be considered when evaluating their results. Neither model directly computes cross-shore transport processes (*e.g.*, aeolian transport, nearshore circulation, rip currents, swash zone dynamics, *etc.*), some of which have been shown to transport material from the beach to the dunes at the Sand Engine MN (Hoonhout and de Vries, 2017; Roest, 2017), and have been shown to transport sediment offshore in GFs (Basco and Pope, 2004). Morton *et al.* (1994) point out that cross-shore sediment transport offshore during high-energy events is often followed by recovery on longer time scales (>5 years), but this limitation should be considered when evaluating

the model results. GenCade also suffers from the limitation of instability when shorelines extend past groin tips and dependence on initial wave conditions when a PDF is used. The latter should be alleviated with the use of a historical wave time series as opposed to a stochastic PDF.

Future research on MN-GF interactions should investigate how geometric parameters of the MN (*e.g.*, initial volume, cross-shore extent, longshore footprint, *etc.*) affect MN-GF interactions. The same can be done with groin field parameters such as the number of groins, groin spacing, cross-shore length of groins, *etc.* The investigation of temporal variations in wave climate affecting these interactions is also recommended as CEM model results of Galveston Island suggest that temporal variation in wave climate may be a contributing factor to the U-shaped coastlines in the groin compartments. It is also worth investigating if MN diffusion remains symmetric under a time varying symmetric wave climate where there is dominant direction in one half of the year and an opposing dominant direction the other half of the year, as is the case with bimodal wave climates. A detailed comparison of one-line model results of a coupled MN-GF system with 3D model results may also prove useful but is outside the scope of this work. Furthermore, the incorporation of additional processes in these one-line models such as cross-shore transport processes and offshore currents (suggested by Valsamidis *et al.*, 2017 to be an important factor in MN evolution) may improve the accuracy of these simulations. Also, modeling can be performed of hypothetical MNs built at specific coastal sites with GFs (*e.g.*, Galveston Island and Miami Beach) to examine the feasibility of mega-nourishment at these sites.

5.5. Conclusions

Mega-nourishment (MN) is an innovative solution for coastal risk reduction where a large volume of sediment is placed at a single site. This sediment is redistributed via natural processes, feeding adjacent beaches. As many beaches that could benefit from a MN are reinforced with a groin field (GF), one-line modeling efforts using the Coastline Evolution Model (CEM) and GENESIS + Cascade (GenCade) have been undertaken to identify the implications of combining these two features. For the presented simulations, the initial beach area of the MN relative to the area of the GF is ~8:1 in all models. Other ratios and geometric arrangements are of course possible, and the current simulations should be considered a starting point of such investigations where scales were based on the Dutch Sand Engine (MN) and Galveston Island, Texas (GF). The diffusion (and thus feeding) rate of the model MN is primarily dependent on the probability of high-angle ($> |45^o|$ relative to shore normal) incident waves (U) in the wave climate with higher probabilities yielding slower diffusion rates. The MN remains diffusive for $U \geq 0.4$. The addition of a MN adjacent to a GF results in shoreline extension in both the nourished region and the GF on multidecadal time scales. Even though GFs inhibit longshore sediment transport, MNs in a diffusive wave climate are capable of feeding beaches on the opposite side of a GF through processes such as bypassing, even with impermeable groins. In the case of an MN updrift of a GF, this feeding may take the form of a reduced erosion signal downdrift of the GF. Multidecadal model results also show that once a MN shoreline diffuses past the groin

tips, the groins have no significant effect on shoreline evolution given the large amount of bypassing. Under high *U*, a modeled erosional hotspot downdrift of a GF remains present when a MN is updrift from the GF up to 25 years. However, the amount of shoreline retreat in this area is diminished relative to a GF alone and can result in net shoreline extension on very large time scales (75 years).

From a design perspective, the placement of the MN relative to the GF is critical in determining evolution of the shoreline, especially in the short term (less than a decade). This is particularly true in asymmetric wave climates ($A \neq 0.5$) where a dominant direction of littoral drift can lead to erosional hotspots downdrift of the GF. Results indicate that placing a MN downdrift of a GF could be a method of mitigating these erosional hotspots on top of all the other benefits offered by MNs. The largest erosive signals arise in CEM when *A* and *U* are high, and thus the largest amount of shoreline extension arises when a MN is built on top of a GF under those conditions. When that is the case, no shoreline retreat is present on long time scales, and the MN diffuses without interaction with the GF. This particular design appears to maximize the amount of shoreline extension resulting from the nourishment diffusion while minimizing any erosive effects generated by LST inhibition from the GF. The erosive hotspot downdrift of a GF can also be mitigated to a lesser degree when a MN is built directly downdrift of a GF under high *A* and *U*. Here, sediment diffusing updrift from the MN is trapped by the downdrift terminal groin, thus filling in the erosional hot spot.

6. GENERAL DISCUSSION AND CONCLUSIONS

Mega-nourishment (MN) is a pioneering method of coastal defense against flooding and mitigation of shoreline erosion. Through MN, a large volume of sediment is placed at a single nourishment site, and natural forces redistribute the sediment along a large stretch of beach thus feeding nearby beaches over time. To date, only one MN has been constructed, the Delftland Sand Motor (SM). Therefore, physical data on MN evolution is very limited, and numerical modeling efforts are necessary to gain a better understanding of how MNs evolve. One-line modeling is a robust method of modeling shoreline evolution and is often the preferred method in engineering applications as it does not have large data or processing requirements. The dissertation's overall goals are to examine the feasibility of using the Coastline Evolution Model (CEM) and GENESIS + CasCade (GenCade) one-line models to explore MN evolution, and to use these models to explore the implications of combining a MN on a coastline with a groin field (GF).

Limited amounts of one-line modeling have been undertaken related to MNs, and thus the performance of CEM and GenCade in simulating MNs is not well understood. To explore these one-line modeling methods for use with MN evolution, modifications are made to the C version (Appendix A) of CEM. These modifications allow for all the topographic variables in the CERC sediment transport equation to be user-specified. They also allow the model to run at a much higher resolution (25 m cell size as opposed to 100+ m cell size) than previous versions. Furthermore, the modifications allow for a

higher resolution wave probability distribution function (PDF) to be used as the wave forcing. The original version used a 4-bin PDF by default, and the modifications allow for a 12-bin PDF to be used.

To assess the feasibility of using CEM and GenCade in MN simulations, GenCade and the modified version of CEM (C version) are used to simulate 3.5 years of evolution of the Sand Motor (SM). The results of these simulations are compared with measured shorelines of the physical SM from March 2013 to September 2016) (3.5 years). The following research questions (RQs) are addressed during this first project:

> RQ1. Much one-line modeling has been performed with traditional nourishments and natural coastlines, but relatively little one-line modeling has been done with MNs. Can MN evolution be reasonably simulated using one-line models (specifically CEM and GenCade)?

It is hypothesized that GenCade and the updated version of CEM are capable of reasonably reproducing observed MN evolution given proper calibration. Results show that given proper parameterization both models can reproduce diffusion rates similar to that observed at the Sand Motor. At best, either model can show a 3.5-year tip retreat within 1 to 5 m of the 171 m trip retreat observed. Modeled mean MN flank slopes are less than 5.0 degrees shallower on left side for both models. Modeled mean flank slopes on the right side are less than 2.0 degrees steeper for CEM and less than 6.0 degrees

shallower for GenCade from that observed after 3.5 years of evolution.

RQ2. The *K* value in the CERC Transport Equation is used to relate available wave power to longshore sediment fluxes. Are commonly used *K* values given in the literature adequate for MN simulation in a one-line model (specifically CEM and GenCade)?

While several methods presented in the literature derive *K* from field data to determine a relationship between immersed weight transport and wave power, Rosati *et al*. (2002) use a linear regression over a very large data set of field observations to obtain the value of $K = 0.92$, which is one of the more recent determinations. It is hypothesized here that this value is adequate to simulate a MN. Model results show that the *K* value affects the rate of MN diffusion with higher *K* values leading to faster diffusion. Of the *K* values commonly used in modeling traditional nourishments and natural coastlines, $K = 0.92$ from Rosati *et al.* (2002) produces the lowest mean measured-modeled RMS differences. However, traditional nourishments are of much smaller scale than MNs. MNs have different coastal dynamics since a very large feature is rapidly introduced onto the coast, which can cause instability to the system. As such, models show that higher-than-usual *K* value may be a better fit for larger-scale nourishments. Artificially increased *K* values can lower measured-modeled root mean square (RMS) differences, though no single value maximizes model performance across all metrics. $K = 1.20$ in

CEM and 1.60 in GenCade minimize RMS differences across the entire domain while *K* = 1.10 in CEM and 2.00 in GenCade minimizes measured-modeled differences in MN tip retreat. It is possible that the ideal *K* value may vary over time as the MN diffuses, though there is no clear trend based on the modeled results at this point. In evaluating model performance, GenCade produces average measured-modeled RMS differences on the order of 50 m. This makes GenCade a useful tool in predicting absolute coastline position.

RQ3. What are the major differences between the shorelines produced by CEM compared to those produced by GenCade when modeling MN evolution?

CEM's wave transformation technique includes a wave shadowing algorithm while GenCade's default wave transformation (used here) does not. It is hypothesized that these wave shadowing algorithms in CEM can lead to areas of sediment starvation that result in asymmetric evolution of a modeled MN under an asymmetric wave climate (as observed at the SM). Model results show that CEM can capture longshore migration of the MN tip (observed to be ~400 m downdrift), indicating that wave shadowing (present in CEM) may be an important component of capturing longshore migration of shoreline features. Note that GenCade (without the wave shadowing algorithms) does not capture this tip migration. CEM also shows asymmetric diffusion of the feature (observed at the SM, though to a lesser extent than modeled) while GenCade produces symmetric diffusion of the MN.

Many beaches that could benefit from a MN are reinforced with groin fields (GFs). It is therefore important to understand how a MN will evolve on a shore with groins. However, no MNs have been constructed in the vicinity of a GF. Therefore, modeling must be undertaken to determine these interactions. Whitley *et al*. (2021) showed that CEM and GenCade are suitable models for MN evolution simulation. However, prior versions of CEM did not have the ability to simulate groins. In order to rectify this, CEM is reprogrammed in MATLAB (Appendix B) and robust groin simulating algorithms are added so that CEM and GenCade can be used to explore the interactions between a MN and a GF. These new CEM groin algorithms are validated by simulating two years (2014 to 2016) of evolution of the Galveston Island GF and comparing them to measured coastlines of the same dates. CEM results are also compared to a well-vetted model, GenCade, as it is the current U.S. industry standard. In general, both models produce similar sawtooth coastline shapes containing groin effects in the groin field. CEM tends to underpredict the low points in the groin field shorelines but can predict the high points under 30 m away from those observed. GenCade greatly overpredicts the high points (30 to 50 m more than those observed). Both CEM and GenCade produced measured-modeled mean RMSDs within the groin compartment of similar order (~12 to 16 m on average), with CEM producing a lower RMSD than GenCade in general. This shows that CEM should be able to simulate groins at least as well as the U.S. industry standard.

The second project investigates the shoreline interactions between a MN and a groin field. CEM and GenCade are used to simulate a MN either adjacent to or on top of

a seven-groin GF on an otherwise straight coastline. The initial MN to GF beach area is ~8:1. During the course of this project, the following research questions (RQs) are addressed:

> RQ4. Many coastlines that would benefit from a MN have been reinforced by groin fields. However, MNs are designed to feed adjacent beaches via LST while groins are designed to inhibit LST. What are the implications of combining these two features on the same coastal system?

It is expected that as a MN diffuses, a nearby groin field will inhibit LST across it, thus reducing the diffusion rate of the MN. However, with such a large supply of sediment added from the MN, it is hypothesized that beaches on the side of the groin field opposite the MN will be fed as there are a number of mechanisms for sediment to be transported past the groins, specifically bypassing and through-passing (due to groin permeability). The modeled MNs built adjacent to a groin field are capable of feeding beaches within the groin field and on the far side of the groin field through processes such as bypassing even with impermeable groins. The addition of a groin field adjacent to a modeled MN can affect its shoreline evolution through sediment accumulation updrift of the groin field and/or with a diminished shoreline extent (or erosion) downdrift of the field.

RQ5. Different coasts around the world can experience vastly different wave climates. What effects do wave climate (specifically, the probability distribution of offshore wave angles) have on shoreline evolution in a coupled MN-groin field system?

An asymmetric wave climate should be accompanied by a dominant direction of LST. When a MN is placed updrift of a groin field, it is expected that this should result in augmented accumulation of sediment updrift of the groin field. Under normal circumstances (with no MN), this should be accompanied by sediment starvation downdrift of the field. However, with such a large sediment supply from the MN, it is hypothesized that the overall rate of erosion updrift of the groin field will be diminished from that without a MN present. Consequently, if a MN is placed downdrift of a groin field, it is still expected that the MN should diffuse in both directions longshore. This should significantly reduce the erosion signal normally present on the downdrift side of the groin field. Furthermore, it is hypothesized that a higher probability of high-angle ($|\phi_0| > 45^0$) incident waves in the wave climate will reduce the rate of MN diffusion. To test this hypothesis, various offshore wave climates are examined while adjusting the probability of a positive versus negative wave angle (asymmetry or *A*) and of a high versus low wave angle (highness or *U*). Model results also show that MN diffusion rates are primarily dependent on *U* with higher *U* values slowing MN diffusion. Note that $U \geq 0.4$ here so that the wave climate remains diffusive as higher values can lead to anti-diffusion of the MN or model instability. On multidecadal time scales, results show that

once a shoreline extends past the groin tips, the groins have no significant effect on shoreline evolution due to the large amount of bypassing.

RQ6. Beaches with groin fields where a dominant direction of littoral drift is present often suffer from an erosional hotspot downdrift of the groin field due to sediment starvation. Can this erosion be mitigated through mega-nourishment, and if so, under what conditions is the mitigation maximized?

It is hypothesized that building the mega-nourishment downdrift of the GF will maximize the mitigation of the erosion as the MN will provide a new source of sediment for the erosional hotspot. When a MN is built updrift of a GF under high U, model results show an erosional hotspot developing downdrift of the GF after 1 to 3 years of evolution. However, this area does experience net shoreline extension on very large time scales (75 years). When a MN is built downdrift of a GF, the erosional hotspot does not develop as there is an influx of sediment in this area from the diffusion MN. This suggests this MN placement could be a method of mitigating erosional hotspots that develop downdrift of a GF when a dominant direction of littoral drift is present. The greatest amount of shoreline extension in the modeled erosional hotspots occur under highly asymmetric wave climates with high *U* values and the MN is built on top of the GF. Under these conditions, the GF has no effect on shoreline evolution as the groin tips are either buried or very close to the shoreline where large amounts of bypassing can occur. As such, no shoreline retreat occurs, and the MN diffuses as if no GF is present.

Despite having seemingly opposite functions (MNs aim to feed beaches through LST while GFs aim to anchor beaches through LST inhibition), much may be gained by combining MNs and GFs. Model results indicate MNs are indeed capable of feeding beaches on the far side of GFs despite sediment inhibition. Furthermore, the models show that combining a MN on a coast reinforced with a GF may reduce or eliminate erosional features generated due to the presence of a GF. However, there are several factors to consider when planning to build a MN on a coastline reinforced with groins. For the MN to effectively feed nearby beaches, the offshore wave climate must remain diffusive ($U \leq 0.5$) as model results show anti-diffusive wave climates resulting in the MN's cross-shore length growing, not diffusing. Furthermore, model results show that the diffusion rate of the MN is dependent on U with higher values slowing MN tip retreat rates. Therefore, it is recommended that project planners consider a study site's offshore wave climate when considering MN construction.

It is recommended that future research efforts compare CEM and GenCade MN evolution with those produced by other one-line models such as LITPACK (DHI, 2009) or UNIBEST (Deltares, 2011). CEM accuracy may be enhanced by the inclusion of additional features that may improve its modeling capabilities. The simulation of cross-shore processes such as aeolian transport (likely important in MN modeling) and rip currents (likely important in GF modeling) should improve CEM accuracy. Valsamidis *et al*. (2017) suggest that the addition of offshore advection to modeling efforts may be important in MN simulation. Given the CEM's sensitivity to wave climate, the use of a more robust wave transformation technique such as SWAN (*e.g.*, Limber *et al.*, 2016) or

STWAVE (*i.e.*, Smith *et al.*, 2001) should improve CEM's accuracy. Furthermore, the addition of sediment sources and sinks are likely important when modeling areas with known sediment sources and sinks, such as Galveston Island (*e.g.*, Ratliff, *et al*. 2018). GenCade accuracy in MN simulation can also be improved with the use of a more advanced wave transformation model such as CMS-Wave (Connell and Permenter, 2013). The use of an offshore contour, the addition of sources and sinks, and the inclusion of cross-shore processes could also improve model accuracy.

Additional research on MN-GF interactions could include the investigation of the impacts of MN geometric parameters (such as initial volume, longshore footprint, *etc*.) on shore evolution. An investigation of the effects of geometric parameters of the groin field (groin length, groin spacing, *etc*.) on shoreline evolution would also prove useful. Furthermore, as simulations of the Galveston Island groin field in Chapter 5 suggest that temporal variation in offshore wave climate may be a contributing factor to coastline shape, a robust investigation into the effects of wave climate temporal variability is recommended.

REFERENCES

Aquaveo, 2017. *SMS User Manual (v12.1): The Surface Water Modeling System*. Aquaveo, LLC, Provo, Utah. 1303p.

Arriaga, J., Rutten, J., Ribas, F., Falqués, A., and Ruessink, G., 2017. Modeling the long-term diffusion and feeding capability of a mega-nourishment. *Coastal Engineering*, 121: 1-13.

Arriaga, J., Ribas, F., Falqués, A., Rutten, J. and Ruessink, G., 2020. Long-term performance of mega-nourishments: Role of directional wave climate and initial geometry. *Journal of Marine Science and Engineering*, *8*(12), p.965.

Ashton, A.D. and Murray, A.B., 2006a. High-angle wave instability and emergent shoreline shapes: 1. Modeling of sand waves, flying spits, and capes. *Journal of Geophysical Research*, 111. doi:10.1029/2005JF000422

Ashton, A.D. and Murray, A.B., 2006b. High-angle wave instability and emergent shoreline shapes: 2. Wave climate analysis and comparisons to nature. *Journal of Geophysical Research*, 111. doi:10.1029/2005JF000423

Ashton, A., Murray, A.B., and Arnault, O., 2001. Formation of coastline features by large-scale instabilities induced by high-angle waves. *Nature*, 414.

Atkins, 2014. Wading depth/depth of closure survey, [Galveston Island] 2014.

Atkins, 2015. Wading depth/depth of closure survey, [Galveston Island] 2015.

Atkins, 2016. Wading depth/depth of closure survey, [Galveston Island] 2016.

Atkins, 2017. Wading depth/depth of closure survey, [Galveston Island] 2017.

Atkins, 2018. *Post-storm Data Collection Surveys, Hurricane and Tropical Storm Harvey, Galveston Island, Texas.* Technical Report for the City of Galveston.

Aptim Environmental & Infrastructure, LLC, 2018. 2018 Galveston Island Topographic and Hydrographic Survey.

Aptim Environmental & Infrastructure, LLC, 2019. 2019 Galveston Island Topographic and Hydrographic Survey.

Aptim Environmental & Infrastructure, LLC, 2019. 2020 Galveston Island Topographic and Hydrographic Survey.

Bakker, W.T., 1969. The dynamics of a coast with a groyne system. *Proceedings of the 11th Coastal Engineering Conference* (New York, N.Y., ASCE), pp. 492-517.

Bakker, W.T., Klein-Breteler, E.H.J. & Roos, A., 1971. The dynamics of a coast with a groyne system, *Proceedings of the 12th Coast Engineering Conference* (New York, New York, ASCE), pp. 1001-1020.

Basco, D.R, 2002. Shore Protection Projects. In King, D.B. (Ed.), *Coastal Engineering Manual* (pp. V-3-1 – V-3-110). Washington, D.C.: U.S. Army Corps of Engineers.

Basco, D.R. and Pope, J., 2004. Groin functional design guidance from the coastal engineering manual. *Journal of Coastal Research*, pp.121-130.

Battjes, J.A. and Stive, M.J.F., 1985. Calibration and verification of a dissipation model for random breaking waves. *Journal of Geophysical Research*, 90, 9159-9167. doi:10.1029/JC090iC05p09159

Bird, E.C.F., 1985. *Coastline Changes. A Global Review*. Chichester, United Kingdom: John Wile-Interscience, 219 pp.

Bodge, K. R., & Kraus, N. C. (1991). Critical examination of longshore transport rate magnitude. In *Coastal Sediments* (pp. 139-155). ASCE.

Bridges, T.S., Wagner, P.W., Burks-Copes, K.A., Bates, M.E., Collier, Z.A., Fischenich, C.J., Gailani, J.Z., Keuck, L.D., Piercy, C.D., Rosati, J.D., Russo, E.J., Shafer, D.J., Suedel, B.C., Vuxton, E.A., and Wamsley, T.V., 2015. *Use of Natural and Nature-Based Features (NNBF) for Coastal Resilience*. Vicksburg, Mississippi: U.S. Army Corps of Engineers, Engineer Research and Development Center, 480p.

Brown, A.C. and McLachlan, A., 2002. Sandy shore ecosystems and the threats facing them: some predictions for the year 2025. *Environmental Conservation*, 62-77.

Carslaw, H.S. and Jaeger, J.C., 1959. *Conduction of heat in solids*. Oxford: Clarendon Press, 520p.

Charlier, R.H. and De Meyer, C.P., 1995. Beach nourishment as efficient coastal protection. *Environmental Management and Health*, 6(5), 26–34. doi:10.1108/09566169510096511

Connell, K.J. and Permenter, R., 2013. *Wave data processing and analysis tools for developing wave boundary forcing for CMS-Wave and GenCade numerical models: Part 1* (No. ERDC/CHL-CHETN-IV-97). Engineer Research and Development Center, Vicksburg, MS, Coastal and Hydraulics Lab.

Cooke, B.C., Jones, A.R., Goodwin, I.D. and Bishop, M.J., 2012. Nourishment practices on Australian sandy beaches: A review. *Journal of Environmental Management*, *113*, pp.319-327.

Crank, J., 1956. *The Mathematics of Diffusion*, Oxford: Clarendon Press, 347p.

Dabees, M. and Kamphuis, J.W., 1999. ONELINE, a numerical model for shoreline change. *Coastal Engineering 1998*, 2668-2681.

Davison, A.T., Nicholls, R.J. and Leatherman, S.P., 1992. Beach nourishment as a coastal management tool: an annotated bibliography on developments associated with the artificial nourishment of beaches. *Journal of Coastal Research*, 984-1022.

De Schipper, M.A., de Vries, S., Stive, M., de Zeeuw, R., Rutten, J., Ruessink, G., Aarninkhof, S., and van Gelder-Maas, C., 2014. Morphological development of a mega-nourishment; first observations at the Sand Motor. *Coastal Engineering Proceedings*, 1(34), 1-6.

De Schipper, M.A., de Vries, S., Ruessink, G., de Zeeuw, R.C., Rutten, J., van Gelder-Maas, C., and Stive, M.J.F., 2016. Initial spreading of a mega feeder nourishment: Observations of the Sand Motor pilot project. *Coastal Engineering*, 111, 23–38. doi:10.1016/j.coastaleng.2015.10.011

De Vries, S., de Schipper, M. A., and Stive, M., 2014. Measured gradients in alongshore sediment transport along the Dutch coast. *Coastal Engineering Proceedings*, 1(34), 43-58.

De Zeeuw, R., de Schipper, M., and de Vries, S., 2017. *Sand Motor Topographic Survey, actual surveyed path*. TU Delft, Dataset. doi:10.4121/uuid:3836e5a5-4fdf-4122-84bd-a9bb679fb84c

Dean, R.G. and Dalrymple, R.A., 2004. *Coastal Processes with Engineering Applications*. New York, NY: Cambridge University Press.

Del Valle, R., Medina, R., and Losada, M.A., 1993. Dependence of coefficient K on grain size, Technical Note No. 3062, *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 119(5), 568-574

Dellapenna, T.M., Allison, M.A., Gill, G.A., Lehman, R.D. and Warnken, K.W., 2006. The impact of shrimp trawling and associated sediment resuspension in mud dominated, shallow estuaries. *Estuarine, Coastal and Shelf Science*, *69*(3-4), pp.519-530.

Deltares, 2011. *UNIBEST-CL+ Manual: Manual for Version 7.1 of the Shoreline Model UNIBEST-CL+*.

DHI [Delft Hydraulics Institute], 2009. *LITPACK: An Integrated Modeling System for Littoral Processes and Coastline Kinetics* (Short Introduction and Tutorial). Published by MIKE by DHI

Duo, E., Casari, A. and Ciavola, P., 2015. Feasibility long-term numerical study of a mega-nourishment in the Northern Adriatic. Coastal and Maritime Mediterranean Conference, Edition 3.

Elko, N., Briggs, T.R., Benedet, L., Robertson, W., Thomson, G., Webb, B.M., Garvey, K., 2021. A Century of U.S. Beach Nourishment, 2021. Ocean & Coastal Management, 199(2021) 105406, ISSN 0964-5691, https://doi.org/10.1016/j.ocecoaman.2020.105406

Ells, K. and Murray, A.B., 2012. Long-term, non-local coastline responses to local shoreline stabilization. *Geophysical Research Letters*, 39(18), L19401. doi:10.1029/2012GL052627

Falqués, A., 2003. On the diffusivity in coastline dynamics. *Geophysical Research Letters*, 30(21). doi:10.1029/2003GL017760

Falqués, A., Calvete, D., 2005. Large-scale dynamics of sandy coastlines: Diffusivity and instability: dynamics of sandy coastlines. *Journal of Geophysical Research*, 110. doi:/10.1029/2004JC002587

Frey, A.E., Connell, K.J., Hanson, H., Larson, M., Thomas, R.C., Munger, S., and Zundel, A., 2012. *GenCade Version 1 Model Theory and User's Guide* (Technical Report ERDC/CHL-TR-12-25). US Army Corps of Engineers Engineer Research and Development Center, Vicksburg, MS, 187p.

Galgano, F. A., Jr., 2004. Long-Term Effectiveness of a Groin and Beach Fill System: A Case Study Using Shoreline Change Maps. *Journal of Coastal Research*, (33), 17.

Goda, Y., Takayama, T. and Suzuki, Y., 1978. Diffraction diagrams for directional random waves. In *Coastal Engineering 1978*, pp. 628-650.

Goda, Y., 2000. *Random Seas and Design of Maritime Structures, 2nd Edition*. New Jersey: World Scientific, 464p.

Goff, J.A., Allison, M.A. and Gulick, S.P., 2010. Offshore transport of sediment during cyclonic storms: Hurricane Ike (2008), Texas Gulf Coast, USA. *Geology*, *38*(4), pp.351-354.

Google Earth, 2022. Imagery of Texas and Galveston Island.

Gornitz, V.M., Daniels, R.C., White, T.W., and Birdwell, K.R., 1994. The Development of a Coastal Risk Assessment Database: Vulnerability to Sea-Level Rise in the U.S. Southeast. *In*: Fink, C.W. (ed.), *Coastal Hazards: Perception, Susceptibility, and Mitigation, Journal of Coastal Research*, Special Issue No. 12, pp. 327–338.

Gravens, M.B., Kraus, N.C., and Hanson, H., 1991. *GENESIS: Generalized model for simulating shoreline change* (Technical Report No. CERC-89-19). Coastal and Hydraulics Laboratory, Vicksburg, MS: US Army Engineer Research and Development Center, 431p.

Gravens, M.B., Ebersole, B.A., Walton, T.L., and Wise, R.A., 2002. Beach Fill Design, *In*: King, D.B. (ed.), *Coastal Engineering Manual*. U.S. Army Corps of Engineers, Washington, D.C., pp. V-4-i – V-4-109.

Griggs, G.B., Tait, J.F., and Corona, W., 1994. The interaction of seawalls and beaches: Seven years of monitoring. *Shore & Beach*, *62*(3), pp.32-38.

Grijm, W., 1961. Theoretical forms of shoreline. *Proceedings of the 7th Coastal Engineering Conference* (New York, New York, ASCE), pp. 219–235.

Hanson, H. and Kraus, N.C., 1989. *GENESIS: Generalized Model for Simulating Shoreline Change. Report 1. Technical Reference* (No. CERC-TR-89-19-1). Coastal Engineering Research Center, Vicksburg, Mississippi, 185p.

Harter, C. and Figlus, J., 2017. Numerical modeling of the morphodynamic response of a low-lying barrier island beach and foredune system inundated during Hurricane Ike using XBeach and CSHORE. *Coastal Engineering*, *120*, pp.64-74.

Hoonhout, B. and de Vries, S., 2017. Field measurements on spatial variations in Aeolian sediment availability at the Sand Motor mega nourishment. *Aeolian Research*, *24*, pp. 93-104.

Kamphuis, J.W., 2010. *Introduction to coastal engineering and management* (Vol. 48). World Scientific.

Komar, P.D., 1983. *CRC Handbook of Coastal Processes and Erosion*. CRC Press, Incorporated.

Komar, P.D. and Inman, D.L., 1970. Longshore sand transport on beaches. *Journal of Geophysical Research*, 75(30), 5914–5927.

Kraus, N.C., 1984. Estimate of breaking wave height behind structures. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, *110*(2), 276-282.

Kraus, N.C, Hanson, H., and Blomgren, S.H., 1994. Modern Functional Design of Groin System, Proceedings, 24th International Conference on Coastal Engineering, ASCE, NY, 1327-1342.

Kroon, A., de Schipper, M., van Gelder, P. and Aarninkhof, S., 2020. Ranking uncertainty: Wave climate variability versus model uncertainty in probabilistic assessment of coastline change. *Coastal Engineering*, 103673.

Larson, M., Hanson, H., and Kraus, N.C., 1987. *Analytical solutions of the one-line model of shoreline change.* Technical Report CERC-87-15, USAE-WES, Coastal Engineering Research Center.

Larson, M., Hanson, H., and Kraus, N.C., 1997. Analytical solutions of one-line model for shoreline change near coastal structures. *Journal of Waterway Port Coastal and Ocean Engineering*, 123, 180-191.

Larson, M., Kraus, N.C., and Hanson, H., 2003. Simulation of regional longshore sediment transport and coastal evolution - the "Cascade" model. *Coastal Engineering 2002*. 2612–2624. doi:10.1142/9789812791306_0218

Leatherman, S.P., 1988. Beach response strategies to accelerated sea-level rise. Proceedings 2nd North American Conference in Preparing for Climate Change. The Climate Institute, Washington, D.C., 353-358.

LeMéhauté, B. and Brebner, A., 1960. *An Introduction to the Mathematical Theories of Two-dimensional Periodic Progressive Gravity Waves*. Civil Engineering Department, Queen's University.

LeMéhauté, B. and Soldate, M., 1977. *Mathematical Modeling of Shoreline Evolution.* Miscellaneous Report 77-10. US Army Corps of Engineers, Fort Belvoir, Coastal Engineering Research Center, 56p.

Lesser, G.R., Roelvink, J.V., Van Kester, J.A.T.M. and Stelling, G.S., 2004. Development and validation of a three-dimensional morphological model. *Coastal Engineering*, *51*(8-9), 883-915.

Limber, P.W., Adams, P.N. and Murray, A.B., 2016. Modeling large-scale shoreline change caused by complex bathymetry in low-angle wave climates. *Marine Geology*, *383*, 55-64.

Lisle, J.T. and Comer, N.N., 2011. *Characterization of Sediments from the Gulf of Mexico and Atlantic Shorelines, Texas to Florida*. USGS Open File Report 2011-1199, 82p.

López-Ruiz, A., Ortega-Sánchez, M., Baquerizo, A. and Losada, M.Á., 2014. A note on alongshore sediment transport on weakly curvilinear coasts and its implications. *Coastal Engineering*, *88*, 143-153.

Luijendijk, A., Hagenaars, G., Ranasinghe, R., Baart, F., Donchyts, G. and Aarninkhof, S., 2018. The state of the world's beaches. *Scientific Reports*, *8*(1), 1-11.

Luijendijk, A.P., Ranasinghe, R., de Schipper, M.A., Huisman, B.A., Swinkels, C.M., Walstra, D.J. and Stive, M.J., 2017. The initial morphological response of the Sand Motor: A process-based modelling study. *Coastal Engineering*, *119*, 1-14.

Mentaschi, L., Vousdoukas, M.I., Pekel, J.F., Voukouvalas, E. and Feyen, L., 2018. Global long-term observations of coastal erosion and accretion. *Scientific Reports*, *8*(1), 1-11.

Mil-Homens, J., Ranasinghe, R., de Vries, J.V.T. and Stive, M.J.F., 2013. Re-evaluation and improvement of three commonly used bulk longshore sediment transport formulas. *Coastal Engineering*, *75*, 29-39.

Moore, L.J., McNamara, D.E., Murray, A.B., and Brenner, O., 2013. Observed changes in hurricane-driven waves explain the dynamics of modern cuspate shorelines. *Geophysical Research Letters,* 40(22), 5867–5871. doi:10.1002/2013GL057311

Morton, R.A., Paine, J.G., and Gibeaut, J.C., 1994. Stages and durations of post-storm beach recovery, southeastern Texas coast, USA. *Journal of Coastal Research*, 10(4), 884–908.

Mulder, J. P., and Tonnon, P. K., 2011. "Sand Engine": Background and design of a mega-nourishment pilot in the Netherlands. *Coastal Engineering Proceedings*, 1(32).

NOAA [National Oceanic and Atmospheric Administration], 2006. *Galveston, Texas Coastal Digital Elevation Model.* National Geophysical Data Center, NESDIS, NOAA, U.S. Department of Commerce, Boulder, Colorado. http://www.ngdc.noaa.gov/dem/squareCellGrid/download/403.

Pelnard-Considère, R., 1956. *Essai de theorie de 1'evolution des forms de rivages en plaged e sablee t de galets*. *4th Journees de l' Hydraulique, les Energiesd e la Mer*, Question III, v. Question III, Rapport No. 1, 289-298.

Peterson, C.H. and Bishop, M.J., 2005. Assessing the environmental impacts of beach nourishment. *BioScience*, 55(10), 887-896. doi:10.1641/0006-3568(2005)055[0887:ATEIOB]2.0.CO;2

Pilkey, O.H. and Clayton, T.D., 1989. Summary of beach replenishment experience on US East Coast barrier islands. *Journal of Coastal Research*, pp.147-159.

Ratliff, K.M., Hutton, E.H. and Murray, A.B., 2018. Exploring wave and sea-level rise effects on delta morphodynamics with a coupled river-ocean model. *Journal of Geophysical Research: Earth Surface*, *123*(11), pp.2887-2900.

Reeve, D.E., 2006. Explicit Expression for Beach Response to Non-Stationary Forcing near a Groyne. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 132, 125-132. doi:10.1061/(ASCE)0733-950X(2006)132:2(125)

Reeve, D.E. and Valsamidis, A., 2014. Analysis of the stability properties of a class of beach evolution models, *Coastal Engineering*, 91, 76-83.

Roest, B., 2017. The influence of the Sand Motor on the sediment transports and budgets of the Delfland coastal cell. Delft, the Netherlands: Delft University of Technology, Master's thesis, 140p.

Rosati, J.D., Walton, T.L., and Bodge, K., 2002. Longshore sediment transport, *In*: King, D.B. (ed.), *Coastal Engineering Manual*. U.S. Army Corps of Engineers, Washington, D.C., pp. III-2-1 – III-2-113.

Slott, J.M., Murray, A.B., Ashton, A.D., and Crowley, T.J., 2006. Coastline responses to changing storm patterns. *Geophysical Research Letters,* 33, L18404. doi:10.1029/2006GL027445

Slott, J.M., Murray, A.B., and Ashton, A.D., 2010. Large-scale responses of complex-shaped coastlines to local shoreline stabilization and climate change. *Journal of Geophysical Research,* 115, F03033. doi:10.1029/2009JF001486

Small, C. and Nicholls, R.J., 2003. A Global Analysis of Human Settlement in Coastal Zones. *Journal of Coastal Research,* 19, 584–599.

Smith, E.R., Wang, P., Ebersole, B.A. and Zhang, J., 2009. Dependence of total longshore sediment transport rates on incident wave parameters and breaker type. *Journal of Coastal Research, 25*, pp.675-683.

Smith, J.M., Sherlock, A.R. and Resio, D.T., 2001. *STWAVE: Steady-state spectral wave model user's manual for STWAVE, Version 3.0* (No. ERDC/CHL-SR-01-1).

Engineer Research and Development Center, Vicksburg, MS, Coastal and Hydraulics Lab.

Speybroeck, J., Bonte, D., Courtens, W., Gheskiere, T., Grootaert, P., Maelfait, J.-P., Mathys, M., Provoost, S., Sabbe, K., Stienen, E.W.M., Lancker, V.V., Vincx, M., and Degraer, S., 2006. Beach nourishment: an ecologically sound coastal defence alternative? A review. *Aquatic Conservation: Marine and Freshwater Ecosystems* 16, 419–435. doi:10.1002/aqc.733

Stive, M.J.F., de Schipper, M.A., Luijendijk, A.P., Aarninkhof, S.G.J., van Gelder-Maas, C., van Thiel de Vries, J.S.M., de Vries, S., Henriquez, M., Marx, S., Ranasinghe, R., 2013. A New Alternative to Saving Our Beaches from Sea-Level Rise: The Sand Motor. *Journal of Coastal Research*, 290, 1001–1008. doi:10.2112/JCOASTRES-D-13-00070.1

Syvitski, J.P., Slingerland, R.L., Burgess, P., Meiburg, E., Murray, A.B., Wiberg, P., Tucker, G. and Voinov, A.A., 2009. Morphodynamic models: An overview. *River, Coastal and Estuarine Morphodynamics, RCEM*, pp.3-20.

[TGLO] Texas General Land Office (2015). *USACE, Texas GLO and Galveston Parks Board Celebrate Completion of Beach Renourishment Project* [Press release]. https://glo.texas.gov/the-glo/news/press-releases/2015/november/usace-texas-glo-and-galveston-parks-board-celebrate-completion-of-beach-renourishment-project.html

Thomas, R.C. and Frey, A.E., 2013. *Shoreline change modeling using one-line models: General model comparison and literature review* (No. ERDC/CHL-CHETN-II-55). Engineer Research and Development Center, Vicksburg, MS, Coastal and Hydraulics Lab.

Tonnon, P.K., Huisman, B.J.A., Stam, G.N. and Van Rijn, L.C., 2018. Numerical modelling of erosion rates, life span and maintenance volumes of mega nourishments. *Coastal Engineering*, *131*, pp.51-69.

van Slobbe, E., de Vriend, H.J., Aarninkhof, S., Lulofs, K., de Vries, M., and Dircke, P., 2013. Building with Nature: in search of resilient storm surge protection strategies. *Natural Hazards, 66*, 1461–1480. doi:10.1007/s11069-013-0612-3

Valsamidis, A. and Reeve, D.E., 2017. Modelling shoreline evolution in the vicinity of a groyne and a river. *Continental Shelf Research*, 132, 49 - 57. doi:10.1016/j.csr.2016.11.010

Valsamidis, A. and Reeve, D.E., 2020. A new approach to analytical modelling of groyne fields. *Continental Shelf Research*, *211*, p.104288.

Valsamidis, A., Cai, Y. and Reeve, D., 2013. Modelling beach-structure interaction using a Heaviside technique: application and validation. *Journal of Coastal Research*, Special Issue No. 65, pp. 410-415.

Valsamidis, A., Reeve, D.E., Cai, Y. and Dodd, N., 2017. On the morphodynamic evolution of mega-nourishment. *In*: *Proceedings of the Coastal Dynamics* (Vol. 2017), pp. 830-840.

Valsamidis, A., Reeve, D., de Schipper, M. and Dodd, N., 2018. Ensemble prediction of mega-nourisment morphodynamic evolution. *Coastal Engineering Proceedings*, (36), pp.41-41.

Valsamidis, A., Figlus, J., Ritt, B. and Reeve, D.E., 2021. Modelling the morphodynamic evolution of Galveston beach, Gulf of Mexico, following Hurricane Ike in 2008. *Continental Shelf Research*, *218*, p.104373.

Valvo, L.M., Murray, A.B., and Ashton, A., 2006. How does underlying geology affect coastline change? An initial modeling investigation. *Journal of Geophysical Research*, 111, F02025. doi:10.1029/2005JF000340

Van den Berg, N., Falqués, A. and Ribas, F., 2012. Modeling large scale shoreline sand waves under oblique wave incidence. *Journal of Geophysical Research: Earth Surface*, *117*(F3).

Van den Berg, N., Falqués, A., Ribas, F. and Caballeria, M., 2014. On the mechanism of wavelength selection of self-organized shoreline sand waves. *Journal of Geophysical Research: Earth Surface*, *119*(3), pp.665-681.

Walton, T.L. and Chiu, T.Y., 1979. A review of analytical techniques to solve the sand transport equation and some simplified solutions. *Coastal Structures '79*, pp. 809-837.

Whitley, A.E., 2014. Improved numerical modeling of large-scale cuspate coastline dynamics. Wilmington, NC: University of North Carolina Wilmington, Master's thesis, 137p.

Whitley, A., 2018. A numerical modeling investigation of spreading time scales for a hypothetical mega-nourishment. *In*: Kothuis, B., Lee, Y., and Brody, S. (eds.) *NSF-PIRE Coastal Flood Reduction Risk Program: Authentic Learning and Transformative Education,* Vol. 1 (2015 – 2017), pp. 194-201.

Whitley, A.E., Figlus, J., and Valsamidis, A. (in review). Numerical modeling of mega-nourishment shoreline interactions with a groin field. *Coastal Engineering.*

Whitley, A.E., Figlus, J., Valsamidis, A. and Reeve, D.E., 2021. One-line modeling of mega-nourishment evolution. *Journal of Coastal Research*, *37*(6), pp.1224-1234.

Wijnberg, K.M., 1995. Morphologic behavior of a barred coast over a period of decades. Utrecht/Amsterdam: Netherlands Geographical Studies and Utrecht University, Ph.D. dissertation, 258p.

Wijnberg, K.M., 2002. Environmental controls on decadal morphologic behaviour of the Holland coast. *Marine Geology,* 189, 227–247. doi:10.1016/S0025-3227(02)00480-2

Williams, A.T., Rangel-Buitrago, N., Pranzini, E. and Anfuso, G., 2018. The management of coastal erosion. *Ocean & Coastal Management*, *156*, 4-20.

Williams, Z.C., McNamara, D.E., Smith, M.D., Murray, A.B., and Gopalakrishnan, S., 2013. Coupled economic-coastline modeling with suckers and free riders. *Journal of Geophysical Research: Earth Surface*, 118, 887–899. doi:10.1002/jgrf.20066

Wind, H.G., 1990. Influence functions, *Proceedings of the 21st International Conference of Coastal Engineering* (New York, New York), pp. 3281-3294.

Zacharioudaki, A. and Reeve, D.E., 2008. Semianalytical Solutions of Shoreline Response to Time-Varying Wave Conditions. *Journal of Waterway, Port, Coastal, and Ocean Engineering*: 134, 265-274. doi:10.1061/(ASCE)0733-950X(2008)134:5(265)

APPENDIX A

COASTLINE EVOLUTION MODEL (C VERSION) CODE

```
/*  CAPERIFFIC
*/
/*  Program To generate capes?? and sandwaves?? using wave
angle relationships  */
/*  Begun by Brad Murray 01/00
*/
/*  Refined by Olivier Arnoult 01/00 - 06/00
*/
/*  Revised and reformed by Andrew Ashton 06/00 -
*/
//  Revised by Andrew Whitley 07/18
/*
*/
/*  Program Notes -
*/
/*      To end program, press 'd' key and 'ESC' key
simultaneously              */
/*      To save current iteration to file, press 's' and
'f' simultaneously     */
/*      To update screen display, press 'p' key
*/
/*
*/

#include <stdlib.h>      /*THIS PROGRAM GONNA MAKE CAPES,
SANDWAVES??*/
#include <stdio.h>
#include <math.h>
#include <time.h>
/*#include <GL/glx.h>
#include <GL/gl.h> */
#ifdef __unix__
# include <unistd.h>
#elif defined _WIN32
# include <windows.h>
#define sleep(x) Sleep(1000 * x)
#endif
//#include <unistd.h>
```

```
//#include <ncurses.h>  // This header allows for the use
of graphics


/*  Run Control Parameters */

int     RunInWindows = 0;           // Run on a WIN32
machine?
#define TimeStep        0.0417      /* days - reflects rate
of sediment transport per time step */
#define OffShoreWvHt    0.85        /* meters */
#define Period          5           /* seconds */
#define Asym            0.5         /*fractional portion of
waves coming from positive (left) direction */
#define Highness        0.5         /* New! .5 = even dist,
> .5 high angle domination */
#define Duration        1           /* Number of time steps
calculations loop at same wave angle */
#define StopAfter       31200       /* Stop after what
number of time steps */
int     StartWRTAfter = 999999999;          // Time steps
to pass before using WRT
int     seed = 44;                  /* random seed  control
value = 1 */

// Initialization parameters
char    StartFromFile = 'y';        /* start from saved
file? */
char    readfilename[24] = "SE_25m_1200Cells";
char    StartFromLine = 'n';        // Start from a saved
lineout file?
char    readlinename[24] = "lineouttest";   // Lineout name
to start from
int     WaveIn = 1;                 /* Input Wave
Distribution file? */
char    readwavename[24] = "SE_MPN.dat";
int     TimesAngleHit[36];
int     NumTimesRun = 100;
int     iii = 0;
int     iv = 0;

// Save control parameters
int     StartSavingAt = 0;          /* time step to begin
saving files */
```

```
int     SaveFile = 1;                /* save full file? */
char    savefilename[24] = "fileout";
int     SaveSpacing = 10000 ;        /* space between saved
files */
int     SaveLine = 1;                /* Save line */
char    savelinename[24] = "lineout";
int     SaveLineSpacing = 100;       /* space between saved
line files */

// Sediment transport parameters
char    UseVariableCERC = 'y';       // Use the variable
CERC equation? --> allows the parameterization of rho_s,
porosity, and K
char    UseKamphius = 'n';           // Use the Kamphius
(1991) LST equation instead of CERC? (disables CERC; NOT
CURRENTLY FUNCTIONAL)
char    UseBailard = 'n';            // Use the Bailard
(1984) equation to determine K? (only works with
UseVariableCERC)
char    UseBattjesStive = 'n';       // Use the Battjes and
Stive (1985) equation to determine KBreak?

float   K = 0.6935 ;                      // CERC formula
emperical constant (overridden if UseBailard)
float   rho = 1000;                  /* kg/m3 - density of
water and dissolved matter                  */
float   rho_s = 2650;                // Denisty of sediment
(kg/m3)
float   porosity = 0.4;              // Sediment porosity (n
in most literature)
float   KBreak = 0.78;               // Coefficient for wave
breaking threshold (breaking wave height / depth at
breaking); originally 0.5; overridden by UseBattjesStive
float   FallVel;                     // Fall velocity (m/s)
of sediment grains (used only in UseBailard at this point)
float   DynViscosity = 1.03E-3;      // Dynamic viscosity of
seawater (Pa*s)
float   DFifty = 281;                // D50 (median grain
size in microns)
float   SedRad;                      // Sediment radius
(calculated from D50)
float   PeakPeriod = 5;              // Peak period (Tp) in
s (for Kamphius formaula)
```

```
/// WRT save control parameters
int     SaveHeight = 0;             // Save breaking wave
height data?
char    saveheightname[24] = "height"; // Save breaking
height name
int     SaveHeightSpacing = 1;  // space between saved wave
height files
int     SaveAngle = 0;              // Save breaking angle
data?
char    saveanglename[24] = "angle";        // Save
breaking angle name
int     SaveAngleSpacing = 1;   // space between saved wave
angle files
int     SaveRelAngle = 0;           // Save relative angle
data
char    SaveRelAnglename[24] = "RelAngle";  // Save
relative angle name
int     SaveRelAngleSpacing = 1;    // space between saved
relative angle files
int     SaveSLAngle = 0;            // Save the shoreline
angle?
char    SaveSLAngleName[24] = "ShoreAngle"; // Save
shoreline angle name
int     SaveSLAngleSpacing = 1; // space between saved
shoreline angle files
int     SaveVolumeOut = 0;          // Save sediment volume
transported out?
char    saveVolumeOutName[24] = "VolumeOut";    // Save sed
volume out name
int     SaveVolumeOutSpacing = 1; // space between saved
sed volume out files
int     SaveSedFluxGradOut = 0;     // Save the sediment
flux gradient?
char    saveSedFluxGradOutName[24] = "SedFluxGrad"; // Save
sed flux gradient out name
int     SaveSedFluxGradOutSpacing = 1;  // space between
saved sed flux grad out files
int     SaveQs = 0;                 // Save the sediment
transport data?
char    SaveQsName[24] = "Qs";      // Save sediment
transport name
int     SaveQsSpacing = 1;      // space between saved sed
transport files
int     SaveDEtaDt = 0;             // Save dEta/dt?
```

```
char    SaveDEtaDtName[24] = "dEta_dt"; // Save dEta/dt
name
int     SaveDEtaDtSpacing = 1;  // space between dEta/dt
files
int     SaveDiffusivity = 0;        // Save the
diffusivity?
char    SaveDiffusivityName[24] = "diffusivity";    // Save
diffusivity name
int     SaveDiffusivitySpacing = 1;     // space between
saved diffusivity files
int     SaveMu = 0;                 // Save the mu
(diffusivity)?
char    SaveMuName[24] = "mu";      // Save mu name
int     SaveMuSpacing = 1;      // space between mu
(diffusivity) files
int     SaveGamma = 0;              // Save the instability
index?
char    SaveGammaName[24] = "gamma";    // Save instability
index name
int     SaveGammaSpacing = 1;       // space between saved
instability index files

// WRT Control Parameters
char    UseSingleInitialAngle = 'n';    // Use a single
initial angle?
float   thetao = 60;                // Single initial angle
in degrees
char    UseShadow = 'y';            // Use the built-in
shadowing routine
char    UseWRT = 'n';               // Use the forward wave
ray tracing (WRT) method to propagate waves?
char    UseRandWaveAngle = 'n';     // Use a random deep
water wave angle for each wave ray in WRT?
char    UseWaveDist = 'n';          // Use wave
distribution in WRT
char    LinkToSedTrans = 'n';       // Link WRT to Sediment
transport model
char    UseRandAngleFluctuation = 'n';  // Use a random
angle fluctionation (+/- 5 deg) each WRT sweep?
char    UseAWTOffshore = 'n';       // Use AWT to propagate
offshore to WRT domain (linked to CEM; *Set LinkToSedTrans
= 'n'!);
char    UseHeightLimit = 'n';       // Limit how large the
wave height can get
```

```
float   beta_limit = 0.2;           // Smallest number beta
is allowed to go
float   height_limit = 2*OffShoreWvHt;  // Largest number
wave height is allowed to go
float   MinBreakDistance = 70;      // Minimum distance (m)
a wave is allowed to go to the shore before breaking
float   delta_s = 1;                // Runge-Kutta step
size in WRT ODE solver (meters)
int     NumSweeps = 4;              // Number of ray sweeps
to run in WRT
int     RollAvgNeighbors = 0;       // Number of neighbors
to use in rolling average calculations (1-5,10)
int     MaxGammaCounter = 100;      // Number of time steps
to go before resetting Gamma calculation arrays
int     DisplayOverallAverages = 0; // Display the overall
averages (H, RelAngle, Qs)?


/* Delta Info */

#define NMouths         0       /* number of river mouths
*/
float   QRiver[NMouths];
#define SedRate 0.00
#define StreamSpot Ymax

/* Aspect Parameters */

#define CellWidth       25.0    /* size of cells (meters)
*/
#define CritBWidth      350.0   /* width barrier maintains
due to overwash (m) important scaling param! */
#define Xmax            200     /* number of cells in x
(cross-shore) direction */
#define Ymax            1200     /* number of cells in y
(longshore) direction */
#define MaxBeachLength  8*Ymax  /* maximum length of arrays
that contain beach data at each time step */
#define ShelfSlope      0.0052   /* slope of continental
shelf */
#define ShorefaceSlope  0.0177    /* for now, linear slope
of shoreface */
                                /* future : shoreface
exponent m^1/3, from depth of ~10m at ~1000 meters */
```

```
#define DepthShoreface  8.22   /* minimum depth of
shoreface due to wave action (meters) */
#define LandHeight      1.90     /* elevation of land above
MHW  */
#define BathySlope      0.5     // Slope of bathymetry
#define Kappa           0.1     // Bathymetry diffusion
coefficient
#define InitBeach       20      /* cell where intial
conditions changes from beach to ocean */
#define InitialDepth    6.0     /* theoretical depth in
meters of continental shelf at x = InitBeach (only used for
Initial Conditions creation) */
#define InitCType       0       /* type of initial conds 0
= sandy, 1 = barrier */
#define InitBWidth      4       /* initial minimum width of
barrier (Cells) */
#define OWType        1         /* 0 = use depth array, 1 =
use geometric rule */
#define OWMinDepth  0.1         /*  littlest overwash of
all */
#define FindCellError   5       /* if we run off of array,
how far over do we try again? */
float   MaxDepth = 18;          // Maximum depth of
bathymetry if not using constant slope
int     StartFlatBathy = 60;        // Number of cells away
from Xmax to have a flat bathymetry (+ towards shore)
float   SedTansLimit =  90;     /* beyond what absolute
slope don't do sed trans (degrees)*/
float   OverwashLimit = 75;     /* beyond what angle don't
do overwash */
int     lngth_bathy = Xmax;     // Bathymetry length
(cross-shore; x-direction)
int     wdth_bathy = Ymax;      // Bathymetry width (long-
shore; y-direction)
char    UseDeanProfile = 'y';       // Use the Dean Profile
for bathymetry?
char    UseDeanMirror = 'n';        // Use the Dean Profile
and mirrored bathymetry (mirrored after going flat)
char    UseFixedBathySlope = 'n';   // Use a fixed slope in
the bathymetry? (otherwise adaptive)
char    DiffuseBathymetry = 'n';    // Diffuse the
bathymetry?
float   T = 5;
float   period = 5;
```

```
/* Plotting Controls */

int CellPixelSize = 4;
int XPlotExtent = Xmax;
int YPlotExtent = Ymax;
int AgeMax = 1000000;       /* Maximum 'age' of cells -
loops back to zero */
int AgeUpdate = 10;         /* Time space for updating age
of non-beach cells */
int AgeShadeSpacing = 10000; /* For graphics - how many
time steps means back to original shade */

/* DeBuggin Parameters */

int     DoGraphics = 0;
int     KeysOn = 0;
int     SaveAge = 0;        /* Save/update age of cells? */
char    PromptStart = 'n';  /* ask prompts for file names,
etc? */
char    OffArray = 'n';     /* Initializing this variable
for later use */
int ScreenTextSpacing = 1000;/* Spacing of writing to
screen in time steps */
int EveryPlotSpacing = 100;
int StartStop = 0;      /* Stop after every iteration 'Q'
to move on */
int InterruptRun = 0;   /* Allow run to be paused by
pressing the 'A' key */
int NoPauseRun = 1;     /* Disbale PauseRun subroutine */
int InitialPert = 0;    /* Start with a bump? */
int InitialSmooth = 0;  /* Smooth starting conditions */
int TestSSInstability = 0; // Test small-scale instability
with small perturbation (InitialSmooth must be = 1)
int StartWBigBumps = 0; // Initial condition: 2 large 100-
cell tall bumps (recommend InitialSmooth = 1)
int WaveAngleSign = 1;  /* used to change sign of wave
angles */
int debug0 = 0;     /* Main program steps */
int debug1 = 0;     /* Find Next Cell */
int debug2 = 0;     /* Shadow Routine */
int debug3 = 0;     /* Determine Angles */
int debug4 = 0;     /* Upwind/Downwind */
int debug5 = 0;     /* Sediment Transport Decisions*/
```

```
int debug6 = 0;      /* Sediment Trans Calculations */
int debug7 = 0;      /* Transport Sweep (move sediment) */
int debug7a = 0;     /* Slope Calcs */
int debug8 = 0;      /* Full/Empty */
int debug9 = 0;      /* FixBeach */
int debug10a = 0;    /* Overwash Tests*/
int debug10b = 0;    /* doing overwash (w/screen) */
int debug11 = 0;     // WRT initial parameters
int debug11a = 0;    // WRT input bathymetry
int debug12 = 0;     // ODE solver output for WRT
int debug13 = 0;     // WRT iterative output
int debug14 = 0;     // WRT stored data
int debug15 = 0;     // WRT Data at wave break
int debug16 = 0;     // Wave height data at each cell from
WRT
int debug16a = 0;    // Number of wave rays that passed
through each cell
int debug16b = 0;    // Beta data at each cell
int debug17 = 0;     // WRT wave breaking criteria
int debug18 = 0;     // Initial wave angle and starting
position for wave rays in WRT
int debug19 = 0;     // Average breaking height and angle
data (from WRT)
int debug20 = 0;     // Wave origin in WRT
int debug21 = 0;     // Display AWT wave height change
int debug22 = 0;     // CERC Parameter debugger
int debug33 = 0;     // Wave angle distribution debugger
int OWflag = 0;      /* debugger */

/* Universal Constants */

#define pi       3.1415926535898
#define exp      2.7182818 /* e */
double Pi =      3.1415926535898;
float g =        9.80665;
float radtodeg =    180.0/pi; /* transform rads to degrees
*/
float PerSecondToPerDay = 86400;    // Multiplying a flux
by this converts a flux per second to a flux per day


/* Overall Shoreface Configuration Arrays - Data file
information */
```

```
char    AllBeach[Xmax][2*Ymax];     /* Flag indicating of
cell is entirely beach ('y'/'n') */
float   PercentFull[Xmax][2*Ymax];  /* Fractional amount of
shore cell full of sediment */
int     Age[Xmax][2*Ymax];          /* Age since cell was
deposited */
float   CellDepth[Xmax][2*Ymax];    /* Depth array (m) (ADA
6/3) */
float   StartingLine[2*Ymax];       // Starting line if
reading a lineout file
FILE    *SaveSandFile;
FILE    *ReadSandFile;
FILE    *ReadWaveFile;


/* Computational Arrays (determined for each time step) */

int     X[MaxBeachLength];              /* X Position of
ith beach element */
int     Y[MaxBeachLength];              /* Y Position of
ith beach element */
char    InShadow[MaxBeachLength];       /* Is ith beach
element in shadow? */
float   ShorelineAngle[MaxBeachLength]; /* Angle between
cell and right (z+1) neighbor  */
float   SurroundingAngle[MaxBeachLength];/* Cell-orientated
angle based upon left and right neighbor */
char    UpWind[MaxBeachLength];         /* Upwind or
downwind condition used to calculate sediment transport */
float   VolumeIn[MaxBeachLength];       /* Sediment volume
into ith beach element */
float   VolumeOut[MaxBeachLength];      /* Sediment volume
out of ith beach element */
float   avg_breaking_height[2*Ymax];    // Average breaking
height at each shoreline cell
float   avg_breaking_angle[2*Ymax];     // Average breaking
angle at each shoreline cell
float   breaking_angles[2*Ymax];        // Breaking angle
for each shorline cell (same as average)
float   breaking_heights[2*Ymax];       // Breaking height
for each shorline cell (same as average)
float   BkAngle[2*Ymax];                // Breaking angle
for each shoreline cell (for linking WRT to CEM)
```

```
float   HeightRollAvg[2*Ymax];          // Rolling average
of breaking heights
float   AngleRollAvg[2*Ymax];           // Rolling average
of breaking angles
float   SedFluxGrad[MaxBeachLength];    // Sediment flux
gradient (VolumeIn - VolumeOut, m^3/s)
float   Diffusivity[MaxBeachLength];    // Diffusivity (mu)
at current time step
float   Mu[MaxBeachLength];             // Diffusivity
(calculated a different way)
float   SumMuDeltaT[MaxBeachLength];    // Sum of
diffusivities * delta t (timestep)
float   SumAbsMuDeltaT[MaxBeachLength]; // Sum of absolute
value of diffusivity  * delta t (timestep)
float   Gamma[MaxBeachLength];          // Instability
index
int     GammaCounter;                   // Counter to keep
track of when to reset time on Gamma calculations.
float   Qs[MaxBeachLength];             // Volumetric
sediment transport at cell border
float   dEta_dt[MaxBeachLength];        // Change in cross-
shore position each time step
float   RelAngle[MaxBeachLength];       // Relative angle
between breaking wave & shoreline angle
float   SLAngle[MaxBeachLength];        // Shoreline angle
float   SumBkHeight[2*Ymax];            // Sum of the
breaking height over entire simulation)
float   SumRelAngle[2*Ymax];            // Sum of the
relative breaking angle (over entire simulation)
float   SumQs[2*Ymax];                  // Sum of Qs (over
entire simulation)
float   TotalAvgBkHeight[2*Ymax];       // Average breaking
height (over entire simulation)
float   TotalAvgRelAngle[2*Ymax];       // Average breaking
angle (over entire simulation)
float   TotalAvgQs[2*Ymax];             // Average Qs (over
entire simulation)
float   AWTWvHeight[Xmax][2*Ymax];      // "Cell" wave
height in AWT

// Compuational Arrays in WRT function
    int beach[2*Ymax];
    float ZZ[Xmax][2*Ymax];
    float DiffusedDepth[Xmax][2*Ymax];
```

```
    float depth[Xmax][2*Ymax];
    int x_pos[Xmax];
    int y_pos[2*Ymax];
    float L[Xmax][2*Ymax];
    float c[Xmax][2*Ymax];
    float cx[Xmax][2*Ymax];
    float c2x2[Xmax][2*Ymax];
    float cy[Xmax][2*Ymax];
    float c2y2[Xmax][2*Ymax];
    float cxcy[Xmax][2*Ymax];
    float k[Xmax][2*Ymax];
    float f[Xmax][2*Ymax];
    float C_g[Xmax][2*Ymax];
    int y_int[2*Ymax];
    int N_rays_hit[2*Ymax]; // Number of wave rays that hit
a particular beach cell
    float cell_wave_height[Xmax][2*Ymax];       // Current
wave height at each cell
    float sum_cell_wave_height[Xmax][2*Ymax];   // Sum of
wave heights at each cell (adds every time ray passes over
cell)
    float avg_cell_wave_height[Xmax][2*Ymax];   // Average
wave height at each cell (over entire simulation)
    float cell_beta[Xmax][2*Ymax];              // Current
beta at each cell
    float sum_cell_beta[Xmax][2*Ymax];          // Sum of
beta information at each cell (adds every time ray passes
over cell)
    float avg_cell_beta[Xmax][2*Ymax];          // Average
beta information at each cell (over entire simulation)
    float breaking_height[2*Ymax];      // Breaking height
for each shoreline cell (single wave ray)
    float sum_breaking_height[2*Ymax];  // Sum of break
heights at each shoreline cell (addes every time a wave ray
hits it)
    float breaking_angle[2*Ymax];       // Breaking angle
for each shoreline cell (single wave ray)
    float sum_breaking_angle[2*Ymax];   // Sum of break
heights at each shoreline cell (addes every time a wave ray
hits it)
    float avg_breaking_angle[2*Ymax];   // Average breaking
angle at each shorline cell
    int NumRaysPerCell[Xmax][2*Ymax];   // Number of rays
that pass through each cell
```

```
    int RayInCell[Xmax][2*Ymax];        // Has a ray passed
through this cell this iteration? (0 or 1)


/* Miscellaneous Global Variables */

int     CurrentTimeStep = 0;    /* Time step of current
calculation */
int     NextX;              /* Global variables used to
iterate FindNextCell in global array - */
int     NextY;              /*  would've used pointer but
wouldn't work */
int     TotalBeachCells;    /* Number of cells describing
beach at particular iteration */
int     ShadowXMax;         /* used to determine maximum
extent of beach cells */
float   WaveAngle;          /* wave angle for current time
step */
int     FindStart;          /* Used to tell FindBeach at
what Y value to start looking */
char    FellOffArray;       /* Flag used to determine if
accidentally went off array */
float   MassInitial;        /* For conservation of mass
calcs */
float   MassCurrent;        /* " */
int     device;
short   button;
long    buttonback;
int     NumWaveBins;        /* For Input Wave - number of
bins  */
int     BinSize;            // Size of wave distribution
bins
float   WaveMax[36];        /* Max Angle for specific bin
*/
float   WaveProb[36];       /* Probability of Certain Bin
*/
float   MaxWaveProb;        // Maximum wave probability
float   xcellwidth;
float   ycellwidth;
float   StartAngle;         // Starting wave angle for WRT
float   StartHeight;        // Starting wave height for WRT
//static WINDOW *mainwnd;
//static WINDOW *screen;
//WINDOW *my_win;
```

```
int current_getch;
int xplotoff;
int yplotoff;


int KEY_P = 93;


/* Function Prototypes */

void    AdjustShore(int i);
void    AgeCells(void);
void    ButtonEnter(void);
void    CalcSedFluxGrad(void);
void    CalcDiffusivity(void);
void    CalcGamma(void);
void    CalcOverallAverages(void);
void    CheckOverwash(int icheck);
void    CheckOverwashSweep(void);
void    DeliverSediment(void);
void    DetermineAngles(void);
void    DetermineSedTransport(void);
void    DisplayAWTWvHeight(void);
void    DoOverwash(int xfrom,int yfrom, int xto, int yto,
float xintto, float yintto, float distance, int ishore);
void    FindBeachCells(int YStart);
char    FindIfInShadow(int icheck, int ShadMax);
void    FindNextCell(int x, int y, int z);
float   FindWaveAngle(void);
void    FixBeach(void);
float   GetOverwashDepth(int xin, int yin, float xintto,
float yintto, int ishore);
void    InitConds(void);
void    InitPert(void);
void    InitBigBumps(void);
void    LoadLineout(void);
float   MassCount(void);
void    OopsImEmpty(int x, int y);
void    OopsImFull(int x, int y);
void    OpenWindow(void);
void    PauseRun(int x, int y, int in);
void    PeriodicBoundaryCopy(void);
void    PrintLocalConds(int x, int y, int in);
float   Raise(float b, float e);
```

```
float   RandAngleFluctuation(void);
float   RandWaveAngle(void);
float   RandZeroToOne(void);
void    ReadSandFromFile(void);
void    ReadWaveIn(void);
void    RunAWTOffShore (void);
void    RunWRT(void);
void    SaveDEtaDtToFile(void);
void    SaveDiffusivityToFile(void);
void    SaveGammaToFile(void);
void    SaveSandToFile(void);
void    SaveLineToFile(void);
void    SaveMuToFile(void);
void    SaveQsToFile(void);
void    SaveRelAngleToFile (void);
void    SaveSedFluxGradOutToFile (void);
void    SaveSLAngleToFile (void);
void    SaveVolumeOutToFile (void);
void    SaveWvHeightToFile (void);
void    SaveWvAngleToFile (void);
void    SedTrans(int From, int To, float ShoreAngle, char
MaxT);
void    ShadowSweep(void);
void    TransportSedimentSweep(void);
int     XMaxBeach(int Max);
void    ZeroVars(void);

/*void  FillUpGap(int X, int Y, int LorR);*/


/* Lets's Go! - Main Program */

int main(void)
{


/* Initialize Variables and Device */

    int xx;          /* duration loop variable */
    int i;           // index for loops
    ShadowXMax =    Xmax-5;


    srand(seed);
```

```
/* Start from file or not? */

    if (PromptStart == 'y')
    {
    printf("shall we start from a file (y or n)? \n");
    scanf("%c", &StartFromFile);

    if (StartFromFile == 'y')
    {
        printf("Starting Filename? \n");
        scanf("%24s", readfilename);
        printf("Saving Filename? \n");
        scanf("%s", savefilename);
        printf("What time step are we starting at?");
        scanf("%d", &CurrentTimeStep);
        ReadSandFromFile();
    }

    if (StartFromFile == 'n')
    {
        printf("Saving Filename? \n");
        scanf("%s", savefilename);
        InitConds();
        if (InitialPert)
        {
        InitPert();
        }
        if (StartWBigBumps)
        {
        InitBigBumps();
        }
        printf("InitConds OK \n");
    }
    }
    else if (StartFromFile == 'y')
    {
    ReadSandFromFile();
    }
    else if (StartFromLine == 'y')
    {
    LoadLineout();
    }
    else
```

```
    {
    InitConds();
    if (InitialPert)
    {
        InitPert();
    }
    if (StartWBigBumps)
    {
        InitBigBumps();
    }
    }


    /* Count Initial Mass */

    PeriodicBoundaryCopy();
    FixBeach();
    MassInitial = MassCount();

    /*if (SaveLine)
      SaveLineToFile();
      if (SaveFile)
      SaveSandToFile();*/

    if(WaveIn)
    ReadWaveIn();

    // Calculate fall velocity (using Stokes settling)
    if (UseBailard == 'y'){

        SedRad = DFifty/(2*1E6);
        FallVel = (2 * (rho_s - rho) * g * Raise(SedRad,2))
/ (9 * DynViscosity);

    }


/* ---------------------------------------    PRIMARY
PROGRAM LOOP   ---------------------------------- */

    SaveLineToFile();

    while (CurrentTimeStep < StopAfter )
    {
```

```
    /*  Time Step iteration - compute same wave angle for
Duration time steps */

    // Activate WRT
    if (CurrentTimeStep >= StartWRTAfter){
        UseWRT = 'y';
        UseShadow = 'n';
        //SaveHeight = 1;
        //SaveLineSpacing = 1;
        //SaveAngle = 1;
        //printf("UseWRT = %c \n", UseWRT);
        //printf("UseShadow = %c \n", UseShadow);
        //printf("\n");
    }

    // THIS IS FOR DEBUGGING
    // if (CurrentTimeStep >= 7700){
        // SaveLineSpacing = 1;
        // SaveHeightSpacig = 1;
        // SaveAngleSpacing = 1;
    // }

    /*  Calculate Wave Angle */

    if (UseSingleInitialAngle == 'y'){
        WaveAngle=thetao*3.14159/180;
    } else {
        WaveAngle = FindWaveAngle();
    }

    if (debug33){
        for (iii = 0; iii < NumWaveBins+1; iii++){
            //printf("%i\n", iii);
            if ((-WaveAngle*180/pi > (WaveMax[iii] -
BinSize) && (-WaveAngle*180/pi <= WaveMax[iii]))){
                TimesAngleHit[iii] = TimesAngleHit[iii] +
1;
                //printf("TimesAngleHit = %i \n",
TimesAngleHit[iii]);
                break;
            }
        }
    }
```

```
    //printf("Wave Angle = %G \n", WaveAngle*radtodeg);

    // Clear all tracks
    for (i = 0; i < MaxBeachLength; i++){
        Qs[i] = 0;
    }

    // Run WRT to determine breaking wave angles and
heights
    if (UseWRT == 'y'){ // Use WRT

        if (UseAWTOffshore == 'n'){
            StartHeight = OffShoreWvHt;
            StartAngle = WaveAngle;
            printf("WaveAngle = %G \n",
WaveAngle*radtodeg);
            printf("StartHeight = %G \n", StartHeight);
        } else if (UseAWTOffshore == 'y'){ // Use AWT to
pre-refract
            printf("Using AWT to propagate waves to WRT
origin... \n");
            RunAWTOffShore();
            printf("WaveAngle = %G \n",
WaveAngle*radtodeg);
            printf("StartAngle = %G \n",
StartAngle*radtodeg);
            printf("StartHeight = %G \n", StartHeight);
        }

        printf("Running WRT... \n");
        RunWRT();
        printf("WRT run complete.  Current time step: %d
\n", CurrentTimeStep); printf("\n");
    } else { // Use AWT

        if (UseAWTOffshore == 'y'){ // Use AWT to pre-
refract
            printf("Using AWT to propagate waves to WRT
origin... \n");
            RunAWTOffShore();
            printf("WaveAngle = %G \n",
WaveAngle*radtodeg);
            printf("StartAngle = %G \n",
StartAngle*radtodeg);
```

```
            printf("StartHeight = %G \n", StartHeight);
        }

    }

    /*  Loop for Duration at the current wave sign and wave
angle */

    for (xx = 0; xx < Duration; xx++)
    {

        /* Text to Screen? */

        if (CurrentTimeStep%ScreenTextSpacing == 0)
        {
        printf("==== WaveAngle: %2.2f  MASS Percent: %1.4f
Time Step: %d\n", 180*(WaveAngle)/pi,
                MassCurrent/MassInitial, CurrentTimeStep);
        }

        PeriodicBoundaryCopy();

        ZeroVars();

        /* Initialize for Find Beach Cells  (make sure
strange beach does not cause trouble */

        FellOffArray = 'y';
        FindStart = 1;

        /*  Look for beach - if you fall off of array, bump
over a little and try again */

        while (FellOffArray == 'y')
        {
        FindBeachCells(FindStart);
        /*printf("FoundCells: %d GetO = %c \n",
FindStart,FellOffArray);*/
        FindStart += FindCellError;
        if (FellOffArray == 'y')
        {
            /*printf("NOODLE  !!!!!FoundCells: %d GetO = %c
\n", FindStart,FellOffArray); */
            /*PauseRun(1,1,-1);*/
```

```
        }

        /* Get Out if no good beach spots exist - finish
program*/

        if (FindStart > Ymax/2+1)
        {
            printf("Stopped Finding Beach - done %d
%d",FindStart,Ymax/2-5);
            SaveSandToFile();
            SaveLineToFile();
            SaveWvHeightToFile();
            SaveWvAngleToFile();
            SaveVolumeOutToFile();
            return 1;
        }
        }

        if (debug0) printf("Foundbeach!: %d \n",
CurrentTimeStep);

        ShadowSweep();
        if (debug0) printf("Shadowswept: %d \n",
CurrentTimeStep);
        DetermineAngles();
        if (debug0) printf("AngleDet: %d \n",
CurrentTimeStep);
        DetermineSedTransport();
        if (debug0) printf("Sed Trans: %d \n",
CurrentTimeStep);
        DisplayAWTWvHeight();
        TransportSedimentSweep();
        if (debug0) printf("Transswept: %d \n",
CurrentTimeStep);

        DeliverSediment();

        FixBeach();
        if (debug0) printf("Fixed Beach: %d \n",
CurrentTimeStep);

        // Calcuate and save the sediment flux gradient

        CalcSedFluxGrad();
```

```
        if (((CurrentTimeStep%SaveVolumeOutSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) &&
SaveVolumeOut)
        {
        SaveVolumeOutToFile ();
        }

        if (((CurrentTimeStep%SaveSedFluxGradOutSpacing ==
0 && CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) &&
SaveSedFluxGradOut)
        {
        SaveSedFluxGradOutToFile ();
        }

        /* OVERWASH */
        /* because shoreline config may have been changed,
need to refind shoreline and recalc angles*/

            ZeroVars();
            /* Initialize for Find Beach Cells  (make sure
strange beach does not cause trouble */

            FellOffArray = 'y';
            FindStart = 1;

            /*  Look for beach - if you fall off of array,
bump over a little and try again */

            while (FellOffArray == 'y')
            {
                FindBeachCells(FindStart);
                /*printf("FoundCells: %d GetO = %c \n",
FindStart,FellOffArray);*/
                FindStart += FindCellError;
                if (FellOffArray == 'y')
                {
                    /*printf("NOODLE  !!!!!FoundCells: %d
GetO = %c \n", FindStart,FellOffArray); */
                    /*PauseRun(1,1,-1);*/
                }
```

```
                /* Get Out if no good beach spots exist -
finish program*/

                if (FindStart > Ymax/2+1)
                {
                    printf("Stopped Finding Beach - done %d
%d",FindStart,Ymax/2-5);
                    SaveSandToFile();
                    return 1;
                }
            }

                /* printf("Foundbeach!: %d \n",
CurrentTimeStep); */

            ShadowSweep();
                if (debug0) printf("Shadowswept: %d \n",
CurrentTimeStep);
            DetermineAngles();
                if (debug0) printf("AngleDet: %d \n",
CurrentTimeStep);
            CheckOverwashSweep();
            FixBeach();


        if ((StartStop)    )
        {
        printf("---- You Paused it, bud ---- \npp");
        PauseRun(1,1,-1);
        }

        if(debug0) printf("End of Time Step: %d \n",
CurrentTimeStep);

        /* Age Empty Cells */

        if ((CurrentTimeStep%AgeUpdate == 0) && SaveAge)
        AgeCells();

        /* Count Mass */

        MassCurrent = MassCount();

        CurrentTimeStep ++;
```

```
        // Calculate Diffusivity, Instability Index, and
Overall Averages (H, RelAngle, & Qs)

        CalcDiffusivity();
        CalcGamma();
        CalcOverallAverages();


        /* SAVE FILE ? */

        if (((CurrentTimeStep%SaveSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveFile)
        {
        SaveSandToFile();
        }

        if (((CurrentTimeStep%SaveLineSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveLine)
        {
        SaveLineToFile();
        }

        if (((CurrentTimeStep%SaveHeightSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveHeight)
        {
        SaveWvHeightToFile ();
        }

        if (((CurrentTimeStep%SaveAngleSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveAngle)
        {
        SaveWvAngleToFile ();
        }

        if (((CurrentTimeStep%SaveRelAngleSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) &&
SaveRelAngle)
        {
```

```
        SaveRelAngleToFile ();
        }

        if (((CurrentTimeStep%SaveSLAngleSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveSLAngle)
        {
        SaveSLAngleToFile ();
        }

        if (((CurrentTimeStep%SaveQsSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveQs)
        {
        SaveQsToFile();
        }

        if (((CurrentTimeStep%SaveDiffusivitySpacing == 0
&& CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) &&
SaveDiffusivity)
        {
        SaveDiffusivityToFile();
        }

        if (((CurrentTimeStep%SaveMuSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveMu)
        {
        SaveMuToFile();
        }

        if (((CurrentTimeStep%SaveGammaSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveGamma)
        {
        SaveGammaToFile();
        }

        if (((CurrentTimeStep%SaveDEtaDtSpacing == 0 &&
CurrentTimeStep >= StartSavingAt)
         || (CurrentTimeStep == StopAfter)) && SaveDEtaDt)
        {
        SaveDEtaDtToFile();
```

```
        }


        /*if (CurrentTimeStep > 14300)
          SaveSandToFile();*/
    }


    }

    // Display overall averages
    if (DisplayOverallAverages){

        // Total Average Breaking Wave Height
        printf("Overall Average Breaking Wave Height =
\n");
        for (i = Ymax/2; i < 3*Ymax/2; i++){
            printf("%G ", TotalAvgBkHeight[i]);
        }
        printf("\n\n");

        // Total Average Relative Breaking Angle
        printf("Overall Average Relative Breaking Angle =
\n");
        for (i = Ymax/2; i < 3*Ymax/2; i++){
            printf("%G ", TotalAvgRelAngle[i]);
        }
        printf("\n\n");

        // Total Average Sediment Flux
        printf("Overall Average Sediment Flux (Qs) = \n");
        for (i = Ymax/2; i < 3*Ymax/2; i++){
            printf("%G ", TotalAvgQs[i]);
        }
        printf("\n\n");

    }

    if (debug33){
        printf("\nTimesAngleHit =");
        for (iv = 0; iv < NumWaveBins+1; iv++){
            printf(" %i ", TimesAngleHit[iv]);
        }
    }
```

```
/* ---------------------------------------- END OF MAIN ----
------------------------------------------------ */

    printf("Run Complete.  Output file: %s \n" ,
savefilename);

}


float FindWaveAngle(void)

/* calculates wave angle for given time step */
// Modified to run in both Windows and UNIX
// Corrected for input of PDFs

{

    float   Angle;          // Wave angle
    float   AngleBin;       // Max wave angle in PDF bin
    float   AngleFluct;     // Random angle fluctuation

    /* Data Input Method */

    float   RandBin;    /* Random number to pick wave angle
bin */
    float   RandAngle;  /* Random number to pick wave angle
within the bin */
    int     flag = 1;
    int     i = 0;
    int     index = 0;
    int     index2 = 0;
    float   WaveMaxProb[36];    // Maximum angle in each
PDF wave bin


    /*  Variables for Asymmetry Method */

    float   AsymRandom;   /* variable used to determine
wave direction for current time step */
    float   AngleRandom;  /* variable used to determine
wave power for current time step */
    int     Sign;         /* defines direction of incoming
waves for current time step, */
```

```
    /* positive from left, negative from right */


    /* Method using input binned wave distribution -
*/
    /* variables WaveProb[], WaveMax[], previously input
from file using ReadWaveIn()   */

    // Determine the wave angle from the input PDF
    if (WaveIn)
    {

        WaveMaxProb[0] = WaveProb[0];

        for (index = 1; index < NumWaveBins+1; index++)
        {
            WaveMaxProb[index] = WaveMaxProb[index-1] +
WaveProb[index];
        }

        RandBin = RandZeroToOne();
        //printf("RandBin = %G\n",RandBin);
        AngleFluct = RandZeroToOne();

        index2 = NumWaveBins;

        while (flag)
        {
            if (RandBin >= WaveMaxProb[index2]){
                AngleBin = WaveMax[index2+1];
                //printf("AngleBin = %G\n", AngleBin);
                flag = 0;
                break;
            }
            index2--;
        }

        Angle = - (AngleFluct * BinSize + (AngleBin -
BinSize)) * (pi/180);
        //printf("Angle = %G \n", Angle*180/pi);
    }
    else{
```

```
    /* Method using wave probability step function
*/

    /*  Determine sign */
    /*      variable Asym will determine fractional
distribution of waves coming from the       */
    /*      positive direction (positive direction coming
from left)  -i.e. fractional wave asymmetry */

    if (RunInWindows) {
        AsymRandom = RandZeroToOne()/10;
    } else {
        AsymRandom = RandZeroToOne();
    }

    if ( AsymRandom <= Asym )
        Sign = 1;
    else
        Sign = -1;


    /*  Determine wave angle */
    /*  New formulation - even distribution */

    if (RunInWindows) {
        AngleRandom = RandZeroToOne() / 10;
    } else {
        AngleRandom = RandZeroToOne();
    }

    if (AngleRandom > Highness)
    {
        Angle = Sign * (((AngleRandom)-Highness)/(1-
Highness)) * pi / 4.0;
    }
    else
    {
        Angle = Sign * (((AngleRandom)/Highness)*pi/4.0 +
pi/4.0);
    }

    /*printf("Highness sub: %f AngleRandom: %f \n",
Highness, AngleRandom);
```

```
      printf("WaveAngle sub: %f Angle: %f \n",
180*(Angle)/pi, AngleRandom);*/

    }
    Angle = WaveAngleSign*Angle;
    /*Angle =1.59/180.0*pi;*/

    // Make sure wave angle is between 89 and -89 degrees
    if (Angle >= (89*pi/180)){
        Angle = Angle - (1*pi/180);
    } else if (Angle <= (-89*pi/180)){
        Angle = Angle + (1*pi/180);
    }

    return Angle;

}

float RandAngleFluctuation(void)

// Creates a random fluctuation in the initial wave angle

{
    float Angle;
    int debug = 0; // local debugger

    Angle = (rand() %10) - 5;

    if (debug) printf("Angle = %G \n", Angle);

    return Angle;
}


void FindBeachCells(int YStart)

/* Determines locations of beach cells moving from left to
right direction  */
/* This function will affect and determine the global
arrays:  X[] and Y[]  */
/* This function calls FindNextCell
*/
/* This will define TotalBeachCells for this time step
*/
```

```
{
    int     y, z, xstart;   /* local iterators */


    /* Starting at left end, find the x - value for first
cell that is 'allbeach' */

    xstart = Xmax -1; y = YStart;

    while (AllBeach[xstart][y] == 'n')
    {
    xstart -= 1;
    }

    xstart += 1;            /* Step back to where partially
full beach */

    X[0] = xstart;  Y[0] = YStart;

    if (debug1) printf("FirsX: %3d  FrstY: %3d  z: 0 \n",
X[0], Y[0]);

    z = 0;

    while ((Y[z] < 2*Ymax -1) && (z < MaxBeachLength-1))
    {
    z++;
    NextX = -2;
    NextY = -2;

    FindNextCell(X[z-1], Y[z-1], z-1);
    X[z] = NextX;
    Y[z] = NextY;

    if (debug1) printf("NextX: %3d  NextY: %3d  z: %d \n",
NextX, NextY, z);

    if (PercentFull[X[z]][Y[z]] == 0)
    {
        printf("\nFINDBEACH: PercentFull Zero x: %d y:
%d\n",X[z],Y[z]);
        /*PauseRun(X[z],Y[z],z);*/
    }
```

```
    /* If return to start point or go off left side of
array, going the wrong direction     */
    /* Jump off and start again closer to middle
*/

    if ((NextY < 1) || ((NextY == Y[0])&&(NextX==X[0])) ||
(z > MaxBeachLength -2))
    {
        /*printf("!!!!!!!Fell
Off!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! x = %d !!!!!!!!!!!!",
NextX);*/
        FellOffArray = 'y';
        ZeroVars();
        return;
    }



    if (z > MaxBeachLength - 3)
    {
        printf("???????????  went to end of MaxBeach!!
????");
    }

    }

    TotalBeachCells = z;
    FellOffArray = 'n';

    if (debug1) printf("Total Beach: %d  \n \n",
TotalBeachCells);

}


void FindNextCell(int x, int y, int z)

/* Function to find next cell that is beach moving in the
general positive X direction */
/* changes global variables NextX and NextY, coordinates
for the next beach cell       */
/* This function will use but not affect the global arrays:
AllBeach [][], X[], and Y[] */
```

```
{

    if ( AllBeach[x-1][y] == 'n')
    /* No beach directly beneath cell */
    {
    if ( AllBeach[x][y-1] == 'y' && AllBeach[x][y+1] ==
'n')
        /* If on right side of protuberance */
    {
        if ( AllBeach[x-1][y-1] == 'y' )
        {   /* Move one inshore */
        NextX = x-1; NextY = y; return;
        }
        else if (AllBeach[x-1][y-1] == 'n' )    /* This is
where shadow procedure was */
        {   /* Back and to the left */
        NextX = x-1; NextY = y-1; return;
        }
        printf("Should've found next cell (1): %d, %d \n",
x, y);
        PauseRun(x, y, z);
    }


    else if ( AllBeach[x][y-1] == 'n' && AllBeach[x][y+1]
== 'y')
        /* If on left side of protuberance */
    {
        if ( AllBeach[x+1][y+1] == 'n' && AllBeach[x+1][y]
== 'n')
        /*  Up and right - move around spit end */
        {
        NextX = x+1; NextY = y+1; return;
        }

        else if (  AllBeach[x+1][y] == 'y')
        /*  On underside of regular or diagonally thin spit
*/
        {
        if ( AllBeach[x+1][y-1] == 'n' && AllBeach[x-1][y-
1] == 'n' && X[z-1]>x)
            /* Reaching end of spit - not going in circles
*/
```

```
        {
            NextX = x-1; NextY = y; return;
        }
        else if (AllBeach[x+1][y-1] == 'n')
            /* This is reaching end of spit */
        {
            NextX = x+1; NextY = y-1; return;
        }
        /* Moving along back side of spit */
        {
            NextX = x; NextY = y-1; return;
        }
        }

        else if ( AllBeach[x+1][y+1] == 'y')
        /* we know ( AllBeach[x+1][y] == 'n') */
        /* Moving straight up */
        /* NEW - we still don't want to go in */
        {
        NextX = x+1; NextY = y; return;
        }

        printf("Should've found next cell (2): %d, %d \n",
x, y);
        PauseRun(x, y ,z);
    }


    if  (AllBeach[x][y-1] == 'n' && AllBeach[x][y+1] ==
'n')
        /* Hanging out - nothing on sides or top - maybe on
corner? */
    {
        if (AllBeach[x-1][y+1] == 'y' && AllBeach[x+1][y]
== 'n')
        /* On left corner of protuberence, move right*/
        {
        NextX = x; NextY = y+1; return;
        }

        else if (AllBeach[x+1][y] == 'y' &&
AllBeach[x+1][y-1] == 'n')
        /* Under protuberance, move around to left and up
*/
```

```
        {
        NextX = x+1; NextY = y-1; return;
        }

        else if (AllBeach[x+1][y] == 'y' &&
AllBeach[x+1][y-1] == 'y')
        /* Under protuberance, move to left */
        {
        NextX = x; NextY = y-1; return;
        }
        printf("Should've found next cell (3): %d, %d \n",
x, y);
        PauseRun(x, y, z);
    }

    else if ( AllBeach[x][y-1] == 'y' && AllBeach[x][y+1]
== 'y' )
        /* thin entrance between spits.  Don't even think
about going in there */
        /* (Similar case to over head and underneath -
don't go in */
        /* check to see which way we were coming in - from
below or from side   */
    {   if (X[z-1] > x)
        /* coming from above */
    {
        if (AllBeach[x+1][y+1] == 'n')
        /* Move right and up*/
        {
        NextX = x+1; NextY = y+1; return;
        }
        else if (AllBeach[x+1][y] == 'n')
        /* Straight up*/
        {
        NextX = x+1; NextY = y; return;
        }
        else if (AllBeach[x+1][y-1] == 'n')
        /* Up and left*/
        /* shouldn't need this, this where coming from */
        {
        NextX = x+1; NextY = y-1; return;
        }
    }
    else if (X[z-1] < x)
```

```
        /* coming from below */
    {
        if (AllBeach[x-1][y-1] == 'n')
        /* move down and left*/
        {
        NextX = x-1; NextY = y-1; return;
        }
        else if (AllBeach[x-1][y] == 'n')
        /*move straight down*/
        {
        NextX = x-1; NextY = y; return;
        }
        else if (AllBeach[x-1][y+1] == 'n')
        /*move straight down*/
        /* shouldn't need this, this would be where coming
from*/
        {
        NextX = x-1; NextY = y+1; return;
        }
    }
    printf("Should've found next cell (3.5): %d, %d \n", x,
y);
    PauseRun(x, y, z);
    }

    printf("Should've found next cell (4): %d, %d \n", x,
y);
    PauseRun(x, y, z);
    }


    else if ( AllBeach[x-1][y] == 'y' && AllBeach[x+1][y]
== 'n')
    /* There is beach beneath cell, nothing over the head
*/
    {
    if ( AllBeach[x][y+1] == 'n')
        /*  Adjacent Cell to right is vacant */
    {
        if ( AllBeach[x-1][y+1] == 'y' )
        /* move straight right */
        {
        NextX = x; NextY = y+1; return;
        }
```

```
        else if ( AllBeach[x-1][y+1] == 'n' )
        /* Move down and to right */
        {
        NextX = x-1; NextY = y+1; return;
        }

        printf("Should've found next cell (5): %d, %d \n",
x, y);
        PauseRun(x, y, z);
    }

    else if ( AllBeach[x][y+1] == 'y')
        /*Brad's note : DON'T REALLY NEED TO REPEAT THIS
(WORKS SAME IN BOTH CASES) */
        /* Right neighbor occupied */
    {
        if ( AllBeach[x+1][y+1] == 'n' )
        /* Move up and to right */
        {
        NextX = x+1; NextY = y+1; return;
        }
        else if ( AllBeach[x+1][y+1] == 'y')
        /* Move straight up */
        {
        NextX = x+1; NextY = y; return;
        }

        printf("Should've found next cell (6): %d, %d \n",
x, y);
        PauseRun(x, y, z);
    }

    printf("Should've found next cell (7): %d, %d \n", x,
y);
    PauseRun(x, y, z);
    }


    else if ( (AllBeach[x-1][y] == 'y') &&
(AllBeach[x+1][y] == 'y'))
    /* There is beach behind cell, and over the head don't
want to go in (will be shadowed anyway */
    /* Need to use last cell to find out if going into left
or right enclosure */
```

```
    {
    /* Fill up that nasty piece of work */

    /*  FillUpGap(x,y, y-Y[z-1]);*/


    if (Y[z-1] < y)
        /* Moving towards right, bump up and over the
problem */
    {
        if (AllBeach[x+1][y-1] == 'n')
        /* Move up and to the left */
        {
        NextX = x+1;  NextY = y-1; return;
        }
        else if (AllBeach[x][y-1] == 'n')
        /* Move directly left */
        {
        NextX = x;  NextY = y-1; return;
        }
        else if (AllBeach[x-1][y-1] == 'n')
        /* Move left and down */
        {
        NextX = x-1;  NextY = y-1; return;
        }
        printf("Should've found next cell (8): %d, %d \n",
x, y);
        PauseRun(x, y, z);
    }

    else if (Y[z-1] > y)
        /* Moving towards left, go back right */
    {
        if (AllBeach[x-1][y+1] == 'n')
        /* Move down and to the right */
        {
        NextX = x-1;  NextY = y+1; return;
        }
        else if (AllBeach[x][y+1] == 'n')
        /* Move directly right */
        {
        NextX = x;  NextY = y+1; return;
        }
        else if (AllBeach[x+1][y+1] == 'n')
```

```
        /* Move right and up */
        {
        NextX = x+1;  NextY = y+1; return;
        }
        printf("Should've found next cell (8): %d, %d \n",
x, y);
        PauseRun(x, y, z);
    }
    printf("Should've found next cell (9): %d, %d \n", x,
y);
    PauseRun(x, y, z);
    }
    printf("Should've found next cell (10): %d, %d \n", x,
y);
    PauseRun(x, y, z);
}


/*void FillUpGap(int X, int Y, int LorR)

*  NOT CURRENTLY BEING USED          *
*  When thin entrance is discovered, fill it up *

{
if (AllBeach[X][Y+2*LorR] == 'n')
{
    AllBeach[X][Y+2*LorR] = 'y';
    PercentFull[X][Y+2*LorR] = 1;
    printf("!!!!!!!!!!!!!!!!!\n     FILLEDERUP: %d, %d, %d
\n", X, Y, LorR);
}
}*/


void ShadowSweep(void)

/*  Moves along beach and tests to see if cells are in
shadow       */
/*  This function will use and determine the Global array:
InShadow[]  */
/*  This function will use and adjust the variable:
ShadowXMax    */
/*  This function will use but not adjust the variable:
TotalBeachCells */
```

```
{

    int i;

    /* Find maximum extent of beach to use as a limit for
shadow searching */

    ShadowXMax = XMaxBeach(ShadowXMax) + 3;

    if (debug2) printf("ShadowXMax: %d   XMaxBeach: %d \n",
ShadowXMax, XMaxBeach(ShadowXMax));

    /* Determine if beach cells are in shadow */

    for (i=0;  i <= TotalBeachCells; i++)
    {
    InShadow[i] = FindIfInShadow(i, ShadowXMax);
    }

}


int XMaxBeach(int Max)

/* Finds extent of beach in x direction                  */
/* Starts searching at a point 3 rows higher than input Max
*/
/* Function returns integer value equal to max extent of
'allbeach' */

{
    int xtest, ytest;

    xtest = Max + 2;
    ytest = 0;

    while (xtest > 0)
    {
    while (ytest < 2 *Ymax)
    {
        if (AllBeach[xtest][ytest] == 'y')
        {
        return xtest;
```

```
        }

        ytest++;
    }

    ytest = 0;

    xtest--;
    }

    printf("***** Should've found Xmax for shadow): %d, %d
***** \n", xtest, ytest);

    return Xmax;

}


char FindIfInShadow(int icheck, int ShadMax)

/*  Function to determine if particular cell xin,yin is in
shadow       */
/*  Returns a character 'y' if yes 'n' if no
*/
/*  New 2/04 - use pixelwise march  - make it faster, more
accurate - aa    */
/*  New 3/04 - correctly take acocunt for sideways and
underneath shadows - aa  */
/*  This function will use but not affect the global
arrays:            */
/*  AllBeach[][] and PercentFull[][]                    */
/*  This function refers to global variable:  WaveAngle
*/

{

    float   slope;          /* search line slope - slope of
zero goes staight forward */
    int ysign;          /* holder for going left or right
alongshore */
    float   x,y;            /* holders for 'real' location
of x and y */
    float   xin, yin;       /* starting 'real' locations */
```

```
    int xinint, yinint;     /* integer locations for
starting location */
    int xtestint,ytestint;  /* cell looking at */
    float   xtest,ytest;        /* 'real' location of
testing */
    float   xout,yout;      /* used in AllBeach check -
exit coordinates */
    int NextXInt, NextYInt; /* holder vairables for cell to
check */
    float   Yup, DistanceUp;    /* when going to next x
cell, what other values */
    float   Xside, DistanceSide;    /* when gpoing to next
y cell,other values */
    int     debug2a = 0;        /* local debuggers */
    int debug2b = 0;

    // Use built-in or WRT shadowing routine?
    if (UseShadow == 'n'){
        // Use WRT shadowing routine
        if (HeightRollAvg[icheck] == 0){
            return 'y';
        } else return 'n';
    } else {
    // Use built-in CEM shadwoing routine

    /* convert angle to a slope and the direction of steps
*/
    /* note that for case of shoreline, positive angle will
be minus y direction */
    /*if (icheck == 106) {debug2a = 1;debug2b=1;}*/

    if (WaveAngle == 0.0)
    {
    /* unlikely, but make sure no div by zero */
    slope = 0.00001;
    }
    else if (fabs(WaveAngle) == 90.0)
    {
    slope = 9999.9;
    }
    else
    {
    slope = fabs(tan(WaveAngle));
    }
```

```
    if (WaveAngle > 0)
    ysign = -1;
    else
    ysign = 1;

    if (debug2a) printf("\nI: %d----------x: %d  Y: %d
Wang:  %f Slope: %f sign: %d \n",
            icheck,
X[icheck],Y[icheck],WaveAngle*radtodeg,slope, ysign);

    /* 03/04 AA: depending on local orientations, starting
point will differ */
    /* so go through scenarios */

    xinint = X[icheck];
    yinint = Y[icheck];

    if (AllBeach[xinint-1][yinint] == 'y' ||
((AllBeach[xinint][yinint-1] == 'y') &&
                          (AllBeach[xinint][yinint+1] ==
'y')) )
    /* 'regular condition' */
    /* plus 'stuck in the middle' situation (unlikely
scenario)*/
    {
    xin = xinint + PercentFull[xinint][yinint];
    yin = yinint + 0.5;
    if (debug2a) printf("-- Regular xin: %f  yin:
%f\n",xin,yin);
    }
    else if (AllBeach[xinint][yinint-1] == 'y')
    /* on right side */
    {
    xin = xinint + 0.5;
    yin = yinint + PercentFull[xinint][yinint];
    if (debug2a) printf("-- Right xin: %f  yin:
%f\n",xin,yin);
    }
    else if (AllBeach[xinint][yinint+1] == 'y')
    /* on left side */
    {
    xin = xinint + 0.5;
    yin = yinint + 1.0 - PercentFull[xinint][yinint];
```

```
    if (debug2a) printf("-- Left xin: %f  yin:
%f\n",xin,yin);
    }
    else if (AllBeach[xinint+1][yinint] == 'y')
    /* gotta be on the bottom now */
    {
    xin = xinint + 1 - PercentFull[xinint][yinint];
    yin = yinint + 0.5;
    if (debug2a) printf("-- Under xin: %f  yin:
%f\n",xin,yin);
    }
    else
    /* debug ain't just an insect */
    {
    printf("Shadowstart Broke !!!! ");
    PauseRun(xinint,yinint,icheck);
    }

    x = xin;
    y = yin;

    while ((floor(x) < ShadMax) && (y > Ymax/2) && (y <
3*Ymax/2))
    {
    NextXInt = floor(x) + 1;
    if (ysign > 0)
        NextYInt = floor(y) + 1;
    else
        NextYInt = ceil(y-1);

    /* moving to next whole 'x' position, what is y
position? */
    Yup = y + (NextXInt-x)*slope * ysign;
    DistanceUp = ((Yup - y)*(Yup - y) + (NextXInt -
x)*(NextXInt - x));

    /* moving to next whole 'y' position, what is x
position? */
    Xside = x + fabs(NextYInt - y) / slope;
    DistanceSide = ((NextYInt - y)*(NextYInt - y) + (Xside
- x)*(Xside - x));

    if (debug2a) printf("x: %f  y: %f  X:%d  Y: %d  Yd: %f
DistD: %f Xs: %f DistS: %f\n",
```

```
                x,y,NextXInt,NextYInt,
Yup,DistanceUp,Xside,DistanceSide);

    if (DistanceUp < DistanceSide)
        /* next cell is the up cell */
    {
        x = NextXInt;
        y = Yup;
        xtestint = NextXInt;
        ytestint = floor(y);
        if (debug2a) printf(" up ");
    }
    else
        /* next cell is the side cell */
    {
        x = Xside;
        y = NextYInt;
        xtestint = floor(x);
        ytestint = y + (ysign-1)/2;
        if (debug2a) printf(" side ");
    }

    if (debug2a) printf("   x: %f  y: %f  xtesti: %d
ytesti: %d \n\n",x,y,xtestint,ytestint);


    /* Now Test */
    /* If AllBeach is along the way, will we pass through
'diamond'?    */
    /* Trick - if crossing through the diamond, will change
quadrants   */
    /* Probably won't get to this one, though
*/

    if  (AllBeach[xtestint][ytestint] == 'y')
    {
        /* use same approach to find exit (could make this
modular)  */
        /* don't change 'x' or 'y' and this will be ok
*/
        NextXInt = floor(x) + 1;
        if (ysign > 0)
        NextYInt = floor(y) + 1;
        else
```

```
        NextYInt = ceil(y-1);

        Yup = y + (NextXInt-x)*slope * ysign;
        DistanceUp = ((Yup - y)*(Yup - y) + (NextXInt -
x)*(NextXInt - x));
        Xside = x + fabs(NextYInt - y) / slope;
        DistanceSide = ((NextYInt - y)*(NextYInt - y) +
(Xside - x)*(Xside - x));

        if (DistanceUp < DistanceSide)
        /* next cell is the up cell */
        {
        xout = NextXInt;
        yout = Yup;
        }
        else
        /* next cell is the side cell */
        {
        xout = Xside;
        yout = NextYInt;
        }

        /*if (debug2a) printf("In Allbeach xin: %2.2f yin:
%2.2f xout: %2.2f yout: %2.2f\n",
          x,y,xout,yout);
          if (debug2a) printf("In Allbeach xin: %2.2f yin:
%2.2f xout: %2.2f yout: %2.2f\n",
          (xout-xtestint-0.5),(x-xtestint-0.5),(yout-
ytestint-0.5),(y-ytestint-0.5));*/

        if(( (xout-xtestint-0.5) * (x-xtestint-0.5) < 0 )
|| ((yout-ytestint-0.5) * (y-ytestint-0.5) < 0))
        {
        if (debug2a) printf("  Shaddowded ");
        return 'y';
        }
    }



    /* Compare a partially full cell's x - distance to a
line projected     */
    /* from the starting beach cell's x-distance
*/
```

```
    /* This assumes that beach projection is in x-direction
(not too bad)   */

    else if ( PercentFull[xtestint][ytestint] > 0 )
    {
        if (AllBeach[xtestint-1][ytestint] == 'y' ||
((AllBeach[xtestint][ytestint-1] == 'y') &&

(AllBeach[xtestint][ytestint+1] == 'y')) )
        /* 'regular' condition */
        /* plus 'stuck in the middle' situation (unlikely
scenario) */
        {
        xtest = xtestint + PercentFull[xtestint][ytestint];
        ytest = ytestint + 0.5;

        if (xtest > (xin  + fabs(ytest-yin)/slope) )
        {
            if (debug2b) printf("Top: sl: %f xt: %2.2f xin:
%2.2f yt: %2.2f yin: %2.2f comp: %2.2f > Thing: %2.2f\n",
                    slope, xtest, xin, ytest, yin, xtest,
(xin  + fabs(ytest-yin)/slope));
            return 'y';
        }
        }
        else if (AllBeach[xtestint][ytestint-1] == 'y')
        /* on right side */
        {
        xtest = xtestint + 0.5;
        ytest = ytestint + PercentFull[xtestint][ytestint];

        if (ytest > (yin + (xtest-xin) * slope))
        {
            if (debug2b) printf("Right:  xt: %f  yt: %f
comp: %f > Thing: %f\n",
                    xtest, ytest, ytest,(yin + (xtest-xin)
* slope));
            return 'y';
        }
        }
        else if (AllBeach[xtestint][ytestint+1] == 'y')
        /* on left side */
        {
        xtest = xtestint + 0.5;
```

```
        ytest = ytestint + 1.0 -
PercentFull[xtestint][ytestint];

        if (ytest < (yin + (xtest-xin) * slope))
        {
            if (debug2b) printf("Left:  xt: %f  yt: %f
comp: %f < Thing: %f\n",
                    xtest, ytest, ytest,(yin + (xtest-xin)
* slope));
            return 'y';
        }
        }
        else if (AllBeach[xtestint+1][ytestint] == 'y')
        /* gotta be on the bottom now */
        {
        xtest = xtestint + 1 -
PercentFull[xtestint][ytestint];
        ytest = ytestint + 0.5;

        if (xtest < (xin  + fabs(ytest-yin)/slope) )
        {
            if (debug2b) printf("Bottom:  xt: %f  yt: %f
comp: %f < Thing: %f\n",
                    xtest, ytest, xtest,(xin  + fabs(ytest-
yin)/slope));

            return 'y';
        }
        }
        else
        /* debug ain't just an insect */
        {
        printf("'Shaddows' not responding xin: %f yin: %f
xt: %f  yt: %f  \n",
               xin,yin,xtest, ytest);
        /*PauseRun(xtestint,ytestint,icheck);*/
        }

    }
    }
    return 'n';
}
}
```

```
void  DetermineAngles(void)

/*  Function to determine beach angles for all beach cells
from left to right       */
/*  By convention, the ShorelineAngle will apply to current
cell and right neighbor */
/*  This function will determine global arrays:
*/
/*      ShorelineAngle[], UpWind[], SurroundingAngle[]
*/
/*  This function will use but not affect the following
arrays and values:      */
/*      X[], Y[], PercentFull[][], AllBeach[][], WaveAngle
*/
/*  ADA Revised underside, SurroundingAngle 6/03, 2/04
fixed                */
/*  ADA Revised angle calc 5/04
*/

{

    int i,j,k;              /* Local loop variables */
    int x2int, y2int;       /* shoreline location vars */
    float x2,x1,y2,y1;      /* shoreline location variables
*/
    int debug3a = 0;        /* local debuggr */


    /* Shoreline Angle Calcs - ADA 05/04 - use correct
'point' to do calcs (like in shaddow */
    /* Set first point                                    */
    /* first angle should be regular one - periodic BC's
should also take care      */

    x2 = X[0] + PercentFull[X[0]][Y[0]];
    y2 = Y[0] + 0.5;

    /* Compute ShorelineAngle[]  */
    /*  not equal to TotalBeachCells because angle between
cell and rt neighbor */

    for (i=0 ; i < TotalBeachCells ; i++)
    {
```

```
    x1 = x2;
    y1 = y2;

    x2int = X[i+1];
    y2int = Y[i+1];

    if (AllBeach[x2int-1][y2int] == 'y' ||
((AllBeach[x2int][y2int-1] == 'y') &&
                          (AllBeach[x2int][y2int+1] == 'y'))
&& (AllBeach[x2int+1][y2int] == 'n'))
        /* 'regular condition' - if between  */
        /* plus 'stuck in the middle' situation (unlikely
scenario)*/
    {
        x2 = x2int + PercentFull[x2int][y2int];
        y2 = y2int + 0.5;
        if (debug3a) printf("-- Regular xin: %f  yin:
%f\n",x2,y2);
    }
    else if ((AllBeach[x2int+1][y2int] == 'y') &&
(AllBeach[x2int-1][y2int] == 'y'))
        /* in a sideways nook (or is that a cranny?) */
    {
        x2 = x2int + 0.5;

        if (AllBeach[x2int][y2int-1] == 'y')
        /* right-facing nook */
        {
        y2 = y2int + PercentFull[x2int][y2int];
        }
        else
        /* left-facing nook */
        {
        y2 = y2int + 1.0 - PercentFull[x2int][y2int];
        }
        if (debug3a) printf("-- Nook  xin: %f  yin:
%f\n",x2,y2);
    }
    else if (AllBeach[x2int][y2int-1] == 'y')
        /* on right side */
    {
        x2 = x2int + 0.5;
        y2 = y2int + PercentFull[x2int][y2int];
```

```
        if (debug3a) printf("-- Right xin: %f  yin:
%f\n",x2,y2);
    }
    else if (AllBeach[x2int][y2int+1] == 'y')
        /* on left side */
    {
        x2 = x2int + 0.5;
        y2 = y2int + 1.0 - PercentFull[x2int][y2int];
        if (debug3a) printf("-- Left xin: %f  yin:
%f\n",x2,y2);
    }
    else if (AllBeach[x2int+1][y2int] == 'y')
        /* gotta be on the bottom now */
    {
        x2 = x2int + 1 - PercentFull[x2int][y2int];
        y2 = y2int + 0.5;
        if (debug3a) printf("-- Under xin: %f  yin:
%f\n",x2,y2);
    }
    else
        /* debug ain't just an insect */
    {
        printf("Shadowstart Broke !!!! ");
        PauseRun(x2int,y2int,i+1);
    }


    /* compute angles */

    if (y2 > y1)
    {
        ShorelineAngle[i] = atan((x2 - x1) / (y2 - y1));
        if (debug3) printf("(R) i = %d  X[i]: %d Y[i]: %d
Percent %3f x: %f y: %f Angle:%f  Deg Angle: %f \n",
                    i, X[i], Y[i],
PercentFull[X[i]][Y[i]],x2,y2,ShorelineAngle[i],
ShorelineAngle[i]*180/pi);
    }
    else if (y2 == y1)
    {
        ShorelineAngle[i] = pi/2.0 * (x1 - x2) / fabs(x2 -
x1);
        if (debug3) printf("(G) i = %d  X[i]: %d Y[i]: %d
Percent %3f x: %f y: %f Angle:%f  Deg Angle: %f \n",
```

```
                    i, X[i], Y[i],
PercentFull[X[i]][Y[i]],x2,y2,ShorelineAngle[i],
ShorelineAngle[i]*180/pi);
    }
    else
        /* y2 < y1 */
    {
        ShorelineAngle[i] = atan((x2 - x1) / (y2 - y1)) -
pi;

        if (ShorelineAngle[i] < - pi)
        {
        ShorelineAngle[i] += 2.0 * pi;
        }
        if (debug3) printf("(U) i = %d  X[i]: %d Y[i]: %d
Percent %3f x: %f y: %f Angle:%f  Deg Angle: %f \n",
                    i, X[i], Y[i],
PercentFull[X[i]][Y[i]],x2,y2,ShorelineAngle[i],
ShorelineAngle[i]*180/pi);
    }

    }

    for (k=1 ; k < TotalBeachCells ; k++)
    {
    /* compute SurroundingAngle array */
    /* 02/04 AA averaging doesn't work on bottom of spits
*/
    /* Use trick that x is less if on bottom of spit -
angles might be different signs as well */

    if ((Y[k-1] - Y[k+1] == 2) &&
        (copysign(ShorelineAngle[k-1],ShorelineAngle[k]) !=
ShorelineAngle[k-1]))
    {
        SurroundingAngle[k] = (ShorelineAngle[k-1] +
ShorelineAngle[k]) / 2 + pi;
        if (SurroundingAngle[k] > pi)
        {
        SurroundingAngle[k] -= 2.0 * pi;
        }
        if (debug4) printf("Under: %d\n",k);
    }
    else
```

```
    {
        SurroundingAngle[k] = (ShorelineAngle[k-1] +
ShorelineAngle[k]) / 2;
    }
    }

    /* Determine Upwind/downwind condition
*/
    /* Note - Surrounding angle is based upon left and
right cell neighbors,    */
    /* and is centered on cell, not on right boundary
*/


    if (debug4) printf("\nUp/Down   Wave Angle:%f\n",
WaveAngle * radtodeg);

    for (j=1 ; j < TotalBeachCells  ; j++)
    {
    if (debug4) printf("i: %d  Shad: %c Ang[i]: %3.1f  Sur:
%3.1f  Effect: %3f  ",
                j,InShadow[j], ShorelineAngle[j]*radtodeg,
                SurroundingAngle[j]*radtodeg, (WaveAngle -
SurroundingAngle[j])*radtodeg);

    if ( fabs(WaveAngle - SurroundingAngle[j]) >=
42.0/radtodeg )
    {
        UpWind[j] = 'u';
        if (debug4) printf("U(1)  ");
    }
    else
    {
        UpWind[j] = 'd';
        if (debug4) printf("D(1)  ");
    }

    if (debug4) printf("\n");

    }

}
```

```
void DetermineSedTransport(void)

/*  Loop function to determine which neigbor/situation to
use for sediment transport calcs  */
/*  Once situation is determined, will use function
SedTrans to determine actual transport  */
/*  This function will call SedTrans which will determine
global arrays:            */
/*      VolumeIn[], VolumeOut[]
*/
/*  This function will use but not affect the following
arrays and values:          */
/*      X[], Y[], InShadow[], UpWind[], ShorelineAngle[]
*/
/*          PercentFull[][], AllBeach[][], WaveAngle
*/

{

    int i;          /* Loop variable */
    float ShoreAngleUsed;   /* Temporary holder for
shoreline angle                 */
    int CalcCell;       /* Cell sediment coming from to go
across boundary i           */
    int Next,Last;      /* Indicators so test can go both
left/right                */
    int Correction;     /* Term needed for shoreline angle
and i+1 case, angle stored at i  */
    char UpWindLocal;   /* Local holder for upwind/downwind
condition               */
    char MaxTrans;      /* Do we need to compute using
maximum transport ?         */
    int DoFlux;     /* Skip sed transport calcs (added
02/04 AA)               */

    int     y_coord;    // Y-coordinate of the ith cell
    float   dQs_dtheta; // partial Qs / partial theta
    float   QL;         // Qs on the left border of the
cell (using central breaking data & left side shoreline
angle)
    float   QR;         // Qs on the right border of the
cell (using central breaking data & right side shoreline
angle)
```

```
    float   SLAngleL;   // Shoreline angle on the left side
of the cell border
    float   SLAngleR;   // Shoreline angle on the right
side of the cell border
    float   rho = 1020;     /* kg/m3 - density of water and
dissolved matter                */


    if (debug5) printf("\nSEDTRANS: %d  @  %f \n\n",
CurrentTimeStep, WaveAngle * radtodeg);

    for (i=1 ; i < TotalBeachCells-1 ; i++)
    {
    if (debug5) printf("\n  i: %d  ",i);

    y_coord = Y[i];     // WRT y-coordinate for the ith
cell

    MaxTrans = 'n';

    /*  Is littoral transport going left or right?  */

    if ((WaveAngle-ShorelineAngle[i]) > 0)
    {
        /*  Transport going right, center on cell to left
side of border     */
        /*  Next cell in positive direction, no correction
term needed      */
        CalcCell = i;
        Next = 1;
        Last = -1;
        Correction = 0;

        if (debug5) printf("RT  %d ",CalcCell);
    }
    else
    {
        /*  Transport going left, center on cell to right
side of border    */
        /*  Next cell in negative direction, correction
term needed         */
        CalcCell = i+1;
        Next = -1;
        Last = 1;
```

```
        Correction = -1;

        if (debug5) printf("LT  %d ",CalcCell);
    }

    if (UseWRT == 'y')
    {
        if (AngleRollAvg[y_coord] - ShorelineAngle[y_coord]
>= 0)
        {
            // Transport going right
            CalcCell = i;
            Next = 1;
            Last = -1;
            Correction = 0;

            if (debug5) printf("RT  %d ",CalcCell);
        }
        else
        {
            // Transport going left
            CalcCell = i+1;
            Next = -1;
            Last = 1;
            Correction = -1;

            if (debug5) printf("LT  %d ",CalcCell);
        }
    }


    if ( InShadow[CalcCell] == 'n')
    {

        /*  Adjustment for maximum transport when passing
through 45 degrees        */
        /*  This adjustment is only made for moving from
downwind to upwind conditions  */
        /*                                              */
        /*  purposefully done before shadow adjustment,
only use maxtran when       */
        /*  transition from dw to up not because of shadow
*/
```

```
        /* keeping transition from uw to dw - does not seem
to be big deal (04/02 AA) */

        if ( ((UpWind[CalcCell] == 'd') &&
(UpWind[CalcCell+Next] == 'u') &&
          (InShadow[CalcCell + Next] == 'n')) ||
         ((UpWind[CalcCell+Last] == 'u') &&
(UpWind[CalcCell] == 'd')
          && (InShadow[CalcCell+Last] == 'n')) )
        {
        MaxTrans = 'y';
        if (debug5) printf("MAXTRAN  ");
        }


        /*  Upwind/Downwind adjustment Make sure sediment
is put into shadows       */
        /*  If Next cell is in shadow, use UpWind condition
*/

        DoFlux = 1;
        UpWindLocal = UpWind[CalcCell];

        if (InShadow[CalcCell+Next] == 'y')
        {
        UpWindLocal = 'u';
        if (debug5) printf("U(2)  ");
        }

        /*  If coming out of shadow, downwind should be
used        */
        /*  HOWEVER- 02/04 AA - if high angle, will result
in same flux in/out problem */
        /*      solution  - no flux for high angle waves */

        if ((InShadow[CalcCell+Last] == 'y') &&(UpWindLocal
== 'u'))
        {
        DoFlux = 0;
        if (debug5) printf("U(X) NOFLUX \n");

        }
```

```
        // If using WRT, determine if upwind or downwind is
to be used
        // if (UseWRT == 'y')
        // {
            // //Calculate shoreline angles on the cell
borders
            // SLAngleR = ShorelineAngle[i];
            // SLAngleL = ShorelineAngle[i-1];

            // // Calculate Qs on the borders of the ith
cell
            // QR =
1.1*rho*Raise(g,3.0/2.0)*Raise(HeightRollAvg[y_coord],2.5)*
cos(AngleRollAvg[y_coord] -
                // SLAngleR)*sin(AngleRollAvg[y_coord] -
SLAngleR)*TimeStep;
            // QL =
1.1*rho*Raise(g,3.0/2.0)*Raise(HeightRollAvg[y_coord],2.5)*
cos(AngleRollAvg[y_coord] -
                // SLAngleL)*sin(AngleRollAvg[y_coord] -
SLAngleL)*TimeStep;

            // //Calculate dQs_dtheta
            // dQs_dtheta = (QR - QL) / (SLAngleR -
SLAngleL);

            // //Determine upwind/downwind
            // if (dQs_dtheta > 0)
            // {
                // UpWindLocal = 'u';
            // }
            // else
            // {
                // UpWindLocal = 'd';
            // }

        // }

        /*  Use upwind or downwind shoreline angle for
calcs            */

        if (UpWindLocal == 'u')
        {
```

```
        ShoreAngleUsed =
ShorelineAngle[CalcCell+Last+Correction];
        if (debug5) printf("UP  ShoreAngle: %3.1f  ",
ShoreAngleUsed * radtodeg);
        }
        else if (UpWindLocal == 'd')
        {
        ShoreAngleUsed =
ShorelineAngle[CalcCell+Correction];
        if (debug5) printf("DN  ShoreAngle: %3.1f  ",
ShoreAngleUsed *radtodeg);
        }


        /* !!! Do not do transport on unerneath c'cause it
gets all messed up */
        if (fabs(ShoreAngleUsed) > SedTansLimit/radtodeg)
        {
        DoFlux = 0;
        }

        /* Send to SedTrans to calculate VolumeIn and
VolumeOut*/


        /* printf("i = %d  Cell: %d NextCell: %d Angle: %f
Trans Angle: %f\n",
           i, CalcCell, CalcCell+Next,
ShoreAngleUsed*180/pi, (WaveAngle -
ShoreAngleUsed)*180/pi); */

        if (debug5) printf("From: %d  To: %d  TransAngle
%3.1f", CalcCell, CalcCell+Next,
                    (WaveAngle - ShoreAngleUsed) *
radtodeg);

        if (DoFlux)
        {
        SedTrans(CalcCell, CalcCell+Next, ShoreAngleUsed,
MaxTrans);
        }
    }

    }
```

```
}


void SedTrans(int From, int To, float ShoreAngle, char
MaxT)

/*  This central function will calcualte the sediment
transported from the cell at From to  */
/*  the cell at To, using the input ShoreAngle
*/
/*  This function will caluclate and determine the global
arrays:                */
/*      VolumeIn[], VolumeOut[], and Qs[]
*/
/*  This function does not use any other arrays
*/
/*  This function will use the global values defining the
wave field:                */
/*  WaveAngle, Period, OffShoreWvHt
*/
/*  Revised 6/02 - New iterative calc for refraction and
breaking, parameters revised       */
{

    /* Coefficients - some of these are important*/

    //float StartDepth = 3*OffShoreWvHt; /* m, depth to
begin refraction calcs (needs to be beyond breakers)    */
    float   StartDepth = MaxDepth + (.001*StartFlatBathy);
// Ensures starting AWT at the edge of the WRT domain
    float   RefractStep = .2;   /* m, step size to iterate
depth for refraction calcs           */

    /* Variables */

    int     Broken = 0;     /* is wave broken yet?
*/
    float   AngleDeep;      /* rad, Angle of waves to shore
at inner shelf  */
    float   Depth = StartDepth; /* m, water depth for
current iteration     */
    float   DepthBreak;     // Depth at wave breaking
```

```
    float   Angle;          /* rad, calculation angle
*/
    float   CDeep;          /* m/s, phase velocity in deep
water        */
    float   LDeep;          /* m, offhsore wavelength
*/
    float   C;          /* m/s, current step phase velocity
*/
    float   kh;         /* wavenumber times depth
*/
    float   n;          /* n                           */
    float   WaveLength;     /* m, current wavelength
*/
    float   WvHeight;       /* m, current wave height
*/
    float   VolumeAcrossBorder; /* m3/day
*/
    int     y_coord;        // y-coordinate of the beach
cell "From"
    int     counter = 0;
    int     debug6a = 1;    // Print volume only
    int     debug19a = 0;   // local debugger
    int     debug19b = 0;   // local debugger

    /* Primary assumption is that waves refract over shore-
parallel contours            */
    /* New algorithm 6/02 iteratively takes wiave onshore
until they break, then computes Qs    */
    /* See notes 06/05/02
*/

    // Set MaxDepth
    if (UseWRT == 'y'){
        StartDepth = MaxDepth + (.001*StartFlatBathy); //
Ensures starting AWT at the edge of the WRT domain
    } else {
        StartDepth = MaxDepth;
    }

    // Prerefract using AWT?
    if (UseAWTOffshore == 'y'){
        AngleDeep = StartAngle - ShoreAngle;
        if (debug6) printf("Wave Angle %2.2f Shore Angle
%2.2f    ",StartAngle*radtodeg, ShoreAngle*radtodeg);
```

```
    } else {
        AngleDeep = WaveAngle - ShoreAngle;
        StartHeight = OffShoreWvHt;
        if (debug6) printf("Wave Angle %2.2f Shore Angle
%2.2f    ",WaveAngle*radtodeg, ShoreAngle*radtodeg);
    }

    if (MaxT == 'y')
    {
    AngleDeep = 42.0 / radtodeg;
    }
    if (debug6) printf("Deep Tranport Angle %2.2f
\n\n",AngleDeep*radtodeg);

    /*  Don't do calculations if over 90 degrees, should be
in shadow  */
    // But only if you're NOT using WRT

    if ((AngleDeep > 0.995*pi/2.0 || AngleDeep < -
0.995*pi/2.0) && (UseWRT == 'n'))
    {
    return;
    }

    else
    {

    y_coord = Y[From];

    SLAngle[y_coord] = ShoreAngle;

    if (UseWRT == 'y'){

    // Use breaking wave height and angle data returned
from WRT for the y-position of Y[From]
    //WvHeight = avg_breaking_height[y_coord];
    //Angle = breaking_angles[y_coord] - pi;
    WvHeight = HeightRollAvg[y_coord];
    //Angle = -(AngleRollAvg[y_coord] - pi) + ShoreAngle;
    Angle = AngleRollAvg[y_coord] - ShoreAngle;

    if (debug19a) printf("y_coord = %i \n", y_coord);
    if (debug19a) printf("WvHeight = %G \n", WvHeight);
```

```
    if (debug19a) printf("Angle = %G \n \n",
Angle*radtodeg);

    if (debug19b) printf("breaking_heights = %G \n",
breaking_heights[y_coord]);
    if (debug19b) printf("avg_breaking_height = %G \n",
avg_breaking_height[y_coord]);
    if (debug19b) printf("breaking_angles = %G \n",
breaking_angles[y_coord]);
    if (debug19b) printf("avg_breaking_angle = %G \n",
avg_breaking_angle[y_coord]);
    if (debug19b) printf("BkAngle = %G \n \n",
BkAngle[y_coord]);

    } else { // Use the Aston Wave Transformation

    /* Calculate Deep Water Celerity & Length, Komar 5.11 c
= gT / pi, L = CT   */

    CDeep = g * Period / (2.0 * pi);
    LDeep = CDeep * Period;
    if (debug6) printf("CDeep = %2.2f LDeep = %2.2f
\n",CDeep, LDeep);

    while(!Broken)
    {
        /* non-iterative eqn for L, from Fenton & McKee
*/

        WaveLength = LDeep *
Raise(tanh(Raise(Raise(2.0*pi/Period,2)*Depth/g,.75)),2.0/3
.0);
        C = WaveLength/Period;
        if (debug6) printf("DEPTH: %2.2f Wavelength = %2.2f
C = %2.2f ", Depth, WaveLength,C);

        /* Determine n = 1/2(1+2kh/tanh(kh)) Komar 5.21
*/
        /* First Calculate kh = 2 pi Depth/L  from k = 2
pi/L       */

        kh =  pi * Depth / WaveLength;
        n =0.5 * ( 1 + 2.0 * kh / sinh(2.0*kh));
        if (debug6) printf("kh: %2.3f  n: %2.3f ", kh, n);
```

```
        /* Calculate angle, assuming shore parallel
contours and no conv/div of rays    */
        /* from Komar 5.47                                */

        Angle = asin(C/CDeep * sin(AngleDeep));
        if (debug6) printf("Angle: %2.2f",Angle*radtodeg);

        /* Determine Wave height from refract calcs - Komar
5.49            */

        WvHeight = StartHeight *
Raise(CDeep*cos(AngleDeep)/(C*2.0*n*cos(Angle)),.5);

        AWTWvHeight[counter][From] = WvHeight;

        if (debug6) printf(" WvHeight : %2.3f\n",WvHeight);

        if (WvHeight > Depth*KBreak){
        Broken = 1;
        counter = 0;
        DepthBreak = Depth;

        HeightRollAvg[y_coord] = WvHeight;
        AngleRollAvg[y_coord] = Angle + ShoreAngle;
        }
        else if (Depth == RefractStep)
        {
        Broken = 1;
        Depth -= RefractStep;
        counter = 0;
        DepthBreak = Depth;

        HeightRollAvg[y_coord] = WvHeight;
        AngleRollAvg[y_coord] = Angle + ShoreAngle;
        }
        else
        Depth -= RefractStep;
        counter++;
    }

    }

    /* Now Determine Transport */
```

```
    /* eq. 9.6b (10.8) Komar, including assumption of sed
density = 2650 kg/m3          */
    /* additional accuracy here will not improve an already
suspect eqn for sed transport   */
    /* (especially with poorly constrained coefficients),
*/
    /* so no attempt made to make this a more perfect
imperfection              */

    if (UseVariableCERC == 'y'){

        if (UseBailard == 'y'){ // Use the Bailard equation
to calculate K
            K = 0.05 + 2.6 * (sin (2*Angle)) * (sin
(2*Angle)) + 0.007 * (KBreak/2) * sqrt(g*DepthBreak) /
FallVel;
        }
        if (UseBattjesStive == 'y'){ // Use the Battjes and
Stive (1985) equation to calculate KBreak
            KBreak = 0.5 + 0.4 * tanh(33 *
OffShoreWvHt/LDeep);
        }
        if (debug22) printf("K = %f , KBreak = %f \n", K,
KBreak);

        VolumeAcrossBorder = fabs( K * (rho * sqrt(g)) /
(16 * sqrt(KBreak) *
                    (rho_s - rho) * (1 - porosity))
*Raise(WvHeight,2.5)*

sin(2*Angle)*PerSecondToPerDay*TimeStep); // This is in m^3
per time step



    } else if (UseKamphius == 'y'){

        VolumeAcrossBorder = fabs(7.3 * Raise(WvHeight,2) *
Raise(PeakPeriod,1.5) *
                Raise(ShorefaceSlope,0.75) * Raise(DFifty,-
0.25) * Raise(sin(2*Angle),0.6));

    } else { // Use default sediment transport
```

```
        VolumeAcrossBorder =
fabs(1.1*rho*Raise(g,3.0/2.0)*Raise(WvHeight,2.5)*
                        cos(Angle)*sin(Angle)*TimeStep);
    }

    VolumeOut[From] = VolumeOut[From] + VolumeAcrossBorder;

    VolumeIn[To] = VolumeIn[To] + VolumeAcrossBorder;

    if (debug6) printf("VolumeAcrossBorder: %f
",VolumeAcrossBorder);
    if (debug6) printf("VolumeIn : %f ",VolumeIn[To]);
    if (debug6) printf("VolumeOut : %f
\n\n",VolumeOut[From]);

    //Qs[From] = VolumeAcrossBorder;
    //Qs[From] = VolumeOut[From];
    //Qs[From] =
1.1*rho*Raise(g,3.0/2.0)*Raise(WvHeight,2.5)*cos(Angle)*sin
(Angle)*TimeStep;
    Qs[y_coord] =
1.1*rho*Raise(g,3.0/2.0)*Raise(WvHeight,2.5)*cos(Angle)*sin
(Angle)*TimeStep;

    //Mu[From] =
1.1*rho*Raise(g,3.0/2.0)*Raise(WvHeight,2.5)*cos(2*Angle)*T
imeStep;
    // Mu[From] = -
1.1*rho*Raise(g,3.0/2.0)*Raise(WvHeight,2.5)*TimeStep
        // *(sin(Angle)*sin(Angle)-
cos(Angle)*cos(Angle))/DepthShoreface;

    RelAngle[y_coord] = Angle;

    }
}


void TransportSedimentSweep(void)

/*  Sweep through cells to place transported sediment
*/
/*  Call function AdjustShore() to move sediment.
*/
```

```
/*  If cell full or overempty, call OopsImFull or
OopsImEmpty()          */
/*  This function doesn't change any values, but the
functions it calls do  */
/*  Uses but doesn't change:  X[], Y[], PercentFull[]
*/
/*  sweepsign added to ensure that direction of actuating
changes does not       */
/*      produce unwanted artifacts (e.g. make sure
symmetrical           */

{

    int i,ii;
    int sweepsign;

    if (RunInWindows) {
        if (RandZeroToOne()*2/10 > 1)
        {
        sweepsign = 1;
        if (debug7) printf("L  ");
        }
        else
        {
        sweepsign = 0;
        if (debug7) printf("R  ");
        }
    } else {
        if (RandZeroToOne()*2 > 1)
        {
        sweepsign = 1;
        if (debug7) printf("L  ");
        }
        else
        {
        sweepsign = 0;
        if (debug7) printf("R  ");
        }
    }


    if (debug7) printf("\n\n TransSedSweep  Ang %f  %d\n",
WaveAngle * radtodeg, CurrentTimeStep);
```

```
    for (i=0; i < TotalBeachCells-1 ; i++)
    {

    if (sweepsign == 1)
        ii = i;
    else
        ii = TotalBeachCells-1-i;

    if (debug7) printf("i: %d  ss: %d  X: %d  Y: %d  In:
%.1f  Out: %.1f\n", ii, sweepsign,
                X[i], Y[i], VolumeIn[i], VolumeOut[i]);

    AdjustShore(ii);

    if (PercentFull[X[ii]][Y[ii]] < 0)
    {
        OopsImEmpty(X[ii],Y[ii]);
    }
    else if (PercentFull[X[ii]][Y[ii]]> 1)
    {
        OopsImFull(X[ii],Y[ii]);
    }
    }

}

void AdjustShore(int i)

/*  Complete mass balance for incoming and ougoing sediment
*/
/*  This function will change the global data array
PercentFull[][]     */
/*  Uses but does not adjust arrays:
*/
/*      VolumeIn[], VolumeOut[], X[], Y[], ShorelineAngle[]
*/
/*  Uses global variables: ShelfSlope, CellWidth,
ShorefaceSlope, InitialDepth  */
/*  NEW - AA 05/04 fully utilize shoreface depths
*/

{

    float   Depth;      /* Depth of convergence*/
```

```
    float   DeltaArea;  /* Holds change in area for cell
(m^2)*/
    float   Distance;   /* distance from shore to intercept
of equilib. profile and overall slope (m)*/
    float   PercentIn;
    float   PercentOut;
    float   PercentSum;
    int Xintint, Yintint;   /* integer representing
location shoreface cell */
    float   Xintfloat,Yintfloat;    /* floaters for
shoreface cell */

    /* variables for loop */
    float   slope;          /* slope of zero goes staight
back */
    int ysign;          /* holder for going left or right
alongshore */
    float   x,y;            /* holders for 'real' location
of x and y */
    int xtest,ytest;        /* cell looking at */
    int NextXInt, NextYInt; /* holder vairables for cell to
check */
    float   Ydown, DistanceDown;    /* when going to next x
cell, what other values */
    float   Xside, DistanceSide;    /* when gpoing to next
y cell,other values */
    int     ShorefaceFlag;      /* flag to see if started
intersecting shoreface cells */

    if (VolumeIn[i] <= VolumeOut[i])
    /* eroding, just have to use shoreface depth */
    {
    Depth = DepthShoreface;
    }
    else
    /* accreting, good god */
    {
    /* where should we intersect shoreface depth ? */

    /* uncomplicated way - assume starting in middle of
cell */
    Distance = DepthShoreface/CellWidth/ShorefaceSlope;
    Xintfloat = X[i] + 0.5 + Distance *
cos(SurroundingAngle[i]);
```

```
    Xintint = floor(Xintfloat);
    Yintfloat = Y[i] + 0.5 - Distance *
sin(SurroundingAngle[i]);
    Yintint = floor(Yintfloat);

    if (debug7a)printf("xs: %d  ys: %d  Xint: %f Xint:%d
Yint: %f Yint: %d  Dint: %f SAng: %f Sin = %f\n",

X[i],Y[i],Xintfloat,Xintint,Yintfloat,Yintint,CellDepth[Xin
tint][Yintint],SurroundingAngle[i]*radtodeg,sin(Surrounding
Angle[i]));


    if ((Yintint < 0) || (Yintint > 2*Ymax))
    {
        Depth = DepthShoreface;
        if ((Yintint > Ymax/2) && (Yintint < 3/2*Ymax))
        {
        printf("Periodic Boundary conditions and Depth Out
of Bounds");
        PauseRun(X[i],Y[i],i);
        }
    }
    else if ((Xintint < 0) || (Xintint > Xmax))
    {
        Depth = DepthShoreface;
        printf("-- Warning - depth location off of x array:
X %d Y %d",Xintint,Yintint);
        PauseRun(X[i],Y[i],i);
    }
    else if (CellDepth[Xintint][Yintint] <= 0)
        /* looking back on land */
    {
        Depth = DepthShoreface;
        if (debug7a) printf("=== Shoreface is Shore, eh?
Accreti:  xs: %d  ys: %d  Xint:%d  Yint: %d  Dint: %f \n",

X[i],Y[i],Xintint,Yintint,CellDepth[Xintint][Yintint]);
    }
    else if (CellDepth[Xintint][Yintint] < DepthShoreface)
    {
        printf("Shallow but underwater Depth
%f",CellDepth[Xintint][Yintint]);
        PauseRun(Xintint,Yintint,01);
```

```
    }
    else
    {
        Depth = CellDepth[Xintint][Yintint];


        /* That was the easy part - now we need to 'fix'
all cells towards shoreface */
        /* probably due to accretion from previous moving
forward */
        /* reuse some of the overwash checking code here */


        if (SurroundingAngle[i] == 0)
        {
        /* unlikely, but make sure no div by zero */
        slope = 0.00001;
        }
        else if (fabs(SurroundingAngle[i]) == 90.0)
        {
        slope = 9999.9;
        }
        else
        {
        slope = fabs(tan(SurroundingAngle[i]));
        }

        if (SurroundingAngle[i] > 0)
        ysign = 1;
        else
        ysign = -1;

        x = Xintfloat;
        y = Yintfloat;
        xtest = Xintint;
        ytest = Yintint;
        ShorefaceFlag = 0;

        while (( CellDepth[xtest][ytest] > DepthShoreface)
&& !(ShorefaceFlag))
        {
        NextXInt = ceil(x) -1;
        if (ysign > 0)
            NextYInt = floor(y) + 1;
```

```
        else
            NextYInt = ceil(y-1);

        /* moving to next whole 'x' position, what is y
position? */
        Ydown = y + (x - NextXInt)*slope * ysign;
        DistanceDown = Raise(((Ydown - y)*(Ydown - y) +
(NextXInt - x)*(NextXInt - x)),.5);

        /* moving to next whole 'y' position, what is x
position? */
        Xside = x - fabs(NextYInt - y) / slope;
        DistanceSide = Raise(((NextYInt - y)*(NextYInt - y)
+ (Xside - x)*(Xside - x)),.5);



        if (DistanceDown < DistanceSide)
            /* next cell is the down cell */
        {
            x = NextXInt;
            y = Ydown;
            xtest = NextXInt-1;
            ytest = floor(y);
        }
        else
            /* next cell is the side cell */
        {
            x = Xside;
            y = NextYInt;
            xtest = floor(x);
            ytest = y + (ysign-1)/2;
        }

        if (CellDepth[xtest][ytest] > DepthShoreface)
            /* Deep hole - fill 'er in - mass came from
previous maths */
        {

            if (debug7a) printf("=== Deep Hole, eh?
Accreti:  xs: %d  ys: %d  Xint:%d  Yint: %d  Dint: %f
Xfill: %d Yfill: %d Dt: %f\n",
```

```
X[i],Y[i],Xintint,Yintint,CellDepth[Xintint][Yintint],xtest
,ytest,
                    CellDepth[xtest][ytest]);
            CellDepth[xtest][ytest] = DepthShoreface;

            /*PauseRun(xtest,ytest,i);*/


        }
        else
            /* stop checking - ostensibly we have hit the
shoreface or shore */
        {
            ShorefaceFlag = 1;

            if (PercentFull[xtest][ytest] > 0)
            /* not good - somehow crossing the shore */
            {
            /*printf("Shoreface is the Beach !!!??");*/
            /*PauseRun(xtest,ytest,-1);*/
            }
        }
        }
    }
    }

    Depth += LandHeight;


    if (Depth < DepthShoreface)
    {
    printf("too deep");
    PauseRun(x,y,-1);
    }

    DeltaArea = (VolumeIn[i] - VolumeOut[i])/Depth;

    PercentFull[X[i]][Y[i]] +=
DeltaArea/(CellWidth*CellWidth);

    PercentIn = VolumeIn[i]/(CellWidth*CellWidth*Depth);
    PercentOut = VolumeOut[i]/(CellWidth*CellWidth*Depth);
    PercentSum = DeltaArea/(CellWidth*CellWidth);
```

```
    if (debug7) printf("  In: %2.4f  Out: %2.4f  Sum:
%2.4f\n", PercentIn, PercentOut, PercentSum);

}


void OopsImEmpty(int x, int y)

/*  If a cell is under-full, this will find source for
desparity and move brach in  */
/*  Function completly changed 5/21/02 sandrevt.c
*/
/*          New Approach - steal from all neighboring
AllBeach cells        */
/*      Backup plan - steal from all neighboring percent
full > 0       */
/*  Function adjusts primary data arrays:
*/
/*      AllBeach[][] and PercentFull[][]
*/


{

    int emptycells = 0;
    int emptycells2 = 0;

    if (debug8) printf("\n      OOPS I'm EMPTY!  X: %d  Y:
%d Per: %f ", x, y, PercentFull[x][y]);

    /* find out how many AllBeaches to take from */

    if (AllBeach[x-1][y] == 'y')
    emptycells += 1;
    if (AllBeach[x+1][y] == 'y')
    emptycells += 1;
    if (AllBeach[x][y-1] == 'y')
    emptycells += 1;
    if (AllBeach[x][y+1] == 'y')
    emptycells += 1;

    if (emptycells > 0)
    {
```

```
    /* Now Move Sediment */

    if (AllBeach[x-1][y] == 'y')
    {
        PercentFull[x-1][y] +=
(PercentFull[x][y])/emptycells;
        AllBeach[x-1][y] = 'n';
        if (debug8) printf ("  MOVEDBACK");
    }
    if (AllBeach[x+1][y] == 'y')
    {
        PercentFull[x+1][y] +=
(PercentFull[x][y])/emptycells;
        AllBeach[x+1][y] = 'n';
        if (debug8) printf ("  MOVEDUP");
    }
    if (AllBeach[x][y-1] == 'y')
    {
        PercentFull[x][y-1] +=
(PercentFull[x][y])/emptycells;
        AllBeach[x][y-1] = 'n';
        if (debug8) printf ("  MOVEDLEFT");
        /*if (debug8) PauseRun(x,y,-1);*/
    }
    if (AllBeach[x][y+1] == 'y')
    {
        PercentFull[x][y+1] +=
(PercentFull[x][y])/emptycells;
        AllBeach[x][y+1] = 'n';
        if (debug8) printf ("  MOVEDRIGHT");
        /*if (debug8) PauseRun(x,y,-1);*/
    }
    }
    else
    {
    /* No full neighbors, so take away from partially full
neighbors */

    if (PercentFull[x-1][y] > 0)
        emptycells2 += 1;
    if (PercentFull[x+1][y] > 0)
        emptycells2 += 1;
    if (PercentFull[x][y-1] > 0)
        emptycells2 += 1;
```

```
    if (PercentFull[x][y+1] > 0)
        emptycells2 += 1;

    if (emptycells2 > 0)
    {

        if (PercentFull[x-1][y] > 0)
        {
        PercentFull[x-1][y] +=
(PercentFull[x][y])/emptycells2;
        if (debug8) printf ("  NOTFULL MOVEDBACK");
        }
        if (PercentFull[x+1][y] > 0)
        {
        PercentFull[x+1][y] +=
(PercentFull[x][y])/emptycells2;
        if (debug8) printf ("  NOTFULL MOVEDUP");
        }
        if (PercentFull[x][y-1] > 0)
        {
        PercentFull[x][y-1] +=
(PercentFull[x][y])/emptycells2;
        if (debug8) printf ("  NOTFULL MOVEDLEFT");
        /*if (debug8) PauseRun(x,y,-1);*/
        }
        if (PercentFull[x][y+1] > 0)
        {
        PercentFull[x][y+1] +=
(PercentFull[x][y])/emptycells2;
        if (debug8) printf ("  NOTFULL MOVEDRIGHT");
        /*if (debug8) PauseRun(x,y,-1);*/
        }
    }
    else
    {
        printf("@@@ Didn't find anywhere to steal sand
from!! x: %d  y: %d\n",x,y);
        PauseRun(x,y,-1);
    }

    }

    AllBeach[x][y] = 'n';
    PercentFull[x][y] = 0.0;
```

```
    CellDepth[x][y] = DepthShoreface;

    if (debug8) printf("\n");

}


void OopsImFull(int x, int y)

/*  If a cell is overfull, push beach out in new direction
*/
/*  Completely revised 5/20/02 sandrevt.c to resolve 0%
full problems, etc.     */
/*  New approach:   put sand wherever 0% full in adjacent
cells         */
/*          if not 0% full, then fill all non-allbeach
*/
/*  Function adjusts primary data arrays:
*/
/*      AllBeach[][] and PercentFull[][]
*/


{

    int fillcells = 0;
    int fillcells2 = 0;

    if (debug8) printf("\n      OOOPPPS I'M FULLL: X: %d
Y: %d Per: %f  ==", x, y, PercentFull[x][y]);
    /*if (debug8) PrintLocalConds(x,y,-1);*/

    /* find out how many cells will be filled up    */

    if (PercentFull[x-1][y] == 0.0)
    fillcells += 1;
    if (PercentFull[x+1][y] == 0.0)
    fillcells += 1;
    if (PercentFull[x][y-1] == 0.0)
    fillcells += 1;
    if (PercentFull[x][y+1] == 0.0)
    fillcells += 1;

    if (fillcells != 0)
```

```
    {
    /* Now Move Sediment */

    if (PercentFull[x-1][y] == 0.0)
    {
        PercentFull[x-1][y] += (PercentFull[x][y]-
1)/fillcells;
        CellDepth[x-1][y] = - LandHeight;
        if (debug8) printf ("  MOVEDBACK");
    }
    if (PercentFull[x+1][y] == 0.0)
    {
        PercentFull[x+1][y] += (PercentFull[x][y]-
1)/fillcells;
        CellDepth[x+1][y] = - LandHeight;
        if (debug8) printf ("  MOVEDUP");
    }
    if (PercentFull[x][y-1] == 0.0)
    {
        PercentFull[x][y-1] += (PercentFull[x][y]-
1)/fillcells;
        CellDepth[x][y-1] = - LandHeight;
        if (debug8) printf ("  MOVEDLEFT");
        /*if (debug8) PauseRun(x,y,-1);*/
    }
    if (PercentFull[x][y+1] == 0.0)
    {
        PercentFull[x][y+1] += (PercentFull[x][y]-
1)/fillcells;
        CellDepth[x][y+1] = - LandHeight;
        if (debug8) printf ("  MOVEDRIGHT");
        /*if (debug8) PauseRun(x,y,-1);*/
    }
    }
    else
    {
    /* No fully empty neighbors, so distribute to partially
full neighbors */

    if (PercentFull[x-1][y] < 1)
        fillcells2 += 1;
    if (PercentFull[x+1][y] < 1)
        fillcells2 += 1;
    if (PercentFull[x][y-1] < 1)
```

```
        fillcells2 += 1;
    if (PercentFull[x][y+1] < 1)
        fillcells2 += 1;

    if (fillcells2 > 0)
    {

        if (PercentFull[x-1][y] < 1)
        {
        PercentFull[x-1][y] += (PercentFull[x][y]-
1)/fillcells2;
        if (debug8) printf ("  MOVEDBACK");
        }
        if (PercentFull[x+1][y] < 1)
        {
        PercentFull[x+1][y] += (PercentFull[x][y]-
1)/fillcells2;
        if (debug8) printf ("  MOVEDUP");
        }
        if (PercentFull[x][y-1] < 1)
        {
        PercentFull[x][y-1] += (PercentFull[x][y]-
1)/fillcells2;
        if (debug8) printf ("  MOVEDLEFT");
        }
        if (PercentFull[x][y+1] < 1)
        {
        PercentFull[x][y+1] += (PercentFull[x][y]-
1)/fillcells2;
        if (debug8) printf ("  MOVEDRIGHT");
        }
    }
    else
    {
        if (debug8) printf("Nobody wants our sand!!! x: %d
y: %d Per: %f\n",x,y,PercentFull[x][y]);
        /*PauseRun(x,y,-1);*/
    }

    }

    AllBeach[x][y] = 'y';
    PercentFull[x][y] = 1.0;
    CellDepth[x][y] = - LandHeight;
```

```
    if (debug8) printf("\n");

}

void FixBeach(void)

/* Hopefully addresses strange problems caused by
filling/emptying of cells */
/* Looks at entire data set                          */
/* Find unattached pieces of sand and moves them back to
the shore      */
/* Takes care of 'floating bits' of sand
*/
/* Also takes care of over/under filled beach pieces
*/
/* Revised 5/21/02 to move sand to all adjacent neighbors
sandrevt.c        */
/* Changes global variable PercentFull[][]
*/
/* Uses but does not change AllBeach[][]
*/
/* sandrevx.c - added sweepsign to reduce chances of
asymmetrical artifacts */


{

    int i,x,y,sweepsign;
    int FixXMax;
    int fillcells3 = 0;

    /*if (debug9) printf("\n\nFIXBEACH      %d     %f\n",
CurrentTimeStep, WaveAngle*radtodeg);*/

    if (RunInWindows) {
        if (RandZeroToOne()*.2 > 1)
        {
        sweepsign = 1;
        if (debug9) printf("fixL  ");
        }
        else
```

```
        {
        sweepsign = 0;
        if (debug9) printf("fixR  ");
        }
    } else {
        if (RandZeroToOne()*2 > 1)
        {
        sweepsign = 1;
        if (debug9) printf("fixL  ");
        }
        else
        {
        sweepsign = 0;
        if (debug9) printf("fixR  ");
        }
    }



    FixXMax = ShadowXMax +
ceil(DepthShoreface/CellWidth/ShorefaceSlope) +3;
    if (FixXMax > Xmax)
    FixXMax = Xmax;

    for (x = FixXMax; x >= 0 ; x--)
    {
    for (i = 0; i <= 2*Ymax ; i++)
    {

        if (sweepsign == 1)
        y = i;
        else
        y = 2*Ymax-i;

        /* ye olde depth fix */
        if ((PercentFull[x][y] <= 0) && (CellDepth[x][y] >
DepthShoreface) &&
        (CellDepth[x-1][y] == DepthShoreface))
        {
        if ((CellDepth[x+1][y] == DepthShoreface) &&
(CellDepth[x][y-1] == DepthShoreface)
            && (CellDepth[x][y+1] == DepthShoreface))
        {
            /* Fill Hole */
```

```
            CellDepth[x][y] = DepthShoreface;
        }
        }
        if (PercentFull[x][y]> 100)
        {
        printf("too full");
        PercentFull[x][y] = 0;
        PauseRun(x,y,-1);
        }



        /* Take care of situations that shouldn't exist */

        if (PercentFull[x][y] < 0)
        {
        AllBeach[x][y] = 'n';
        if (debug9 && y != 0) printf("\nUnder 0 Percent X:
%d  Y: %d Percent: %f\n", x,y,PercentFull[x][y]);
        OopsImEmpty(x,y);
        printf("Underzerofill");
        SaveLineToFile();
        SaveWvHeightToFile();
        SaveWvAngleToFile();
        /*PauseRun(x,y,-1);*/
        }

        if (PercentFull[x][y] > 1)
        {
        AllBeach[x][y] = 'y';
        CellDepth[x][y] = - LandHeight;
        if (debug9 && y != 0) printf("\nOver 100 Percent X:
%d  Y: %d Per: %f\n"
                        ,x,y, PercentFull[x][y]);
        OopsImFull(x,y);
        }

        if (((PercentFull[x][y] >=0) && (PercentFull[x][y]
<1)) && (AllBeach[x][y] == 'y'))
        {
        AllBeach[x][y] = 'n';
        CellDepth[x][y] = - LandHeight;
        if (debug9 && y != 0) printf("\nALLBeachProb X: %d
Y: %d\n", x,y);
```

```
        }


        /* Take care of 'loose' bits of sand */

        fillcells3 = 0;

        if ( (PercentFull[x][y] != 0) && (PercentFull[x-
1][y] < 1) && (PercentFull[x+1][y] < 1) &&
         (PercentFull[x][y+1] < 1) && (PercentFull[x][y-1]
< 1) && (AllBeach[x][y] =='n'))
        /* Beach in cell, but bottom, top, right, and left
neighbors not all full */
        {
        if (debug9 && y != 0) printf("\nFB Moved loose bit
of sand,  X: %d  Y: %d  Per: %f  ",
                         x, y, PercentFull[x][y]);

        /* distribute to partially full neighbors */

        if ((PercentFull[x-1][y] < 1) && (PercentFull[x-
1][y] > 0))
            fillcells3 += 1;
        if ((PercentFull[x+1][y] < 1) &&
(PercentFull[x+1][y] > 0))
            fillcells3 += 1;
        if ((PercentFull[x][y-1] < 1) && (PercentFull[x][y-
1] > 0))
            fillcells3 += 1;
        if ((PercentFull[x][y+1] < 1) &&
(PercentFull[x][y+1] > 0))
            fillcells3 += 1;

        if ((fillcells3 > 0))
        {

            if ((PercentFull[x-1][y] < 1) &&
(PercentFull[x-1][y] > 0))
            {
            PercentFull[x-1][y] +=
(PercentFull[x][y])/fillcells3;
            if (debug9) printf ("  MOVEDBACK");
            }
```

```
            if ((PercentFull[x+1][y] < 1) &&
(PercentFull[x+1][y] > 0))
            {
            PercentFull[x+1][y] +=
(PercentFull[x][y])/fillcells3;
            if (debug9) printf ("  MOVEDUP");
            }
            if ((PercentFull[x][y-1] < 1) &&
(PercentFull[x][y-1] > 0))
            {
            PercentFull[x][y-1] +=
(PercentFull[x][y])/fillcells3;
            if (debug9) printf ("  MOVEDLEFT");
            /*if (debug9) PauseRun(x,y,-1);*/
            }
            if ((PercentFull[x][y+1] < 1) &&
(PercentFull[x][y+1] > 0))
            {
            PercentFull[x][y+1] +=
(PercentFull[x][y])/fillcells3;
            if (debug9) printf ("  MOVEDRIGHT");
            /*if (debug9) PauseRun(x,y,-1);*/
            }
        }
        else
        {
            printf("Loner fixbeach breakdown - mass
disintegrated x: %d  y: %d\n",x,y);
            if (debug9)
            PauseRun(x,y,-1);
        }

        PercentFull[x][y] = 0;
        AllBeach[x][y] = 'n';
        CellDepth[x][y] = DepthShoreface;

        if (debug9) printf("\n");


        /* If we have overfilled any of the cells in this
loop, need to OopsImFull() */

        if (PercentFull[x-1][y] > 1)
        {
```

```
            OopsImFull(x-1,y);
            if (debug9) printf("    Below Overfilled\n");
        }
        if (PercentFull[x][y-1] > 1)
        {
            OopsImFull(x,y-1);
            if (debug9) printf("    Left Side
Overfilled\n");
        }
        if (PercentFull[x][y+1] > 1)
        {
            OopsImFull(x,y+1);
            if (debug9) printf("    Right Side
Overfilled\n");
        }
        if (PercentFull[x+1][y+1] > 1)
        {
            OopsImFull(x+1,y+1);
            if (debug9) printf("    Top Overfilled\n");
        }

        }

        /*if ((AllBeach[x][y] =='y') && (PercentFull[x-
1][y] < 1) && (PercentFull[x+1][y] < 1)
          && (PercentFull[x][y-1] < 1) &&
(PercentFull[x][y+1] < 1)
          && (AllBeach[x-1][y-1] == 'n') && (AllBeach[x-
1][y+1] == 'n') &&
          (AllBeach[x+1][y+1] == 'n') && (AllBeach[x+1][y-
1] == 'n') )

          {
          printf("%% Booger !! x: %d  y: %d", x,y);
          PauseRun(x,y,-1);
          }*/

    }
    }


}
```

```
float MassCount(void)

/* Counts the total volume occupied by beach cells  */
/* Uses same algorhythm as AdjustShore              */
/* returns a float of the total sum                 */
/* Uses AllBeach[][] and PercentFull[][]            */
/* and InitialDepth, CellWidth, ShelfSlope          */

{

    int     x,y;
    float   Mass = 0;
    /*float     MassHere;
      float     refdepth;

      refdepth = InitialDepth;*/

    for (x=0; x < Xmax ; x++)
    {
    for(y=Ymax/2; y < 3 * Ymax /2; y++)
    {
        /*if ((PercentFull[x][y] > 0) && (PercentFull[x][y]
< 1.0))
          MassHere = PercentFull[x][y] * (refdepth -
CellDepth[x][y]) +
           (1 - PercentFull[x][y])*(refdepth -
DepthShoreface);
          else
          MassHere = refdepth - CellDepth[x][y];*/

        Mass += PercentFull[x][y];
    }
    }

    return Mass;

}


float Raise(float b, float e)

/* function calulates b to the e power */
/* pow has problems if b <= 0 */
```

```
{
    if (b>0)
    return powf(b,e);
    else
    return -powf(fabs(b),e);
}


float RandZeroToOne(void)

/* function will return a random number equally distributed
between zero and one */
/* currently this function has no seed */
// Modified to run on both UNIX and WIN32 machines

{
    if (RunInWindows) {
        return rand() %10;
    }
    else {
        return rand()/(Raise(2,31)-1);
    }

}


void InitConds(void)

/* Creates initial beach conditions
*/
/* Flat beach with zone of AllBeach = 'y' separated by
AllBeach = 'n'       */
/* Bounding layer set to random fraction of fullness
*/

{
    int     x,y;
    printf("Condition Initial \n");

    if (InitCType == 0)
    /* 'Regular Initial cons - beach backed by sandy land
*/
    {
```

```
    for (y = 0; y <= 2*Ymax; y++)
        for (x = 0; x <= Xmax; x++)
        {

        CellDepth[x][y] = InitialDepth + ((x-InitBeach) *
CellWidth * ShelfSlope);

        if (x < InitBeach)
        {
            PercentFull[x][y] = 1;
            AllBeach[x][y] = 'y';
            CellDepth[x][y] = - LandHeight;
        }
        else if (x == InitBeach)
        {
            if (InitialSmooth)
            {
            PercentFull[x][y] = .5;
            if (TestSSInstability) {
                PercentFull[x][(Ymax-1)] = .6;
                PercentFull[x][Ymax] = .6;
            }
            }
            else
            {
            if (RunInWindows) {
                PercentFull[x][y] = RandZeroToOne()/10;
            } else {
                PercentFull[x][y] = RandZeroToOne();
            }
            printf("x: %d  Y: %d  Per:
%f\n",x,y,PercentFull[x][y]);
            }
            AllBeach[x][y] = 'n';
            CellDepth[x][y] = - LandHeight;
        }
        else if (x > InitBeach)
        {
            PercentFull[x][y] = 0;
            AllBeach[x][y] = 'n';
            if (CellDepth[x][y] < DepthShoreface)
            {
            CellDepth[x][y] = DepthShoreface;
            }
```

```
        }
        else
        {
        printf("WTF! x: %d  Y: %d  Per:
%f\n",x,y,PercentFull[x][y]);
        PauseRun(x,y,-1);
        }

        Age[x][y] = 0;
        }
    }

    else if (InitCType == 1)
    /* 'Simple Barrier' type initial condition - island
backed by lagoon at slope of shelf */
    {

    for (y = 0; y <= 2*Ymax; y++)
    {
        for (x = 0; x < Xmax; x++)
        {

        CellDepth[x][y] = InitialDepth + ((x-InitBeach) *
CellWidth * ShelfSlope);

        if (CellDepth[x][y] <= 0)
            /* This must be land due to continental shelf
intersection */
        {
            PercentFull[x][y] = 1.0;
            AllBeach[x][y] = 'y';
            CellDepth[x][y] = - LandHeight;
        }
        else if (x > InitBeach)
            /* Shoreward of beach - enforce ShorefaceDepth
if necessary */
        {
            PercentFull[x][y] = 0;
            AllBeach[x][y] = 'n';
            if (CellDepth[x][y] < DepthShoreface)
            {
            CellDepth[x][y] = DepthShoreface;
            }
        }
```

```
        else if (x == InitBeach)
            /* Beach */
        {
            if (InitialSmooth)
            {
            PercentFull[x][y] = .5;
            if (TestSSInstability) {
                PercentFull[x][(Ymax-1)] = .6;
                PercentFull[x][Ymax] = .6;
            }
            }
            else
            {
            if (RunInWindows) {
                PercentFull[x][y] = RandZeroToOne()/10;
            } else {
                PercentFull[x][y] = RandZeroToOne();
            }
            /*printf("x: %d  Y: %d  Per:
%f\n",x,y,PercentFull[x][y]);*/
            }
            AllBeach[x][y] = 'n';
            CellDepth[x][y] = - LandHeight;
        }
        else if ((x < InitBeach) && (x > InitBeach -
InitBWidth - 1))
            /* Island */
        {
            PercentFull[x][y] = 1.0;
            AllBeach[x][y] = 'y';
            CellDepth[x][y] = - LandHeight;
        }
        else if (x == InitBeach - InitBWidth -1)
            /* Back of Barrier */
        {
            if (InitialSmooth)
            {
            PercentFull[x][y] = .5;
            }
            else
            {
            if (RunInWindows) {
                PercentFull[x][y] = RandZeroToOne()/10;
            } else {
```

```
                PercentFull[x][y] = RandZeroToOne();
            }
            printf("x: %d  Y: %d  Per:
%f\n",x,y,PercentFull[x][y]);
            }
            AllBeach[x][y] = 'n';
            CellDepth[x][y] = - LandHeight;
        }
        else if (x < InitBeach - InitBWidth -1)
            /* Lagoon at depth of shelf slope  */
        {
            PercentFull[x][y] = 0;
            AllBeach[x][y] = 'n';
        }
        if (PercentFull[x][y] > 1)
        {
            printf("x: %d  Y: %d  Per:
%f\n",x,y,PercentFull[x][y]);
            PauseRun(x,y,-1);
        }
        Age[x][y] = 0;
        }
    }
    }

}


void InitPert(void)

/* Andrew's initial bump */


{

    int x,y;
    int PWidth = 40;
    int PHeight = 40;
    int PYstart = 100;

    if (InitialPert == 1)
    /* Square perturbation */
    {
```

```
    /* Fill AllBeach areas */

    for (x = InitBeach ; x <= InitBeach + PHeight ; x++)
    {
        for (y = PYstart ; y <= PYstart + PWidth ; y++)
        {
        PercentFull[x][y] = 1.0;
        AllBeach[x][y] = 'y';
        }
    }


    /* PercentFull Top */

    for (y = PYstart -1; y <= PYstart + PWidth +1; y++)
    {
        if (RunInWindows) {
            PercentFull[InitBeach + PHeight + 1][y] =
RandZeroToOne()/10;
        } else {
            PercentFull[InitBeach + PHeight + 1][y] =
RandZeroToOne();
        }
    }

    /* PercentFull Sides */

    for (x = InitBeach ; x <= InitBeach + PHeight ; x++)
    {
        if (RunInWindows) {
            PercentFull[x][PYstart-1] = RandZeroToOne()/10;
            PercentFull[x][PYstart+PWidth + 1] =
RandZeroToOne()/10;
        } else {
            PercentFull[x][PYstart-1] = RandZeroToOne();
            PercentFull[x][PYstart+PWidth + 1] =
RandZeroToOne();

        }
    }
    }

    else if (InitialPert == 2)
    /* Another Perturbation  - steep point */
```

```
    {

    x = InitBeach;

    PercentFull[x][17] = 0.8;
    PercentFull[x][18] = 1.0;
    AllBeach[x][18] = 'y';
    PercentFull[x][19] = 0.8;

    x = InitBeach + 1;

    PercentFull[x][17] = 0.6;
    PercentFull[x][18] = 1.0;
    AllBeach[x][18] = 'y';
    PercentFull[x][19] = 0.6;

    x = InitBeach + 2;

    PercentFull[x][17] = 0.2;
    PercentFull[x][18] = 1.0;
    AllBeach[x][18] = 'y';
    PercentFull[x][19] = 0.2;

    x = InitBeach + 3;

    PercentFull[x][18] = 0.3;

    }

}

void InitBigBumps(void)

// Start initial condition with two large 100-cell tall
Bumps

{
    int x,y,i;
    int BumpHeight = 100; // Initial x (cross-shore)
position of bump tip
    int BumpWidth = 30; // Half of bump's width (in cells)

    printf("Starting with two tall initial bumps\n");
```

```
    // First bump

    // Center of bump(tip)
    PercentFull[InitBeach+BumpHeight][3*Ymax/4] = 0.5;
    for (x = InitBeach; x < InitBeach+BumpHeight; x++){
        PercentFull[x][3*Ymax/4] = 1;
    }

    for (i = 1; i < BumpWidth; i++){
        // Right side of bump
        PercentFull[InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth)][(3*Ymax/4)+i] = 0.5;
        for (x = InitBeach; x < InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth); x++){
            PercentFull[x][(3*Ymax/4)+i] = 1;
        }

        // Left side of bump
        PercentFull[InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth)][(3*Ymax/4)-i] = 0.5;
        for (x = InitBeach; x < InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth); x++){
            PercentFull[x][(3*Ymax/4)-i] = 1;
        }
    }


    // Second bump

    // Center of bump(tip)
    PercentFull[InitBeach+BumpHeight][5*Ymax/4] = 0.5;
    for (x = InitBeach; x < InitBeach+BumpHeight; x++){
        PercentFull[x][5*Ymax/4] = 1;
    }

    for (i = 1; i < BumpWidth; i++){
        // Right side of bump
        PercentFull[InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth)][(5*Ymax/4)+i] = 0.5;
        for (x = InitBeach; x < InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth); x++){
            PercentFull[x][(5*Ymax/4)+i] = 1;
        }
```

```
        // Left side of bump
        PercentFull[InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth)][(5*Ymax/4)-i] = 0.5;
        for (x = InitBeach; x < InitBeach+BumpHeight-
(i*BumpHeight/BumpWidth); x++){
            PercentFull[x][(5*Ymax/4)-i] = 1;
        }
    }

}


void PeriodicBoundaryCopy(void)

// Simulates periodic boundary conditions by copying middle
section to front and end of arrays

{
    int x,y;

    for (y = Ymax; y < 3*Ymax/2; y++)
    for (x = 0; x < Xmax; x++)
    {
        AllBeach[x][y-Ymax] = AllBeach[x][y];
        PercentFull[x][y-Ymax] = PercentFull[x][y];
        Age[x][y-Ymax] = Age[x][y];
        CellDepth[x][y-Ymax] = CellDepth[x][y];
    }
    for (y = Ymax/2; y <= Ymax; y++)
    for (x = 0; x < Xmax; x++)
    {
        AllBeach[x][y+Ymax] = AllBeach[x][y];
        PercentFull[x][y+Ymax] = PercentFull[x][y];
        Age[x][y+Ymax] = Age[x][y];
        CellDepth[x][y+Ymax] = CellDepth[x][y];
    }

    // Correct PeriodicBoundaryCopy bug
    for (x = 0; x < Xmax; x++){
        AllBeach[x][0] = AllBeach[x][Ymax];
        PercentFull[x][0] = PercentFull[x][Ymax];
    }

}
```

```
void ZeroVars(void)

/* Resets all arrays recalculated at each time step to
'zero' conditions */

{

    int z;

    for (z=0; z < MaxBeachLength; z++)
    {
    X[z] = -1;
    Y[z] = -1;
    InShadow[z] = '?';
    ShorelineAngle[z] = -999;
    SurroundingAngle[z] = -998;
    UpWind[z] = '?';
    VolumeIn[z] = 0;
    VolumeOut[z] = 0;
    }
}


void ReadSandFromFile(void)

/*  Reads saved output file, AllBeach[][] & PercentFull[][]
*/

{
    int x,y;

    ReadSandFile = fopen(readfilename,"r");printf("CHECK
READ \n");


    for (y = Ymax/2; y < 3*Ymax/2; y++)
    {

    for (x=0; x<Xmax; x++)
    {
        fscanf(ReadSandFile, " %f", &PercentFull[x][y]);
```

```
            if (PercentFull[x][y] >= 1.0)
            AllBeach[x][y] = 'y';
            else
            AllBeach[x][y] = 'n';
        }
        }

        for (y = Ymax/2; y < 3*Ymax/2; y++)
        for (x=0; x<Xmax; x++)
            fscanf(ReadSandFile, " %f", &CellDepth[x][y]);


        if (SaveAge)

        for (y = Ymax/2; y < 3*Ymax/2; y++)
        {
            for (x=0; x<Xmax; x++)
            {
            fscanf(ReadSandFile, " %d", &Age[x][y]);
            }
        }

        /*PrintLocalConds(5,5,-1);*/
        fclose(ReadSandFile);
        printf("file read!");

        PeriodicBoundaryCopy();

}


void LoadLineout(void)

// Loads information from a lineout file to start a run
// Modifies global arrays PercentFull[][], CellDepth[][]
StartingLine[]

{

        int x,y;
        int HitBeach;   // Indicator for when the x-direction
loop has reached the beach;
```

```
    int debugloc = 0;   // Localdebugger

    // Read the lineout file and save as StartingLine[]
    ReadSandFile = fopen(readlinename,"r");
    printf("CHECK READ \n");

    for (y = Ymax/2; y < 3*Ymax/2; y++){
        fscanf(ReadSandFile, " %f", &StartingLine[y]);
    }

    fclose(ReadSandFile);
    printf("Line file read!");

    // Convert StartingLine[] to PercentFull[][] and
CellDepth[][]
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x = 0; x < Xmax; x++){
            if (x == 0){
                PercentFull[x][y] = 1;
                AllBeach[x][y] = 'y';
                CellDepth[x][y] = - LandHeight;
                HitBeach = 0;
            } else if ((x > InitBeach + StartingLine[y]) &&
(x < InitBeach + StartingLine[y] + 1)){
                    PercentFull[x][y] = StartingLine[y] -
(x-1) + InitBeach - 0.5;
                    AllBeach[x][y] = 'n';
                    CellDepth[x][y] = - LandHeight;
                    HitBeach = 1;
            } else if (HitBeach == 0){
                PercentFull[x][y] = 1;
                AllBeach[x][y] = 'y';
                CellDepth[x][y] = - LandHeight;
            } else if (HitBeach == 1){
                PercentFull[x][y] = 0;
                AllBeach[x][y] = 'n';
                CellDepth[x][y] = InitialDepth + ((x-
InitBeach) * CellWidth * ShelfSlope);
            }
        }
    }

    // Make adjustmets
    for (y = Ymax/2; y < 3*Ymax/2; y++){
```

```
        for (x = 0; x < Xmax; x++){
            if (PercentFull[x][y] < 0){
                PercentFull[x-1][y] = PercentFull[x][y] +
1;
                AllBeach[x-1][y] = 'n';
                CellDepth[x-1][y] = - LandHeight;

                PercentFull[x][y] = 0;
                AllBeach[x][y] = 'n';
                CellDepth[x][y] = InitialDepth + ((x-
InitBeach) * CellWidth * ShelfSlope);
            }
        }
    }
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x = 0; x < Xmax; x++){
            if ((CellDepth[x][y] < DepthShoreface) &&
(PercentFull[x][y] == 0)){
                CellDepth[x][y] = DepthShoreface;
            }
        }
    }

    // Establish Age
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x = 0; x < Xmax; x++){
            Age[x][y] = 0;
        }
    }

    // Debug
    if (debugloc){
    printf("\nStartingLine = \n");
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        printf("%G ", StartingLine[y]);
    } printf("\n");

    printf("\nPercentFull = \n");
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x = 0; x < Xmax; x++){
            printf("%G ", PercentFull[x][y]);
        } printf("\n");
    }
```

```
    printf("\nCellDepth = \n");
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x = 0; x < Xmax; x++){
            printf("%G ", CellDepth[x][y]);
        } printf("\n");
    }
    }

    PeriodicBoundaryCopy();
}


void SaveSandToFile(void)

/*  Saves current AllBeach[][] and PercentFull[][] data
arrays to file      */
/*  Save file name will add extension '.' and the
CurrentTimeStep       */

{
    int x,y;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s.%d", savefilename,
CurrentTimeStep);
    printf( "Saving as: %s      ", savename );


    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y= Ymax/2; y< 3*Ymax/2; y++)
    for (x=0; x<Xmax; x++)
        fprintf(SaveSandFile, " %f", PercentFull[x][y]);

    for (y= Ymax/2; y< 3*Ymax/2; y++)
    for (x=0; x<Xmax; x++)
```

```
        fprintf(SaveSandFile, " %f", CellDepth[x][y]);

    if (SaveAge)
    for (y=Ymax/2; y< 3*Ymax/2; y++)
        for (x=0; x<Xmax; x++)
        fprintf(SaveSandFile, " %d", Age[x][y]);


    fclose(SaveSandFile);
    printf("--- regular file saved! ----\n\n");

}


void SaveLineToFile(void)

/*  Saves data line of shoreline position rather than
entire array      */
/*  Main concern is to have only one data point at each
alongshore location */
/*  Save file name will add extension '.' and the
CurrentTimeStep       */

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", savelinename,
CurrentTimeStep);
    printf( "Saving as: %s                  ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
```

```
    {

    x = Xmax-1;
    xtop = Xmax;

    /* step back to where we encounter allbeach */
    while(AllBeach[x][y] == 'n')
    {
        x -= 1;
    }

    /* if on side of shape, need to average */
    if (PercentFull[x+2][y] > 0)
    {
        xtop = x+1;
        while(PercentFull[xtop][y] > 0)
        {
        xtop +=1;
        }

        xsave = x;

        for (i=x+1; i<xtop ; i++)
        xsave += PercentFull[i][y];

    }
    /* otherwise Regular Beach Condition */
    else
    {
        xsave = x + PercentFull[x+1][y];
    }

    /* note this assumes average of beach locations should
be 0.5 percentfull */
    fprintf(SaveSandFile, " %f", xsave - InitBeach + 0.5);

    /*printf("y %d , xtop = %d xsave = %f
\n",y,xtop,xsave);
      if (xtop != Xmax)
      PauseRun(x+1,y,-1);           */

    }

    fclose(SaveSandFile);
```

```
    printf("line file saved!\n\n");

}


void SaveWvHeightToFile (void)

// Saves the breaking wave height data to a file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", saveheightname,
CurrentTimeStep);
    printf( "Saving as: %s                  ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = HeightRollAvg[y];

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Height file saved!\n\n");

}
```

```
void SaveWvAngleToFile (void)

// Saves the breaking wave angle data to a file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", saveanglename,
CurrentTimeStep);
    printf( "Saving as: %s                  ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = (AngleRollAvg[y]*radtodeg);

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Angle file saved!\n\n");

}

void SaveRelAngleToFile (void)

// Saves the calculated relative angle data to a file

{
```

```
    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", SaveRelAnglename,
CurrentTimeStep);
    printf( "Saving as: %s                  ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = RelAngle[y]*radtodeg;

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Relative Angle file saved!\n\n");

}

void SaveSLAngleToFile (void)

// Saves the shoreline angle data to a file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");
```

```
    sprintf(savename, "%s%d", SaveSLAngleName,
CurrentTimeStep);
    printf( "Saving as: %s                    ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = SLAngle[y]*radtodeg;

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Shoreline Angle file saved!\n\n");

}


void SaveVolumeOutToFile (void)

// Saves the sediment volume transported out of each cell
to a file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", saveVolumeOutName,
CurrentTimeStep);
    printf( "Saving as: %s                    ", savename );
```

```
    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = VolumeOut[y];

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Volume file saved!\n\n");

}

void SaveSedFluxGradOutToFile(void)

// Saves the sediment flux gradient in each beach cell to a
file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", saveSedFluxGradOutName,
CurrentTimeStep);
    printf( "Saving as: %s                 ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
```

```
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = SedFluxGrad[y];

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("SedFlux file saved!\n\n");

}

void SaveGammaToFile(void)

// Saves the sediment flux gradient in each beach cell to a
file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", SaveGammaName,
CurrentTimeStep);
    printf( "Saving as: %s                  ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {
```

```
    xsave = Gamma[y];

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Gamma file saved!\n\n");

}

void SaveDiffusivityToFile(void)

// Saves the diffusivity in each beach cell to a file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", SaveDiffusivityName,
CurrentTimeStep);
    printf( "Saving as: %s                  ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = Diffusivity[y];

    fprintf(SaveSandFile, " %f", xsave);

    }
```

```
    fclose(SaveSandFile);
    printf("Diffusivity file saved!\n\n");

}

void SaveMuToFile(void)

// Saves the mu in each beach cell to a file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", SaveMuName, CurrentTimeStep);
    printf( "Saving as: %s                    ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = Mu[y];

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Mu file saved!\n\n");

}

void SaveQsToFile(void)
```

```
// Saves sediment transport (Qs) to file

{

    int y,x,xtop,i;
    float   xsave;
    char    savename[40];

    printf("\n saving \n ");

    sprintf(savename, "%s%d", SaveQsName, CurrentTimeStep);
    printf( "Saving as: %s                    ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = Qs[y];

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Sediment transport file saved!\n\n");

}

void SaveDEtaDtToFile(void)

// Saves the dEta/dt data to file

{
    int y,x,xtop,i;
    float   xsave;
    char    savename[40];
```

```
    printf("\n saving \n ");

    sprintf(savename, "%s%d", SaveDEtaDtName,
CurrentTimeStep);
    printf( "Saving as: %s                  ", savename );

    SaveSandFile = fopen(savename, "w");
    if (!SaveSandFile)
    {
    printf("problem opening output file\n");
    exit(1);
    }


    for (y=Ymax/2; y < 3*Ymax/2; y++)
    {

    xsave = dEta_dt[y];

    fprintf(SaveSandFile, " %f", xsave);

    }

    fclose(SaveSandFile);
    printf("Sediment transport file saved!\n\n");

}



void PrintLocalConds(int x, int y, int in)

/* Prints Local Array Conditions aound x,y */

{

    int i,j,k,isee;
    float vol = CellWidth*CellWidth*DepthShoreface;

    printf("\n x: %d  y: %d  z: %d\n\n", x,y,in);

    /* if not given location along beach, look to see if
along beach */
```

```
if (in<0)
{
for (i=0; i <= TotalBeachCells; i++)
    if ((X[i]==x) && (Y[i]==y))
    isee = i;
}
else
isee = in;


for (i= x+2 ; i > x-3 ; i--)
{
for (j = y-2 ; j < y+3 ; j++)
{
    printf("    %d,%d", i , j);
}
printf("\n");
}
printf("\n");

for (i= x+2 ; i > x-3 ; i--)
{
for (j = y-2 ; j < y+3 ; j++)
{
    printf("    %f", CellDepth[i][j]);
    if (CellDepth[i][j] == DepthShoreface)
    printf("y");
    else
    printf("n");
}
printf("\n");
}
printf("\n");

for (i= x+2 ; i > x-3 ; i--)
{
for (j = y-2 ; j < y+3 ; j++)
{
    printf("    %c", AllBeach[i][j]);
}
printf("\n");
}
```

```
    printf("\n");

    for (i= x+2 ; i > x-3 ; i--)
    {
    for (j = y-2 ; j < y+3 ; j++)
    {
        printf("    %2.5f",PercentFull[i][j]);
    }
    printf("\n");
    }

    printf("\n");

    printf(" %d  ", in );

    if (isee>=0)
    {
    for (k = in-3; k <in+4; k++)
    {
        printf("    %2d: %2d,%2d", k , X[k] , Y[k]);
    }
    printf("\n\n\n");

    printf("Wave Angle: %f\n\n",WaveAngle*radtodeg);
    printf("i       %d      %d      !%d     %d      %d\n",
           in-2, in-1,in,in+1,in+2);
    printf("Shadow      %c      %c      %c      %c
%c\n",
           InShadow[in-2], InShadow[in-1],InShadow[in],
InShadow[in+1], InShadow[in+2]);
    printf("Upwind      %c      %c      %c      %c
%c\n",
           UpWind[in-2],  UpWind[in-1],UpWind[in],
UpWind[in+1],  UpWind[in+2]);
    printf("Angle       %2.2f       %2.2f       %2.2f
%2.2f       %2.2f\n",
           ShorelineAngle[in-2]*radtodeg,ShorelineAngle[in-
1]*radtodeg, ShorelineAngle[in]*radtodeg,

ShorelineAngle[in+1]*radtodeg,ShorelineAngle[in+2]*radtodeg
);
    printf("SurrAngle   %2.2f       %2.2f       %2.2f
%2.2f       %2.2f\n",
```

```
            SurroundingAngle[in-2]*radtodeg,
SurroundingAngle[in-1]*radtodeg,
SurroundingAngle[in]*radtodeg,
            SurroundingAngle[in+1]*radtodeg,
SurroundingAngle[in+2]*radtodeg);
    printf("Vol In      %2.2f       %2.2f       %2.2f
%2.2f       %2.2f\n",
            VolumeIn[in-2], VolumeIn[in-
1],VolumeIn[in],VolumeIn[in+1],VolumeIn[in+2]);
    printf("Vol Out     %2.2f       %2.2f       %2.2f
%2.2f       %2.2f\n",
            VolumeOut[in-2], VolumeOut[in-1],
VolumeOut[in],VolumeOut[in+1],VolumeOut[in+2]);
    printf("Diff        %2.2f       %2.2f       %2.2f
%2.2f       %2.2f\n",
            VolumeIn[in-2]-VolumeOut[in-2], VolumeIn[in-1]-
VolumeOut[in-1], VolumeIn[in]-VolumeOut[in],
            VolumeIn[in+1]-VolumeOut[in+1],VolumeIn[in+2]-
VolumeOut[in+2]);
    printf("Frac Diff   %2.3f       %2.3f       %2.3f
%2.3f       %2.3f\n",
            (VolumeIn[in-2]-VolumeOut[in-2])/vol,
(VolumeIn[in-1]-VolumeOut[in-1])/vol,
            (VolumeIn[in]-VolumeOut[in])/vol,
(VolumeIn[in+1]-VolumeOut[in+1])/vol,
            (VolumeIn[in+2]-VolumeOut[in+2])/vol);

    }

    printf("\n");




}


void PauseRun(int x, int y, int in)

/* Pauses run intil the 'q' key is pressed  */
/* Can Print or Plot Out Useful info        */
```

```
{

    int xsee=1,ysee=-1,isee=-1,i;

    printf("\nPaused x: %d  y: %d Time:
%d\n",x,y,CurrentTimeStep);

    /*if (SaveLine) SaveLineToFile();
      else SaveSandToFile();*/

    if(NoPauseRun)
    return;

    sleep(5);
    printf("\nend Pause\n");

}


void ButtonEnter(void)

{

    char newdigit = 'z';
    int flag = 0;
    int i = 1;
    char digits[7] = "";

    /*printf("Flag1 %d\n",flag);*/

    printf("Press <Space> to Start\n");



    printf("Enter Digit and <Space> (<Z> to finish)\n");



    i += 1;
    sprintf(digits,"%s%c",digits,newdigit);
    printf("\nCurrent %s\n",digits);
    newdigit = 'z';
```

```
}


void AgeCells(void)

/* Age Cells */

{

    int x,y;

    for (y = 0; y < 2*Ymax; y++)
    for (x=0; x<Xmax; x++)
        if (PercentFull[x][y] == 0)
        {
        Age[x][y] = CurrentTimeStep%AgeMax;
        }

}


void ReadWaveIn(void)

/* Input Wave Distribution */

{

    int i;

    for (i=0 ; i<= 36; i++)
    {
    WaveMax[i] =0;
    WaveProb[i] = 0;
    }

    ReadWaveFile = fopen(readwavename,"r");printf("CHECK
READ WAVE\n");

    fscanf(ReadWaveFile, " %d \n", &NumWaveBins);
    fscanf(ReadWaveFile, " %d \n", &BinSize);
    fscanf(ReadWaveFile, " %G \n", &MaxWaveProb);
```

```
    printf("Wave Bins %d \n",NumWaveBins);
    printf("Bin Size %d degrees \n", BinSize);
    printf("Max Wave Probability %G \n", MaxWaveProb);

    WaveMax[0] = -90;
    WaveProb[0] = 0;

    for (i=1 ; i<= NumWaveBins ; i++)
    {
    fscanf(ReadWaveFile, " %f %f", &WaveMax[i] ,
&WaveProb[i]);
    printf("i= %d  Wave= %f Prob= %f \n",i, WaveMax[i],
WaveProb[i]);
    }

    fclose(ReadWaveFile);
    printf("wave file read! \n");

}

void    DeliverSediment(void)

/* Simple 'first approximation of sediment delivery */
/* At certain alongshore location, add certain amount of
sed to the coast */

    {

    int x,y;

    x = 0;
    y = StreamSpot;

    while (AllBeach[x][y] == 'y')
    {
        x += 1;
    }

    PercentFull[x][y] += SedRate;

    }
```

```
void CheckOverwashSweep(void)

    /* Just a loop to call overwash check founction
CheckOverwash                    */
    /* Nothing done here, but can be down when
CheckOVerwash is called              */


    {

    int i,ii;       /* local loop variable */
    int sweepsign;

    if (RunInWindows) {
        if (RandZeroToOne()*.2 > 1)
        {
            sweepsign = 1;
            if (debug10a) printf("L  ");
        }
        else
        {
            sweepsign = 0;
            if (debug10a) printf("R  ");
        }
    } else {
        if (RandZeroToOne()*2 > 1)
        {
            sweepsign = 1;
            if (debug10a) printf("L  ");
        }
        else
        {
            sweepsign = 0;
            if (debug10a) printf("R  ");
        }
    }

    OWflag = 0;
    for (i=1; i < TotalBeachCells-1 ; i++)
    {
        if (sweepsign == 1)
            ii = i;
        else
            ii = TotalBeachCells-1-i;
```

```
        /* To do test shoreline should be facing seaward
*/
        /* don't worry about shadow here, as overwash is
not set to a time scale with AST   */

        if ((fabs(SurroundingAngle[ii]) <
(OverwashLimit/radtodeg))  && (InShadow[ii] == 'n'))
        {
            CheckOverwash(ii);
        }

    }


    /*if (OWflag) PauseRun(1,1,-1);*/

}




void CheckOverwash(int icheck)

    /* New 1/04 ADA - Step back pixelwise in direction of
Surrounding Angle to check needage    */
    /* If too short, calls DoOverwash, which will move some
sediment                  */
    /* Uses AllBeach[][] and PercentFull[][] (can be
changed when DoOVerwash is called      */
    /* Need to change sweepsign because filling cells
should affect neighbors           */
    /* 'x' and 'y' hold real-space values, will be mapped
onto ineger array             */

    {

    float   slope;          /* slope of zero goes staight
back */
    int ysign;          /* holder for going left or right
alongshore */
    float   x,y;            /* holders for 'real' location
of x and y */
    float   xin, yin;       /* starting 'real' locations */
```

```
    int xtest,ytest;        /* cell looking at */
    float   xint,yint;      /* intercepts of overwash line
in overwashable cell */
    int NextXInt, NextYInt; /* holder vairables for cell to
check */
    float   Ydown, DistanceDown;    /* when going to next x
cell, what other values */
    float   Xside, DistanceSide;    /* when gpoing to next
y cell,other values */
    float   checkdistance;      /* distance of checking
line- minimum, not actual width, ends loop */
    float   measwidth;      /* actual barrier width between
cells */
    int     AllBeachFlag;       /* flag to see if overwash
line has passed over at least one AllBeach cell */

    /* convert angle to a slope and the direction of steps
*/
    /* note that for case of shoreline, positive angle will
be minus y direcyion */

    /*if(icheck == 122)
        debug10a = 1;
    else
        debug10a = 0;*/

    if (SurroundingAngle[icheck] == 0.0)
    {
        /* unlikely, but make sure no div by zero */
        slope = 0.00001;
    }
    else if (fabs(SurroundingAngle[icheck]) == 90.0)
    {
        slope = 9999.9;
    }
    else
    {
        slope = fabs(tan(SurroundingAngle[icheck]));
    }

    if (SurroundingAngle[icheck] > 0)
        ysign = 1;
    else
        ysign = -1;
```

```
        if (debug10a) printf("\nI: %d------------ Surr: %f
%f Slope: %f sign: %d \n",
         icheck,
SurroundingAngle[icheck],SurroundingAngle[icheck]*radtodeg,
slope, ysign);


    if (AllBeach[X[icheck]-1][Y[icheck]] == 'y' ||
((AllBeach[X[icheck]][Y[icheck]-1] == 'y') &&
        (AllBeach[X[icheck]][Y[icheck]+1] == 'y')) )
    /* 'regular condition' */
    /* plus 'stuck in the middle' situation (unlikely
scenario)*/
    {
        xin = X[icheck] +
PercentFull[X[icheck]][Y[icheck]];
        yin = Y[icheck] + 0.5;
    }
    else if (AllBeach[X[icheck]][Y[icheck]-1] == 'y')
    /* on right side */
    {
        xin = X[icheck] + 0.5;
        yin = Y[icheck] +
PercentFull[X[icheck]][Y[icheck]];
        if (debug10a) printf("-- Right xin: %f  yin:
%f\n",xin,yin);
    }
    else if (AllBeach[X[icheck]][Y[icheck]+1] == 'y')
    /* on left side */
    {
        xin = X[icheck] + 0.5;
        yin = Y[icheck] + 1.0 -
PercentFull[X[icheck]][Y[icheck]];
        if (debug10a) printf("-- Left xin: %f  yin:
%f\n",xin,yin);
    }
    else
    /* underneath, no overwash */
    {
        return;
    }

    x = xin;
```

```
    y = yin;
    checkdistance = 0 ;
    AllBeachFlag = 0;

    while ((checkdistance < CritBWidth) && (y > 0) && (y <
2*Ymax) && (x > 1))
    {
        NextXInt = ceil(x) -1;
        if (ysign > 0)
            NextYInt = floor(y) + 1;
        else
            NextYInt = ceil(y-1);

        /* moving to next whole 'x' position, what is y
position? */
        Ydown = y + (x - NextXInt)*slope * ysign;
        DistanceDown = Raise(((Ydown - y)*(Ydown - y) +
(NextXInt - x)*(NextXInt - x)),.5);

        /* moving to next whole 'y' position, what is x
position? */
        Xside = x - fabs(NextYInt - y) / slope;
        DistanceSide = Raise(((NextYInt - y)*(NextYInt - y)
+ (Xside - x)*(Xside - x)),.5);

            if (debug10a) printf("x: %f  y: %f  X:%d  Y: %d
Yd: %f  DistD: %f Xs: %f DistS: %f\n",
            x,y,NextXInt,NextYInt,
Ydown,DistanceDown,Xside,DistanceSide);

        if (DistanceDown < DistanceSide)
        /* next cell is the down cell */
        {
            x = NextXInt;
            y = Ydown;
            xtest = NextXInt-1;
            ytest = floor(y);
                /*if (debug10a) printf(" down ");*/
        }
        else
        /* next cell is the side cell */
        {
            x = Xside;
            y = NextYInt;
```

```
            xtest = floor(x);
            ytest = y + (ysign-1)/2;
                /*if (debug10a) printf(" side ");*/
        }

        /*if ((debug10a) && (DoGraphics ==
'y'))PutPixel(ytest*CellPixelSize,xtest*CellPixelSize,0,0,2
00);*/

        checkdistance = Raise(((x - xin)*(x - xin) +  (y -
yin)*(y - yin)),.5) * CellWidth;
        if (AllBeach[xtest][ytest] == 'y')
            AllBeachFlag = 1;

        if (debug10a) printf("  x: %f  y: %f  xtest: %d
ytest: %d check: %f\n\n",x,y,xtest,ytest,checkdistance);

        if ((AllBeach[xtest][ytest] == 'n') &&
(AllBeachFlag) && !(((X[icheck]-xtest) > 1) || (abs(ytest -
Y[icheck]) > 1)))
        /* if passed through an allbeach and a neighboring
partial cell, jump out, only bad things follow */
        {
            return;
        }

        if((AllBeach[xtest][ytest] == 'n') &&
(AllBeachFlag) && (xtest < X[icheck]) &&
            (((X[icheck]-xtest) > 1) || (abs(ytest -
Y[icheck]) > 1)))
        /* Looking for shore cells, but don't want
immediate neighbors, and go backwards */
        /* Also mush pass though an allbeach cell along the
way */
        {

            if (AllBeach[xtest+1][ytest] == 'y')
            /* 'regular condition' - UNDERNEATH, here */
            {
                xint = (xtest + 1 -
PercentFull[xtest][ytest]);
                yint = yin + (xin - xint)*ysign * slope;

                if ((yint > ytest+1.0) || (yint < ytest))
```

```
                /* This cell isn't actually an overwash
cell */
                {
                    measwidth = CritBWidth;
                    if (debug10a) printf("-- Regunder
Cancelled  xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f
yint: %f sl: %2.2fMMeas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,slope,measwidth);
                }
                else
                {
                    measwidth = CellWidth * Raise((xint -
xin)*(xint - xin)+ (yint - yin)*(yint - yin),0.5);

                    if (debug10a) printf("-- Regunder Over
xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f yint: %f sl:
%2.2fMeas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,slope,measwidth);
                }
            }
            else if (AllBeach[xtest][ytest-1] == 'y')
            /* on right side */
            {
                yint = (ytest + PercentFull[xtest][ytest]);
                xint = xin - fabs(yin - yint)/ slope;

                if (xint < xtest)
                /* This cell isn't actually an overwash
cell */
                {
                    measwidth = CritBWidth;

                    if (debug10a) printf("-- Right
Cancelled  xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f
yint: %f sl: %2.2fMMeas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,slope,measwidth);
                }
                else
                {
                    measwidth = CellWidth * Raise((xint -
xin)*(xint - xin)+ (yint - yin)*(yint - yin),0.5);
```

```
                    if (debug10a) printf("-- Right Over
xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f yint: %f sl:
%2.2fMMeas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,slope,measwidth);
                }
            }
            else if (AllBeach[xtest][ytest+1] == 'y')
            /* on left side */
            {
                yint = (ytest + 1 -
PercentFull[xtest][ytest]);
                xint = xin - fabs(yin - yint)/ slope;

                if (xint < xtest)
                /* This cell isn't actually an overwash
cell */
                {
                    measwidth = CritBWidth;

                    if (debug10a) printf("-- Left cancelled
xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f yint: %f sl:
%2.2fMMeas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,slope,measwidth);
                }
                else
                {
                    measwidth = CellWidth * Raise((xint -
xin)*(xint - xin)+ (yint - yin)*(yint - yin),0.5);

                    if (debug10a) printf("-- Left Over
xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f yint: %f sl:
%2.2fMMeas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,slope,measwidth);
                }
            }
            else if (AllBeach[xtest-1][ytest] == 'y')
            /* 'regular condition' */
            /* plus 'stuck in the middle' situation */
            {
```

```
                xint = (xtest  +
PercentFull[xtest][ytest]);
                yint = yin + (xin - xint)*ysign * slope;

                if ((yint > ytest+1.0) || (yint < ytest))
                /* This cell isn't actually an overwash
cell */
                {
                    measwidth = CritBWidth;
                    if (debug10a) printf("-- RegularODD
Cancelled  xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f
yint: %f Meas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,measwidth);
                }
                else
                {
                    measwidth = CellWidth * Raise((xint -
xin)*(xint - xin)+ (yint - yin)*(yint - yin),0.5);

                    if (debug10a) printf("-- RegularODD
Over  xin: %2.2f  yin: %2.2f xt:%d yt: %d xint: %f yint: %f
Meas: %3.2f\n",

xin,yin,xtest,ytest,xint,yint,measwidth);
                    /*PauseRun(xtest,ytest,icheck);*/
                }
            }
            else if (PercentFull[xtest][ytest] > 0)
            /* uh oh - not good situation, no allbeach on
sides */
            /* assume this is an empty cell, */
            {
                xint = x;
                yint = y;

                measwidth = CellWidth * Raise((xint -
xin)*(xint - xin)+ (yint - yin)*(yint - yin),0.5);

                printf("-- Some Odd Over  xin: %2.2f  yin:
%2.2f xt:%d yt: %d xint: %f yint: %f Meas: %3.2f Ang: %f
Abs: %f\n",
```

```
                xin,yin,xtest,ytest,xint,yint,measwidth,
SurroundingAngle[icheck]*radtodeg,fabs(SurroundingAngle[ich
eck])*radtodeg);
                /*PauseRun(xtest,ytest,icheck);*/
            }
            else
            /* empty cell - oughta fill er up  - fill max
barrier width*/
            {
                xint = x;
                yint = y;
                measwidth = CritBWidth - CellWidth;

                printf("-- Empty Odd Over  xin: %2.2f  yin:
%2.2f xt:%d yt: %d xint: %f yint: %f Meas: %3.2f Ang: %f
Abs: %f\n",
                xin,yin,xtest,ytest,xint,yint,measwidth,
SurroundingAngle[icheck]*radtodeg,fabs(SurroundingAngle[ich
eck])*radtodeg);
                /*PauseRun(xtest,ytest,icheck); */
            }

            checkdistance = measwidth;

            if (measwidth < CritBWidth)
            {

DoOverwash(X[icheck],Y[icheck],xtest,ytest,xint,yint,measwi
dth,icheck);
                /* jump out of loop */
                OWflag = 1;
                return;
            }


        }

        }
/*  while(!getbutton(GKEY)){}*/

}
```

```
void DoOverwash(int xfrom,int yfrom, int xto, int yto,
float xintto, float yintto, float widthin, int ishore)

    /*  given a cell where overwash is needed ,move
sediment back  *** ADA 09/03, rev 01/04 */
    /*  for 'true' overwash based on shoreline angles
*/
    /*  will change and use PercentFull[][] and AllBeach
[][]               */

{
    float BBneed, delBB, delShore;  /* local variables */
    float MaxOver = 0.2;         /*Maximum overwash step
size (enforced at backbarrier) */
    /*float DepthBackBarrier = 6.0;      m current set
depth for backbarrier (temp - make into function)*/
    float DepthBB;           /* holds effective backbarrier
depth */
    short   vertex[2];

    DepthBB =
GetOverwashDepth(xto,yto,xintto,yintto,ishore);

    /* calculated value of most that backbarrier ca nmove
given geometry (true, non-iterative solution) */


    if (DepthBB == DepthShoreface)
    {
        BBneed = MaxOver;
    }
    else
    {
        BBneed = (CritBWidth - widthin) / CellWidth /  (1 -
(DepthBB / DepthShoreface));
    }


    if (BBneed <= MaxOver)
    /* do all overwash */
    {
        delShore = BBneed * DepthBB / DepthShoreface;
        delBB = BBneed;
    }
```

```
    else
    /* only do overwash to max change) */
    {
        delShore = MaxOver * DepthBB / DepthShoreface ;
        delBB = MaxOver;

    }


    if (debug10b) printf("** Overwash From X: %d  Y: %d
To: X: %d Y: %d Width: %f \n"
        , xfrom, yfrom,xto,yto,widthin );
    if (debug10b) printf("DepthBB: %f  BBNeed: %f DelShore:
%f  DelBB: %f\n",
            DepthBB, BBneed,delShore,delBB );
    /*if (DepthBB == DepthShoreface) PauseRun(xto,yto,-
1);*/

    if (debug10b && (DoGraphics == 'y'))
    {
        /*bgnpolygon();
            RGBcolor(250,0,0);
            vertex[0] = (yfrom+0.2)*CellPixelSize;
            vertex[1] = (xfrom+.5)*CellPixelSize;
            v2s(vertex);
            vertex[0] = (yfrom+0.8)*CellPixelSize;
            v2s(vertex);
            vertex[0] = (yto+0.8)*CellPixelSize;
            vertex[1] = (xto+0.5)*CellPixelSize;
            v2s(vertex);
            vertex[0] = (yto+0.3)*CellPixelSize;;
            v2s(vertex);
            vertex[0] = (yfrom+0.2)*CellPixelSize;
            vertex[1] = (xfrom+.5)*CellPixelSize;
            v2s(vertex);
            endpolygon();*/
    }

    PercentFull[xto][yto] += delBB;
    PercentFull[xfrom][yfrom] -= delShore;

    if (PercentFull[xto][yto] > 1)
    {
        OopsImFull(xto,yto);
```

```
    }
    if (PercentFull[xfrom][yfrom] < 0)
    {
        OopsImEmpty(xfrom,yfrom);
    }

    if (debug10b) PauseRun(xto,yto,-1);

}


float GetOverwashDepth(int xin, int yin, float xinfl, float
yinfl, int ishore)

    /* Rountine finds corresponding overwash depths
*/
    /*      OWType = 0 take the depth at neightbor to the
backing cell      */
    /*      OWType = 1 geometric rule based upon distance
from back to shoreline    */
    /* AA 5/04                                    */
{
    int xdepth;
    float   Depth;
    float   BBDistance;  /* Distance from backshore to next
shore */
    float   slope;          /* slope of zero goes staight
back */
    int ysign;          /* holder for going left or right
alongshore */
    float   x,y;            /* holders for 'real' location
of x and y */
    int xtest,ytest;        /* cell looking at */
    int NextXInt, NextYInt; /* holder vairables for cell to
check */
    float   Ydown, DistanceDown;    /* when going to next x
cell, what other values */
    float   Xside, DistanceSide;    /* when gpoing to next
y cell,other values */
    int     BackFlag;       /* Flag to indicate if hit
backbarrier */
    int Backi = -1;     /* i for backbarrier intersection
*/
    int i,j;            /* counters */
```

```
    int FoundFlag;      /* Backbarrier intersection flag */
    float   AngleSin, AngleUsed;


    if (OWType == 0)
    /* Use Cell Depths for overwash depths */
    {
        xdepth = xin;
        Depth = CellDepth[xdepth][yin];

        while ((Depth < 0) && (xdepth > 0))
        {
            Depth = CellDepth[xdepth][yin];
            /*printf("-- Overwash depth problem - Here = %f
Next = %f",CellDepth[xdepth][yto],CellDepth[xdepth-1][yto]
);
            PauseRun(xdepth,yto,-1);*/
            xdepth --;
        }

        if (Depth == DepthShoreface)
        {
            Depth = 6.0;
        }

        return Depth;

    }
    else if (OWType == 1)
    /* Geometric relation to determine depth through
intersection of shorefaces     */
    /* look in line determined by shoreline slope - reuse
stepping function (again) */
    {
        x = xinfl;
        y = yinfl;

        if (SurroundingAngle[ishore] == 0.0)
        {
            /* unlikely, but make sure no div by zero */
            slope = 0.00001;
        }
        else if (fabs(SurroundingAngle[ishore]) == 90.0)
        {
```

```
            slope = 9999.9;
        }
        else
        {
            slope = fabs(tan(SurroundingAngle[ishore]));
        }

        BackFlag = 0;
        if (SurroundingAngle[ishore] > 0)
            ysign = 1;
        else
            ysign = -1;

        while ((!BackFlag) && (y > 0) && (y < 2*Ymax) && (x
> 1))
        {
            NextXInt = ceil(x) -1;
            if (ysign > 0)
                NextYInt = floor(y) + 1;
            else
                NextYInt = ceil(y-1);

            /* moving to next whole 'x' position, what is y
position? */
            Ydown = y + (x - NextXInt)*slope * ysign;
            DistanceDown = Raise(((Ydown - y)*(Ydown - y) +
(NextXInt - x)*(NextXInt - x)),.5);

            /* moving to next whole 'y' position, what is x
position? */
            Xside = x - fabs(NextYInt - y) / slope;
            DistanceSide = Raise(((NextYInt - y)*(NextYInt
- y) + (Xside - x)*(Xside - x)),.5);

                if (debug10b) printf("x: %f  y: %f  X:%d
Y: %d  Yd: %f  DistD: %f Xs: %f DistS: %f\n",
                x,y,NextXInt,NextYInt,
Ydown,DistanceDown,Xside,DistanceSide);

            if (DistanceDown < DistanceSide)
            /* next cell is the down cell */
            {
                x = NextXInt;
                y = Ydown;
```

```
                xtest = NextXInt-1;
                ytest = floor(y);
            }
            else
            /* next cell is the side cell */
            {
                x = Xside;
                y = NextYInt;
                xtest = floor(x);
                ytest = y + (ysign-1)/2;
            }

            if (PercentFull[xtest][ytest] > 0)
                BackFlag = 1;
        }

        /* Try to find the i for the cell found */
        /* If you have a better idea how to do this, go
ahead */

        i = 2;
        FoundFlag = 0;

        while ((i < TotalBeachCells-1) && !(FoundFlag))
        {
            if ((X[i] == xtest) && (Y[i] == ytest))
            {
                FoundFlag = 1;
                Backi = i;
            }
            i ++;
        }


        if (!BackFlag)
        /* The search for the backbarrier went out of
bounds - not good, assume big = depthshoreface    */
        /* Periodic B.C.'s should make this not so
important                          */
        {
            Depth = DepthShoreface;
            if (debug10b) printf("\nbackbarrier out of
bounds: xin: %d yin: %d xbi: %d ybi: %d xinf: %f yinf: %f
Per: %f Dist:  Depth: %f\n",
```

```
            xin, yin, xtest, ytest, xinfl,
yinfl,PercentFull[xtest][ytest], Depth);
            /*PauseRun(xin,yin,-1);*/
        }
        else
        {
            BBDistance =  Raise(((xinfl - xtest-
PercentFull[xtest][ytest])*
            (xinfl - xtest-PercentFull[xtest][ytest])) +
            ((yinfl - ytest - 0.5)*(yinfl - ytest -
0.5)),.5);

            if(!FoundFlag)
            /* The backbarrier intersection isn't on the
shoreline */
            /* Assume 1/2 of the length applies to this
case */
            {
                Depth = BBDistance/2 * ShorefaceSlope *
CellWidth;
                if (debug10b) printf("\nNot Found backi: %d
bx: %d by: %d Depth:%f",
                    Backi,xtest,ytest,Depth);
            }
            else
            /* Use the fancy geometry thing */
            {
                AngleUsed = 0;
                for (j = -1; j <2 ; j ++)
                {
                    AngleUsed += SurroundingAngle[Backi+j];
                }
                AngleUsed = AngleUsed/5;

                if (fabs(AngleUsed) > pi/4.0)
                {
                    AngleUsed = pi/4.0;
                    if (debug10b) printf("Big Angle");
                    /*PauseRun(X[Backi],Y[Backi],Backi);*/
                }

                AngleSin = sin(pi/2.0 -
fabs(SurroundingAngle[ishore] + AngleUsed));
```

```
                Depth = BBDistance * AngleSin / (1 +
AngleSin);

        if (debug10b) printf("\nBack Angle backi: %d bx: %d
by: %d BackA: %f AngU: %f Asin: %f L/2: %f Depth:%f",

Backi,X[Backi],Y[Backi],SurroundingAngle[ishore]*radtodeg,A
ngleUsed*radtodeg,AngleSin,
                    BBDistance/2.0,Depth);

            }
        }

        if (Depth < OWMinDepth)
        {
            Depth = OWMinDepth;
        }
        else if (Depth > DepthShoreface)
        {
            Depth = DepthShoreface;
        }
            if (debug10b) printf("\nOverwash Depth2: xin:
%d yin: %d xbi: %d ybi: %d xinf: %f yinf: %f Per: %f Dist:
%f  Depth: %f\n",
            xin, yin, xtest, ytest, xinfl,
yinfl,PercentFull[xtest][ytest],BBDistance, Depth);
        return Depth;
    }

    printf("OWDepth all broken");
    PauseRun(xin,yin,-1);
    return DepthShoreface;
}


float RandWaveAngle(void)

// Function returns a random initial wave angle for thetao
between 90 and 270 degrees

{
    float Angle;
    float RandNumber;
```

```
    RandNumber = rand() % 100;  // Random number between 0
and 100
    Angle = ( (RandNumber * 1.79) + 89 ) * pi/180;

    //RandNumber = RandZeroToOne();
    //RandAngle = rand() % 270 + 90;
    //Angle = RandAngle * pi/180;

    return Angle;
}

void    CalcSedFluxGrad(void)

// Calculates the sediment flux gradient and change is
shoreline position
//
// Uses but does not change the global arrays VolumeIn[]
and VolumeOut[]
// Affects the global arrays SedFluxGrad[] and dEta_dt[]

{
    int y;
    int debugloc = 0; // local debugger

    if (debugloc) printf("SedFluxGrad = \n");

    for (y = Ymax/2; y < 3*Ymax/2; y++){
        //SedFluxGrad[y] = VolumeIn[y] - VolumeOut[y];
        SedFluxGrad[y] = VolumeOut[y] - VolumeIn[y];

        if (debugloc) printf("%G ", SedFluxGrad[y]);
    }

    if (debugloc) printf("dEta_dt = \n");

    for (y = Ymax/2; y < 3*Ymax/2; y++){
        dEta_dt[y+1] = - SedFluxGrad[y] / DepthShoreface;
        //dEta_dt[y] = SedFluxGrad[y] / DepthShoreface;

        if (debugloc) printf("%G ", dEta_dt[y]);
    }


}
```

```
void CalcDiffusivity(void)

// Calculates the diffusivity and sum of (Diffusivity *
Delta t) of each beach cell for current time step
//
// Uses but does not affect global array SedFluxGrad[]
// Affects global arrays Diffusivity[], SumMuDeltaT[], and
SumAbsMuDeltaT[]
// Affects global variable GammaCounter

{
    int y;
    float K2 = 0.34;    // in m^(3/5)*s^(-6/5)
    float DeltaTheta;   // Change in shore angle
    float dTheta_dx;    // Change is shoreline angle w/
respect to x
    int debugloc = 0;   // local debugger


    if (debugloc) printf("Diffusivity = \n");

    for (y = Ymax/2; y < 3*Ymax/2; y++){
        // Diffusivity[y] = (SedFluxGrad[y] * (
(6/5)*(pow(sin(WaveAngle-ShorelineAngle[y]),2))
            // - pow(cos(WaveAngle-ShorelineAngle[y]),2) )
) / (DepthShoreface * (cos(WaveAngle-ShorelineAngle[y]))
            // * (sin(WaveAngle-ShorelineAngle[y])) );
        Diffusivity[y] = ( K2 * pow(period,0.2) *
pow(OffShoreWvHt, (12/5))
            * pow(cos(WaveAngle-ShorelineAngle[y]),(6/5)) *
sin(WaveAngle-ShorelineAngle[y])
            * ( (6/5)* pow(sin(WaveAngle-
ShorelineAngle[y]),2) - pow(cos(WaveAngle-
ShorelineAngle[y]),2) ) )
            / (DepthShoreface * (cos(WaveAngle-
ShorelineAngle[y])) * (sin(WaveAngle-ShorelineAngle[y])) );
        //Diffusivity[y] = -(Qs[y+1] - Qs[y]) /
(DepthShoreface * (ShorelineAngle[y+1] -
ShorelineAngle[y]));
        //Diffusivity[y] = -(Qs[y+1] - Qs[y-1]) /
(DepthShoreface * (ShorelineAngle[y+1] - ShorelineAngle[y-
1]));
```

```
        //DeltaTheta = (ShorelineAngle[y+1] -
ShorelineAngle[y]);
        DeltaTheta = (ShorelineAngle[y+1] -
ShorelineAngle[y-1]);

        if (DeltaTheta == 0){
            Mu[y] = 0;
        } else {
            //Mu[y] = -(Qs[y+1] - Qs[y]) / (DepthShoreface
* DeltaTheta);
            //if (Mu[y] > 2000) Mu[y] = 0;
            Mu[y] = -(Qs[y+1] - Qs[y-1]) / (DepthShoreface
* DeltaTheta);
        }

        //dTheta_dx = DeltaTheta/CellWidth;

        //Mu[y] = dEta_dt[y] / dTheta_dx;

        // Determine when to reset arrays to calcualte
Gamma
        if (GammaCounter == 0){
            GammaCounter = 1;
        } else if (GammaCounter == MaxGammaCounter){
            GammaCounter = 0;
            SumMuDeltaT[y] = 0;
            SumAbsMuDeltaT[y] = 0;
        } else {
            GammaCounter++;
        }

        //SumMuDeltaT[y] = SumMuDeltaT[y] + (Diffusivity[y]
* TimeStep);
        //SumAbsMuDeltaT[y] = SumAbsMuDeltaT[y] +
((fabs(Diffusivity[y])) * TimeStep);
        SumMuDeltaT[y] = SumMuDeltaT[y] + (Mu[y] *
TimeStep);
        SumAbsMuDeltaT[y] = SumAbsMuDeltaT[y] +
((fabs(Mu[y])) * TimeStep);

        //if (debugloc) printf("%G ", Diffusivity[y]);
        if (debugloc) printf("%G ", Mu[y]);

    }
```

```
}

void CalcGamma(void)

// Calculates the instability index of each beach cell for
current time step
//
// Uses but does not affect global arrays SumMuDeltaT[] and
SumAbsMuDeltaT[]
// Affects global arrays Gamma[]

{
    int y;
    int debugloc = 0; // local debugger

    if (debugloc) printf("Instability Index = \n");

    for (y = Ymax/2; y < 3*Ymax/2; y++){
        Gamma[y] = SumMuDeltaT[y] / SumAbsMuDeltaT[y];

        if (debugloc) printf("%G ", Gamma[y]);
    }
}

void RunAWTOffShore (void)

// Uses the Ashton Wave Transformation to propagate
offshore waves to start of WRT domain
//
// Uses global variable WaveAngle, Period, and OffShoreWvHt
// Uses and modifies global variables StartAngle and
StartHeight

{
    /* Coefficients - some of these are important*/

    float   StartDepth = 200;   /* m, depth to begin
refraction calcs (needs to be beyond breakers) */
    float   RefractStep = .2;   /* m, step size to iterate
depth for refraction calcs           */
    //float     KBreak = 0.5;       /* coefficient for wave
breaking threshold              */
    float   rho = 1020;     /* kg/m3 - density of water and
dissolved matter                */
```

```
    /* Variables */

    float   AngleDeep;      /* rad, Angle of waves to shore
at inner shelf  */
    float   Depth = StartDepth; /* m, water depth for
current iteration     */
    float   Angle;          /* rad, calculation angle
*/
    float   CDeep;          /* m/s, phase velocity in deep
water         */
    float   LDeep;          /* m, offhsore wavelength
*/
    float   C;          /* m/s, current step phase velocity
*/
    float   kh;         /* wavenumber times depth
*/
    float   n;          /* n                           */
    float   WaveLength;     /* m, current wavelength
*/
    float   WvHeight;       /* m, current wave height
*/
    int     y_coord;        // y-coordinate of the beach
cell "From"
    int     debug6a = 1;    // Print volume only
    int     debug19a = 0;   // local debugger
    int     debug19b = 0;   // local debugger

    AngleDeep = WaveAngle;

    /* Calculate Deep Water Celerity & Length, Komar 5.11 c
= gT / pi, L = CT   */

    CDeep = g * Period / (2.0 * pi);
    LDeep = CDeep * Period;
    if (debug6) printf("CDeep = %2.2f LDeep = %2.2f
\n",CDeep, LDeep);


    // Run while loop until you reach the edge of WRT
domain (at MaxDepth)
    while (Depth >= MaxDepth + (.001*StartFlatBathy)){
```

```
        /* non-iterative eqn for L, from Fenton & McKee
*/

        WaveLength = LDeep *
Raise(tanh(Raise(Raise(2.0*pi/Period,2)*Depth/g,.75)),2.0/3
.0);
        C = WaveLength/Period;
        if (debug6) printf("DEPTH: %2.2f Wavelength = %2.2f
C = %2.2f ", Depth, WaveLength,C);

        /* Determine n = 1/2(1+2kh/tanh(kh)) Komar 5.21
*/
        /* First Calculate kh = 2 pi Depth/L  from k = 2
pi/L       */

        kh =  pi * Depth / WaveLength;
        n =0.5 * ( 1 + 2.0 * kh / sinh(2.0*kh));
        if (debug6) printf("kh: %2.3f  n: %2.3f ", kh, n);

        /* Calculate angle, assuming shore parallel
contours and no conv/div of rays    */
        /* from Komar 5.47                              */

        Angle = asin(C/CDeep * sin(AngleDeep));
        if (debug6) printf("Angle: %2.2f",Angle*radtodeg);

        /* Determine Wave height from refract calcs - Komar
5.49            */

        WvHeight = OffShoreWvHt *
Raise(CDeep*cos(AngleDeep)/(C*2.0*n*cos(Angle)),.5);
        if (debug6) printf(" WvHeight : %2.3f\n",WvHeight);

        if (Depth == RefractStep){
            Depth -= RefractStep;
        } else
        Depth -= RefractStep;
    }

    // Determine starting wave angle & height for WRT
    StartAngle = Angle;
    StartHeight = WvHeight;

}
```

```
void CalcOverallAverages(void)

// Calculates the average breaking height, relative
breaking angle, and Qs over the entire
// simulation

{
    int i;

    for (i = 0; i < 2*Ymax; i++){
        SumBkHeight[i] = SumBkHeight[i] + HeightRollAvg[i];
        SumRelAngle[i] = SumRelAngle[i] + RelAngle[i];
        SumQs[i] = SumQs[i] + Qs[i];
    }

    if (CurrentTimeStep > 0){
        for (i = 0; i < 2*Ymax; i++){
            TotalAvgBkHeight[i] =
SumBkHeight[i]/CurrentTimeStep;
            TotalAvgRelAngle[i] =
SumRelAngle[i]/CurrentTimeStep;
            TotalAvgQs[i] = SumQs[i]/CurrentTimeStep;
        }
    }

}

void DisplayAWTWvHeight(void)

// Displays the AWT wave height as it changes during
propagation

{

    int x,y;

    if (debug21){

        printf("AWTWvHeight = \n");

        for (y = Ymax/2; y < 3*Ymax/2; y++){
            for (x = 0; x < Xmax; x++){
                printf("%G ", AWTWvHeight[x][y]);
```

```
            }
            printf("\n");
        }

    }

}



void RunWRT(void)

// Runs the Wave Ray Tracer (foward wave ray tracing)
//
// Uses the global arrays PercentFull[][] and AllBeach[][]
// May utilize global variable WaveAngle
// Affects the global arrays breaking_heights[] and
breaking_angles[][]

{

    //float     KBreak = 0.5;       // coefficient for wave
breaking threshold

    // Initialize wave parameters
    float Ho = StartHeight; // Initial deep water wave
height
    int x;
    int y;
    int i;

    // Find the cross-shore position of the beach for all
columns
    if (debug11) printf("beach = ");
    for (x = 0; x < Xmax; x++){
        //for (y = Ymax/2 - 1; y < 3*Ymax/2; y++){
        for (y = 0; y < 2*Ymax; y++){
            if ((PercentFull[x][y] > 0) &&
(PercentFull[x][y] < 1)){
                beach[y] = x + 1;
                if (debug11) printf("%i ", beach[y]);
            }
        }
```

```
    } if (debug11) printf("\n");
    if (debug11) printf("\n");

    // Define depth at all cells
    float calc_slope; // Calculated bathymetry slope
    int start_slope; // y-coordinate of start of slope
    float Xo; // Distance from coastline to edge of beach
cell
    float Zo; // Initial depth at 1st beach cell
    float Zf; // Depth where the bathymetry starts to
flatten out
    int counter;
    for (y = 0; y < 2*Ymax; y++){
        for (x = 0; x < Xmax + 1; x++){
            if (AllBeach[x][y] == 'y'){
                ZZ[x][y] = 0;
                counter = 0;
            } else if (UseDeanProfile == 'y'){
                if ((AllBeach[x][y] == 'n') &&
(AllBeach[(x-1)][y] == 'y')) {
                    Xo = CellWidth * (1-PercentFull[x][y]);
                    Zo = .1 * pow((Xo/2), (.66667)); // Not
exactly center of cell, but that's ok
                    ZZ[x][y] = Zo;
                    counter = 1;
                } else if ((AllBeach[x][y] == 'n') &&
(AllBeach[(x-1)][y] == 'n')) {
                    if (x >= (Xmax - StartFlatBathy)){
                        ZZ[x][y] = ZZ[x-1][y] + .002;
                    } else if (counter < 10) {
                        ZZ[x][y] = .1 * pow( (Xo +
(CellWidth * counter) - (CellWidth/2)), (.66667));
                        counter++;
                    } else if (counter = 10) {
                        calc_slope = (MaxDepth - ZZ[x-
1][y])/(Xmax - StartFlatBathy - x);
                        Zf = ZZ[x-1][y] + calc_slope;
                        ZZ[x][y] = Zf;
                        counter++;
                    } else if (counter > 10) {
                        ZZ[x][y] = ZZ[x-1][y] + calc_slope;
                        counter++;
                    }
                }
```

```
            } else if (UseDeanMirror == 'y'){
                if ((AllBeach[x][y] == 'n') &&
(AllBeach[(x-1)][y] == 'y')) {
                    Xo = CellWidth * (1-PercentFull[x][y]);
                    Zo = .1 * pow(((CellWidth - Xo)/2),
(.66667)); // Not exactly center of cell, but that's ok
                    ZZ[x][y] = Zo;
                    counter = 1;
                } else if ((AllBeach[x][y] == 'n') &&
(AllBeach[(x-1)][y] == 'n')) {
                    if (counter <= 10) {
                        //ZZ[x][y] = Zo + .1 *
pow((CellWidth * counter), (.66667));
                        ZZ[x][y] = .1 * pow( (Xo +
(CellWidth * (counter-1)) - (CellWidth/2)), (.66667));
                        counter++;
                    } else if (counter > 10) {
                        ZZ[x][y] = ZZ[x-1][y] +
.002*CellWidth;
                        counter++;
                    }
                }
            } else if ((x >= Xmax - StartFlatBathy) &&
(UseFixedBathySlope == 'n')){
                ZZ[x][y] = MaxDepth + Zo;
            } else if ((AllBeach[x][y] == 'n') &&
(AllBeach[(x-1)][y] == 'y') && (UseFixedBathySlope == 'y'))
{
                Zo = BathySlope*(1-PercentFull[x][y]);
                ZZ[x][y] = Zo;
                counter = 1;
            } else if ((AllBeach[x][y] == 'n') &&
(AllBeach[(x-1)][y] == 'n') && (UseFixedBathySlope == 'y'))
{
                counter++;
                ZZ[x][y] = Zo + (BathySlope * counter);
            } else if ((AllBeach[x][y] == 'n') &&
(AllBeach[(x-1)][y] == 'y')) {
                start_slope = x;
                calc_slope =  MaxDepth/(Xmax -
StartFlatBathy - start_slope);
                Zo = calc_slope*(1-PercentFull[x][y]);
                ZZ[x][y] = Zo;
                counter = 1;
```

```
            } else if (AllBeach[x][y] == 'n') {
                counter++;
                ZZ[x][y] = (calc_slope * counter) + Zo;
            }
        }
    }

    // Make sure that no cells have a depth greater than
MaxDepth
    for (y = 0; y < 2*Ymax; y++){
        for (x = 0; x < Xmax + 1; x++){
            if (ZZ[x][y] > MaxDepth){
                ZZ[x][y] = MaxDepth;
            }
        }
    }

    // Diffuse the bathymetry and establish depth
    if ((CurrentTimeStep > StartWRTAfter) &&
(DiffuseBathymetry == 'y')){
        for (y = Ymax/2; y < 3*Ymax/2; y++){
            DiffusedDepth[Xmax-1][y] = ZZ[Xmax-1][y];
            depth[Xmax-1][y] = DiffusedDepth[Xmax-1][y];
            for (x = 0; x < Xmax - 1; x++){
                DiffusedDepth[x][y] = DiffusedDepth[x][y] +
(TimeStep*Kappa/CellWidth*CellWidth)*(ZZ[x+1][y]
                    + ZZ[x-1][y] + ZZ[x][y+1] + ZZ[x][y-1]
- 4*ZZ[x][y]);
                depth[x][y] = DiffusedDepth[x][y];
            }
        }
    } else if ((CurrentTimeStep <= StartWRTAfter) &&
(DiffuseBathymetry == 'y')){
        for (y = Ymax/2; y < 3*Ymax/2; y++){
            DiffusedDepth[Xmax-1][y] = ZZ[Xmax-1][y];
            depth[Xmax-1][y] = DiffusedDepth[Xmax-1][y];
            for (x = 0; x < Xmax - 1; x++){
                DiffusedDepth[x][y] = ZZ[x][y] +
(TimeStep*Kappa/CellWidth*CellWidth)*(ZZ[x+1][y] +
                    ZZ[x-1][y] + ZZ[x][y+1] + ZZ[x][y-1] -
4*ZZ[x][y]);
                depth[x][y] = DiffusedDepth[x][y];
            }
        }
```

```
    } else {
        for (y = Ymax/2; y < 3*Ymax/2; y++){
            for (x = 0; x < Xmax; x++){
                depth[x][y] = ZZ[x][y];
            }
        }
    }

    // Copy bathymetry for periodic boundaries
    for (y = Ymax; y < 3*Ymax/2; y++){
        for (x = 0; x < Xmax; x++){
            DiffusedDepth[x][y-Ymax] = DiffusedDepth[x][y];
            depth[x][y-Ymax] = depth[x][y];
        }
    }
    for (y = Ymax/2; y <= Ymax; y++){
        for (x = 0; x < Xmax; x++){
            DiffusedDepth[x][y+Ymax] = DiffusedDepth[x][y];
            depth[x][y+Ymax] = depth[x][y];
        }
    }

    for (x = 0; x < Xmax; x++){
        DiffusedDepth[x][0] = DiffusedDepth[x][Ymax];
        depth[x][0] = depth[x][Ymax];
    }

    // Display depth at all cells
    if (debug11) printf("ZZ = \n");
    for (y=0; y < 2*Ymax; y++){
        for (x=0; x < Xmax; x++){
            if (debug11) printf("%.1f ", ZZ[x][y]);
        } if (debug11) printf("\n");
    } if (debug11) printf("\n");

    if (DiffuseBathymetry == 'y'){
        if (debug11) printf("DiffusedDepth = \n");
        for (y=0; y < 2*Ymax; y++){
            for (x=0; x < Xmax; x++){
                if (debug11) printf("%.1f ",
DiffusedDepth[x][y]);
            } if (debug11) printf("\n");
        } if (debug11) printf("\n");
    }
```

```
    if (debug11 || debug11a) printf("depth = \n");
    for (y=0; y < 2*Ymax; y++){
        for (x=0; x < Xmax; x++){
            if (debug11 || debug11a) printf("%G ",
depth[x][y]);
        } if (debug11 || debug11a) printf("\n");
    } if (debug11 || debug11a) printf("\n");



    // Define additional parameters
    int dx = CellWidth;
    int dy = CellWidth;
    //int x_int = (CellWidth * Xmax) - (2 * CellWidth) +
(CellWidth/2);
    int x_int = (CellWidth * Xmax) - (2 * CellWidth);
    int beta_int = 1;
    int sigma_int = 0;
    float h_int = StartHeight;
    int sf = 0;
    int si = (CellWidth/4);
    int ds = 0.5;
    if (debug11) printf("Additional parameters: \n");
    if (debug11) printf("dx = %i \n", dx);
    if (debug11) printf("dy = %i \n", dy);
    if (debug11) printf("x_int = %i \n", x_int);
    if (debug11) printf("y_int = ");
    for (i=0; i < (Ymax*2); i++){
        //y_int[i] = ((i + 1) * CellWidth) + (CellWidth/2);
        y_int[i] = (i + 1) * CellWidth;
        if (debug11) printf("%i ", y_int[i]);
    } if (debug11) printf("\n");
        if (debug11) printf("beta_int = %i \n", beta_int);
    if (debug11) printf("sigma_int = %i \n", sigma_int);
    if (debug11) printf("h_int = %.1f \n", h_int);
    if (debug11) printf("g = %.2f \n", g);
    if (debug11) printf("sf = %i \n", sf);
    if (debug11) printf("si = %i \n", si);
    if (debug11) printf("ds = %i \n", ds);

    float period_now = T;
    float period = period_now;
    float wavelength_int = (g * pow(period, 2)) / (2 * Pi);
```

```
    float velocity_int = pow(wavelength_int, 2) /
pow(period, 2);
    if (debug11) printf("period now = %.1f \n",
period_now);
    if (debug11) printf("period = %.1f \n", period);
    if (debug11) printf("wavelength_int = %.2f \n",
wavelength_int);
    if (debug11) printf("velocity_int = %.2f \n",
velocity_int);
    if (debug11) printf("\n");

    // Define x- and y-position matrices
    if (debug11) printf("x_pos = \n");
    for (i=0; i < Xmax; i++){
        //x_pos[i] = 1 + (CellWidth * i);
        x_pos[i] = CellWidth * i;
        if (debug11) printf("%i ", x_pos[i]);
    } if (debug11) printf("\n");
    if (debug11) printf("\n");

    if (debug11) printf("y_pos = \n");
    for (i=0; i < 2*Ymax+1; i++){
        //y_pos[i] = 1 + (CellWidth * i);
        y_pos[i] = CellWidth * i;
        if (debug11) printf("%i ", y_pos[i]);
    } if (debug11) printf("\n");
    if (debug11) printf("\n");

    // Define the deep water wavelength (L_0)
    float L_0;
    int ii;
    if (debug11) printf("L_0 = ");
    L_0 = ((g*T*T)/(2*Pi));
    if (debug11) printf("%.2f \n", L_0);
    if (debug11) printf("\n");

    // Define the shallow water wavelength (L) for all
cells
    if (debug11) printf("L = \n");
    for (ii=0; ii < 2*Ymax; ii++){
        for (i=0; i < Xmax; i++){
            L[i][ii] = L_0 * pow( (tanh(
pow((2*Pi/T)*(2*Pi/T)*(depth[i][ii] / g) , .75) )) ,
(0.6666667) );
```

```
            if (debug11) printf("%.2f ", L[i][ii]);
        } if (debug11) printf("\n");
    } if (debug11) printf("\n");

    // Define wave celerity (C or c) for all cells
    if (debug11) printf("c = \n");
    for (ii=0; ii < 2*Ymax; ii++){
        for (i=0; i < Xmax; i++){
            c[i][ii] = (L[i][ii])/T;
            if (debug11) printf("%G ", c[i][ii]);
        } if (debug11) printf("\n");
    } if (debug11) printf("\n");

    //Run loop to fill cx and c2x2 with gradients of c & cx
    // cx
    if (debug11) printf("cx = \n");
    for (ii=0; ii < 2*Ymax; ii++){
        for (i=0; i < Xmax; i++){
            if (PercentFull[i][ii] >= 1){
                cx[i][ii] = 0;
            } else {
                cx[i][ii] = (c[i+1][ii] - c[i-
1][ii])/(2*dx); // Old way of doing grads
                // cx[i][ii] = (c[(i+1)][ii] -
c[i][ii])/dx;
                cx[Xmax-1][ii] = cx[Xmax-2][ii];
            }
            if (debug11) printf("%G ", cx[i][ii]);
        } if (debug11) printf("\n");
    } if (debug11) printf("\n");

    // c2x2
    if (debug11) printf("c2x2 = \n");
    for (ii=0; ii < 2*Ymax; ii++){
        for (i=0; i < Xmax; i++){
            if (PercentFull[i][ii] >= 1){
                c2x2[i][ii] = 0;
            } else {
                c2x2[i][ii] = (((c[i+1][ii] - c[i][ii])/dx)
- ((c[i][ii] - c[i-1][ii])/dx))/dx;
                // c2x2[i][ii] = (cx[(i+1)][ii] -
cx[i][ii])/dx;
                //c2x2[i][ii] = (cx[i+1][ii] - cx[i-
1][ii])/(2*dx); // Old way of doing grads
```

```
            }
            if (debug11) printf("%G ", c2x2[i][ii]);
        } if (debug11) printf("\n");
    } if (debug11) printf("\n");


    // Run loop to fill cy, c2y2, & cxcy with appropriate
gradients

    // cy
    if (debug11) printf("cy = \n");
    // for (i=0; i < Xmax; i++){
        // cy[i][0] = (c[i][1] - c[i][0])/dy;
        // if (debug11) printf("%G ", cy[i][0]);
    // } if (debug11) printf("\n");
    // for (ii=1; ii < (2*Ymax - 1); ii++){
        // for (i=0; i < Xmax; i++){
            // cy[i][ii] = (c[i][(ii+1)] - c[i][ii])/dy;
            // if (debug11) printf("%G ", cy[i][ii]);
        // } if (debug11) printf("\n");
    // }
    // for (i=0; i < Xmax; i++){
        // cy[i][(2*Ymax-1)] = (c[i][0] - c[i][(2*Ymax-
1)])/dy;
        // if (debug11) printf("%G ", cy[i][(2*Ymax-1)]);
    // } if (debug11) printf("\n");
    for (i=0; i < Xmax; i++){
        cy[i][0] = (c[i][1] - c[i][(2*Ymax - 1)])/(2*dy);
// Old way
        if (debug11) printf("%G ", cy[i][0]);
    } if (debug11) printf("\n");
    for (ii=1; ii < (2*Ymax - 1); ii++){
        for (i=0; i < Xmax; i++){
            cy[i][ii] = (c[i][ii+1] - c[i][ii-1])/(2*dy);
            if (debug11) printf("%G ", cy[i][ii]);
        } if (debug11) printf("\n");
    }
    for (i=0; i < Xmax; i++){
        cy[i][(2*Ymax-1)] = (c[i][0] - c[i][(2*Ymax -
2)])/(2*dy);
        if (debug11) printf("%G ", cy[i][(2*Ymax-1)]);
    } if (debug11) printf("\n");
    if (debug11) printf("\n");
```

```
    // c2y2
    if (debug11) printf("c2y2 = \n");
    // for (i=0; i < Xmax; i++){
        // c2y2[i][0] = (cy[i][1] - cy[i][0])/dy;
        // if (debug11) printf("%G ", c2y2[i][0]);
    // } if (debug11) printf("\n");
    // for (ii=1; ii < (2*Ymax - 1); ii++){
        // for (i=0; i < Xmax; i++){
            // c2y2[i][ii] = (cy[i][(ii+1)] -
cy[i][ii])/dy;
            // if (debug11) printf("%G ", c2y2[i][ii]);
        // } if (debug11) printf("\n");
    // }
    // for (i=0; i < Xmax; i++){
        // c2y2[i][(2*Ymax - 1)] = (cy[i][(2*Ymax - 1)] -
cy[i][0])/dy;
        // if (debug11) printf("%G ", c2y2[i][(2*Ymax -
1)]);
    // } if (debug11) printf("\n");
    for (i=0; i < Xmax; i++){
        //c2y2[i][0] = (cy[i][1] - cy[i][2*Ymax -
1])/(2*dy);
        c2y2[i][0] = (((c[i][1] - c[i][0])/dy) - ((c[i][1]
- c[i][2*Ymax-1])/dy))/dy;
        if (debug11) printf("%G ", c2y2[i][0]);
    } if (debug11) printf("\n");
    for (ii=1; ii < (2*Ymax - 1); ii++){
        for (i=0; i < Xmax; i++){
            //c2y2[i][ii] = (cy[i][ii+1] - cy[i][ii-
1])/(2*dy);
            c2y2 [i][ii] = (((c[i][ii+1] - c[i][ii])/dy) -
((c[i][ii] - c[i][ii-1])/dy))/dy;
            if (debug11) printf("%G ", c2y2[i][ii]);
        } if (debug11) printf("\n");
    }
    for (i=0; i < Xmax; i++){
        //c2y2[i][(2*Ymax - 1)] = (cy[i][0] - cy[i][2*Ymax
- 2])/(2*dy);
        c2y2[i][2*Ymax-1] = (((c[i][0] - c[i][2*Ymax-
1])/dy) - ((c[i][2*Ymax-1] - c[i][2*Ymax-2])/dy))/dy;
        if (debug11) printf("%G ", c2y2[i][2*Ymax - 1]);
    } if (debug11) printf("\n");
    if (debug11) printf("\n");
```

```
    // cxcy
    float cxcy1;    float cxcy2;
    float cxcy3;    float cxcy4;
    if (debug11) printf("cxcy = \n");
    // for (i=0; i < Xmax; i++){
        // cxcy[i][0] = (cx[i][1] - cx[i][0])/dy;
        // if (debug11) printf("%G ", cxcy[i][0]);
    // } if (debug11) printf("\n");
    // for (ii=1; ii < (2*Ymax - 1); ii++){
        // for (i=0; i < Xmax; i++){
            // cxcy[i][ii] = (cx[i][(ii+1)] -
cx[i][ii])/dy;
            // if (debug11) printf("%G ", cxcy[i][ii]);
        // } if (debug11) printf("\n");
    // }
    // for (i=0; i < Xmax; i++){
        // cxcy[i][(2*Ymax - 1)] = (cx[i][0] -
cx[i][(2*Ymax - 1)])/dy;
        // if (debug11) printf("%G ", cxcy[i][(2*Ymax -
1)]);
    // } if (debug11) printf("\n");

    for (i=0; i < Xmax-1; i++){
            if (PercentFull[i][ii] >= 1){
                cxcy[i][ii] = 0;
            } else {
                //cxcy[i][0] = (cx[i][1] - cx[i][(2*Ymax -
1)])/(2*dy); // OLD WAY
                cxcy1 = (((c[i+1][1] - c[i][1])/dx) -
((c[i+1][0] - c[i][0])/dx))/dy;
                cxcy2 = (((c[i][1] - c[i-1][1])/dx) -
((c[i][0] - c[i-1][0])/dx))/dy;
                cxcy3 = (((c[i][0] - c[i-1][0])/dx) -
((c[i][2*Ymax-1] - c[i-1][2*Ymax-1])/dx))/dy;
                cxcy4 = (((c[i+1][0] - c[i][0])/dx) -
((c[i+1][2*Ymax-1] - c[i][2*Ymax-1])/dx))/dy;
                cxcy[i][0] = (cxcy1 + cxcy2 + cxcy3 +
cxcy4)/4;
            }
        if (debug11) printf("%G ", cxcy[i][0]);
    } if (debug11) printf("\n");
    for (ii=1; ii < (2*Ymax - 1); ii++){
        for (i = 0; i < Xmax; i++) {
            if (PercentFull[i][ii] >= 1){
```

```
                cxcy[i][ii] = 0;
            } else {
                //cxcy[i][ii] = (cx[i][(ii+1)] - cx[i][(ii-
1)])/(2*dy); // OLD WAY
                cxcy1 = (((c[i+1][ii+1] - c[i][ii+1])/dx) -
((c[i+1][ii] - c[i][ii])/dx))/dy;
                cxcy2 = (((c[i][ii+1] - c[i-1][ii+1])/dx) -
((c[i][ii] - c[i-1][ii])/dx))/dy;
                cxcy3 = (((c[i][ii] - c[i-1][ii])/dx) -
((c[i][ii-1] - c[i-1][ii-1])/dx))/dy;
                cxcy4 = (((c[i+1][ii] - c[i][ii])/dx) -
((c[i+1][ii-1] - c[i][ii-1])/dx))/dy;
                cxcy[i][ii] = (cxcy1 + cxcy2 + cxcy3 +
cxcy4)/4;
            }
            if (debug11) printf("%G ", cxcy[i][ii]);
        } if (debug11) printf("\n");
    }
    for (i=0; i < Xmax; i++){
        if (PercentFull[i][ii] >= 1){
            cxcy[i][ii] = 0;
        } else {
            //cxcy[i][(2*Ymax - 1)] = (cx[i][0] -
cx[i][(2*Ymax - 2)])/(2*dy); // OLD WAY
            cxcy1 = (((c[i+1][0] - c[i][0])/dx) -
((c[i+1][2*Ymax - 1] - c[i][2*Ymax - 1])/dx))/dy;
            cxcy2 = (((c[i][0] - c[i-1][0])/dx) -
((c[i][2*Ymax - 1] - c[i-1][2*Ymax - 1])/dx))/dy;
            cxcy3 = (((c[i][2*Ymax-1] - c[i-1][2*Ymax-
1])/dx) - ((c[i][2*Ymax - 2] - c[i-1][2*Ymax-2])/dx))/dy;
            cxcy4 = (((c[i+1][2*Ymax-1] - c[i][2*Ymax-
1])/dx) - ((c[i+1][2*Ymax-2] - c[i][2*Ymax-2])/dx))/dy;
            cxcy[i][2*Ymax-1] = (cxcy1 + cxcy2 + cxcy3 +
cxcy4)/4;
        }
        if (debug11) printf("%G ", cxcy[i][(2*Ymax - 1)]);
    } if (debug11) printf("\n");
    if (debug11) printf("\n");


    // Define wavenumber (k) for each cell
    //float k_d;
    float extra_depth;
    float depth_now;
```

```
    //k_d = (pow(2*3.14159265358, 2)) / (g * pow(period,
2));
    // printf("k_d = %.2f \n", k_d);
    // printf("\n");

    if (debug11) printf("k = \n");
    for (ii=0; ii < 2*Ymax; ii++){
        for (i=0; i < Xmax; i++){
            depth_now = depth[i][ii];
            if (depth_now <= 0){
                k[i][ii] = 0;
                if (debug11) printf("%G ", k[i][ii]);
            } else if (depth_now > 0){
                k[i][ii] = 2*pi/L[i][ii];
                if (debug11) printf("%G ", k[i][ii]);
            }
        }  if (debug11) printf("\n");
    } if (debug11) printf("\n");

    // Calculate group velocity for all cells
    if (debug11) printf("C_g = \n");
    for (ii=0; ii < 2*Ymax; ii++){
        for (i=0; i < Xmax; i++){
            if (c[i][ii] <= 0){
                C_g[i][ii] = 0;
                if (debug11) printf("%G ", C_g[i][ii]);
            } else if (c[i][ii] > 0){
                C_g[i][ii] = (c[i][ii] / 2) *
(1+(2*k[i][ii]*depth[i][ii])/(sinh(2*k[i][ii]*depth[i][ii])
));
                if (debug11) printf("%G ", C_g[i][ii]);
            }
        } if (debug11) printf("\n");
    } if (debug11) printf("\n");


////////////////////////////////////////////////////////////
//////////////////////////////////////
    if (debug0) printf("------- Main WRT Loop ------- \n");
    if (debug0) printf("\n");

    // Initialize loop parameters
    int wave_origin;    // y-coordinate of wave's origin
    float theta_int;
```

```
    float x_now;        // Current value for x (x-position
of ray)
    float y_now;        // Current value for y (y-position
of ray)
    float theta_now;    // Current value for theta (angle
of ray)
    float beta_now;     // Current value for beta (distance
between rays)
    float sigma_now;    // Current value for sigma (change
in beta wrt time)
    float h_now;        // Current value for H (wave
height)
    int x_crd = (Xmax-1);   // Current cell x-coordinate
    int y_crd = 0;          // Current cell y-coordinate
    float h_b;
    int z;
    float break_depth;
    // float x_store[9000];
    // float y_store[9000];
    // float theta_store[9000];
    // float beta_store[9000];
    // float sigma_store[9000];
    // float h_store[9000];
    float beta_calc;
    float theta_deg;        // theta_now in degrees
    int N_rays = 0;         // Number of wave rays shot
during simulation
    char DataForAllBeachCells = 'n';    // Specifies
whether or not there is waveheight data at all beach cells
    float shore_position;   // Shore position

    int debug21a = 0;   // Local debugger (to see if beta
goes negative)


    // CLEAR ALL TRACKS
    for (y = 0; y < 2*Ymax; y++){
        avg_breaking_height[y] = 0;
        avg_breaking_angle[y] = 0;
        breaking_angles[y] = 0;
        breaking_heights[y] = 0;
        breaking_angle[y] = 0;
        breaking_height[y] =0;
        sum_breaking_angle[y] = 0;
```

```
            sum_breaking_height[y] = 0;
            N_rays_hit[y] = 0;

            for (x = 0; x < Xmax; x++){
                cell_wave_height[x][y] = 0;
                sum_cell_wave_height[x][y] = 0;
                avg_cell_wave_height[x][y] = 0;
                cell_beta[x][y] = 0;
                sum_cell_beta[x][y] = 0;
                avg_cell_beta[x][y] = 0;
                NumRaysPerCell[x][y] = 0;
                RayInCell[x][y] = 0;
            }
        }
        N_rays = 0;

        // Run loop until there is waveheight data for all
beach cells
        int sweep;
        for (sweep = 1; sweep < (NumSweeps + 1); sweep++){
        //while (DataForAllBeachCells == 'n'){

            printf("Sweep number: %i \n", sweep);
            //x_now = x_int + ((1-sweep) *
(CellWidth/NumSweeps));
            //printf("x_now = %G \n", x_now);

            // Use a random deep-water wave angle?
            float theta;
            if (UseRandWaveAngle == 'y'){
                theta = RandWaveAngle();
            } else if (UseWaveDist == 'y'){
                theta = FindWaveAngle();
                theta = theta + pi;
            } else if (LinkToSedTrans == 'y'){
                //theta = WaveAngle + pi;
                theta = pi - WaveAngle;
            } else if (UseAWTOffshore == 'y'){
                theta = pi - StartAngle;
            } else {
                theta = (180-thetao)*pi/180;  // Initial deep
water wave angle;
                //theta = pi;
            }
```

```
        if (debug18) printf("Initial angle = %G degrees
\n", theta*180/pi);

        // Makes sure theta is in the domain (to account
for fluctuation change & remove very high angles)
        if (theta < (96 * (pi/180))) {
            theta = 96 * (pi/180);
            printf("Adjusting initial angle to %G \n",
theta*radtodeg);
        } else if (theta > (264 * (pi/180))) {
            theta = 264 * (pi/180);
            printf("Adjusting initial angle to %G \n",
theta*radtodeg);
        }

        // Use a random angle fluctuation?
        float fluctuation;
        fluctuation = RandAngleFluctuation() * (pi/180);
        if (UseRandAngleFluctuation == 'y') theta = theta +
fluctuation;

        if (debug18) printf("theta = %G , fluctuation = %G
\n", theta*radtodeg, fluctuation*radtodeg);
        if (debug18) printf("\n");

    // Run loop to solve differential equations
    if (debug0) printf("--- ODE Solver Loop --- \n");

    //for (wave_origin = (Ymax/2) - 1; wave_origin <
3*Ymax/2; wave_origin++){
    for (wave_origin = 0; wave_origin < (2*Ymax-1);
wave_origin++){
    //for (wave_origin = (Ymax/2) - 1 - (Ymax/4);
wave_origin < 3*Ymax/2 + (Ymax/4); wave_origin++){
    //for (wave_origin = 50; wave_origin < 51;
wave_origin++){

        //Use a random deep-water wave angle? (*** PUT THIS
IN IF USING WHILE LOOP!! ***)
        // if (UseRandWaveAngle == 'y'){
            // theta = RandWaveAngle();
        // } else if (UseWaveDist == 'y'){
            // theta = FindWaveAngle();
            // theta = theta + pi;
```

```
            // theta_d = theta*180/pi;
            // if (debug18) printf("For wave ray
originating at y = %i: \n", wave_origin);
            // if (debug18) printf("Initial angle = %G
degrees \n", theta_d);
            // if (debug18) printf("\n");
        // } else theta = thetao;


        // Determine origin adjustment (y-direction) based
on angle sign
        // This ensures that the wave rays never start on a
cell border
        float y_adjust;
        if (theta <= 180*pi/180) {
            y_adjust = .0001;
        } else if (theta > 180*pi/180) {
            y_adjust = -.0001;
        }

        // Establish parameters for each loop iteration
        float C_o; // Initial "deep water" wave celerity
        C_o = g * Period / (2.0 * pi);
        //theta = thetao;
        theta_int = theta;
        x_now = x_int - 1;
        //x_now = x_int + ((1-sweep) *
(CellWidth/NumSweeps));
        //y_now = y_int[wave_origin];
        y_now = y_int[wave_origin] + ((1-sweep) *
(CellWidth/NumSweeps)) + y_adjust;
        theta_now = theta_int;
        beta_now = beta_int;
        sigma_now = sigma_int;
        h_now = h_int;


        if (debug13) printf("x_now = %G \n", x_now);
        if (debug13) printf("y_now = %G \n", y_now);
        if (debug13) printf("theta_int = %.1f \n",
theta_int);
        if (debug13) printf("beta_now = %.1f \n",
beta_now);
```

```
        if (debug13) printf("sigma_now = %.1f \n",
sigma_now);
        if (debug13) printf("h_now = %.1f \n", h_now);

        // Store the _now parameters
        // x_store[0] = x_now;
        // y_store[0] = y_now;
        // theta_store[0] = theta_now;
        // beta_store[0] = beta_now;
        // sigma_store[0] = sigma_now;

        h_b = h_int;
        z = 0;
        break_depth = 70;
        depth_now = 999999;

        // (Re-)establish cell_wave_height[][] as all zeros
(done at beginning of each iteration)
        for (x=0; x < Xmax; x++){
            for (y=0; y < 2*Ymax; y++){
                cell_wave_height[x][y] = 0;
                cell_beta[x][y] = 0;
                RayInCell[x][y] = 0;
            }
        }

        // (Re-)establish breaking_angle[] and
breaking_height[] as all zeros for each iteration
        for (y=0; y < 2*Ymax; y++){
            breaking_height[y] = 0;
            breaking_angle[y] = 0;
        }

        // Make sure this occurs before waves break
        int WaveBreak = 0;
        while (WaveBreak == 0){
        //while (h_now < KBreak*depth_now){
        //while ((h_now < depth_now) && (x_now > 0)){
            if (break_depth < 5){
                ds = 10;
            }

            // Determine current cell coordinates
([x_crd][y_crd])
```

```
            for(i = Xmax; i > 0; i--){ // determines x_crd
                if ((x_now >= x_pos[i]) && (x_now <
(x_pos[i] + dx))){
                    x_crd = i;
                }
            }

            if (debug13) printf("x_crd = %i \n", x_crd);
            for(i = 2*Ymax; i > 0; i--){
                if ((y_now >= y_pos[i]) & (y_now <
(y_pos[i] + dy))){
                    y_crd = i;
                }
            }
            if (debug13) printf("y_crd = %i \n", y_crd);
            if (debug13) printf("\n");

            // Determine which gradients to use (dependent
on wave position in cell)
            float cx_ode;       float cy_ode;
            float c2x2_ode;     float c2y2_ode;
            float cxcy_ode;

            for (i = 0; i < Xmax; i++){
                if ( (x_now >= x_pos[i]) && (x_now <=
(x_pos[i] + (CellWidth/2))) ){
                    cx_ode = cx[x_crd][y_crd];
                    c2x2_ode = c2x2[x_crd][y_crd];
                } else if ( (x_now >= x_pos[i]) && (x_now
>= (x_pos[i] + (CellWidth/2))) ){
                    //cx_ode = cx[x_crd+1][y_crd];
                    //c2x2_ode = c2x2[x_crd+1][y_crd];
                    cx_ode = cx[x_crd][y_crd];
                    c2x2_ode = c2x2[x_crd][y_crd];
                }
            }

            for (i = 0; i < 2*Ymax; i++){
                if ((y_now >= y_pos[i]) && (y_now <=
(y_pos[i] + (CellWidth/2)))){
                    cy_ode = cy[x_crd][y_crd];
                    c2y2_ode = c2y2[x_crd][y_crd];
                    cxcy_ode = cxcy[x_crd][y_crd];
```

```
                } else if ((y_now >= y_pos[i]) && (y_now >=
(y_pos[i] + (CellWidth/2)))){
                    // cy_ode = cy[x_crd][y_crd-1];
                    // c2y2_ode = c2y2[x_crd][y_crd-1];
                    // cxcy_ode = cxcy[x_crd][y_crd-1];
                    cy_ode = cy[x_crd][y_crd];
                    c2y2_ode = c2y2[x_crd][y_crd];
                    cxcy_ode = cxcy[x_crd][y_crd];
                }
            }

            // Break out if c or C_g <= 0
            if (c[x_crd][y_crd] <= 0 || C_g[x_crd][y_crd]
<=0) {
                printf("c or C_g has gone to zero! \n");
                h_now = 0;
                break;
            }

            // Use Runge-Kutta method to solve ODE's
            int N;
            N = (si - sf)/delta_s;
            //if (debug13) printf("N = %i \n", N);
            float x_ode[2*N];
            float y_ode[2*N];
            float theta_ode[2*N];
            float beta_ode[2*N];
            float sigma_ode[2*N];
            float s_ode[2*N];

            float x_k1; float x_k2; float x_k3; float x_k4;
            float y_k1; float y_k2; float y_k3; float y_k4;
            float theta_k1; float theta_k2; float theta_k3;
float theta_k4;
            float beta_k1;  float beta_k2;  float beta_k3;
float beta_k4;
            float sigma_k1; float sigma_k2; float sigma_k3;
float sigma_k4;

            // Establish initial conditions
            x_ode[0] = x_now;
            y_ode[0] = y_now;
            theta_ode[0] = theta_now;
            beta_ode[0] = beta_now;
```

```
            sigma_ode[0] = sigma_now;
            s_ode[0] = sf;

            // Run Runga-Kutta
            for(i=0; i < N; i++){
                theta_k1 =
(((1/c[x_crd][y_crd]))*cx_ode*(sin(theta_ode[i])) -
cos(theta_ode[i])*cy_ode) * delta_s;
                theta_k2 =
((((1/c[x_crd][y_crd]))*cx_ode*(sin(theta_ode[i] +
0.5*theta_k1)) - cos(theta_ode[i] + 0.5*theta_k1)*cy_ode))
* delta_s;
                theta_k3 =
((((1/c[x_crd][y_crd]))*cx_ode*(sin(theta_ode[i] +
0.5*theta_k2)) - cos(theta_ode[i] + 0.5*theta_k2)*cy_ode))
* delta_s;
                theta_k4 =
((((1/c[x_crd][y_crd]))*cx_ode*(sin(theta_ode[i] +
theta_k3)) - cos(theta_ode[i] + theta_k3)*cy_ode)) *
delta_s;

                x_k1 = (cos(theta_ode[i])) * delta_s;
                x_k2 = (cos(theta_ode[i] + 0.5*theta_k1)) *
delta_s;
                x_k3 = (cos(theta_ode[i] + 0.5*theta_k2)) *
delta_s;
                x_k4 = (cos(theta_ode[i] + theta_k3)) *
delta_s;

                y_k1 = (sin(theta_ode[i])) * delta_s;
                y_k2 = (sin(theta_ode[i] + 0.5*theta_k1)) *
delta_s;
                y_k3 = (sin(theta_ode[i] + 0.5*theta_k2)) *
delta_s;
                y_k4 = (sin(theta_ode[i] + theta_k3)) *
delta_s;

                sigma_k1 = ( -(-
(cos(theta_ode[i]))*(cx_ode/c[x_crd][y_crd]) -
(sin(theta_ode[i]))*(cy_ode/c[x_crd][y_crd])) *
sigma_ode[i]
                    - (
sin(theta_ode[i])*sin(theta_ode[i])*(c2x2_ode/c[x_crd][y_cr
d]) -
```

```
2*sin(theta_ode[i])*cos(theta_ode[i])*(cxcy_ode/c[x_crd][y_
crd])
                    +
cos(theta_ode[i])*cos(theta_ode[i])*(c2y2_ode/c[x_crd][y_cr
d])) * beta_ode[i] ) * delta_s;

                beta_k1 = sigma_ode[i] * delta_s;

                sigma_k2 = ( -(-
(cos(theta_ode[i]+0.5*theta_k1))*(cx_ode/c[x_crd][y_crd]) -
(sin(theta_ode[i]+0.5*theta_k1))*(cy_ode/c[x_crd][y_crd]))
* (sigma_ode[i]+0.5*sigma_k1)
                    - (
sin(theta_ode[i]+0.5*theta_k1)*sin(theta_ode[i]+0.5*theta_k
1)*(c2x2_ode/c[x_crd][y_crd]) -
2*sin(theta_ode[i]+0.5*theta_k1)*cos(theta_ode[i]+0.5*theta
_k1)
                    *(cxcy_ode/c[x_crd][y_crd]) +
cos(theta_ode[i]+0.5*theta_k1)*cos(theta_ode[i]+0.5*theta_k
1)*(c2y2_ode/c[x_crd][y_crd])) * (beta_ode[i]+0.5*beta_k1)
) * delta_s;

                beta_k2 = (sigma_ode[i] + 0.5*sigma_k1) *
delta_s;

                sigma_k3 = ( -(-
(cos(theta_ode[i]+0.5*theta_k2))*(cx_ode/c[x_crd][y_crd]) -
(sin(theta_ode[i]+0.5*theta_k2))*(cy_ode/c[x_crd][y_crd]))
* (sigma_ode[i]+0.5*sigma_k2)
                    - (
sin(theta_ode[i]+0.5*theta_k2)*sin(theta_ode[i]+0.5*theta_k
2)*(c2x2_ode/c[x_crd][y_crd]) -
2*sin(theta_ode[i]+0.5*theta_k2)*cos(theta_ode[i]+0.5*theta
_k2)
                    *(cxcy_ode/c[x_crd][y_crd]) +
cos(theta_ode[i]+0.5*theta_k2)*cos(theta_ode[i]+0.5*theta_k
2)*(c2y2_ode/c[x_crd][y_crd])) * (beta_ode[i]+0.5*beta_k2)
) * delta_s;

                beta_k3 = (sigma_ode[i] + 0.5*sigma_k2) *
delta_s;

                sigma_k4 = ( -(-
(cos(theta_ode[i]+theta_k3))*(cx_ode/c[x_crd][y_crd]) -
```

```
(sin(theta_ode[i]+theta_k3))*(cy_ode/c[x_crd][y_crd])) *
(sigma_ode[i]+sigma_k3)
                    - (
sin(theta_ode[i]+theta_k3)*sin(theta_ode[i]+theta_k3)*(c2x2
_ode/c[x_crd][y_crd]) -
2*sin(theta_ode[i]+theta_k3)*cos(theta_ode[i]+theta_k3)
                    *(cxcy_ode/c[x_crd][y_crd]) +
cos(theta_ode[i]+theta_k3)*cos(theta_ode[i]+theta_k3)*(c2y2
_ode/c[x_crd][y_crd])) * (beta_ode[i]+beta_k3) ) * delta_s;

                beta_k4 = (sigma_ode[i] + sigma_k3) *
delta_s;

                x_ode[i+1] = x_ode[i] +
((0.166666666666667)*(x_k1+2*x_k2+2*x_k3+x_k4));
                y_ode[i+1] = y_ode[i] +
((0.166666666666667)*(y_k1+2*y_k2+2*y_k3+y_k4));
                theta_ode[i+1] = theta_ode[i] +
((0.166666666666667)*(theta_k1+2*theta_k2+2*theta_k3+theta_
k4));
                beta_ode[i+1] = beta_ode[i] +
((0.166666666666667)*(beta_k1+2*beta_k2+2*beta_k3+beta_k4))
;
                sigma_ode[i+1] = sigma_ode[i] +
((0.166666666666667)*(sigma_k1+2*sigma_k2+2*sigma_k3+sigma_
k4));
                s_ode[i+1] = s_ode[i] + delta_s;

                // Periodic Boundaries
                // Make sure that the wave ray does not run
off the domain (have it loop around, like in a VG)
                if (y_ode[N-1] < CellWidth){
                    y_ode[N-1] = 2*Ymax*CellWidth -
CellWidth + y_ode[N-1];
                } else if (y_ode[N-1] > (2*Ymax*CellWidth -
CellWidth)){
                    y_ode[N-1] = y_ode[N-1] -
(2*Ymax*CellWidth - CellWidth);
                }

            } // END of Runga-Kutta

            // Display the results of the ODE solver
```

```
            if (debug0) printf("--- ODE solver has run ---
\n");
            if (debug12) printf("x_ode = \n");
            for (i=0; i < N; i++){
                if (debug12) printf("%G ", x_ode[i]);
            } if (debug12) printf("\n");
            if (debug12) printf("\n");
            if (debug12) printf("y_ode = \n");
            for (i=0; i < N; i++){
                if (debug12) printf("%G ", y_ode[i]);
            } if (debug12) printf("\n");
            if (debug12) printf("\n");
            if (debug12) printf("theta_ode = \n");
            for (i=0; i < N; i++){
                if (debug12) printf("%G ", theta_ode[i]);
            } if (debug12) printf("\n");
            if (debug12) printf("\n");
            if (debug12) printf("beta_ode = \n");
            for (i=0; i < N; i++){
                if (debug12) printf("%G ", beta_ode[i]);
            } if (debug12) printf("\n");
            if (debug12) printf("\n");
            if (debug12) printf("sigma_ode = \n");
            for (i=0; i < N; i++){
                if (debug12) printf("%G ", sigma_ode[i]);
            } if (debug12) printf("\n");
            if (debug12) printf("\n");
            if (debug12) printf("c[x_crd][y_crd] = %G \n",
c[x_crd][y_crd]);
            if (debug12) printf("cx[x_crd][y_crd] = %G \n",
cx[x_crd][y_crd]);
            if (debug12) printf("cy[x_crd][y_crd] = %G \n",
cy[x_crd][y_crd]);
            if (debug12) printf("c2x2[x_crd][y_crd] = %G
\n", c2x2[x_crd][y_crd]);
            if (debug12) printf("c2y2[x_crd][y_crd] = %G
\n", c2y2[x_crd][y_crd]);
            if (debug12) printf("cxcy[x_crd][y_crd] = %G
\n", cxcy[x_crd][y_crd]);
            if (debug12) printf("C_g = %G \n",
C_g[x_crd][y_crd]);
            if (debug12) printf("x_crd = %i \n", x_crd);
            if (debug12) printf("y_crd = %i \n", y_crd);
            if (debug12) printf("\n");
```

```
            // Establish new _now parameters based on
output of of ODE solver
            x_now = x_ode[N-1];
            y_now = y_ode[N-1];
            theta_now = theta_ode[N-1];
            beta_now = beta_ode[N-1];
            sigma_now = sigma_ode[N-1];
            depth_now = depth[x_crd][y_crd];
            if (debug13) printf("x_now = %G \n", x_now);
            if (debug13) printf("y_now = %G \n", y_now);
            if (debug13) printf("theta_now = %G \n",
theta_now);
            if (debug13) printf("beta_now = %G \n",
beta_now);
            if (debug13) printf("sigma_now = %G \n",
sigma_now);
            if (debug13) printf("depth_now = %G \n",
depth_now);

            // Store the current _now parameters
            // x_store[z] = x_now;
            // y_store[z] = y_now;
            // theta_store[z] = theta_now;
            // beta_store[z] = beta_now;
            // sigma_store[z] = sigma_now;
            // h_store[z] = h_now;

            // Take si, sf, and z to the next iteration
            si = si + ds;
            sf = sf + ds;
            z = z + 1;

            // Re-establish current cell coordinates (x_crd
& y_crd)
            for(i = Xmax; i > 0; i--){ // determines x_crd
                if ((x_now >= x_pos[i]) && (x_now <=
(x_pos[i] + dx))){
                    x_crd = i;
                }
            }

            if (debug13) printf("x_crd = %i \n", x_crd);
```

```
            for(i = Ymax*2; i > 0; i--){
                if ((y_now >= y_pos[i]) & (y_now <=
(y_pos[i] + dy))){
                    y_crd = i;
                }
            } if (debug13) printf("y_crd = %i \n", y_crd);

            depth_now = depth[x_crd][y_crd];

            // Break out of 'for' loop if wave ray enters
cell adjacent to the beach
            // if (depth_now <= 2*BathySlope) {
                // if (debug17) printf("Wave ray has run
into the beach! \n");
                // if (debug17) printf("\n");
                // break;
            // }

            // Determine shore position (for current y-
coordinate)
            float beach_location;
            for (x=0; x < Xmax; x++){
                if ((PercentFull[x][y_crd] > 0) &&
(PercentFull[x][y_crd] < 1)){
                    beach_location = (x +
PercentFull[x][y_crd])*CellWidth;
                    //printf("beach_location = %G \n",
beach_location);
                }
            }

            // Determine Distance to Beach
            float x_beach_dist; // Distance to the beach in
the y-direction
            float y_beach_dist; // Distance to the beach in
the y-direction
            float DistToBeach; // Closest distance to beach
(either x or y)

            x_beach_dist = fabsf(x_now - beach_location);

            // If adjacent cell is a beach or land cell in
the y-direction,
            // determine the distance to it
```

```
            if (PercentFull[x_crd][y_crd+1] > 0) {
                y_beach_dist = fabsf(y_crd*CellWidth -
y_now);
            } else if (PercentFull[x_crd][y_crd-1] > 0){
                y_beach_dist = fabsf(y_now -
y_crd*CellWidth);
            } else if ((PercentFull[x_crd][y_crd+1] > 0) &&
(PercentFull[x_crd][y_crd-1] > 0)){
                if (PercentFull[x_crd][y_crd+1] >=
PercentFull[x_crd][y_crd-1]){
                    y_beach_dist = fabsf(y_crd*CellWidth -
y_now);
                } else {
                    y_beach_dist = fabsf(y_now -
y_crd*CellWidth);
                }
            } else {
                y_beach_dist = CellWidth*Xmax*10; // Make
it so big it'll never be > x_beach_dist
            }

            // Use whichever distance (x or y) is smaller
at actual distance to beach
            if (x_beach_dist > y_beach_dist){
                DistToBeach = y_beach_dist;
                if ((DistToBeach < CellWidth+(CellWidth/2))
&& (debug17)) printf("Using y-direction distance to
shore\n");
            } else {
                DistToBeach = x_beach_dist;
                if ((DistToBeach < CellWidth+(CellWidth/2))
&& (debug17)) printf("Using x-direction distance to
shore\n");
            }


            // Determine "true" depth (as opposed to depth
in cell) --> Determines wave breaking
            float true_depth;
            if (DistToBeach < CellWidth+(CellWidth/2)) {
                true_depth =
.1*(pow((DistToBeach),.666666667));
                if (debug17) printf("true_depth = %G \n",
true_depth);
```

```
                // Break wave if wave height is greater
than true depth
                if (h_now >= (KBreak * true_depth)){
                    if (debug17) {
                        printf("BREAK WAVE!!!! True depth =
%G, wave height = %G \n", true_depth, h_now);
                        printf("x location = %G , y
location = %G , \ndistance to shore = %G \n", x_now, y_now,
DistToBeach);
                    }
                    WaveBreak = 1;
                    break;
                } else if (DistToBeach <=
MinBreakDistance){
                    if (debug17) {
                        printf("BREAK WAVE!!!! True depth =
%G, wave height = %G \n", true_depth, h_now);
                        printf("x location = %G , distance
to shore = %G \n", x_now, DistToBeach);
                    }
                    WaveBreak = 1;
                    break;
                }
            }

            // Define beta_calc (used to calculate h_now)
            if (beta_now <= beta_limit){
                beta_calc = beta_limit;
                //beta_calc = -beta_now;
                if (debug21a) printf("Beta has gone below
%.1f! \n ", beta_limit);
            } else beta_calc = beta_now;

            // Calculate current wave height (h_now)
            //h_now = StartHeight *
sqrt((C_o)/(2*C_g[x_crd][y_crd])) * sqrt(1/beta_calc);

            float kh;
            float n; // C*n = C_g
            kh = pi * depth[x_crd][y_crd] /
L[x_crd][y_crd];
```

```
            if (debug13) printf("depth = %G  ",
depth[x_crd][y_crd]);
            if (debug13) printf("L = %G  ",
L[x_crd][y_crd]);
            if (debug13) printf("kh = %G  ", kh);

            n = 0.5 * (1 + 2 * kh / sinh(2 * kh));

            if (debug13) printf("n = %G  ", n);

            h_now = StartHeight *
sqrt((C_o)/(2*n*c[x_crd][y_crd])) * sqrt(1/beta_calc);

            if (debug13) printf("h_now = %G \n", h_now);

            cell_wave_height[x_crd][y_crd] = h_now;
            cell_beta[x_crd][y_crd] = beta_now;


            // Has a ray passed through this cell?
            RayInCell[x_crd][y_crd] = 1;

            h_b = h_now;
            theta_deg = theta_now*180/pi;

            if (debug13) printf("theta_deg = %G \n",
theta_deg);
            if (debug13) printf("c[x_crd][y_crd] = %G \n",
c[x_crd][y_crd]);
            if (debug13) printf("C_g[x_crd][y_crd] = %G
\n", C_g[x_crd][y_crd]);
            if (debug13) printf("cx[x_crd][y_crd] = %G \n",
cx[x_crd][y_crd]);
            if (debug13) printf("cy[x_crd][y_crd] = %G \n",
cy[x_crd][y_crd]);
            if (debug13) printf("c2x2[x_crd][y_crd] = %G
\n", c2x2[x_crd][y_crd]);
            if (debug13) printf("c2y2[x_crd][y_crd] = %G
\n", c2y2[x_crd][y_crd]);
            if (debug13) printf("cxcy[x_crd][y_crd] = %G
\n", cxcy[x_crd][y_crd]);
            if (debug13) printf("h_now = %G \n", h_now);
            if (debug13) printf("depth_now = %G \n",
depth_now);
```

```
            if (debug13) printf("Current Time Step: %i \n",
CurrentTimeStep);
            if (debug13) printf("\n");
            if (debug13) printf("z = %i \n", z);

        } // END OF WHILE LOOP (includes ODE Solver)

        // Display the stored results
        // if (debug14) printf("x_store = \n");
        // for (i=0; i < z; i++){
            // if (debug14) printf("%G ", x_store[i]);
        // } if (debug14) printf("\n");
        // if (debug14) printf("\n");

        // if (debug14) printf("y_store = \n");
        // for (i=0; i < z; i++){
            // if (debug14) printf("%G ", y_store[i]);
        // } if (debug14) printf("\n");
        // if (debug14) printf("\n");

        // if (debug14) printf("theta_store = \n");
        // for (i=0; i < z; i++){
            // if (debug14) printf("%G ", theta_store[i]);
        // } if (debug14) printf("\n");
        // if (debug14) printf("\n");

        // if (debug14) printf("beta_store = \n");
        // for (i=0; i < z; i++){
            // if (debug14) printf("%G ", beta_store[i]);
        // } if (debug14) printf("\n");
        // if (debug14) printf("\n");

        // if (debug14) printf("sigma_store = \n");
        // for (i=0; i < z; i++){
            // if (debug14) printf("%G ", sigma_store[i]);
        // } if (debug14) printf("\n");
        // if (debug14) printf("\n");

        // if (debug14) printf("h_store = \n");
        // for (i=0; i < z; i++){
            // if (debug14) printf("%G ", h_store[i]);
        // } if (debug14) printf("\n");
        // if (debug14) printf("\n");
```

```
        // Display braking wave data for this wave ray
        if (debug15) printf("for wave_origin = %i : \n",
wave_origin);
        if (debug15) printf("Breaking wave angle = %G \n",
theta_deg);
        if (debug15) printf("Breaking wave height = %G \n",
h_now);
        if (debug15) printf("Breaking coordinates: [%i][%i]
\n", x_crd, y_crd);
        if (debug15) printf("\n");

        // Store breaking wave data for this wave ray
        breaking_angle[y_crd] = theta_now;
        breaking_height[y_crd] = h_now;

        // Calculate sum of breaking wave heights and
angles in each shoreline cell
        for (y=0; y < 2*Ymax; y++){
            sum_breaking_height[y] = sum_breaking_height[y]
+ breaking_height[y];
            //sum_breaking_angle[y] = sum_breaking_angle[y]
+ breaking_angle[y];
        }

        // Calculate average breaking angle for that
particular beach cell
        N_rays_hit[y_crd] = N_rays_hit[y_crd] + 1;

        avg_breaking_angle[y_crd] =
((avg_breaking_angle[y_crd]*(N_rays_hit[y_crd] - 1))
                + breaking_angle[y_crd])/N_rays_hit[y_crd];


        // printf("Average breaking angle at y = %i is %G
\n", y_crd, avg_breaking_angle[y_crd]);
        // printf("Number of wave rays hit at y = %i is %i
\n", y_crd, N_rays_hit[y_crd]);
        // printf("\n");

        // Calculate the number of wave rays that passed
through each cell
        for (x=0; x < Xmax; x++){
            for (y=0; y < 2*Ymax; y++){
```

```
                NumRaysPerCell[x][y] = NumRaysPerCell[x][y]
+ RayInCell[x][y];
            }
        }

        // Calculate sum of wave heights and betas in each
cells
        for (x=0; x < Xmax; x++){
            for (y=0; y < 2*Ymax; y++){
                sum_cell_wave_height[x][y] =
sum_cell_wave_height[x][y] + cell_wave_height[x][y];
                sum_cell_beta[x][y] = sum_cell_beta[x][y] +
cell_beta[x][y];
            }
        }

        N_rays++;

    } ////////// Shoot next wave ray //////////


    // Calculate average cell wave height and cell beta for
all cells
    for (x=0; x < Xmax; x++){
        for (y=0; y < 2*Ymax+1; y++){
            //avg_cell_wave_height[x][y] =
sum_cell_wave_height[x][y]/N_rays;
            //avg_cell_wave_height[x][y] =
sum_cell_wave_height[x][y]/NumSweeps;
            if (NumRaysPerCell[x][y] == 0){
                avg_cell_wave_height[x][y] = 0;
                avg_cell_beta[x][y] = 0;
            } else {
                avg_cell_wave_height[x][y] =
sum_cell_wave_height[x][y]/(NumRaysPerCell[x][y]);
                avg_cell_beta[x][y] =
sum_cell_beta[x][y]/(NumRaysPerCell[x][y]);
            }
        }
    }

    // Calculate average breaking wave height [and angle]
for each shoreline cell
    for (y = 0; y < 2*Ymax; y++){
```

```
        //avg_breaking_angle[y] =
sum_breaking_angle[y]/N_rays;
        //avg_breaking_height[y] =
sum_breaking_height[y]/sweep;
        //avg_breaking_height[y] =
sum_breaking_height[y]/N_rays;

        if (N_rays_hit[y] > 0){
            avg_breaking_height[y] =
sum_breaking_height[y]/N_rays_hit[y];
        } else avg_breaking_height[y] = 0;

        //printf("avg_breaking_height = %G, y = %i \n",
avg_breaking_height[y], y);
        //printf("sum_breaking_height = %G, N_rays_hit =
%i, y = %i \n", sum_breaking_height[y], N_rays_hit[y], y);

        if (avg_breaking_height[y] > height_limit) {
            avg_breaking_height[y] = height_limit;
            printf("Wave height has gone over %G ! \n",
height_limit);
        }

    }

    // Determine if there is wave height data for all beach
cells
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        if (avg_breaking_height[y] > 0){
            DataForAllBeachCells = 'y';
        } else if (avg_breaking_height[y] == 0){
            DataForAllBeachCells = 'n';
            //printf("No wave height data for at least one
beach cell! \n");
            //printf("Running loop again. \n");
            //printf("\n");
            break;
        }
    }

    } // Ends sweep 'while' or 'for' loop (depending on
which method is used)

    // Average right and left neighbors if no data for cell
```

```
    // if ((avg_breaking_height[0] == 0) &&
(avg_breaking_height[1] == 0)
            // && (avg_breaking_height[Ymax] != 0) &&
(avg_breaking_height[2] != 0)){
        // avg_breaking_height[0] = (avg_breaking_height[2]
+ avg_breaking_height[Ymax])/2;
        // avg_breaking_angle[0] = (avg_breaking_angle[2] +
avg_breaking_angle[Ymax])/2;
        // printf("Modified breaking data at 0 \n");
    // } else if ((avg_breaking_height[0] == 0) &&
(avg_breaking_height[1] == 0)
            // && (avg_breaking_height[Ymax-1] != 0) &&
(avg_breaking_height[2] != 0)){
        // avg_breaking_height[0] = (avg_breaking_height[2]
+ avg_breaking_height[Ymax-1])/2;
        // avg_breaking_angle[0] = (avg_breaking_angle[2] +
avg_breaking_angle[Ymax-1])/2;
        // printf("Modified breaking data at 0 \n");
    // } else
    if ((avg_breaking_height[0] == 0) &&
(avg_breaking_height[Ymax] != 0)
            && (avg_breaking_height[1] != 0)){
        avg_breaking_height[0] = (avg_breaking_height[1] +
avg_breaking_height[Ymax])/2;
        avg_breaking_angle[0] = (avg_breaking_angle[1] +
avg_breaking_angle[Ymax])/2;
        printf("Modified breaking data at 0 \n");
    }
    for (y = 1; y < (2*Ymax) - 1; y++){
        // if ((avg_breaking_height[y] == 0) &&
(avg_breaking_height[y+1] == 0)
                // && (avg_breaking_height[y-1] != 0) &&
(avg_breaking_height[y+2] != 0)) {
            // avg_breaking_height[y] =
(avg_breaking_height[y+2] + avg_breaking_height[y-1])/2;
            // avg_breaking_height[y+1] =
(avg_breaking_height[y+2] + avg_breaking_height[y-1])/2;
            // avg_breaking_angle[y] =
(avg_breaking_angle[y+2] + avg_breaking_angle[y-1])/2;
            // avg_breaking_angle[y+1] =
(avg_breaking_angle[y+2] + avg_breaking_angle[y-1])/2;
            // printf("Modified breaking data at %i \n",y);
        // } else
```

```
        if ((avg_breaking_height[y] == 0) &&
(avg_breaking_height[y-1] != 0)
                && (avg_breaking_height[y+1] != 0)) {
            avg_breaking_height[y] =
(avg_breaking_height[y+1] + avg_breaking_height[y-1])/2;
            avg_breaking_angle[y] =
(avg_breaking_angle[y+1] + avg_breaking_angle[y-1])/2;
            printf("Modified breaking data at %i \n",y);
        }
    }
    // if ((avg_breaking_height[Ymax] == 0) &&
(avg_breaking_height[0] == 0)
            // && (avg_breaking_height[1] != 0) &&
(avg_breaking_height[Ymax-1] != 0)){
        // avg_breaking_height[Ymax] =
(avg_breaking_height[1] + avg_breaking_height[Ymax-1])/2;
        // avg_breaking_angle[Ymax] =
(avg_breaking_angle[1] + avg_breaking_angle[Ymax-1])/2;
        // printf("Modified breaking data at %i \n",Ymax);
    // } else if ((avg_breaking_height[Ymax] == 0) &&
(avg_breaking_height[0] == 0)
            // && (avg_breaking_height[1] == 0) &&
(avg_breaking_height[Ymax-1] != 0)){
        // avg_breaking_height[Ymax] =
(avg_breaking_height[2] + avg_breaking_height[Ymax-1])/2;
        // avg_breaking_angle[Ymax] =
(avg_breaking_angle[2] + avg_breaking_angle[Ymax-1])/2;
        // printf("Modified breaking data at %i \n",Ymax);
    // } else
    if ((avg_breaking_height[Ymax] == 0)
            && (avg_breaking_height[1] != 0) &&
(avg_breaking_height[Ymax-1] != 0)){
        avg_breaking_height[Ymax] = (avg_breaking_height[1]
+ avg_breaking_height[Ymax-1])/2;
        avg_breaking_angle[Ymax] = (avg_breaking_angle[1] +
avg_breaking_angle[Ymax-1])/2;
        printf("Modified breaking data at %i \n",Ymax);
    }

    // Fill breaking angle data matrices
    for (y = 0; y < 2*Ymax; y++){
    //for (y = Ymax/2; y < 3*Ymax/2; y++){
        breaking_angles[y] = avg_breaking_angle[y];
        breaking_heights[y] = avg_breaking_height[y];
```

```
            BkAngle[y] = avg_breaking_angle[y] - pi;
        }

        // Average right and left neighbors if no data for cell
        for (y = 1; y < (2*Ymax) - 1; y++){
            if ((avg_breaking_height[y] == 0) &&
(avg_breaking_height[y+1] == 0)) {
                avg_breaking_height[y] =
(avg_breaking_height[y+2] + avg_breaking_height[y-1])/2;
                avg_breaking_height[y+1] =
(avg_breaking_height[y+2] + avg_breaking_height[y-1])/2;
                avg_breaking_angle[y] =
(avg_breaking_angle[y+2] + avg_breaking_angle[y-1])/2;
                avg_breaking_angle[y+1] =
(avg_breaking_angle[y+2] + avg_breaking_angle[y-1])/2;
            } else if (HeightRollAvg[y] == 0) {
                avg_breaking_height[y] =
(avg_breaking_height[y+1] + avg_breaking_height[y-1])/2;
                avg_breaking_angle[y] =
(avg_breaking_angle[y+1] + avg_breaking_angle[y-1])/2;
            }
        }

        // Calculate rolling averages for breaking height and
angle
        if (RollAvgNeighbors == 0){
            for (y = 0; y < 2*Ymax; y++){
                HeightRollAvg[y] = avg_breaking_height[y];
                AngleRollAvg[y] = avg_breaking_angle[y];
            }
        }

        else if (RollAvgNeighbors == 1){
        HeightRollAvg[0] = (avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0]
            + avg_breaking_height[1]) / 3;

        AngleRollAvg[0] = (avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0]
            + avg_breaking_angle[1]) / 3;

        for (y = 1; y < 2*Ymax - 1; y++){
            HeightRollAvg[y] = (avg_breaking_height[y-1] +
avg_breaking_height[y]
```

```
            + avg_breaking_height[y+1]) / 3;
        AngleRollAvg[y] = (avg_breaking_angle[y-1] +
avg_breaking_angle[y]
            + avg_breaking_angle[y+1]) / 3;
    }

    HeightRollAvg[2*Ymax-1] = (avg_breaking_height[2*Ymax-
2] + avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0]) / 3;

    AngleRollAvg[2*Ymax-1] = (avg_breaking_angle[2*Ymax-2]
+ avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0]) / 3;
    }


    else if (RollAvgNeighbors == 2){
    HeightRollAvg[0] = (avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]) / 5;
    HeightRollAvg[1] = (avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]) / 5;

    AngleRollAvg[0] = (avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]) / 5;
    AngleRollAvg[1] = (avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]) / 5;

    for (y = 2; y < 2*Ymax - 2; y++){
        HeightRollAvg[y] = (avg_breaking_height[y-2] +
avg_breaking_height[y-1] + avg_breaking_height[y]
            + avg_breaking_height[y+1] +
avg_breaking_height[y+2]) / 5;
        AngleRollAvg[y] = (avg_breaking_angle[y-2] +
avg_breaking_angle[y-1] + avg_breaking_angle[y]
            + avg_breaking_angle[y+1] +
avg_breaking_angle[y+2]) / 5;
```

```
    }

    HeightRollAvg[2*Ymax-1] = (avg_breaking_height[2*Ymax-
3] + avg_breaking_height[2*Ymax-2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]) / 5;
    HeightRollAvg[2*Ymax-2] = (avg_breaking_height[2*Ymax-
4] + avg_breaking_height[2*Ymax-3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]) /
5;

    AngleRollAvg[2*Ymax - 1] = (avg_breaking_angle[2*Ymax-
3] + avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]) / 5;
    AngleRollAvg[2*Ymax - 2] = (avg_breaking_angle[2*Ymax-
4] + avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]) / 5;
    }

    else if (RollAvgNeighbors == 3) {
    HeightRollAvg[0] = (avg_breaking_height[2*Ymax-3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3]) / 7;
    HeightRollAvg[1] = (avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4])
/ 7;
    HeightRollAvg[2] = (avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]
        + avg_breaking_height[5]) / 7;
    HeightRollAvg[3] = (avg_breaking_height[0] +
avg_breaking_height[1] + avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
```

```
        + avg_breaking_height[6]) / 7;

    AngleRollAvg[0] = (avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3]) / 7;
    AngleRollAvg[1] = (avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4]) / 7;
    AngleRollAvg[2] = (avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]
        + avg_breaking_angle[5]) / 7;
    AngleRollAvg[3] = (avg_breaking_angle[0] +
avg_breaking_angle[1] + avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6]) / 7;

    for (y = 3; y < 2*Ymax - 3; y++){
        HeightRollAvg[y] = (avg_breaking_height[y-3]
            + avg_breaking_height[y-2] +
avg_breaking_height[y-1] + avg_breaking_height[y]
            + avg_breaking_height[y+1] +
avg_breaking_height[y+2] + avg_breaking_height[y+3]) / 7;
        AngleRollAvg[y] = (avg_breaking_angle[y-3]
            + avg_breaking_angle[y-2] +
avg_breaking_angle[y-1] + avg_breaking_angle[y]
            + avg_breaking_angle[y+1] +
avg_breaking_angle[y+2] + avg_breaking_angle[y+3]) / 7;
    }

    HeightRollAvg[2*Ymax-3] = (avg_breaking_height[2*Ymax-
6] + avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0]) / 7;
```

```
    HeightRollAvg[2*Ymax-2] = (avg_breaking_height[2*Ymax-
5] + avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]
        + avg_breaking_height[1]) / 7;
    HeightRollAvg[2*Ymax-1] = (avg_breaking_height[2*Ymax-
4] + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2]) / 7;

    AngleRollAvg[2*Ymax-3] = (avg_breaking_angle[2*Ymax-6]
+ avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-
4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0]) / 7;
    AngleRollAvg[2*Ymax-2] = (avg_breaking_angle[2*Ymax-5]
+ avg_breaking_angle[2*Ymax-4] + avg_breaking_angle[2*Ymax-
3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1]) / 7;
    AngleRollAvg[2*Ymax-1] = (avg_breaking_angle[2*Ymax-4]
+ avg_breaking_angle[2*Ymax-3] + avg_breaking_angle[2*Ymax-
2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2]) / 7;

    }

    else if (RollAvgNeighbors == 4) {
    HeightRollAvg[0] = (avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4])
/ 9;
    HeightRollAvg[1] = (avg_breaking_height[2*Ymax-3]
```

```
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5])
/ 9;
    HeightRollAvg[2] = (avg_breaking_height[2*Ymax-2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]
        + avg_breaking_height[5] + avg_breaking_height[6])
/ 9;
    HeightRollAvg[3] = (avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7])
/ 9;
    HeightRollAvg[4] = (avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5] +
avg_breaking_height[6]
        + avg_breaking_height[7] + avg_breaking_height[8])
/ 9;

    AngleRollAvg[0] = (avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4]) /
9;
    AngleRollAvg[1] = (avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4] + avg_breaking_angle[5]) /
9;
    AngleRollAvg[2] = (avg_breaking_angle[2*Ymax-2]
```

```
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]
        + avg_breaking_angle[5] + avg_breaking_angle[6]) /
9;
    AngleRollAvg[3] = (avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7]) /
9;
    AngleRollAvg[4] = (avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4] + avg_breaking_angle[5] +
avg_breaking_angle[6]
        + avg_breaking_angle[7] + avg_breaking_angle[8]) /
9;

    for (y = 4; y < 2*Ymax - 4; y++){
        HeightRollAvg[y] = (avg_breaking_height[y-4] +
avg_breaking_height[y-3]
            + avg_breaking_height[y-2] +
avg_breaking_height[y-1] + avg_breaking_height[y]
            + avg_breaking_height[y+1] +
avg_breaking_height[y+2] + avg_breaking_height[y+3]
            + avg_breaking_height[y+4]) / 9;
        AngleRollAvg[y] = (avg_breaking_angle[y-4] +
avg_breaking_angle[y-3]
            + avg_breaking_angle[y-2] +
avg_breaking_angle[y-1] + avg_breaking_angle[y]
            + avg_breaking_angle[y+1] +
avg_breaking_angle[y+2] + avg_breaking_angle[y+3]
            + avg_breaking_angle[y+4]) / 9;
    }

    HeightRollAvg[2*Ymax-4] = (avg_breaking_height[2*Ymax-
8]
        + avg_breaking_height[2*Ymax-7] +
avg_breaking_height[2*Ymax-6] + avg_breaking_height[2*Ymax-
5]
```

```
        + avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3] + avg_breaking_height[2*Ymax-
2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0]) / 9;
    HeightRollAvg[2*Ymax-3] = (avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1])
/ 9;
    HeightRollAvg[2*Ymax-2] = (avg_breaking_height[2*Ymax-
6]
        + avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4] + avg_breaking_height[2*Ymax-
3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2])
/ 9;
    HeightRollAvg[2*Ymax-1] = (avg_breaking_height[2*Ymax-
5]
        + avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3] + avg_breaking_height[2*Ymax-
2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3])
/ 9;

    AngleRollAvg[2*Ymax-4] = (avg_breaking_angle[2*Ymax-8]
        + avg_breaking_angle[2*Ymax-7] +
avg_breaking_angle[2*Ymax-6] + avg_breaking_angle[2*Ymax-5]
        + avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3] + avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0]) / 9;
    AngleRollAvg[2*Ymax-3] = (avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
```

```
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1]) /
9;
    AngleRollAvg[2*Ymax-2] = (avg_breaking_angle[2*Ymax-6]
        + avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4] + avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2]) /
9;
    AngleRollAvg[2*Ymax-1] = (avg_breaking_angle[2*Ymax-5]
        + avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3] + avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3]) /
9;

    }

    else if (RollAvgNeighbors == 5) {
    HeightRollAvg[0] = (avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]) / 11;
    HeightRollAvg[1] = (avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5] +
avg_breaking_height[6]) / 11;
    HeightRollAvg[2] = (avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
```

```
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]
        + avg_breaking_height[5] + avg_breaking_height[6] +
avg_breaking_height[7]) / 11;
    HeightRollAvg[3] = (avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7] +
avg_breaking_height[8]) / 11;
    HeightRollAvg[4] = (avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5] +
avg_breaking_height[6]
        + avg_breaking_height[7] + avg_breaking_height[8] +
avg_breaking_height[9]) / 11;
    HeightRollAvg[5] = (avg_breaking_height[0] +
avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]
        + avg_breaking_height[5] + avg_breaking_height[6] +
avg_breaking_height[7]
        + avg_breaking_height[8] + avg_breaking_height[9] +
avg_breaking_height[10]) / 11;

    AngleRollAvg[0] = (avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]) / 11;
    AngleRollAvg[1] = (avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
```

```
        + avg_breaking_angle[4] + avg_breaking_angle[5] +
avg_breaking_angle[6]) / 11;
    AngleRollAvg[2] = (avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]
        + avg_breaking_angle[5] + avg_breaking_angle[6] +
avg_breaking_angle[7]) / 11;
    AngleRollAvg[3] = (avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7] +
avg_breaking_angle[8]) / 11;
    AngleRollAvg[4] = (avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4] + avg_breaking_angle[5] +
avg_breaking_angle[6]
        + avg_breaking_angle[7] + avg_breaking_angle[8] +
avg_breaking_angle[9]) / 11;
    AngleRollAvg[5] = (avg_breaking_angle[0] +
avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]
        + avg_breaking_angle[5] + avg_breaking_angle[6] +
avg_breaking_angle[7]
        + avg_breaking_angle[8] + avg_breaking_angle[9] +
avg_breaking_angle[10]) / 11;

    for (y = 5; y < 2*Ymax - 5; y++){
        HeightRollAvg[y] = (avg_breaking_height[y-5] +
avg_breaking_height[y-4] + avg_breaking_height[y-3]
            + avg_breaking_height[y-2] +
avg_breaking_height[y-1] + avg_breaking_height[y]
            + avg_breaking_height[y+1] +
avg_breaking_height[y+2] + avg_breaking_height[y+3]
            + avg_breaking_height[y+4] +
avg_breaking_height[y+5]) / 11;
```

```
        AngleRollAvg[y] = (avg_breaking_angle[y-5] +
avg_breaking_angle[y-4] + avg_breaking_angle[y-3]
            + avg_breaking_angle[y-2] +
avg_breaking_angle[y-1] + avg_breaking_angle[y]
            + avg_breaking_angle[y+1] +
avg_breaking_angle[y+2] + avg_breaking_angle[y+3]
            + avg_breaking_angle[y+4] +
avg_breaking_angle[y+5]) / 11;
    }

    HeightRollAvg[2*Ymax-5] = (avg_breaking_height[2*Ymax-
10] + avg_breaking_height[2*Ymax-9]
        + avg_breaking_height[2*Ymax-8] +
avg_breaking_height[2*Ymax-7] + avg_breaking_height[2*Ymax-
6]
        + avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4] + avg_breaking_height[2*Ymax-
3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]) /
11;
    HeightRollAvg[2*Ymax-4] = (avg_breaking_height[2*Ymax-
9] + avg_breaking_height[2*Ymax-8]
        + avg_breaking_height[2*Ymax-7] +
avg_breaking_height[2*Ymax-6] + avg_breaking_height[2*Ymax-
5]
        + avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3] + avg_breaking_height[2*Ymax-
2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]) / 11;
    HeightRollAvg[2*Ymax-3] = (avg_breaking_height[2*Ymax-
8] + avg_breaking_height[2*Ymax-7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]) / 11;
    HeightRollAvg[2*Ymax-2] = (avg_breaking_height[2*Ymax-
7] + avg_breaking_height[2*Ymax-6]
```

```
        + avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4] + avg_breaking_height[2*Ymax-
3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]) / 11;
    HeightRollAvg[2*Ymax-1] = (avg_breaking_height[2*Ymax-
6] + avg_breaking_height[2*Ymax-5]
        + avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3] + avg_breaking_height[2*Ymax-
2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]) / 11;

    AngleRollAvg[2*Ymax-5] = (avg_breaking_angle[2*Ymax-10]
+ avg_breaking_angle[2*Ymax-9]
        + avg_breaking_angle[2*Ymax-8] +
avg_breaking_angle[2*Ymax-7] + avg_breaking_angle[2*Ymax-6]
        + avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4] + avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]) / 11;
    AngleRollAvg[2*Ymax-4] = (avg_breaking_angle[2*Ymax-9]
+ avg_breaking_angle[2*Ymax-8]
        + avg_breaking_angle[2*Ymax-7] +
avg_breaking_angle[2*Ymax-6] + avg_breaking_angle[2*Ymax-5]
        + avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3] + avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]) / 11;
    AngleRollAvg[2*Ymax-3] = (avg_breaking_angle[2*Ymax-8]
+ avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]) / 11;
    AngleRollAvg[2*Ymax-2] = (avg_breaking_angle[2*Ymax-7]
+ avg_breaking_angle[2*Ymax-6]
```

```
        + avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4] + avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]) / 11;
    AngleRollAvg[2*Ymax-1] = (avg_breaking_angle[2*Ymax-6]
+ avg_breaking_angle[2*Ymax-5]
        + avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3] + avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]) / 11;

    }

    else if (RollAvgNeighbors == 10) {
    HeightRollAvg[0] = (avg_breaking_height[2*Ymax-10] +
avg_breaking_height[2*Ymax-9]
        + avg_breaking_height[2*Ymax-8] +
avg_breaking_height[2*Ymax-7] + avg_breaking_height[2*Ymax-
6]
        + avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4] + avg_breaking_height[2*Ymax-
3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5] +
avg_breaking_height[6]
        + avg_breaking_height[7] + avg_breaking_height[8] +
avg_breaking_height[9]
        + avg_breaking_height[10]) / 21;
    HeightRollAvg[1] = (avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8]
        + avg_breaking_height[2*Ymax-7] +
avg_breaking_height[2*Ymax-6] + avg_breaking_height[2*Ymax-
5]
        + avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3] + avg_breaking_height[2*Ymax-
2]
```

```
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]
        + avg_breaking_height[5] + avg_breaking_height[6] +
avg_breaking_height[7]
        + avg_breaking_height[8] + avg_breaking_height[9] +
avg_breaking_height[10]
        + avg_breaking_height[11]) / 21;
    HeightRollAvg[2] = (avg_breaking_height[2*Ymax-8]
        + avg_breaking_height[2*Ymax-7] +
avg_breaking_height[2*Ymax-6] + avg_breaking_height[2*Ymax-
5]
        + avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3] + avg_breaking_height[2*Ymax-
2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]
        + avg_breaking_height[5] + avg_breaking_height[6] +
avg_breaking_height[7]
        + avg_breaking_height[8] + avg_breaking_height[9] +
avg_breaking_height[10]
        + avg_breaking_height[11] +
avg_breaking_height[12]) / 21;
    HeightRollAvg[3] = (avg_breaking_height[2*Ymax-7] +
avg_breaking_height[2*Ymax-6]
        + avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7] +
avg_breaking_height[8]
        + avg_breaking_height[9] + avg_breaking_height[10]
        + avg_breaking_height[11] + avg_breaking_height[12]
+ avg_breaking_height[13]) / 21;
    HeightRollAvg[4] = (avg_breaking_height[2*Ymax-6]
```

```
        + avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5] +
avg_breaking_height[6]
        + avg_breaking_height[7] + avg_breaking_height[8] +
avg_breaking_height[9]
        + avg_breaking_height[10] + avg_breaking_height[11]
+ avg_breaking_height[12]
        + avg_breaking_height[13] +
avg_breaking_height[14]) / 21;
    HeightRollAvg[5] = (avg_breaking_height[2*Ymax-5] +
avg_breaking_height[2*Ymax-4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0] + avg_breaking_height[1]
        + avg_breaking_height[2] + avg_breaking_height[3] +
avg_breaking_height[4]
        + avg_breaking_height[5] + avg_breaking_height[6] +
avg_breaking_height[7]
        + avg_breaking_height[8] + avg_breaking_height[9] +
avg_breaking_height[10]
        + avg_breaking_height[11] + avg_breaking_height[12]
        + avg_breaking_height[13] + avg_breaking_height[14]
+ avg_breaking_height[15]) / 21;
    HeightRollAvg[6] = (avg_breaking_height[2*Ymax-4] +
avg_breaking_height[2*Ymax-3]
        + avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1] + avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5] +
avg_breaking_height[6]
        + avg_breaking_height[7] + avg_breaking_height[8] +
avg_breaking_height[9]
        + avg_breaking_height[10] + avg_breaking_height[11]
+ avg_breaking_height[12]
```

```
        + avg_breaking_height[13] + avg_breaking_height[14]
+ avg_breaking_height[15]
        + avg_breaking_height[16]) / 21;
    HeightRollAvg[7] = (avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2]
        + avg_breaking_height[2*Ymax-1] +
avg_breaking_height[0]
        + avg_breaking_height[1] + avg_breaking_height[2] +
avg_breaking_height[3]
        + avg_breaking_height[4] + avg_breaking_height[5] +
avg_breaking_height[6]
        + avg_breaking_height[7] + avg_breaking_height[8] +
avg_breaking_height[9]
        + avg_breaking_height[10] + avg_breaking_height[11]
+ avg_breaking_height[12]
        + avg_breaking_height[13] + avg_breaking_height[14]
+ avg_breaking_height[15]
        + avg_breaking_height[16] +
avg_breaking_height[17]) / 21;
    HeightRollAvg[8] = (avg_breaking_height[2*Ymax-2] +
avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7] +
avg_breaking_height[8]
        + avg_breaking_height[9] + avg_breaking_height[10]
+ avg_breaking_height[11]
        + avg_breaking_height[12] + avg_breaking_height[13]
+ avg_breaking_height[14]
        + avg_breaking_height[15] + avg_breaking_height[16]
+ avg_breaking_height[17]
        + avg_breaking_height[16]) / 21;
    HeightRollAvg[9] = (avg_breaking_height[2*Ymax-1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7] +
avg_breaking_height[8]
        + avg_breaking_height[9] + avg_breaking_height[10]
+ avg_breaking_height[11]
```

```
        + avg_breaking_height[12] + avg_breaking_height[13]
+ avg_breaking_height[14]
        + avg_breaking_height[15] + avg_breaking_height[16]
+ avg_breaking_height[17]
        + avg_breaking_height[16] +
avg_breaking_height[17]) / 21;


    AngleRollAvg[0] = (avg_breaking_angle[2*Ymax-10] +
avg_breaking_angle[2*Ymax-9]
        + avg_breaking_angle[2*Ymax-8] +
avg_breaking_angle[2*Ymax-7] + avg_breaking_angle[2*Ymax-6]
        + avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4] + avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4] + avg_breaking_angle[5] +
avg_breaking_angle[6]
        + avg_breaking_angle[7] + avg_breaking_angle[8] +
avg_breaking_angle[9]
        + avg_breaking_angle[10]) / 21;
    AngleRollAvg[1] = (avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8]
        + avg_breaking_angle[2*Ymax-7] +
avg_breaking_angle[2*Ymax-6] + avg_breaking_angle[2*Ymax-5]
        + avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3] + avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]
        + avg_breaking_angle[5] + avg_breaking_angle[6] +
avg_breaking_angle[7]
        + avg_breaking_angle[8] + avg_breaking_angle[9] +
avg_breaking_angle[10]
        + avg_breaking_angle[11]) / 21;
    AngleRollAvg[2] = (avg_breaking_angle[2*Ymax-8]
        + avg_breaking_angle[2*Ymax-7] +
avg_breaking_angle[2*Ymax-6] + avg_breaking_angle[2*Ymax-5]
        + avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3] + avg_breaking_angle[2*Ymax-2]
```

```
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]
        + avg_breaking_angle[5] + avg_breaking_angle[6] +
avg_breaking_angle[7]
        + avg_breaking_angle[8] + avg_breaking_angle[9] +
avg_breaking_angle[10]
        + avg_breaking_angle[11] + avg_breaking_angle[12])
/ 21;
    AngleRollAvg[3] = (avg_breaking_angle[2*Ymax-7] +
avg_breaking_angle[2*Ymax-6]
        + avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7] +
avg_breaking_angle[8]
        + avg_breaking_angle[9] + avg_breaking_angle[10]
        + avg_breaking_angle[11] + avg_breaking_angle[12] +
avg_breaking_angle[13]) / 21;
    AngleRollAvg[4] = (avg_breaking_angle[2*Ymax-6]
        + avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4] + avg_breaking_angle[5] +
avg_breaking_angle[6]
        + avg_breaking_angle[7] + avg_breaking_angle[8] +
avg_breaking_angle[9]
        + avg_breaking_angle[10] + avg_breaking_angle[11] +
avg_breaking_angle[12]
        + avg_breaking_angle[13] + avg_breaking_angle[14])
/ 21;
    AngleRollAvg[5] = (avg_breaking_angle[2*Ymax-5] +
avg_breaking_angle[2*Ymax-4]
```

```
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0] + avg_breaking_angle[1]
        + avg_breaking_angle[2] + avg_breaking_angle[3] +
avg_breaking_angle[4]
        + avg_breaking_angle[5] + avg_breaking_angle[6] +
avg_breaking_angle[7]
        + avg_breaking_angle[8] + avg_breaking_angle[9] +
avg_breaking_angle[10]
        + avg_breaking_angle[11] + avg_breaking_angle[12]
        + avg_breaking_angle[13] + avg_breaking_angle[14] +
avg_breaking_angle[15]) / 21;
    AngleRollAvg[6] = (avg_breaking_angle[2*Ymax-4] +
avg_breaking_angle[2*Ymax-3]
        + avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1] + avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4] + avg_breaking_angle[5] +
avg_breaking_angle[6]
        + avg_breaking_angle[7] + avg_breaking_angle[8] +
avg_breaking_angle[9]
        + avg_breaking_angle[10] + avg_breaking_angle[11] +
avg_breaking_angle[12]
        + avg_breaking_angle[13] + avg_breaking_angle[14] +
avg_breaking_angle[15]
        + avg_breaking_angle[16]) / 21;
    AngleRollAvg[7] = (avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2]
        + avg_breaking_angle[2*Ymax-1] +
avg_breaking_angle[0]
        + avg_breaking_angle[1] + avg_breaking_angle[2] +
avg_breaking_angle[3]
        + avg_breaking_angle[4] + avg_breaking_angle[5] +
avg_breaking_angle[6]
        + avg_breaking_angle[7] + avg_breaking_angle[8] +
avg_breaking_angle[9]
        + avg_breaking_angle[10] + avg_breaking_angle[11] +
avg_breaking_angle[12]
        + avg_breaking_angle[13] + avg_breaking_angle[14] +
avg_breaking_angle[15]
        + avg_breaking_angle[16] + avg_breaking_angle[17])
/ 21;
```

```
    AngleRollAvg[8] = (avg_breaking_angle[2*Ymax-2] +
avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7] +
avg_breaking_angle[8]
        + avg_breaking_angle[9] + avg_breaking_angle[10] +
avg_breaking_angle[11]
        + avg_breaking_angle[12] + avg_breaking_angle[13] +
avg_breaking_angle[14]
        + avg_breaking_angle[15] + avg_breaking_angle[16] +
avg_breaking_angle[17]
        + avg_breaking_angle[16]) / 21;
    AngleRollAvg[9] = (avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7] +
avg_breaking_angle[8]
        + avg_breaking_angle[9] + avg_breaking_angle[10] +
avg_breaking_angle[11]
        + avg_breaking_angle[12] + avg_breaking_angle[13] +
avg_breaking_angle[14]
        + avg_breaking_angle[15] + avg_breaking_angle[16] +
avg_breaking_angle[17]
        + avg_breaking_angle[16] + avg_breaking_angle[17])
/ 21;


    for (y = 10; y < 2*Ymax - 10; y++){
        HeightRollAvg[y] = (avg_breaking_height[y-10] +
avg_breaking_height[y-9]
            + avg_breaking_height[y-8] +
avg_breaking_height[y-7] + avg_breaking_height[y-6]
            + avg_breaking_height[y-5] +
avg_breaking_height[y-4] + avg_breaking_height[y-3]
            + avg_breaking_height[y-2] +
avg_breaking_height[y-1] + avg_breaking_height[y]
            + avg_breaking_height[y+1] +
avg_breaking_height[y+2] + avg_breaking_height[y+3]
```

```
            + avg_breaking_height[y+4] +
avg_breaking_height[y+5]  + avg_breaking_height[y+6]
            + avg_breaking_height[y+7] +
avg_breaking_height[y+8] + avg_breaking_height[y+9]
            + avg_breaking_height[y+10]) / 21;
        AngleRollAvg[y] = (avg_breaking_angle[y-10] +
avg_breaking_angle[y-9] + avg_breaking_angle[y-8]
            + avg_breaking_angle[y-7] +
avg_breaking_angle[y-6] + avg_breaking_angle[y-5]
            + avg_breaking_angle[y-4] +
avg_breaking_angle[y-3] + avg_breaking_angle[y-2]
            + avg_breaking_angle[y-1] +
avg_breaking_angle[y] + avg_breaking_angle[y+1]
            + avg_breaking_angle[y+2] +
avg_breaking_angle[y+3] + avg_breaking_angle[y+4]
            + avg_breaking_angle[y+5] +
avg_breaking_angle[y+6] + avg_breaking_angle[y+7]
            + avg_breaking_angle[y+8] +
avg_breaking_angle[y+9] + avg_breaking_angle[y+10]) / 21;
    }


    HeightRollAvg[2*Ymax-10] = (avg_breaking_height[2*Ymax-
20] + avg_breaking_height[2*Ymax-19]
        + avg_breaking_height[2*Ymax-18] +
avg_breaking_height[2*Ymax-17] +
avg_breaking_height[2*Ymax-16]
        + avg_breaking_height[2*Ymax-15] +
avg_breaking_height[2*Ymax-14] +
avg_breaking_height[2*Ymax-13]
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0]) / 21;
```

```
    HeightRollAvg[2*Ymax-9] = (avg_breaking_height[2*Ymax-
19] + avg_breaking_height[2*Ymax-18]
        + avg_breaking_height[2*Ymax-17] +
avg_breaking_height[2*Ymax-16]
        + avg_breaking_height[2*Ymax-15] +
avg_breaking_height[2*Ymax-14] +
avg_breaking_height[2*Ymax-13]
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1])
/ 21;
    HeightRollAvg[2*Ymax-8] = (avg_breaking_height[2*Ymax-
18]
        + avg_breaking_height[2*Ymax-17] +
avg_breaking_height[2*Ymax-16]
        + avg_breaking_height[2*Ymax-15] +
avg_breaking_height[2*Ymax-14] +
avg_breaking_height[2*Ymax-13]
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]) / 21;
    HeightRollAvg[2*Ymax-7] = (avg_breaking_height[2*Ymax-
17] + avg_breaking_height[2*Ymax-16]
```

```
        + avg_breaking_height[2*Ymax-15] +
avg_breaking_height[2*Ymax-14] +
avg_breaking_height[2*Ymax-13]
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3]) / 21;
    HeightRollAvg[2*Ymax-6] = (avg_breaking_height[2*Ymax-
16]
        + avg_breaking_height[2*Ymax-15] +
avg_breaking_height[2*Ymax-14] +
avg_breaking_height[2*Ymax-13]
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4])
/ 21;
    HeightRollAvg[2*Ymax-5] = (avg_breaking_height[2*Ymax-
15] + avg_breaking_height[2*Ymax-14] +
avg_breaking_height[2*Ymax-13]
```

```
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]) / 21;
    HeightRollAvg[2*Ymax-4] = (avg_breaking_height[2*Ymax-
14] + avg_breaking_height[2*Ymax-13]
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6]) / 21;
    HeightRollAvg[2*Ymax-3] = (avg_breaking_height[2*Ymax-
13]
        + avg_breaking_height[2*Ymax-12] +
avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
```

```
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7])
/ 21;
    HeightRollAvg[2*Ymax-2] = (avg_breaking_height[2*Ymax-
12] + avg_breaking_height[2*Ymax-11] +
avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7] +
avg_breaking_height[8]) / 21;
    HeightRollAvg[2*Ymax-1] = (avg_breaking_height[2*Ymax-
11] + avg_breaking_height[2*Ymax-10]
        + avg_breaking_height[2*Ymax-9] +
avg_breaking_height[2*Ymax-8] + avg_breaking_height[2*Ymax-
7]
        + avg_breaking_height[2*Ymax-6] +
avg_breaking_height[2*Ymax-5] + avg_breaking_height[2*Ymax-
4]
        + avg_breaking_height[2*Ymax-3] +
avg_breaking_height[2*Ymax-2] + avg_breaking_height[2*Ymax-
1]
        + avg_breaking_height[0] + avg_breaking_height[1] +
avg_breaking_height[2]
```

```
        + avg_breaking_height[3] + avg_breaking_height[4] +
avg_breaking_height[5]
        + avg_breaking_height[6] + avg_breaking_height[7] +
avg_breaking_height[8]
        + avg_breaking_height[9]) / 21;


    AngleRollAvg[2*Ymax-10] = (avg_breaking_angle[2*Ymax-
20] + avg_breaking_angle[2*Ymax-19]
        + avg_breaking_angle[2*Ymax-18] +
avg_breaking_angle[2*Ymax-17] + avg_breaking_angle[2*Ymax-
16]
        + avg_breaking_angle[2*Ymax-15] +
avg_breaking_angle[2*Ymax-14] + avg_breaking_angle[2*Ymax-
13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0]) / 21;
    AngleRollAvg[2*Ymax-9] = (avg_breaking_angle[2*Ymax-19]
+ avg_breaking_angle[2*Ymax-18]
        + avg_breaking_angle[2*Ymax-17] +
avg_breaking_angle[2*Ymax-16]
        + avg_breaking_angle[2*Ymax-15] +
avg_breaking_angle[2*Ymax-14] + avg_breaking_angle[2*Ymax-
13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1]) /
21;
    AngleRollAvg[2*Ymax-8] = (avg_breaking_angle[2*Ymax-18]
```

```
        + avg_breaking_angle[2*Ymax-17] +
avg_breaking_angle[2*Ymax-16]
        + avg_breaking_angle[2*Ymax-15] +
avg_breaking_angle[2*Ymax-14] + avg_breaking_angle[2*Ymax-
13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]) / 21;
    AngleRollAvg[2*Ymax-7] = (avg_breaking_angle[2*Ymax-17]
+ avg_breaking_angle[2*Ymax-16]
        + avg_breaking_angle[2*Ymax-15] +
avg_breaking_angle[2*Ymax-14] + avg_breaking_angle[2*Ymax-
13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3]) / 21;
    AngleRollAvg[2*Ymax-6] = (avg_breaking_angle[2*Ymax-16]
        + avg_breaking_angle[2*Ymax-15] +
avg_breaking_angle[2*Ymax-14] + avg_breaking_angle[2*Ymax-
13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
```

```
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4]) /
21;
    AngleRollAvg[2*Ymax-5] = (avg_breaking_angle[2*Ymax-15]
+ avg_breaking_angle[2*Ymax-14] +
avg_breaking_angle[2*Ymax-13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]) / 21;
    AngleRollAvg[2*Ymax-4] = (avg_breaking_angle[2*Ymax-14]
+ avg_breaking_angle[2*Ymax-13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6]) / 21;
    AngleRollAvg[2*Ymax-3] = (avg_breaking_angle[2*Ymax-13]
        + avg_breaking_angle[2*Ymax-12] +
avg_breaking_angle[2*Ymax-11] + avg_breaking_angle[2*Ymax-
10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
```

```
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7]) /
21;
    AngleRollAvg[2*Ymax-2] = (avg_breaking_angle[2*Ymax-12]
+ avg_breaking_angle[2*Ymax-11] +
avg_breaking_angle[2*Ymax-10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7] +
avg_breaking_angle[8]) / 21;
    AngleRollAvg[2*Ymax-1] = (avg_breaking_angle[2*Ymax-11]
+ avg_breaking_angle[2*Ymax-10]
        + avg_breaking_angle[2*Ymax-9] +
avg_breaking_angle[2*Ymax-8] + avg_breaking_angle[2*Ymax-7]
        + avg_breaking_angle[2*Ymax-6] +
avg_breaking_angle[2*Ymax-5] + avg_breaking_angle[2*Ymax-4]
        + avg_breaking_angle[2*Ymax-3] +
avg_breaking_angle[2*Ymax-2] + avg_breaking_angle[2*Ymax-1]
        + avg_breaking_angle[0] + avg_breaking_angle[1] +
avg_breaking_angle[2]
        + avg_breaking_angle[3] + avg_breaking_angle[4] +
avg_breaking_angle[5]
        + avg_breaking_angle[6] + avg_breaking_angle[7] +
avg_breaking_angle[8]
        + avg_breaking_angle[9]) / 21;
    }

    else {
```

```
        printf("Rolling average neighbor unspecified!
\nUsing raw averages. \n");
        for (y = 0; y < 2*Ymax; y++){
            HeightRollAvg[y] = avg_breaking_height[y];
            AngleRollAvg[y] = avg_breaking_angle[y];
        }
    }

    // Average right and left neighbors if no data for cell
    // if ((HeightRollAvg[0] == 0) && (HeightRollAvg[1] ==
0) && (HeightRollAvg[Ymax] != 0)){
        // HeightRollAvg[0] = (HeightRollAvg[2] +
HeightRollAvg[Ymax])/2;
        // AngleRollAvg[0] = (AngleRollAvg[2] +
AngleRollAvg[Ymax])/2;
        // printf("Modified Rolling Average at 0 \n");
    // } else if ((HeightRollAvg[0] == 0) &&
(HeightRollAvg[1] == 0) && (HeightRollAvg[Ymax] == 0)){
        // HeightRollAvg[0] = (HeightRollAvg[2] +
HeightRollAvg[Ymax-1])/2;
        // AngleRollAvg[0] = (AngleRollAvg[2] +
AngleRollAvg[Ymax-1])/2;
        // printf("Modified Rolling Average at 0 \n");
    // } else if ((HeightRollAvg[0] == 0) &&
(HeightRollAvg[Ymax] != 0)){
        // HeightRollAvg[0] = (HeightRollAvg[1] +
HeightRollAvg[Ymax])/2;
        // AngleRollAvg[0] = (AngleRollAvg[1] +
AngleRollAvg[Ymax])/2;
        // printf("Modified Rolling Average at 0 \n");
    // }
    // for (y = 1; y < (2*Ymax) - 1; y++){
        // if ((HeightRollAvg[y] == 0) &&
(HeightRollAvg[y+1] == 0)) {
            // HeightRollAvg[y] = (HeightRollAvg[y+2] +
HeightRollAvg[y-1])/2;
            // HeightRollAvg[y+1] = (HeightRollAvg[y+2] +
HeightRollAvg[y-1])/2;
            // AngleRollAvg[y] = (AngleRollAvg[y+2] +
AngleRollAvg[y-1])/2;
            // AngleRollAvg[y+1] = (AngleRollAvg[y+2] +
AngleRollAvg[y-1])/2;
            // printf("Modified Rolling Average at %i
\n",y);
```

```
        // } else if (HeightRollAvg[y] == 0) {
            // HeightRollAvg[y] = (HeightRollAvg[y+1] +
HeightRollAvg[y-1])/2;
            // AngleRollAvg[y] = (AngleRollAvg[y+1] +
AngleRollAvg[y-1])/2;
            // printf("Modified Rolling Average at %i
\n",y);
        // }
    // }
    // if ((HeightRollAvg[Ymax] == 0) && (HeightRollAvg[0]
== 0) && (HeightRollAvg[1] != 0)){
        // HeightRollAvg[Ymax] = (HeightRollAvg[1] +
HeightRollAvg[Ymax-1])/2;
        // AngleRollAvg[Ymax] = (AngleRollAvg[1] +
AngleRollAvg[Ymax-1])/2;
        // printf("Modified Rolling Average at %i
\n",Ymax);
    // } else if ((HeightRollAvg[Ymax] == 0) &&
(HeightRollAvg[0] == 0) && (HeightRollAvg[1] == 0)){
        // HeightRollAvg[Ymax] = (HeightRollAvg[2] +
HeightRollAvg[Ymax-1])/2;
        // AngleRollAvg[Ymax] = (AngleRollAvg[2] +
AngleRollAvg[Ymax-1])/2;
        // printf("Modified Rolling Average at %i
\n",Ymax);
    // } else if (HeightRollAvg[Ymax] == 0){
        // HeightRollAvg[Ymax] = (HeightRollAvg[1] +
HeightRollAvg[Ymax-1])/2;
        // AngleRollAvg[Ymax] = (AngleRollAvg[1] +
AngleRollAvg[Ymax-1])/2;
        // printf("Modified Rolling Average at %i
\n",Ymax);
    // }


    // Convert angles back to CEM orientation
    for (y = 0; y < 2*Ymax; y++){
        AngleRollAvg[y] = pi - AngleRollAvg[y];
    }

    //printf("Thetao = %G \n", (pi-thetao)*radtodeg);

    // If the breaking angle is over 90 degrees, consider
that "shadowed"
```

```
    for (y = 0; y < 2*Ymax; y++){
        if (AngleRollAvg[y] > 90*pi/180){
            if (debug2) printf("Cell y = %i is in shadow!",
y);
            AngleRollAvg[y] = 0;
            HeightRollAvg[y] = 0;
        }
    }

    // Display average cell wave height data
    if (debug16) printf("Average wave height at each cell
is: \n");
    //for (y = 0; y < 2*Ymax; y++){
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x=0; x < Xmax; x++){
            if (debug16) printf("%G ",
avg_cell_wave_height[x][y]);
        } if (debug16) printf("\n");
    } if (debug16) printf("\n");

    // Display the number of wave rays that passed through
each cell
    if (debug16a) printf("Number of wave rays taht passed
through each cell: \n");
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x=0; x < Xmax; x++){
            if (debug16a) printf("%i ",
NumRaysPerCell[x][y]);
        } if (debug16a) printf("\n");
    } if (debug16a) printf("\n");

    // Display the average beta data
    if (debug16b) printf("Average beta at ecah cell is:
\n");
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        for (x=0; x < Xmax; x++){
            if (debug16b) printf("%G ",
avg_cell_beta[x][y]);
        } if (debug16b) printf("\n");
    } if (debug16b) printf("\n");


    // Display average breaking wave height and angle for
each shoreline cell
```

```
    if (debug19) printf("Average breaking wave angle data
is: \n");
    //for (y = 0; y < 2*Ymax; y++){
    for (y = Ymax/2; y < 3*Ymax/2; y++){
      if (debug19) printf("%G ", avg_breaking_angle[y]*radtodeg);
    } if (debug19) printf("\n");
    if (debug19) printf("\n");
    if (debug19) printf("Average breaking wave height data: \n");
    //for (y = 0; y < 2*Ymax; y++){
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        if (debug19) printf("%G ", avg_breaking_height[y]);
    } if (debug19) printf("\n");
    if (debug19) printf("\n");

    // Display average breaking wave height and angle for each shoreline cell
    if (debug19) printf("Rolling average breaking wave angle data is: \n");
    //for (y = 0; y < 2*Ymax; y++){
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        if (debug19) printf("%G ", AngleRollAvg[y]*radtodeg);
    } if (debug19) printf("\n");
    if (debug19) printf("\n");
    if (debug19) printf("Rolling average breaking wave height data: \n");
    //for (y = 0; y < 2*Ymax; y++){
    for (y = Ymax/2; y < 3*Ymax/2; y++){
        if (debug19) printf("%G ", HeightRollAvg[y]);
    } if (debug19) printf("\n");
    if (debug19) printf("\n");

    // Determine which beach cells are shadowed
    if (UseShadow == 'n'){
        if (debug2) printf("Using WRT shadowing scheme... \n");
        if (debug2) printf("InShadow = \n");
        for (y = 0; y < 2*Ymax; y++){
            if (HeightRollAvg[y] == 0) {
                InShadow[y] = 'y';
            } else InShadow[y] = 'n';
            if (debug2) printf("%c ", InShadow[y]);
        } if (debug2) printf("\n");
    }

}
```

APPENDIX B

COASTLINE EVOLUTION MODEL (MATLAB VERSION) CODE

```
% Coastline Evolution Model (CEM)
% MATLAB Version
%
% Version 2.28.2
%
% Original program by Brad Murray, Oliver Arnoult, and
Andrew Ashton
% Reprogrammed by Andrew E. Whitley
%
% Note: if using an input PDF or starting a simulation from
input
% conditions, the appropriate matrices must already be
loaded into the
% MATLAB workspace.
%
% Input files:
%   WavePDF     - input wave PDF
%       1st column: maximum wave angle (deg) for bin
%       2nd column: fractional probability of wave angle
bin
%   PercentFull - fractional amount of sediment in each
domain cell
%   AllBeach    - flag indicating of cell is entirely beach
('y'/'n')
%   Age         - Age since cell was deposited
%   CellDepth   - Depth array (m)
%
% Required function files:
%   CEM_AdjustShore.m
%   CEM_AgeCells.m
%   CEM_CalcGroinBypassParams.m
%   CEM_AWTRefractToBreak.m
%   CEM_AWTRefractToDepth.m
%   CEM_CalcVolInOut.m
%   CEM_CheckOverwash.m
%   CEM_CheckOverwashSweep.m
%   CEM_CreateGroins.m
%   CEM_DetermineAngles.m
%   CEM_DetermineMeanWvDir.m
%   CEM_DetermineSedTransport.m
```

```
%   CEM_DoOverwash.m
%   CEM_FindCoastline.m
%   CEM_FindGroinBeachCells.m
%   CEM_FindIfInShadow.m
%   CEM_FindWaveAngle.m
%   CEM_FindWaveAngleWaveIn.m
%   CEM_FixBeach.m
%   CEM_GroinShadowing.m
%   CEM_InitConds.m
%   CEM_InitCondsFromLine.m
%   CEM_InitialPert.m
%   CEM_MassCount.m
%   CEM_MassCountBetweenPoints.m
%   CEM_MATLABFindBeachCells.m
%   CEM_OopsImEmpty.m
%   CEM_OopsImFull.m
%   CEM_PeriodicBoundaryCopy.m
%   CEM_SaveLineToFile.m
%   CEM_SaveSandToFile.m
%   CEM_SedTrans.m
%   CEM_ShadowSweep.m
%   CEM_StraightBoundaries.m
%   CEM_TransportSedimentSweep.m
%   CEM_XMaxBeach.m

% Clear workspace (except for persisting input variables)
clearvars -except WavePDF PercentFull CellDepth AllBeach
Age GroinInputData ...
    InputLine Coastline0 MassInitial MassCountInput
MassBetweenPointsInitial...
    CumVolumeIn CumVolumeOut

%%% CONTROLS AND PARAMETERS %%%

% Run Control Parameters
TimeStep = 1/24;          % Time step in days
StartingTimeStep = 0;    % Starting time step
StopAfter = 6*24*365;           % Stop after what number of
time steps
seed = 44;               % Random number generator seed (can
set to 'default' to return to MATLAB default')
RunOverwash = 0;          % Run overwash algorithm?
TimeModel = 1;           % Time the model?
```

```
UseFixBeachFlag = 0;    % Use the FixBeachFlag to run
CEM_FixBeach twice?

% Initialization Parameters
StartFromFile = 0;      % Start from an input file (or
matrix in this case)?
StartFromLine = 1;      % Start from an input coastline?
                            % Name of input coastline:
'InputLine'
LoadCellDepth = 0;      % Load the CellDepth matrix?
LoadAge = 0;            % Load the Age matrix?
LoadAllBeach = 0;       % Load the AllBeach matrix?
WaveIn = 1;             % Input a wave PDF?
readwavename = WavePDF;       % Input PDF file (matrix)
                                 % Column 1: Maximum wave
angle in bin
                                 % Column 2: Wave
probability for bin

   % Note that positive wave angles approach from the left,
and negative approach from the right

% Output File Parameters
SaveFile = 0;           % Save files?
savefilename = 'fileout';
SaveLine = 1;           % Save shoreline files
savelinename = 'lineout';
StartSavingAt = 1;      % Time step to start saving files
SaveSpacing = 5000;     % Time steps between save files
SaveLineSpacing = 730;  % Time steps between saving line
files
SaveLineASCII = 1;        % Save coastline line output
files in ASCII?
SaveFinalData = 1;      % Save the final data?

% Wave and Wave PDF Parameters
UseSingleWaveAngle = 0;     % Use a single wave angle for
the entire simulation? (otherwise uses a PDF)
SingleOffshoreAngle = 20;   % Single offshore wave angle
(deg)

OffShoreWvHt = 0.7141;       % Offshore wave height (m)
Period = 4.1636;             % Offshore wave period (s)
```

```
Asym = '';                % Asymmetry of wave PDF
(fractional portion of waves coming from positive (left)
direction)
Highness = '';          % Highness of wave PDF (fractional
portion of waves coming from high angle (> 45 deg))
Duration = 1;           % Number of time step calculations
loop at same wave angle
BinSize = 15;           % Size of wave PDF bins (deg)
MaxDepth = 15.8 ;          % Maximum depth to start
refraction (m)
RefractStep = 0.2;      % Wave refraction step size (depth,
m)
AgeMax = 1000000;       % Maximum 'age' of cells - loops
back to zero

% Sediment Transport Parameters
UseVariableCERC = 1;        % Use the variable CERC
equation? --> allows the parameterization of rho_s,
porosity, and K

MaxVol = 5000;               % Maximum volume that can be
transported across a single border (m^3)

K = 0.92;                   % CERC formula emperical
constant (overridden if UseBailard)
rho = 1000;                 % kg/m3 - density of water and
dissolved matter
rho_s = 2650;               % Denisty of sediment (kg/m3)
porosity = 0.4;             % Sediment porosity (n in most
literature)
KBreak = 0.5352;               % Coefficient for wave
breaking threshold (breaking wave height / depth at
breaking); originally 0.5; overridden by UseBattjesStive


% Periodic Boundary Condition Controls
UsePeriodicBoundaries = 0;      % Use periodic boundaries?
(PeriodicBoundaryCopy is applied each time step)

InitialStraigthBoundaries = 1;  % Have the initial
boundaries be straight on the extremities?
ForceStraightBoundaries = 0;    % Force straight boundaries
on the extremites? (copies every time step)
```

```
                                    % Note: if both
periodic boundaries and
                                    % forcing straight
boudnaries are off,
                                    % no copying of the
extremeties takes
                                    % place
PeriodicCopyGroins = 0;          % Periodic boundary copy
the groins? (CURRENTLY NOT FUNCTIONAL)

% Aspect Parameters
CellWidth = 25;             % Size of cells (m)
Xmax = 75;                  % Number of cells in x (cross-
shore) direction
Ymax = 1000;                  % Number of cells in y
(longshore) direction
MaxBeachLength = 8*Ymax;    % Maximumk length of arrays
that contain beach data
ShelfSlope = 0.00051;         % Slope of continental shelf
ShorefaceSlope = 0.015;      % Linear slope of shoreface
DepthShoreface = 8;      % Minimum depth of shoreface due
to wave action (m)
InitBeach = 20;             % Cell where initial conditions
changes from beach to ocean
InitialDepth = 8;           % Theoretical depth (m) of
continental shelf at x = InitBeach
LandHeight = 1.31 ;             % Elevtaion of land above
MHW
InitCType = 0;              % Type of initial conditions (0
= sandy; 1 = barrier);
InitBWidth = 4;             % Intiial min width of barrier
(cells)
OWType = 1;                 % 0 = use depth array, 1 = use
geometric rule
OWMinDepth = 0.1;           % Smallest overwash of all
FindCellError = 5;          % if we run off of array, how
far over do we try again?
SedTransLimit = 90;          % Beyond what absolute slope
don't do sed trans (deg)
CritBWidth = 350;           % Width barrier maintains due
to overwash (m) important scaling parameter!
OverwashLimit = 75;         % beyond what angle don't do
overwash
```

```
% Gaussian Coastline Parameters
StartWithGaussian = 0;      % Start with a Gaussian-shaped
coastline
GaussMult = 800;        % Gaussian function multiplier;
controls the magnitude (height) of the Gaussian function
GaussSigma = 50;        % "Standard deviation" for
Gaussian; controls with width of the function
GaussCenter = 200;       % "Mean" for the Gaussian;
controls the center of the function (in cells)

% Groin Controls and Parameters
UseGroins = 1;              % Include groins in the
simulation
UseGroinBypassing = 1;      % Include groin bypassing in
the simulation?
UseGroinDiffraction = 1;    % Include groin diffraction in
the simulation?
UseGroinShadowBlock = 0;    % Have the groin shadow block
all incoming wave energy?
UseGeometricDiffAng = 0;    % Geometrically determine
breaking wave angle from diffraction? (if not, use Eq 15.11
in Kamphuis, 2010)
UseBreakDepthAsD_LT = 1;      % Use the modeling wave
breaking depth as depth of longshore transport (D_LT)?

GroinStart = 1;             % Cross-shore starting position
of groins (cells)
GroinData = GroinInputData; % Groin input data
                              % Column 1: Alongshore
position of groins (cells, indicates groin on right side of
cell)
                              % Column 2: Cross-shore
length of groins (cells)
GroinPermeability = 0.5;      % Fractional amount of
sediment allowed to pass through groins (range 0 [0%] to 1
[100%])
DeanProfileA = .053;        % Dean profile scaling
parameter (used in Groin Bypassing)
A_w = 1.27;                 % Factor to convert waveheight
to H_1/10 for depth of longshore transport calc (1.27 for
Hs; only used if UseBreakDepthAsD_LT=0)

% Sea Level Rise (SLR) Parameters
```

```
UseConstantSLR = 0;          % Use constant sea level rise
algorithm?
RateSLR_Const = 0.63;        % Rate of constant SLR (cm/yr)

% Plot Controls
ShowPFAnim = 0;              % Show an animation of
PercentFull matrix every time step?
ShowLineAnim = 1;            % Show an animation of the
Coastline every time step?
PlotInitialCoastline = 1;      % Show the original
coastline on the line animation?
PauseAnim = 0;               % Pause the animation every
timestep?

% Mass counting parameters
MassCountBetweenPoints = 0;     % Count mass in between
specified points?
MassCountPoints = '';   % Matrix containing end points for
mass counting extents
                                        % (N X 2) Matrix
containing end
                                        % points (LS cells)
in which to
                                        % count sediment
mass

% Debugging parameters
SaveAge = 0;                 % Save the age of cells?
ScreenTextSpacing = 100;    % Spacing of writing to the
screen in time steps
NoPauseRun = 1;              % Disable PauseRun subroutine
InitialPert = 0;             % Start with a small bump?
InitialSmooth = 1;           % Smooth starting conditions
WaveAngleSign = 1;           % Used to change the sign on
wve angles
debug0 = 0;                  % Main progrm stpes
debug1 = 0;                  % Find Next Cell
debug2 = 0;                  % Shadow routine
debug3 = 0;                  % Determine angles
debug3a = 0;                 % Wave angle adjustment
debug3b = 1;                 % PDF calling
debug4 = 0;                  % Upwind/downwind
debug5 = 0;                  % Sediment transport decisions
```

```
debug6 = 0;                 % Sediment transport
calculations
debug6a = 0;                % To/From sediment transport
volumes
debug7 = 0;                 % Transport sweep (moving
sediment)
debug7a = 0;                % slope calculations
debug8 = 0;                 % Full/empty
debug9 = 0;                 % Fix beach
debug10a = 0;               % Overwash tests
debug10b = 0;               % Doing overwash
debug40 = 0;                % Groin location
debug40a = 0;               % Transport over groins
(includes bypassing)
debug41 = 0;                % Groin diffraction shadowing
debug41a = 0;               % Groin diffraction
calculations
debug42 = 0;                % Mass count between
subroutines (domain)
debug43 = 0;                % Mass count between points
(once per timestep)
debug43a = 0;               % Mass count between points
subroutines
OWFlag = 0;                 % debugger

% Universal Constants
g = 9.80665;                % Acceleration due to gravity
(m/s^2)
radtodeg = 180/pi;          % Transform radians to degrees
degtorad = pi/180;          % Transform degrees to radians

% Computational Arrays (determined each time step)
X = zeros(1,MaxBeachLength);        % X position (cross-
shore) of ith beach element
Y = zeros(1,MaxBeachLength);        % Y position
(alongshore) of ith beach element
InShadow = char(ones(1,MaxBeachLength)*'?');     % Is the
ith beach element in shadow?
ShorelineAngle = zeros(1,MaxBeachLength);   % Angle between
cell and right (z+1) neighbor
SurroundingAngle = zeros(1,MaxBeachLength);  % Cell-
orietned angle based upon left and right neighbor
UpWind = char(ones(1,MaxBeachLength)*'?');  % upwind or
downwind condition used to calculated sediment transport
```

```
% VolumeIn = zeros(1,MaxBeachLength);      % Sediment volume
into ith beach element
% VolumeOut = zeros(1,MaxBeachLength);     % Sediment volume
out of ith beach element
CumVolumeIn = zeros(1,2*Ymax);      % Cumulative sediment
volume into every y position
CumVolumeOut = zeros(1,2*Ymax);      % Cumulative sediment
volume out of every y position
MeanVolumeIn = zeros(1,2*Ymax);      % Mean sediment volume
into every y position
MeanVolumeOut = zeros(1,2*Ymax);      % Cumulative sediment
volume out of every y position

% Miscellaneous Variables
CurrentTimeStep = StartingTimeStep;         % Current time
step

% End controls and parameters

%% Begin main program

if (TimeModel)
    tic
end

% Initialize variables
ShadowXMax = Xmax -5;
rng(seed);
YStart = (Ymax/2) + 1;  % Starting y position in coastline
of interest
YEnd = 3*Ymax/2;        % Ending y position in coastline of
interest


%%% LOAD FILE
if (StartFromLine == 1)

    % Make the initial conditions
    [PercentFull, CellDepth, AllBeach, Age] =
CEM_InitCondsFromLine(InputLine,...

InitialDepth,InitBeach,CellWidth,ShelfSlope,LandHeight,Dept
hShoreface,...
```

```
UsePeriodicBoundaries,InitialStraigthBoundaries,Xmax,Ymax);

    fprintf('Line file read! \n');
    %pause


elseif (StartFromFile == 0)

    % Initialize overall shoreface configuation arrays
    AllBeach = char(ones(2*Ymax,Xmax) * 'n');       % Flag
indicating of cell is entirely beach ('y'/'n')
    PercentFull = zeros(2*Ymax,Xmax);   % Fractional amount
of shore cell full of sediment
    Age = zeros(2*Ymax,Xmax);           % Age since cell
was deposited
    CellDepth = zeros(2*Ymax,Xmax);     % Depth array (m)

    % Make the initial conditions
    [PercentFull, CellDepth, AllBeach, Age] =
CEM_InitConds(PercentFull,...

CellDepth,AllBeach,Age,InitialDepth,InitBeach,InitBWidth,Ce
llWidth,...

ShelfSlope,LandHeight,DepthShoreface,InitialSmooth,InitialP
ert,...

InitCType,Xmax,Ymax,StartWithGaussian,GaussMult,GaussSigma,
GaussCenter);

    fprintf('Initial Conditions OK \n');
    %pause

elseif (StartFromFile == 1)
    % Starting from a pre-loaded matrix
    % Will AT LEAST need PercentFull
    % If CellDepth, Age, or AllBeach are not loaded, they
are created below

    % Make the CellDepth Matrix
    if (LoadCellDepth == 0)

        CellDepth = zeros(2*Ymax,Xmax);
```

```
        for y = 1:2*Ymax
            for x = 1:Xmax
                CellDepth(y,x) = InitialDepth + ((x-
InitBeach) * CellWidth * ShelfSlope);

                if (PercentFull(y,x) > 0)

                    CellDepth(y,x) = -LandHeight;

                elseif (CellDepth(y,x) < DepthShoreface)
                    CellDepth(y,x) = DepthShoreface;
                end
            end
        end
    end

    % Make the Age matrix
    if (LoadAge == 0)
        Age = zeros(2*Ymax,Xmax);
    end

    % Make the AllBeach Matrix
    if (LoadAllBeach == 0)

        AllBeach = char(ones(2*Ymax,Xmax) * 'n');

        for y = 1:2*Ymax
            for x = 1:Xmax

                if (PercentFull(y,x) >= 1)
                    AllBeach(y,x) = 'y';
                else
                    AllBeach(y,x) = 'n';
                end

            end
        end
    end

else
    fprintf('Loading file algorithm busted!!!\n');
    beep
    pause
```

```
end

%% Initialize Model Domain

% Periodic boundary copy
if (ForceStraightBoundaries)

    [PercentFull, CellDepth, AllBeach, Age] =
CEM_StraightBoundaries(PercentFull,CellDepth,AllBeach,Age,Y
max);

elseif (UsePeriodicBoundaries)

    [PercentFull, CellDepth, AllBeach, Age] =
CEM_PeriodicBoundaryCopy(PercentFull,...
        CellDepth,AllBeach,Age,Xmax,Ymax);

end

% Groin Creation
if (UseGroins)
    [GroinCells,GroinX,GroinY,GroinXPlot,GroinYPlot] =
CEM_CreateGroins(Xmax,Ymax,GroinData,GroinStart,InitBeach,P
eriodicCopyGroins);
else
    GroinCells = zeros(2*Ymax,Xmax);  % Matrix showing
cells that have a groin on the right boundary
end

% Fix Beach
[PercentFull, AllBeach, CellDepth, FixBeachFlag] =
CEM_FixBeach(PercentFull, AllBeach, CellDepth,...

GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlope,CellWid
th,Xmax,Ymax,LandHeight,debug8,debug9);

% Mass Count
if (StartingTimeStep <= 1)
    [MassInitial] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
    MassCurrent = MassInitial;

    if (MassCountBetweenPoints)
```

```
        [MassBetweenPointsInitial] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
        MassBetweenPointsCurrent =
MassBetweenPointsInitial;
    end
end


%% Input the wave PDF

if (WaveIn)

    fprintf('CHECK READ WAVE\n');

    WaveMax = readwavename(:,1);
    WaveProb = readwavename(:,2);
    NumWaveBins = length(WaveMax);

    fprintf('Wave PDF imported!\n');

    BinProbabilitiy = zeros(1,NumWaveBins);
    BinProbabilitiy(1) = WaveProb(1);
    for ii = 2:NumWaveBins
        BinProbabilitiy(ii) = WaveProb(ii) +
BinProbabilitiy(ii-1);
    end

end

% Preallocate to rrack the number of times each wave angle
bin is called
if ((debug3 || debug3b) && WaveIn)
    TimesAngleHit = zeros(2,length(WaveProb));
    TimesAngleHit(2,:) = WaveProb;
% elseif (debug3)
%     WaveMax = [-45 0 45 90];
%     WaveProb = [(((1-Asym)/2)+(Highness/2))/2 (((1-
Asym)/2)+((1-Highness)/2))/2 ...
%         (((Asym)/2)+((1-Highness)/2))/2
(((Asym)/2)+(Highness/2))/2];
%
%     TimesAngleHit = zeros(2,4);
%     TimesAngleHit(2,:) = WaveProb;
```

```
end


%% ----------------    PRIMARY PROGRAM LOOP   -------------
----------------

% Save the initial coastline
if ((StartingTimeStep == 0) && (SaveLine ||
PlotInitialCoastline))
    [Coastline0] =
CEM_SaveLineToFile(PercentFull,AllBeach,InitBeach,...

Xmax,Ymax,CurrentTimeStep,savelinename,SaveLineASCII);
end

% Determine the mean wave angles
if (UseGroins && UseGroinDiffraction)
    CEM_DetermineMeanWvDir
else
    PosMWD_0 = 0;     % Positive mean offshore wave
direction (radians)
    NegMWD_0 = 0;     % Negative mean offshore wave
direction (radians)
end


while (CurrentTimeStep < StopAfter)
    % Time Step iteration - compute same wave angle for
Duration time steps

    % Calculate the Wave Angle
    %%% May want to make these outside functions, though
I've had problems
    %%% doing that in the past
    if (UseSingleWaveAngle)
        % Use a single wave angle for the duration of the
simulation
        WaveAngle = SingleOffshoreAngle*degtorad;

    elseif(WaveIn)
        % Use the imported Wave PDF
```

```
        %WaveAngle =
CEM_FindWaveAngleWaveIn(WaveProb,BinProbabilitiy,BinSize);
        %WaveAngle*radtodeg

        % Initialize
        flag = 1;
        index = 1;

        % Choose a random PDF bin
        RandBin = rand(1);

        while(flag)
            if (RandBin <=  BinProbabilitiy(index))
                AngleBin = WaveMax(index);
                flag = 0;
                break
            end
            index = index + 1;

        end

        % Create a random angle fluctuation
        AngleFluct = rand(1);

        % Calculate the offshore wave angle
        Angle = (AngleFluct * BinSize + (AngleBin -
BinSize));
        WaveAngle = Angle*degtorad;

    else

        % Method using wave probability step function (CEM
default)

        % Determine sign
            % variable Asym will determine fractional
distribution of waves
            % coming from the positive direction (positive
direction coming
            % from left)  -i.e. fractional wave asymmetry

        AsymRandom = rand(1);

        if (AsymRandom <= Asym)
```

```
            Sign = 1;
        else
            Sign = -1;
        end

        % Determine wave angle

        AngleRandom = rand(1);

        if (AngleRandom > Highness)
            Angle = Sign * (((AngleRandom)-Highness)/(1-
Highness)) * pi / 4.0;
        else
            Angle = Sign * (((AngleRandom)/Highness)*pi/4.0
+ pi/4.0);
        end

         WaveAngle = WaveAngleSign*Angle;

    end % End wave angle determination

    % Make sure wave angle is between 89 and -89 degrees

    if (WaveAngle >= (89*degtorad))
        Angle = Angle - (1*degtorad);
        if (debug3a)
            fprintf('Adjusted WaveAngle by -1 deg\n');
        end
    elseif (WaveAngle <= (-89*degtorad))
        Angle = Angle + (1*degtorad);
        if (debug3a)
            fprintf('Adjusted WaveAngle by +1 deg\n');
        end
    end


    % Determine how many times each bin in PDF is called
    if ((debug3 || debug3b) && WaveIn)
        for iii = 1:length(WaveProb)
            if ((WaveAngle*radtodeg > (WaveMax(iii) -
BinSize)) && ...
                    (WaveAngle*radtodeg <= WaveMax(iii)))
                TimesAngleHit(1,iii) = TimesAngleHit(1,iii)
+ 1;
```

```
                break;
            end
        end
    end

    for xx = 1:Duration

        % Text to screen?
        if (mod(CurrentTimeStep,ScreenTextSpacing) == 0)
            fprintf(['==== WaveAngle:
',num2str(WaveAngle*radtodeg),...
                ' MASS Percent:
',num2str(MassCurrent/MassInitial),...
                ' Time Step:
',num2str(CurrentTimeStep),'\n']);
        end

        % Periodic boundary copy
        if (ForceStraightBoundaries)

            [PercentFull, CellDepth, AllBeach, Age] =
CEM_StraightBoundaries(PercentFull,CellDepth,AllBeach,Age,Y
max);

        elseif (UsePeriodicBoundaries)

            [PercentFull, CellDepth, AllBeach, Age] =
CEM_PeriodicBoundaryCopy(PercentFull,...
                CellDepth,AllBeach,Age,Xmax,Ymax);

        end

        % Zero Vars
        %%% May want to make this a function
        X(:,:) = -1;
        Y(:,:) = -1;
        InShadow(:,:) = '?';
        ShorelineAngle(:,:) = -999;
        SurroundingAngle(:,:) = -998;
        UpWind(:,:) = '?';

        if (debug42)
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
```

```
            fprintf('TS:%i, Mass before
FixBeach:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end

        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, before FixBeach,
MassPer between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
        end

        % Fix Beach
        [PercentFull, AllBeach, CellDepth, FixBeachFlag] =
CEM_FixBeach(PercentFull, AllBeach, CellDepth,...

GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlope,CellWid
th,Xmax,Ymax,LandHeight,debug8,debug9);

        % Run Fix Beach again if necessary
        if (UseFixBeachFlag && FixBeachFlag)
            [PercentFull, AllBeach, CellDepth,
FixBeachFlag] = CEM_FixBeach(PercentFull, AllBeach,
CellDepth,...

GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlope,CellWid
th,Xmax,Ymax,LandHeight,debug8,debug9);
        end

        if (debug42)
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
            fprintf('TS:%i, Mass after
FixBeach:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end
```

```
        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, after FixBeach, MassPer
between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
        end

        % Find beach cells using MATLAB function
        [X, Y, TotalBeachCells] =
CEM_MATLABFindBeachCells(PercentFull,AllBeach,InitBeach,Xma
x,Ymax);

        % Find which beach cells have a groin on the border
        if (UseGroins)
            [GroinRightOfBeach,GroinLeftOfBeach] =
CEM_FindGroinBeachCells(X,Y,GroinCells,TotalBeachCells,Ymax
,debug40);
        else
            GroinRightOfBeach = zeros(1,TotalBeachCells);
            GroinLeftOfBeach = zeros(1,TotalBeachCells);
        end

        % Run ShadowSweep
        [InShadow, ShadowXMax] = CEM_ShadowSweep
(InShadow,ShadowXMax,TotalBeachCells,AllBeach,PercentFull,X
max,Ymax,WaveAngle,X,Y,CurrentTimeStep,debug2);

        if (debug0)
            fprintf(['Shadowswept:
',num2str(CurrentTimeStep),'\n']);
        end
```

```
        % DetermineAngles
        [ShorelineAngle,UpWind,SurroundingAngle] =
CEM_DetermineAngles(X,Y,PercentFull,AllBeach,WaveAngle,InSh
adow,TotalBeachCells,ShorelineAngle,SurroundingAngle,Ymax,d
ebug3,debug4);


        if (debug0)
            fprintf(['AngleDet:
',num2str(CurrentTimeStep),'\n']);
        end

        % Determine shadowing from groin diffraction
        if (UseGroins)


[InDiffShadow,GroinTipX,GroinTipY,DiffPhi_b,DiffK_b] =
CEM_GroinShadowing(X,...

WaveAngle,OffShoreWvHt,Period,MaxDepth,GroinData,GroinStart
,...

TotalBeachCells,DeanProfileA,RefractStep,PeriodicCopyGroins
,...

Ymax,UseGroinDiffraction,UseGroinShadowBlock,UseGeometricDi
ffAng,PosMWD_0,NegMWD_0,...
                CellWidth,KBreak,debug6);
        else
            InDiffShadow = zeros(1,TotalBeachCells);
            DiffPhi_b = 0;
            DiffK_b = 0;
        end

%         beep
%         fprintf('Paused\n');
%         pause

        % Determine parameters for groin bypassing
        if (UseGroins && UseGroinBypassing)
            CEM_CalcGroinBypassParams
        else
            D_G = zeros(1,TotalBeachCells); % depth of
water at groin tips
```

```
            D_G = D_G + 999;
        end

        if (debug0)
            fprintf(['GroinsSwept:
',num2str(CurrentTimeStep),'\n']);
        end

        % DetermineSedTransport
        CEM_DetermineSedTransport

        if (debug0)
            fprintf(['Sed Trans:
',num2str(CurrentTimeStep),'\n']);
        end

        % Calculate the cumulate and mean sediment
transport volumes
        CEM_CalcVolInOut;

        if (debug42)
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
            fprintf('TS:%i, Mass Before Transport
Sweep:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end

        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, before transport sweep,
MassPer between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
        end
```

```
        % Run TransportSedimentSweep
        CEM_TransportSedimentSweep

        if (debug0)
            fprintf(['Transswept:
',num2str(CurrentTimeStep),'\n']);
        end

        if (debug42)
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
            fprintf('TS:%i, Mass After Transport
Sweep:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end

        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, after transport sweep,
MassPer between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
        end

        % Fix Beach
        [PercentFull, AllBeach, CellDepth, FixBeachFlag] =
CEM_FixBeach(PercentFull, AllBeach, CellDepth,...

GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlope,CellWid
th,Xmax,Ymax,LandHeight,debug8,debug9);

        % Run Fix Beach again if necessary
        if (UseFixBeachFlag && FixBeachFlag)
            [PercentFull, AllBeach, CellDepth,
FixBeachFlag] = CEM_FixBeach(PercentFull, AllBeach,
CellDepth,...
```

```
GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlope,CellWid
th,Xmax,Ymax,LandHeight,debug8,debug9);
        end
        if (debug0)
            fprintf(['Fixed Beach:
',num2str(CurrentTimeStep),'\n']);
        end

        if (debug42)
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
            fprintf('TS:%i, Mass after
FixBeach:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end

        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, after FixBeach, MassPer
between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
        end

        %%% OVERWASH %%%
        if (RunOverwash)
            % because shoreline config may have been
changed, need to refind
            % shoreline and recalc angles

            % ZeroVars
            %%% May want to make this a function
            X(:,:) = -1;
            Y(:,:) = -1;
            InShadow(:,:) = '?';
```

```
            ShorelineAngle(:,:) = -999;
            SurroundingAngle(:,:) = -998;
            UpWind(:,:) = '?';
            VolumeIn(:,:) = 0;
            VolumeOut(:,:) = 0;

            % Periodic boundary copy
            if (ForceStraightBoundaries)
                [PercentFull, CellDepth, AllBeach, Age] =
CEM_StraightBoundaries(PercentFull,CellDepth,AllBeach,Age,Y
max);
            elseif (UsePeriodicBoundaries)
                [PercentFull, CellDepth, AllBeach, Age] =
CEM_PeriodicBoundaryCopy(PercentFull,...
                    CellDepth,AllBeach,Age,Xmax,Ymax);
            end

            % Initialize for Find Beach Cells  (make sure
strange beach does not cause trouble
            FellOffArray = 'y';
            FindStart = 1;

            % Look for beach - if you fall off of array,
bump over a little and try again
            while (FellOffArray == 'y')

                [X, Y, FellOffArray, TotalBeachCells,
InShadow, ShorelineAngle, ...
                    SurroundingAngle, UpWind, VolumeIn,
VolumeOut] = CEM_FindBeachCells(X,Y,...

Xmax,Ymax,FindStart,AllBeach,PercentFull,MaxBeachLength,InS
hadow,...

ShorelineAngle,SurroundingAngle,UpWind,VolumeIn,VolumeOut,d
ebug1);

                FindStart = FindStart + FindCellError;

                % Get Out if no good beach spots exist -
finish program
                if (FindStart > (Ymax/2) + 1)
                    fprintf(['Stopped Finding Beach - done,
FindStart: ',...
```

```
                        num2str(FindStart),' Ymax/2 -
5:',num2str((Ymax/2) - 5)]);
                    beep
                    pause
                    return
                end

            end

            % Run ShadowSweep
            [InShadow, ShadowXMax] = CEM_ShadowSweep
(InShadow,ShadowXMax,...

TotalBeachCells,AllBeach,PercentFull,Xmax,Ymax,WaveAngle,X,
Y,debug2);

            if (debug0)
                fprintf(['Shadowswept:
',num2str(CurrentTimeStep),'\n']);
            end

            % DetermineAngles
            [ShorelineAngle,UpWind,SurroundingAngle] =
CEM_DetermineAngles(X,Y,PercentFull,AllBeach,WaveAngle,InSh
adow,TotalBeachCells,ShorelineAngle,SurroundingAngle,Ymax,d
ebug3,debug4);

            if (debug0)
                fprintf(['AngleDet:
',num2str(CurrentTimeStep),'\n']);
            end

            % CheckOverwashSweep
            [PercentFull, AllBeach, CellDepth, OWFlag] =
CEM_CheckOverwashSweep(AllBeach,PercentFull,CellDepth,X,Y,.
..

SurroundingAngle,InShadow,OverwashLimit,TotalBeachCells,Cri
tBWidth,CellWidth,DepthShoreface,LandHeight,OWFlag,OWType,Y
max,debug8,debug10a,debug10b);

        end % end overwash algorithm

        if (debug42)
```

```
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
            fprintf('TS:%i, Mass before
FixBeach:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end

        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, before FixBeach,
MassPer between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
        end

        % Fix Beach
        [PercentFull, AllBeach, CellDepth, FixBeachFlag] =
CEM_FixBeach(PercentFull, AllBeach, CellDepth,...

GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlope,CellWid
th,Xmax,Ymax,LandHeight,debug8,debug9);

        % Run Fix Beach again if necessary
        if (UseFixBeachFlag && FixBeachFlag)
            [PercentFull, AllBeach, CellDepth,
FixBeachFlag] = CEM_FixBeach(PercentFull, AllBeach,
CellDepth,...

GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlope,CellWid
th,Xmax,Ymax,LandHeight,debug8,debug9);
        end

        if (debug42)
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
```

```
            fprintf('TS:%i, Mass after
FixBeach:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end

        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, after FixBeach, MassPer
between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
        end

        % Account for SLR
        if (UseConstantSLR)

            [PercentFull] =
CEM_ConstantSLRAdjust(PercentFull,RateSLR_Const,TimeStep,Sh
orefaceSlope,CellWidth,Xmax,Ymax);

        end

        if (debug42)
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);
            fprintf('TS:%i, Mass After SLR
Adjust:%G\n',CurrentTimeStep,MassCurrent/MassInitial);
        end

        if (debug43a)
            [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
            MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;
```

```
                for PointSet =
1:length(MassCountPoints(:,1))
                    fprintf('TS: %i, after SLR adjust,
MassPer between %i and %i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
            end

            % Age Empty Cells
            if (SaveAge)
                [Age] =
CEM_AgeCells(Age,PercentFull,AgeMax,CurrentTimeStep,Xmax,Ym
ax);
            end

            % Count Mass
            [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries);

            if (MassCountBetweenPoints)
                [MassBetweenPointsCurrent] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax);
                MassBetweenPointsPercentage =
MassBetweenPointsCurrent./MassBetweenPointsInitial;

                if (debug43 && MassCountBetweenPoints)
                    NumPointSets =
length(MassCountPoints(:,1));

                for PointSet = 1:NumPointSets
                    fprintf('TS: %i, MassPer between %i and
%i == %G\n',...

CurrentTimeStep,MassCountPoints(PointSet,1),MassCountPoints
(PointSet,2),MassBetweenPointsPercentage(PointSet));
                end
            end
        end

        CurrentTimeStep = CurrentTimeStep + 1;
```

```
        % Determine the coastline
        [Coastline] =
CEM_FindCoastline(PercentFull,AllBeach,InitBeach,Xmax,Ymax)
;

        % SAVE FILE
        if (((mod(CurrentTimeStep,SaveSpacing) == 0 &&
CurrentTimeStep...
                >= StartSavingAt) || (CurrentTimeStep ==
StopAfter)) && SaveFile)


CEM_SaveSandToFile(PercentFull,Age,CellDepth,AllBeach,Curre
ntTimeStep,SaveAge,savefilename);

        end

        % Save the shorelines
         if (((mod(CurrentTimeStep,SaveLineSpacing) == 0 &&
CurrentTimeStep...
                >= StartSavingAt) || (CurrentTimeStep ==
StopAfter)) && SaveLine)

            [Coastline] =
CEM_SaveLineToFile(PercentFull,AllBeach,InitBeach,Xmax,Ymax
,CurrentTimeStep,savelinename,SaveLineASCII);

         end

    end % End of duration for each time step

    %%% GRAPHICS AND ANIMATION %%%
    % Prepare the Years
    CurrentYear = (CurrentTimeStep)*(TimeStep/365);

    % PercentFull Animation
    if (ShowPFAnim)
        figure(1);

imagesc(rot90(PercentFull(((Ymax/2)+1):(3*Ymax/2),:)));
        title(['TimeStep: ',num2str(CurrentTimeStep),'
Year: ',num2str(CurrentYear)]);
        xlabel('Y (cells)');
        ylabel('X (cells)');
```

```
            drawnow;

            % Pause animation?
            if (PauseAnim)
                pause;
            end

        end

        % Coastline Animation
        if (ShowLineAnim)

            % Make the figure
            figure(2);

            % Plot the Intial coastline
            if (PlotInitialCoastline)
                plot(Coastline0,'--k');
                hold on;
            end

            % Plot the current coastline
            plot(Coastline);
            title(['TimeStep: ',num2str(CurrentTimeStep),'
Year: ',num2str(CurrentYear)]);
            xlabel('Y (cells)');
            ylabel('X (cells)');

            % Plot the groin locations
            if (UseGroins)
                scatter(GroinXPlot,GroinYPlot,'s','r');
            end

            % Make the legend
            if (PlotInitialCoastline && UseGroins)
                legend('Initial Coastline','Current
Coastline','Groin');
            elseif (PlotInitialCoastline)
               legend('Initial Coastline','Current Coastline');
            end

            % Draw now
            drawnow;
            hold off;
```

```
        % Pause animation?
        if (PauseAnim)
            pause;
        end


    end

%     figure(1);
%     plot(InDiffShadow)
%     title(['TimeStep:',num2str(CurrentTimeStep),'
WvAngle:',num2str(WaveAngle*radtodeg)])

end % End of main program loop

% ------------------------ END OF MAIN --------------------
-------

% Make TimesAngleHit(1,:) be a percent occurance
if (debug3 || debug3b)
    TimesAngleHit(1,:) =
TimesAngleHit(1,:)/CurrentTimeStep;
end

if (TimeModel)
    TimeToRunSec = toc
end

if (SaveFinalData)
    save('FinalData')
end

fprintf('Run Complete.');
fprintf('\n END OF LINE\n');
%%% END OF LINE %%%
```

The following is code for MATLAB functions necessary to run CEM.

```
% CEM_AdjustShore.m
%
% CEM Function: Complete mass balance for incoming and
ougoing sediment
```

```
% NEW - AA 05/04 fully utilize shoreface depths

%radtodeg = 180/pi;          % Transform radians to degrees

if (VolumeIn(i) <= VolumeOut(i))
    % eroding, just have to use shoreface depth

    Depth = DepthShoreface;

else
    % accreting, oh my gosh!!!

    % where should we intersect shoreface depth ?

    % uncomplicated way - assume starting in middle of cell
    Distance = DepthShoreface/CellWidth/ShorefaceSlope;
    Xintfloat = X(i) + 0.5 + Distance *
cos(SurroundingAngle(i));
    Xintint = floor(Xintfloat);
    Yintfloat = Y(i) + 0.5 - Distance *
sin(SurroundingAngle(i));
    Yintint = floor(Yintfloat);

    if (debug7a)
        fprintf(['xs: ',num2str(X(i)),' ys:
',num2str(Y(i)),' Xint: ',...
            num2str(Xintfloat),' Xint: ',num2str(Xintint),'
Yint: ',...
            num2str(Yintfloat),' Yint: ',num2str(Yintint),'
Dint: ',...
            num2str(CellDepth(Yintint,Xintint)),' SAng:
',...
            num2str(SurroundingAngle(i)*radtodeg),' Sin =
',...
            num2str(SurroundingAngle(i)),'\n']);
    end

    if ((Yintint <= 1) || (Yintint > 2*Ymax))

        Depth = DepthShoreface;

        if ((Yintint > Ymax/2) && (Yintint < 3/2*Ymax))
            fprintf('Periodic Boundary conditions and Depth
Out of Bounds\n');
```

```
            %pause;
        end

    elseif ((Xintint <= 1) || (Xintint > Xmax))
        Depth = DepthShoreface;
        %fprintf(['-- Warning - depth location off of x
array: X ',num2str(Xintint),' Y ',num2str(Yintint),'\n']);
        %pause

    elseif (CellDepth(Yintint,Xintint) <= 0)
        %looking back on land

        Depth = DepthShoreface;
        if (debug7a)
            fprintf(['=== Shoreface is Shore, eh? Accreti:
xs: ',...
                num2str(X(i)),' ys: ',num2str(Y(i)),'
Xint::',num2str(Xintint),...
                ' Yint:',num2str(Yintint),' Dint:
',num2str(CellDepth(Yintint,Xintint)),'\n']);
        end

    elseif (CellDepth(Yintint,Xintint) < DepthShoreface)
        fprintf(['Shallow but underwater Depth
',num2str(CellDepth(Yintint,Xintint))]);
        %pause

    else

        Depth = CellDepth(Yintint,Xintint);

        % That was the easy part - now we need to 'fix' all
cells towards shoreface
        % probably due to accretion from previous moving
forward
        % reuse some of the overwash checking code here

        % calculate the slope
        if (SurroundingAngle(i) == 0)
            % unlikely, but make sure no div by zero
            slope = 0.00001;
        elseif (abs(SurroundingAngle(i)*radtodeg) == 90)
            slope = 9999.9;
        else
```

```
            slope = abs(tan(SurroundingAngle(i)));
        end

        % determine the ysign
        if (SurroundingAngle(i) > 0)
            ysign = 1;
        else
            ysign = -1;
        end

        % Initialize variable for while loop
        x = Xintfloat;
        y = Yintfloat;
        xtest = Xintint;
        ytest = Yintint;
        ShorefaceFlag = 0;

        while ( (CellDepth(ytest,xtest) > DepthShoreface)
&& (ShorefaceFlag == 0))

            NextXInt = ceil(x) -1;
            if (ysign > 0)
                NextYInt = floor(y) + 1;
            else
                NextYInt = ceil(y-1);
            end

            % moving to next whole 'x' position, what is y
position?
            Ydown = y + (x - NextXInt)*slope * ysign;
            DistanceDown = ((Ydown - y)*(Ydown - y) +
(NextXInt - x)*(NextXInt - x))^0.5;

            % moving to next whole 'y' position, what is x
position?
            Xside = x - abs(NextYInt - y) / slope;
            DistanceSide = ((NextYInt - y)*(NextYInt - y) +
(Xside - x)*(Xside - x))^0.5;

            if (DistanceDown < DistanceSide)
                % next cell is the down cell */

                x = NextXInt;
                y = Ydown;
```

```
                xtest = NextXInt-1;
                ytest = floor(y);
            else
                %  next cell is the side cell

                x = Xside;
                y = NextYInt;
                xtest = floor(x);
                ytest = y + (ysign-1)/2;
            end

            if (CellDepth(ytest,xtest) > DepthShoreface)
                % Deep hole - fill 'er in - mass came from
previous maths

                if (debug7a)
                    fprintf(['=== Deep Hole, eh? Accreti:
xs: ',...
                        num2str(X(i)),' ys:
',num2str(Y(i)),' Xint::',...
                        num2str(Xintint),'
Yint:',num2str(Yintint),...
                        ' Dint:
',num2str(CellDepth(Yintint,Xintint)),'\n']);
                end
                CellDepth(ytest,xtest) = DepthShoreface;
                %pause

            else
                % stop checking - ostensibly we have hit
the shoreface or shore

                ShorefaceFlag = 1;

                if (PercentFull(ytest,xtest) > 0)
                    % not good - somehow crossing the shore

                    fprintf('Shoreface is the Beach
!!!??\n');
                    %beep;
                    %pause;
                end

            end
```

```
        end % end while loop

    end

end

Depth = Depth + LandHeight;

if (Depth == 0)
    fprintf(['\nDepth went to zero!!! xs: ',
num2str(X(i)),' ys: ',num2str(Y(i))]);
    return;
end

if (Depth < DepthShoreface)
    fprintf('too deep ');
    %pause
end

DeltaArea = (VolumeIn(i) - VolumeOut(i))/Depth;

PercentFull(Y(i),X(i)) = PercentFull(Y(i),X(i)) +
DeltaArea/(CellWidth*CellWidth);

%fprintf(['(',num2str(X(i)),',',num2str(Y(i)),') DeltaArea:
',num2str(DeltaArea),' PercentFull:
',num2str(PercentFull(Y(i),X(i))),'\n']);

% Alert if it goes to infinity
if (isinf(PercentFull(Y(i),X(i))))
    beep
    fprintf(['PercentFull went to infinity!!!
x:',num2str(X(i)),' y:',num2str(Y(i))]);
    %PercentFull(Y(i),X(i)) = 0.5;
    pause;
end

if (debug7)
    PercentIn = VolumeIn(i)/(CellWidth*CellWidth*Depth);
    PercentOut = VolumeOut(i)/(CellWidth*CellWidth*Depth);
    PercentSum = DeltaArea/(CellWidth*CellWidth);
    fprintf(['   In: ',num2str(PercentIn),'   Out:
',num2str(PercentOut),...
```

```
        '    Sum: ',num2str(PercentSum),'\n']);
end


function [Age] =
CEM_AgeCells(Age,PercentFull,AgeMax,CurrentTimeStep,Xmax,Ym
ax)

% CEM Function: Ages the cells (returns matrix 'Age')

for y = 1:2*Ymax
    for x = 1:Xmax

        if (PercentFull(y,x) == 0)

            Age(y,x) = mod(CurrentTimeStep,AgeMax);

        end
    end
end


function [WvHeight,WvAngle] =
CEM_AWTRefractToBreak(WaveAngle,OffShoreWvHt,Period,MaxDept
h,KBreak,ShoreAngle,RefractStep,debug6)

% CEM Function: Refracts an offshore wave though the
Asthon-Murray Wave
% Transformation (AWT) from an offshore depth to a
specified depth.
% Waveheight and angle at this point are returned.

% Coefficients
g = 9.80665;                % Acceleration due to gravity
(m/s^2)

% Initialize Variables
counter = 0;
Broken = 0;
radtodeg = 180/pi;          % Transform radians to degrees
%degtorad = pi/180;          % Transform degrees to radians

% Use the Ashton-Murray Wave Transformation to determine
wave breaking data
```

```
% Primary assumption is that waves refract over shore-
parallel contours
% New algorithm 6/02 iteratively takes wave onshore until
they break, then computes Qs

% Initialize offshore wave conditions
StartDepth = MaxDepth;
AngleDeep = WaveAngle - ShoreAngle;
StartHeight = OffShoreWvHt;
Depth = StartDepth;

if (debug6)
    fprintf(['Wave Angle: ',num2str(WaveAngle*radtodeg),'
Shore Angle: ',...
        num2str(ShoreAngle*radtodeg),'   ']);
end


% Begin refraction
% Calculate Deep Water Celerity & Length, Komar 5.11 c = gT
/ pi, L = CT

CDeep = g * Period / (2.0 * pi);
LDeep = CDeep * Period;

if (debug6)
    fprintf(['CDeep = ',num2str(CDeep),' LDeep =
',num2str(LDeep),'\n']);
end

while (Broken == 0)

    % non-iterative eqn for L, from Fenton & McKee

    Wavelength = LDeep * (tanh(
(((2*pi/Period)^2)*(Depth/g))^(.75))) ^(2/3);
    C = Wavelength / Period;

    if (debug6)
        fprintf(['DEPTH: ',num2str(Depth),' Wavelength =
',...
            num2str(Wavelength),' C = ',num2str(C),' ']);
    end
```

```
    % Determine n = 1/2(1+2kh/tanh(kh)) Komar 5.21
    % First Calculate kh = 2 pi Depth/L  from k = 2 pi/L

    kh = pi*Depth/Wavelength;
    n = 0.5 * (1 + 2 * kh / sinh(2*kh));

    if (debug6)
        fprintf(['kh: ',num2str(kh),' n: ',num2str(n),'
']);
    end

    % Calculate angle, assuming shore parallel contours and
no conv/div
    % of rays from Komar 5.47

    Angle = asin(C/CDeep * sin(AngleDeep));

    if (debug6)
        fprintf(['Angle: ',num2str(Angle*radtodeg)]);
    end

    % Determine Wave height from refract calcs - Komar 5.49

    WvHeight = StartHeight * sqrt(abs(CDeep *
cos(AngleDeep) / (C * 2 * n * cos(Angle))));

    if (debug6)
        fprintf([' WvHeight: ',num2str(WvHeight),'\n']);
    end

    if (WvHeight > Depth*KBreak)
        Broken = 1;
        counter = 0;
        DepthBreak = Depth;
    elseif (Depth == RefractStep)
        Broken = 1;
        Depth = Depth - RefractStep;
        counter = 0;
        DepthBreak = Depth;
    else
        Depth = Depth - RefractStep;
        counter = counter + counter;
    end
```

```
end % end while wave not broken loop

% Return the angle relative to the y-axis (not the
shoreline)
WvAngle = Angle + ShoreAngle;


function [WvHeight,WvAngle] =
CEM_AWTRefractToDepth(WaveAngle,OffShoreWvHt,Period,MaxDept
h,DepthStop,ShoreAngle,RefractStep,debug6)

% CEM Function: Refracts an offshore wave though the
Asthon-Murray Wave
% Transformation (AWT) from an offshore depth to a
specified depth.
% Waveheight and angle at this point are returned.

% Coefficients
g = 9.80665;                % Acceleration due to gravity
(m/s^2)

% Initialize Variables
counter = 0;
Broken = 0;
radtodeg = 180/pi;          % Transform radians to degrees
%degtorad = pi/180;          % Transform degrees to radians

% Use the Ashton-Murray Wave Transformation to determine
wave breaking data
% Primary assumption is that waves refract over shore-
parallel contours
% New algorithm 6/02 iteratively takes wave onshore until
they break, then computes Qs

% Initialize offshore wave conditions
StartDepth = MaxDepth;
AngleDeep = WaveAngle - ShoreAngle;
StartHeight = OffShoreWvHt;
Depth = StartDepth;

if (debug6)
    fprintf(['Wave Angle: ',num2str(WaveAngle*radtodeg),'
Shore Angle: ',...
        num2str(ShoreAngle*radtodeg),'   ']);
```

```
end


% Begin refraction
% Calculate Deep Water Celerity & Length, Komar 5.11 c = gT
/ pi, L = CT

CDeep = g * Period / (2.0 * pi);
LDeep = CDeep * Period;

if (debug6)
    fprintf(['CDeep = ',num2str(CDeep),' LDeep =
',num2str(LDeep),'\n']);
end

while (Broken == 0)

    % non-iterative eqn for L, from Fenton & McKee

    Wavelength = LDeep * (tanh(
(((2*pi/Period)^2)*(Depth/g))^(.75))) ^(2/3);
    C = Wavelength / Period;

    if (debug6)
        fprintf(['DEPTH: ',num2str(Depth),' Wavelength =
',...
            num2str(Wavelength),' C = ',num2str(C),' ']);
    end

    % Determine n = 1/2(1+2kh/tanh(kh)) Komar 5.21
    % First Calculate kh = 2 pi Depth/L  from k = 2 pi/L

    kh = pi*Depth/Wavelength;
    n = 0.5 * (1 + 2 * kh / sinh(2*kh));

    if (debug6)
        fprintf(['kh: ',num2str(kh),' n: ',num2str(n),'
']);
    end

    % Calculate angle, assuming shore parallel contours and
no conv/div
    % of rays from Komar 5.47
```

```
    Angle = asin(C/CDeep * sin(AngleDeep));

    if (debug6)
        fprintf(['Angle: ',num2str(Angle*radtodeg)]);
    end

    % Determine Wave height from refract calcs - Komar 5.49

    WvHeight = StartHeight * sqrt(abs(CDeep *
cos(AngleDeep) / (C * 2 * n * cos(Angle))));

    if (debug6)
        fprintf([' WvHeight: ',num2str(WvHeight),'\n']);
    end

    if (Depth <= DepthStop)
        Broken = 1;
        counter = 0;
    elseif (Depth == RefractStep)
        Broken = 1;
        Depth = Depth - RefractStep;
        counter = 0;
    else
        Depth = Depth - RefractStep;
        counter = counter + counter;
    end

end % end while wave not broken loop

% Return the angle relative to the y-axis (not the
shoreline)
WvAngle = Angle + ShoreAngle;



% CEM_CalcGroinBypassParams.m

% CEM Function: determines the bypassing parameters for
groins.
% Specifically, the water depth at the groin tip is
calcualted.

% Constants and Parameters
debugloc = 0;               % local debugger
```

```
% Initialzie
D_G = zeros(1,TotalBeachCells); % water depth at groin tip


% Find D_G for all groins

for GroinNumber = 1:length(GroinTipY)

    % Loop for all beach cells
    for i = 1:TotalBeachCells

        if (Y(i) == GroinTipY(GroinNumber))
            % There is a groin on the right side of the
cell at Y(i)

            % Determine the cross-shore distance between
shoreline and groin tip
            x_SLPos = X(i) + PercentFull(Y(i),X(i));   %
cross-shore position of shoreline (cells)
            x_GrTipPos = GroinTipX(GroinNumber)+0.5;    %
cross-shore position of groin tip (cells)

            if (x_SLPos >= x_GrTipPos)
                % the shoreline is past (or up to) groin
tip
                D_G(i) = 0;

                if (debugloc)
                    fprintf('X:%G Y:%G ',X(i),Y(i));
                    fprintf(['SL pos up to groin tip: D_G =
',num2str(D_G(i)),'\n']);
                    fprintf('X = %G, PF = %G, x_SLPos = %G,
x_GrTipPos =
%G\n',X(i),PercentFull(Y(i),X(i)),x_SLPos,x_GrTipPos);
                    beep
                    pause
                end

            else
                % Calculate the cross-shore distance to
calculate BYP
```

```
                CSDist = (x_GrTipPos - x_SLPos) *
CellWidth;    % Cross-shore distance between shoreline and
groin tip (m)

                % Calculate the water depth at the groin
tip (from Dean
                % profile)
                D_G(i) = DeanProfileA * (CSDist ^ (2/3));

                if (debugloc)
                    fprintf('X:%G Y:%G ',X(i),Y(i));
                    fprintf(['CSDist: ',num2str(CSDist),'
D_G: ',num2str(D_G(i)),'\n']);
                end

            end

        elseif (Y(i) == GroinTipY(GroinNumber)+1)
            % There is a groin on the left side of the cell
at Y(i)


            % Determine the cross-shore distance between
shoreline and groin tip
            x_SLPos = X(i) + PercentFull(Y(i),X(i));   %
cross-shore position of shoreline (cells)
            x_GrTipPos = GroinTipX(GroinNumber)+0.5;    %
cross-shore position of groin tip (cells)

            if (x_SLPos >= x_GrTipPos)
                % the shoreline is past (or up to) groin
tip
                D_G(i) = 0;

                if (debugloc)
                    fprintf('X:%G Y:%G ',X(i),Y(i));
                    fprintf(['SL pos up to groin tip: D_G =
',num2str(D_G(i)),'\n']);
                    fprintf('X = %G, PF = %G, x_SLPos = %G,
x_GrTipPos =
%G\n',X(i),PercentFull(Y(i),X(i)),x_SLPos,x_GrTipPos);
                    beep
                    pause
                end
```

```
            else
                % Calculate the cross-shore distance to
calculate BYP

                CSDist = (x_GrTipPos - x_SLPos) *
CellWidth;    % Cross-shore distance between shoreline and
groin tip (m)

                % Calculate the water depth at the groin
tip (from Dean
                % profile)
                D_G(i) = DeanProfileA * (CSDist ^ (2/3));

                if (debugloc)
                    fprintf('X:%G Y:%G ',X(i),Y(i));
                    fprintf(['CSDist: ',num2str(CSDist),'
D_G: ',num2str(D_G(i)),'\n']);
                end

            end

        end

    end

end


% CEM_CalcVolInOut.m
%
% CEM Function: Calculates the cumulative and overall mean
sediment volumes
% transported in and out of every cell

% Calculate the cumulative volume in and out of every beach
element
for i = 1:length(Y)

    ypos = Y(i);

    CumVolumeIn(ypos) = CumVolumeIn(ypos) + VolumeIn(i);
    CumVolumeOut(ypos) = CumVolumeOut(ypos) + VolumeOut(i);
```

```
end



% Calculate the mean volume in to and out of every beach
element
MeanVolumeIn = CumVolumeIn/(CurrentTimeStep -
StartingTimeStep);
MeanVolumeOut = CumVolumeIn/(CurrentTimeStep -
StartingTimeStep);


function [PercentFull, AllBeach, CellDepth, OWFlag] =
CEM_CheckOverwash(icheck,AllBeach,PercentFull,CellDepth,X,Y
,...

SurroundingAngle,CritBWidth,CellWidth,DepthShoreface,LandHe
ight,TotalBeachCells,OWFlag,OWType,Ymax,debug8,debug10a,deb
ug10b)

% CEM Function: Step back pixelwise in direction of
Surrounding Angle to
% check needage
% If too short, calls DoOverwash, which will move some
sediment
% Need to change sweepsign because filling cells should
affect neighbors
% 'x' and 'y' hold real-space values, will be mapped onto
ineger array

if (SurroundingAngle(icheck) == 0)
    % unlikely, but make sure no div by zero
    slope = 0.00001;
    %abs(tan((SurroundingAngle(icheck+1) +
SurroundingAngle(icheck-1))/2));

elseif (abs(SurroundingAngle(icheck)) == 90)
    slope = 9999.9;

else
    slope = abs(tan(SurroundingAngle(icheck)));
end
```

```
if (SurroundingAngle(icheck) > 0)
    ysign = 1;
else
    ysign = -1;
end

if (debug10a)
    fprintf(['\nI: ',num2str(icheck),'------------- Surr:
',...
        num2str(SurroundingAngle(icheck)),' ',...
        num2str(SurroundingAngle(icheck)*radtodeg),' Slope:
',...
        num2str(slope),' sign: ',num2str(ysign),'\n']);
end

if ((AllBeach(Y(icheck),X(icheck)-1) == 'y') || ...
        ((AllBeach(Y(icheck)-1,X(icheck)) == 'y') && ...
        (AllBeach(Y(icheck)+1,X(icheck)) == 'y')))
    % 'regular condition'
    % plus 'stuck in the middle' situation (unlikely
scenario)

    xin = X(icheck) + PercentFull(Y(icheck),X(icheck));
    yin = Y(icheck) + 0.5;

elseif (AllBeach(Y(icheck)-1,X(icheck)) == 'y')
    % on right side

    xin = X(icheck) + 0.5;
    yin = Y(icheck) + PercentFull(Y(icheck),X(icheck));

    if (debug10a)
        fprintf(['--Right xin: ',num2str(xin),' yin:
',num2str(yin),'\n']);
    end

elseif (AllBeach(Y(icheck)+1,X(icheck)) == 'y')
    % on left side

    xin = X(icheck) + 0.5;
    yin = Y(icheck) + 1 - PercentFull(Y(icheck),X(icheck));

    if (debug10a)
```

```
        fprintf(['--Left xin: ',num2str(xin),' yin:
',num2str(yin),'\n']);
    end
else
    % underneath, no overwash

    return

end

% initialize while loop
x = xin;
y = yin;
checkdistance = 0;
AllBeachFlag = 0;

while ((checkdistance < CritBWidth) && (y > 0) && (y <
2*Ymax) && (x > 1))

    NextXInt = ceil(x) -1;

    if (ysign > 0)
        NextYInt = floor(y) + 1;
    else
        NextYInt = ceil(y-1);
    end

    % moving to next whole 'x' position, what is y
position?
    Ydown = y + (x - NextXInt)*slope * ysign;
    DistanceDown = ((Ydown - y)*(Ydown - y) + (NextXInt -
x)*(NextXInt - x))^(0.5);

    % moving to next whole 'y' position, what is x
position?

    Xside = x - abs(NextYInt - y) / slope;
    DistanceSide = ((NextYInt - y)*(NextYInt - y) + (Xside
- x)*(Xside - x))^(0.5);

    if (debug10a)
        fprintf(['x: ',num2str(x),' y: ',num2str(y),' X:
',num2str(NextXInt),...
```

```
            ' Y: ',num2str(NextYInt),'Yd:
',num2str(Ydown),' DistD: ',...
            num2str(DistanceDown),' Xs: ',num2str(Xside),'
DistS: ',...
            num2str(DistanceSide),'\n']);
    end

    if (DistanceDown < DistanceSide)
        % next cell is the down cell

        x = NextXInt;
        y = Ydown;
        xtest = NextXInt-1;
        ytest = floor(y);

    else
        % next cell is the side cell

        x = Xside;
        y = NextYInt;
        xtest = floor(x);
        ytest = y + (ysign-1)/2;

    end

    % make sure ytest doesn't do to 0
    if (ytest == 0)
        ytest = 2*Ymax;
    end

    checkdistance = CellWidth * (((x - xin)*(x - xin) +  (y
- yin)*(y - yin))^.5);

    if (AllBeach(ytest,xtest) == 'y')
        AllBeachFlag = 1;
    end

    if (debug10a)
        fpritnf(['  x: ',num2str(x),' y: ',num2str(y),'
xtest: ',...
            num2str(xtest),' ytest: ',num2str(ytest),'
check: ',...
            num2str(checkdistance),'\n\n']);
    end
```

```
    if ((AllBeach(ytest,xtest) == 'n') && (AllBeachFlag) &&
...
            (xtest < X(icheck)) && ((X(icheck)-xtest > 1)
|| ...
            (abs(test - Y(icheck)) > 1)))
        % Looking for shore cells, but don't want immediate
neighbors, and go backwards
        % Also mush pass though an allbeach cell along the
way

        if (AllBeach(ytest,xtest+1) == 'y')
            % 'regular condition' - UNDERNEATH, here

            xint = (xtest + 1 - PercentFull(ytest,xtest));
            yint = yin + (xin - xint)*ysign*slope;

            if ((yint > ytest + 1) || (yint < ytest))
                %  This cell isn't actually an overwash
cell

                measwidth = CritBWidth;

                if (debug10a)
                    fprintf(['-- Regunder Cancelled  xin:
',num2str(xin),...
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end

            else

                measwidth = CellWidth * ((xint - xin)*(xint
- xin) + (yint - yin)*(yint - yin)) ^ (0.5);

                 if (debug10a)
                    fprintf(['-- Regunder Over  xin:
',num2str(xin),...
```

```
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end

            end

        elseif (AllBeach(ytest-1,xtest) == 'y')
            % on right side

            yint = ytest + PercentFull(ytest,xtest);
            xint = xin - abs(yin - yint)/ slope;


            if (xint < xtest)
                % This cell isn't actually an overwash cell

                measwidth = CritBWidth;

                if (debug10a)
                    fprintf(['-- Right Cancelled  xin:
',num2str(xin),...
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end

            else

                measwidth = CellWidth * ((xint - xin)*(xint
- xin)+ (yint - yin)*(yint - yin))^(0.5);

                if (debug10a)
                    fprintf(['-- Right Over  xin:
',num2str(xin),...
```

```
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end

            end

        elseif (AllBeach(ytest+1,xtest) == 'y')
            % on left side

            yint = (ytest + 1 - PercentFull(ytest,xtest));
            xint = xin - abs(yin - yint)/slope;

            if (xint < xtest)
                %  This cell isn't actually an overwash
cell

                measwidth = CritBWidth;

                if (debug10a)
                    fprintf(['-- Left Cancelled  xin:
',num2str(xin),...
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end

            else

                measwidth = CellWidth * ((xint - xin)*(xint
- xin)+ (yint - yin)*(yint - yin)) ^0.5;

                if (debug10a)
                    fprintf(['-- Left Over  xin:
',num2str(xin),...
```

```
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end

            end

        elseif (AllBeach(ytest,xtest-1) == 'y')
            % 'regular condition'
            % plus 'stuck in the middle' situation

            xint = xtest + PercentFull(ytest,xtest);
            yint = yin + (xin - xint)*ysign * slope;

            if ((yint > ytest+1) || (yint < ytest))
                % This cell isn't actually an overwash cell

                measwidth = CritBWidth;

                if (debug10a)
                    fprintf(['-- RegularODD Cancelled  xin:
',num2str(xin),...
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end

            else

                measwidth = CellWidth * ((xint - xin)*(xint
- xin)+ (yint - yin)*(yint - yin))^0.5;

                if (debug10a)
                    fprintf(['-- RegularODD Over  xin:
',num2str(xin),...
```

```
                        ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                        num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                        num2str(yint),' sl:
',num2str(slope),' MMeas: ',...
                        num2str(measwidth),'\n']);
                end
            end

        elseif (PercentFull(ytest,xtest) > 0)
            % uh oh - not good situation, no allbeach on
sides
            % assume this is an empty cell

            xint = x;
            yint = y;

            measwidth = CellWidth * ((xint - xin)*(xint -
xin)+ (yint - yin)*(yint - yin))^(0.5);

            if (debug10a)
                fprintf(['-- Some Odd Over  xin:
',num2str(xin),...
                    ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                    num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                    num2str(yint),' sl: ',num2str(slope),'
MMeas: ',...
                    num2str(measwidth),'\n']);
            end

        else
            % empty cell - oughta fill er up  - fill max
barrier width

            xint = x;
            yint = y;
            measwidth = CritBWidth - CellWidth;


            if (debug10a)
```

```
                fprintf(['-- Empty Odd Over  xin:
',num2str(xin),...
                    ' yin: ',num2str(yin),' xt:
',num2st(xtest),' yt: ',...
                    num2str(ytest),' xint:
',num2str(xint),' yint: ',...
                    num2str(yint),' sl: ',num2str(slope),'
MMeas: ',...
                    num2str(measwidth),'\n']);
            end

        end

        checkdistance = measwidth;

        if (measwidth < CritBWidth)

            % DoOverwash
            [PercentFull, AllBeach, CellDepth] =
CEM_DoOverwash(X(icheck),Y(icheck),xtest,ytest,xint,yint,me
aswidth,icheck,...

PercentFull,AllBeach,X,Y,CellDepth,DepthShoreface,LandHeigh
t,SurroundingAngle,TotalBeachCells,OWType,debug8,debug10b);

            OWFlag = 1;
            return;
        end

    end

end % end while loop


function [PercentFull, AllBeach, CellDepth, OWFlag] =
CEM_CheckOverwashSweep(AllBeach,PercentFull,CellDepth,X,Y,.
..

SurroundingAngle,InShadow,OverwashLimit,TotalBeachCells,Cri
tBWidth,CellWidth,DepthShoreface,LandHeight,OWFlag,OWType,Y
max,debug8,debug10a,debug10b)

% CEM Function: Just a loop to call overwash check
founction CheckOverwash
```

```
% Nothing done here, but can be down when CheckOVerwash is
called

% Initialize
radtodeg = 180/pi;          % Transform radians to degrees
RandNum = rand(1);

if (RandNum*2 > 1)
    sweepsign = 1;

    if (debug10a)
        fprintf('L  ');
    end
else
    sweepsign =  0;

    if (debug10a)
        fprintf('R  ');
    end
end

for i = 3:TotalBeachCells - 2

    if (sweepsign == 1)
        ii = i;
    else
        ii = TotalBeachCells - 1;
    end

    if (( abs(SurroundingAngle(ii)) <
OverwashLimit/radtodeg) && (InShadow(ii) == 'n'))

        % CheckOverwash
        [PercentFull, AllBeach, CellDepth, OWFlag] =
CEM_CheckOverwash(ii,AllBeach,PercentFull,CellDepth,X,Y,...

SurroundingAngle,CritBWidth,CellWidth,DepthShoreface,LandHe
ight,TotalBeachCells,OWFlag,OWType,Ymax,debug8,debug10a,deb
ug10b);


    end
end
```

```
function [PercentFull] =
CEM_ConstantSLRAdjust(PercentFull,RateSLR_Const,TimeStep,Sh
orefaceSlope,CellWidth,Xmax,Ymax)

% CEM function: accounts for shoreline retreat from a
constant sea level
% rise.

loc_debug = 0;  % local debugger

% Find change in water level for every time step
%DeltaEta = RateSLR_Const * (TimeStep / (365*100)); %
change in SL per timestep (m)
DeltaEta = RateSLR_Const * (TimeStep / (CellWidth * 100 *
365)); % change in SL per timestep (cells)

% Find the change in PercentFull (F) every timestep due to
SLR
DeltaF = DeltaEta / ShorefaceSlope;

% Loop for all PercentFull to determine beach locations
for y = 1:2*Ymax
    for x = 1:Xmax

        if ((PercentFull(y,x) < 1) && (PercentFull(y,x) >
0))
            % We have found a beach cell

            if (loc_debug)
                fprintf('Adjusting PercentFull at (%i,%i)
by %d\n',y,x,DeltaF);
                Old = PercentFull(y,x);
            end

            % Adjust the PercentFull due to the change in
sea level
            %PercentFull(y,x) = ((CellWidth *
PercentFull(y,x)) - (ShorefaceSlope*DeltaEta)) / CellWidth;
            PercentFull(y,x) = PercentFull(y,x) - DeltaF;

            if (loc_debug)
                New = PercentFull(y,x);
                ChangeInPF(y) = New - Old;
```

```
                %fprintf(['Delta PF:
',num2str(ChangeInPF(y)),'\n']);
            end

        end
    end
end


function [GroinPos,GroinX,GroinY,GroinXPlot,GroinYPlot] =
CEM_CreateGroins(Xmax,Ymax,GroinData,GroinStart,InitBeach,P
eriodicCopyGroins)

% CEM Function: Creates the Gron Position Arrays.
%
% Creates the following variables:
%   GroinPos - shows groin positions
%       (=1 if groin on right side of cell)
%   GroinX - x-coordinates of the cells that contain groins
%   GroinY - y-coordinates of the cells that contain groins
%   GroinX - x-coordinates of groins (for plotting only)
%   GroinY - y-coordinates of groins (for plotting only)


% Initialize
debug = 0;  % local debugger

% Make the input groin data matrix
GroinInputData = GroinData;

% Make the groin position matrix
GroinPos = zeros(2*Ymax,Xmax);  % Matrix

for i = 1:length(GroinInputData(:,1))
    %GroinPos(GroinInputData(:,1)+(Ymax/2)+1,GroinStart) =
1;

GroinPos(GroinInputData(i,1)+(Ymax/2),GroinStart:GroinStart
+GroinInputData(i,2)) = 1;
end

% Find the x & y coordinates of groin positions (for
plotting)
[GroinXPlot, GroinYPlot] = find(GroinPos==1);
```

```
GroinXPlot = GroinXPlot - (Ymax/2) + 0.5;
GroinYPlot = GroinYPlot - InitBeach - 1;

if (debug)
    figure;
    scatter(GroinXPlot,GroinYPlot,'s','r');
end


% Periodic Boundary Copy
if (PeriodicCopyGroins)
    GroinPos(1:Ymax/2,:) = GroinPos(Ymax+1:3*Ymax/2,:);
    GroinPos((3*Ymax/2)+1:2*Ymax,:) =
GroinPos((Ymax/2)+1:Ymax,:);
end

if (debug)
    figure;
    imagesc(GroinPos);
end

% Find the x & y coordinates of groin positions
[GroinY, GroinX] = find(GroinPos==1);

%
% if (debug)
%     figure;
%     scatter(GroinX,GroinY,'x','g');
% end


function [ShorelineAngle,UpWind,SurroundingAngle] =
CEM_DetermineAngles(X,Y,PercentFull,AllBeach,WaveAngle,InSh
adow,TotalBeachCells,ShorelineAngle,SurroundingAngle,Ymax,d
ebug3,debug4)

% CEM Function: Function to determine beach angles for all
beach cells from
% left to right.  By convention, the ShorelineAngle will
apply to current
% cell and right neighbor.

% pause
```

```
radtodeg = 180/pi;          % Transform radians to degrees
degtorad = pi/180;          % Transform degrees to radians
debug3a = 0;                % local debugger

% Shoreline Angle Calcs - ADA 05/04 - use correct 'point'
to do calcs (like in shaddow
% Set first point
% first angle should be regular one - periodic BC's should
also take care

x2 = X(1) + PercentFull(Y(1),X(1));
y2 = Y(1) + 0.5;




%% Compute ShorelineAngle
% not equal to TotalBeachCells because angle between cell
and rt neighbor

for i = 1:TotalBeachCells

%     if (i == TotalBeachCells)
%         pause;
%     end

    % Account for periodic Boundary Conditions
    if (i == 1)
        iplus = i+1;
    elseif (i == TotalBeachCells)
        iplus = 1;
    else
        iplus = i+1;
    end

    x1 = x2;
    y1 = y2;

    x2int = X(iplus);
    y2int = Y(iplus);

    % Account for periodic Boundary Conditions
    if (y2int == 2*Ymax)
        y2intPlus = 1;
```

```
    else
        y2intPlus = y2int+1;
    end
    if (y2int == 1)
        y2intMinus = 2*Ymax;
    else
        y2intMinus = y2int-1;
    end

    if (AllBeach(y2int,x2int-1) == 'y' ||
((AllBeach(y2intMinus,x2int) == 'y')...
            && (AllBeach(y2intPlus,x2int) == 'y')) &&
(AllBeach(y2int,x2int+1) == 'n'))
        % 'regular condition' - if between
        % plus 'stuck in the middle' situation (unlikely
scenario)

        x2 = x2int + PercentFull(y2int,x2int);
        y2 = y2int + 0.5;
        if (debug3a)
            fprintf(['-- Regular xin: ',num2str(x2),' yin:
',num2str(y2),'\n']);
        end

    elseif ((AllBeach(y2int,x2int+1) == 'y') &&
(AllBeach(y2int,x2int-1) == 'y'))
        % in a sideways nook (or is that a cranny?)

        x2 = x2int + 0.5;

        if (AllBeach(y2intMinus,x2int) == 'y')
            % right-facing nook

            y2 = y2int + PercentFull(y2int,x2int);

        else
            % left-facing nook

            y2 = y2int + 1.0 - PercentFull(y2int,x2int);

        end

        if (debug3a)
```

```
            fprintf(['-- Nook xin: ',num2str(x2),' yin:
',num2str(y2),'\n']);
        end


    elseif (AllBeach(y2intMinus,x2int) == 'y')
        % on right side

        x2 = x2int + 0.5;
        y2 = y2int + PercentFull(y2int,x2int);

        if (debug3a)
            fprintf(['-- Right xin: ',num2str(x2),' yin:
',num2str(y2),'\n']);
        end

    elseif (AllBeach(y2intPlus,x2int) == 'y')
        % on left side

        x2 = x2int + 0.5;
        y2 = y2int + 1.0 - PercentFull(y2int,x2int);
        if (debug3a)
            fprintf(['-- Left xin: ',num2str(x2),' yin:
',num2str(y2),'\n']);
        end


    elseif (AllBeach(y2int,x2int+1) == 'y')
        % gotta be on the bottom now

        x2 = x2int + 1 - PercentFull(y2int,x2int);
        y2 = y2int + 0.5;
         if (debug3a)
            fprintf(['-- Under xin: ',num2str(x2),' yin:
',num2str(y2),'\n']);
         end

    else
        % debug ain't just an insect

        fprintf('Determine Angles broke!!!! \n');
        return;
    end
```

```
    %  compute angles

     % Account for periodic Boundary Conditions
    if (y1 > 2*Ymax)

        y1 = y1 - 2*Ymax;
        ShorelineAngle(i) =  atan((x2 - x1) / abs(y2 -
y1));

        if (debug3)
            fprintf(['Rt.boundary (R) i = ',num2str(i),'
X(i): ',num2str(X(i)),' Y(i): ',...
                num2str(Y(i)),' PercentFull:
',num2str(PercentFull(Y(i),X(i))),...
                ' x: ',num2str(x2),' y: ',num2str(y2),'
SLAng: ',...
                num2str(ShorelineAngle(i)*180/pi),'\n']);
        end

    elseif (y2 > y1)

        ShorelineAngle(i) =  atan((x2 - x1) / (y2 - y1));

        if (debug3)
            fprintf(['(R) i = ',num2str(i),' X(i):
',num2str(X(i)),' Y(i): ',...
                num2str(Y(i)),' PercentFull:
',num2str(PercentFull(Y(i),X(i))),...
                ' x: ',num2str(x2),' y: ',num2str(y2),'
SLAng: ',...
                num2str(ShorelineAngle(i)*180/pi),'\n']);
        end

    elseif (y2 == y1)

        ShorelineAngle(i) = pi/2.0 * (x1 - x2) / abs(x2 -
x1);

        if (debug3)
            fprintf(['(G) i = ',num2str(i),' X(i):
',num2str(X(i)),' Y(i): ',...
                num2str(Y(i)),' PercentFull:
',num2str(PercentFull(Y(i),X(i))),...
```

```
                ' x: ',num2str(x2),' y: ',num2str(y2),'
SLAng: ',...
                num2str(ShorelineAngle(i)*180/pi),'\n']);
        end

    else
        % y2 < y1

        ShorelineAngle(i) =  atan((x2 - x1) / (y2 - y1)) -
pi;

        if (ShorelineAngle(i) < -pi)
            ShorelineAngle(i) = ShorelineAngle(i) + 2*pi;
        end

        if (debug3)
            fprintf(['(U) i = ',num2str(i),' X(i):
',num2str(X(i)),' Y(i): ',...
                num2str(Y(i)),' PercentFull:
',num2str(PercentFull(Y(i),X(i))),...
                ' x: ',num2str(x2),' y: ',num2str(y2),'
SLAng: ',...
                num2str(ShorelineAngle(i)*180/pi),'\n']);
        end

    end

end

%% loop through to make sure that shoreline angle does not
reach 90 deg
for iii = 1:TotalBeachCells

    if (iii == TotalBeachCells)
        iiiPlus1 = 1;
    else
        iiiPlus1 = iii+1;
    end
    if (iii == 1)
        iiiMinus1 = TotalBeachCells;
    else
        iiiMinus1 = iii-1;
    end
```

```
    if ((ShorelineAngle(iii) >= 89*(pi/180)) ||
(ShorelineAngle(iii) <= -89*(pi/180)))
        ShorelineAngle(iii) =
mean(ShorelineAngle(iiiMinus1:iiiPlus1));
    end
end



%% compute SurroundingAngle array
% 02/04 AA averaging doesn't work on bottom of spits
% Use trick that x is less if on bottom of spit - angles
might be different signs as well

for k = 1:TotalBeachCells

    if (k == TotalBeachCells)
        kPlus1 = 1;
    else
        kPlus1 = k+1;
    end
    if (k == 1)
        kMinus1 = TotalBeachCells;
    else
        kMinus1 = k-1;
    end


    if ((Y(kMinus1) - Y(kPlus1) == 2) &&
(ShorelineAngle(kMinus1)*sign(ShorelineAngle(k)) ~=
ShorelineAngle(kMinus1)))

        SurroundingAngle(k) = (ShorelineAngle(kMinus1) +
ShorelineAngle(k)) / 2 + pi;

        if (SurroundingAngle(k) > pi)
            SurroundingAngle(k) = SurroundingAngle(k) -
2*pi;
        end

        if (debug4)
            fprintf(['Under: ',num2str(k),'\n']);
        end
```

```
    else
        SurroundingAngle(k) = (ShorelineAngle(kMinus1) +
ShorelineAngle(k))/2;
    end

end

%% Determine Upwind/downwind condition
% Note - Surrounding angle is based upon left and right
cell neighbors,
% and is centered on cell, not on right boundary

if (debug4)
    fprintf(['\nUp/Down    Wave Angle:
',num2str(WaveAngle*radtodeg),'\n']);
end

for j = 1:TotalBeachCells

    if (debug4)
        fprintf(['j: ',num2str(j),' Shad: ',InShadow(j),'
Ang: ',...
            num2str(ShorelineAngle(j)*radtodeg),' SurAng:
',...
            num2str(SurroundingAngle(j)*radtodeg),' Effect:
',...
            num2str((WaveAngle -
SurroundingAngle(j))*radtodeg),'\n']);
    end

    if (abs(WaveAngle - SurroundingAngle(j)) >=
42.0*degtorad)

        UpWind(j) = 'u';

        if (debug4)
            fprintf('U(1)    ');
        end
    else

        UpWind(j) = 'd';
        if (debug4)
            fprintf('D(1)    ');
```

```
        end

    end

    if (debug4)
        fprintf('\n');
    end

end

%% Get rid of any NaN's

% ShorelineAngle(isnan(ShorelineAngle)) = 0;
% SurroundingAngle(isnan(SurroundingAngle)) = 0;


% CEM_DetermineMeanWvDir.m
%
% CEM-MATLAB Function: determines the mean wave directions
(left and
% rightward approaching) for diffraction calculations from
Kamphuis (2000).

% Initialize function
radtodeg = 180/pi;          % Transform radians to degrees
degtorad = pi/180;          % Transform degrees to radians
MWD_GroinTip = 0;     % Mean wave direction at groin tip
(radians)



% Determine the mean offshore wave angle
% Positive and negative (Asymmetry) not taken into account
if ((UseSingleWaveAngle == 0) && (WaveIn == 0))
    % Use A & U (4-bin PDF)
    % This should be determined only from highness

    PosMWD_0_Deg = (Highness + 0.5) * 45;    % Mean wave
direction offshore in degrees
    NegMWD_0_Deg = -PosMWD_0_Deg;


elseif ((UseSingleWaveAngle == 1) && (WaveIn == 0))
    % Use the single wave angle
```

```
    PosMWD_0_Deg = SingleOffshoreAngle;
    NegMWD_0_Deg = -PosMWD_0_Deg;

elseif ((UseSingleWaveAngle == 0) && (WaveIn == 1))
    % Use the input wave PDF

    NumBins = length(WavePDF(:,1));
    counter = 0;

    for i = 1:NumBins
        if (WavePDF(i,1) <=0)
            % Wave angle is negative
            NegPDF(i,1) = WavePDF(i,1);
            NegPDF(i,2) = WavePDF(i,2);
            counter = counter + 1;
        elseif (WavePDF(i,1) > 0)
            % wave angle is positive
            PosPDF(i - counter,1) = WavePDF(i,1);
            PosPDF(i - counter,2) = WavePDF(i,2);
        else
            % something is wrong!
            beep
            fprintf('CEM_DetermineMeanWvDir broke!!!! \n')
            pause
        end
    end

    clear i

    % Find the positive offshore MWD
    PosMWD_0_DegSum = 0;
    for ii = 1:length(PosPDF(:,1))
       PosMWD_0_DegSum = PosMWD_0_DegSum + ((PosPDF(ii,1) -
(BinSize/2)) * PosPDF(ii,2));
    end
    PosMWD_0_Deg = PosMWD_0_DegSum / sum(PosPDF(:,2));

    clear ii

    % Find the negative offshore MWD
    NegMWD_0_DegSum = 0;
    for ii = 1:length(PosPDF(:,1))
       NegMWD_0_DegSum = NegMWD_0_DegSum + ((NegPDF(ii,1) -
(BinSize/2)) * NegPDF(ii,2));
```

```
    end
    NegMWD_0_Deg = NegMWD_0_DegSum / sum(NegPDF(:,2));

else
    % something is wrong!
    beep
    fprintf('CEM_DetermineMeanWvDir broke!!!! \n')
    pause
end

PosMWD_0 = PosMWD_0_Deg*degtorad;           % Mean positive
wave direction offshore in radians
NegMWD_0 = NegMWD_0_Deg*degtorad;           % Mean negative
wave direction offshore in radians


% CEM_DetermineSedTransport.m

% CEM function: Loop function to determine which
neigbor/situation to use
% for sediment transport calcs.  Once situation is
determined, will use
% function SedTrans to determine actual transport.

% Initialize function
VolumeIn = zeros(1,TotalBeachCells);     % Sediment volume
into ith beach element
VolumeOut = zeros(1,TotalBeachCells);    % Sediment volume
ou of ith beach element
GroinTransRight = 0;
GroinTransLeft = 0;
GroinTipDepth = 0;

debug5a = 0;        % Local Debugger (periodic boundary
conditions)



if (debug5)
    fprintf(['\nSEDTRANS: ',num2str(CurrentTimeStep),' @
',...
        num2str(WaveAngle*radtodeg),'\n\n']);
end
```

```
for i = 1:TotalBeachCells

    GroinTransport = 0;          % Is transport occuring
over a terminal groin? (assume no at first)

   if (debug5)
       fprintf(['\n   i: ',num2str(i),'   ']);
   end

   %y_coord = Y(i);     % WRT y-coordinate for the ith cell
(if using WRT)

   MaxTrans = 'n';

   % Is littoral transport going left or right?

   if ((WaveAngle - ShorelineAngle(i)) > 0)
       % Transport going right, center on cell to left side
of border
       % Next cell in positive direction, no correction
term needed
       CalcCell = i;
       Next = 1;
       Last = -1;
       Correction = 0;
       %TransRight = 1;

       if (debug5)
           fprintf(['RT   ',num2str(CalcCell),' ']);
       end
   else
       % Transport going left, center on cell to right side
of border
       % Next cell in negative direction, correction term
needed
       CalcCell = i+1;
       Next = -1;
       Last = 1;
       Correction = -1;
       %TransRight = 0;

       if (debug5)
           fprintf(['LT   ',num2str(CalcCell),' ']);
       end
```

```
    end

    % Acccount for periodic boundary conditions
    if (CalcCell+Next == TotalBeachCells + 1)
        CalcCellPlusNext = 1;

        if (debug5a)
            fprintf('CalcCell: %i, CalcCellPlusNext: %i
\n',CalcCell,CalcCellPlusNext);
            beep
            pause
        end

    elseif (CalcCell+Next == 0)
        CalcCellPlusNext = TotalBeachCells;

        if (debug5a)
            fprintf('CalcCell: %i, CalcCellPlusNext: %i
\n',CalcCell,CalcCellPlusNext);
            beep
            pause
        end

    else
        CalcCellPlusNext = CalcCell+Next;
    end

    if (CalcCell+Last > TotalBeachCells)
        CalcCellPlusLast = CalcCell+Last-TotalBeachCells;

        if (debug5a)
            fprintf('CalcCell: %i, CalcCellPlusLast: %i
\n',CalcCell,CalcCellPlusLast);
            beep
            pause
        end

    elseif (CalcCell+Last < 1)
        CalcCellPlusLast = CalcCell+Last+TotalBeachCells;

        if (CalcCellPlusLast <= 0)
            fprintf('CalcCellPlusLast has gone below 1!\n')
            beep
        end
```

```
        if (debug5a)
            fprintf('CalcCell: %i, CalcCellPlusLast: %i
\n',CalcCell,CalcCellPlusLast);
            beep
            pause
        end

    else
        CalcCellPlusLast = CalcCell+Last;
    end

    % Account for periodic boundary conditions for [To] and
[From]
    if (CalcCell == TotalBeachCells + 1)
        CalcCellOrig = CalcCell;
        CalcCell = 1;
    end

    if (CalcCellPlusNext == TotalBeachCells + 1)
        CalcCellPlusNext = 1;
    elseif (CalcCellPlusNext == 0)
        CalcCellPlusNext = TotalBeachCells;
    end

    if (InShadow(CalcCell) == 'n')
        % Adjustment for maximum transport when passing
through 45 degrees
        % This adjustment is only made for moving from
downwind to upwind conditions
        %
        % purposefully done before shadow adjustment, only
use maxtran when
        % transition from dw to up not because of shadow
        % keeping transition from uw to dw - does not seem
to be big deal (04/02 AA)

        if ( ( (UpWind(CalcCell) == 'd') &&
(UpWind(CalcCellPlusNext) == 'u') && ...
                (InShadow(CalcCellPlusNext) == 'n') ) || ...
                ( (UpWind(CalcCellPlusLast) == 'u') &&
(UpWind(CalcCell) == 'd')...
                && (InShadow(CalcCellPlusLast) == 'n') ) )
```

```
            MaxTrans = 'y';

            if (debug5)
                fprintf('MAXTRANS   ');
            end
        end

        % Upwind/Downwind adjustment Make sure sediment is
put into shadows
        % If Next cell is in shadow, use UpWind condition

        DoFlux = 1;
        UpWindLocal = UpWind(CalcCell);

        if (InShadow(CalcCellPlusNext) == 'y')
            UpWindLocal = 'u';
            if (debug5)
                fprintf('U(2)   ');
            end
        end

        % If coming out of shadow, downwind should be used
        % HOWEVER- 02/04 AA - if high angle, will result in
same flux in/out problem
        %    solution  - no flux for high angle waves

        if ((InShadow(CalcCellPlusLast) == 'y') &&
(UpWindLocal == 'u'))
            DoFlux = 0;
            if (debug5)
                fprintf('U(X) NOFLUX \n');
            end
        end

        if (UpWindLocal == 'u')

            % Account for periodic boundary conditions
            CalcCellPlusLastPlusCorrection =
CalcCellPlusLast+Correction;

%             if (debug5a)
%                 fprintf('CalcCell: %i,
CalcCellPlusLastPlusCorrection: %i
\n',CalcCell,CalcCellPlusLastPlusCorrection);
```

```
%                beep
%                pause
%            end

            if ((CalcCellPlusLastPlusCorrection) <= 0)
                CalcCellPlusLastPlusCorrection =
TotalBeachCells;

%                if (debug5a)
%                    fprintf('CalcCell: %i,
CalcCellPlusLastPlusCorrection: %i
\n',CalcCell,CalcCellPlusLastPlusCorrection);
%                    beep
%                    pause
%                end

            end

            ShoreAngleUsed =
ShorelineAngle(CalcCellPlusLastPlusCorrection);

            if (debug5)
                fprintf(['UP   ShoreAngle:
',num2str(ShoreAngleUsed*radtodeg)]);
            end

        elseif (UpWindLocal == 'd')

            % Account for periodic boundary conditions
            CalcCellPlusCorrection = CalcCell+Correction;

            if (CalcCellPlusCorrection <= 0)
                CalcCellPlusCorrection = TotalBeachCells;
            end

            ShoreAngleUsed =
ShorelineAngle(CalcCellPlusCorrection);

            if (debug5)
                fprintf(['DN   ShoreAngle:
',num2str(ShoreAngleUsed*radtodeg)]);
            end
        end
```

```
        % !!! Do not do transport on unerneath c'cause it
gets all messed up
        if (abs(ShoreAngleUsed) > SedTransLimit/radtodeg)
            DoFlux = 0;
        end

        % Determine if transport is occuring over a groin
        if (CalcCell < CalcCellPlusNext)
            % Transport is going right

            if (GroinRightOfBeach(CalcCell) == 1)
                % Transport is over a groin

                GroinTransport = 1;
                GroinTransRight = 1;
                GroinTipDepth = D_G(CalcCell);
            end

        elseif (CalcCell > CalcCellPlusNext)
            % Transport is going left

            if (GroinLeftOfBeach(CalcCell) == 1)
                % Transport is over a groin

                GroinTransport = 1;
                GroinTransLeft = 1;
                GroinTipDepth = D_G(CalcCell);
            end
        else
            fprintf('WTF is going on with Left/Right
transport?!?!? \n');
            beep
            pause
        end

        % See if sediment transport is occuring in a zone of
wave
        % diffraction
        if ((CalcCell < CalcCellPlusNext) &&
(InDiffShadow(CalcCell) == 1))
            % Transport going right
            DiffShadowed = 1;                          % Flag
indicating diffractive effects
```

```
            DiffPhi = DiffPhi_b(CalcCell);    % Break angle
due to diffraction
            DiffKb = DiffK_b(CalcCell);       % Diffraction
coefficient
        elseif ((CalcCell > CalcCellPlusNext) &&
(InDiffShadow(CalcCellPlusNext) == 1))
            % Transport is going left
            DiffShadowed = 1;                 % Flag
indicating diffractive effects
            DiffPhi = DiffPhi_b(CalcCellPlusNext);    % Break
angle due to diffraction
            DiffKb = DiffK_b(CalcCellPlusNext);       %
Diffraction coefficient
        else
            % Breaking waves not influenced by diffraction
            DiffShadowed = 0;
            DiffPhi = 0;
            DiffKb = 0;
        end

        % Send to SedTrans to calculate VolumeIn and
VolumeOut

        if (debug5)
            fprintf(['From: ',num2str(CalcCell),' To:
',num2str(CalcCellPlusNext),...
                ' TransAngle: ',num2str((WaveAngle -
ShoreAngleUsed)*radtodeg)]);
        end


        if (debug40a && GroinTransport)
            if (GroinTransRight)
                fprintf(['Trans over groin -- RT
From:',num2str(CalcCell),' To:
',num2str(CalcCellPlusNext),'\n']);
            elseif (GroinTransLeft)
                fprintf(['Trans over groin -- LT
From:',num2str(CalcCell),' To:
',num2str(CalcCellPlusNext),'\n']);
            end
        end

        if (DoFlux)
```

```
            % Call SedTrans
            [VolumeIn, VolumeOut] = CEM_SedTrans(CalcCell,
CalcCellPlusNext, ...
                ShoreAngleUsed, MaxTrans, Y, WaveAngle,
Period, OffShoreWvHt, ...
                MaxDepth, RefractStep, KBreak, TimeStep,
VolumeIn, VolumeOut, ...

UseVariableCERC,K,rho,rho_s,porosity,GroinTransport,...

GroinPermeability,GroinTipDepth,DiffShadowed,DiffPhi,DiffKb
,MaxVol,...
                A_w,UseBreakDepthAsD_LT,debug6,debug40a,
debug41a);

            if (debug5a)

                if ((CalcCell > TotalBeachCells) ||
(CalcCell+Next <= 0))
                    fprintf('From: %i, To: %i, VolumeIn: %G,
VolumeOut: %G \n',CalcCell,CalcCellPlusNext,
VolumeIn(CalcCell), VolumeOut(CalcCell));
                    beep
                    pause
                end
            end


        end

    end % end if in shadow is no

end % end for loop


function [PercentFull, AllBeach, CellDepth] =
CEM_DoOverwash(xfrom,yfrom,xto,yto,xintto,yintto,widthin,is
hore,...

PercentFull,AllBeach,X,Y,CellDepth,DepthShoreface,LandHeigh
t,SurroundingAngle,TotalBeachCells,OWType,debug8,debug10b)
```

```
% CEM Function: given a cell where overwash is needed, move
sediment back
% for 'true' overwash based on shoreline angles


[DepthBB] = CEM_GetOverwashDepth
(xto,yto,xintto,yintto,ishore,...

PercentFull,CellDepth,X,Y,SurroundingAngle,DepthShoreface,T
otalBeachCells,OWType,debug10b);

% calculated value of most that backbarrier ca nmove given
geometry (true, non-iterative solution)


if (DepthBB == DepthShoreface)

    BBneed = MaxOver;

else

    BBneed = (CritBWidth - widthin) / CellWidth /  (1 -
(DepthBB / DepthShoreface));

end

if (BBneed <= MaxOver)
    % do all overwash

    delShore = BBneed * DepthBB / DepthShoreface;
    delBB = BBneed;

else
    % only do overwash to max change)

    delShore = MaxOver * DepthBB / DepthShoreface ;
    delBB = MaxOver;

end

if (debug10b)
    fprintf(['** Overwash From X: ',num2str(xfrom),' Y:
',num2str(yfrom),...
```

```
        ' To: X: ',num2str(xto),' Y: ',num2str(yto),'
Width: ',num2str(widthin),'\n']);
    fprintf(['DepthBB: ',num2str(DepthBB),' BBNeed:
',num2str(BBneed),...
        ' DelShore: ',num2str(delShore),' DelBB:
',num2str(delBB),'\n']);
end

% Adjust PercentFull matrix
PercentFull(yto,xto) = PercentFull(yto,xto) + delBB;
PercentFull(yfrom,xfrom) = PercentFull(yfrom,xfrom) -
delShore;

% Adjust for empty and full cells
if (PercentFull(yto,xto) > 1)

    [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,AllBeach,CellDepth,xto,yto,LandH
eight,debug8);

end
if (PercentFull(yto,xto) < 0)

    [PercentFull, AllBeach, CellDepth] =
CEM_OopsImEmpty(PercentFull,AllBeach,CellDepth,xto,yto,Dept
hShoreface,debug8);

end


function [Coastline] =
CEM_FindCoastline(PercentFull,AllBeach,InitBeach,Xmax,Ymax)

% CEM Function: finds the coastline from PercentFull and
AllBeach

% Initialize
ShoreLength = (3*Ymax/2) - (Ymax/2+1);
Coastline = zeros(1,ShoreLength);
index = 1;

% Determine the shoreline
for y = (Ymax/2+1):(3*Ymax/2)
    x = Xmax-1;
```

```
    xtop = Xmax;

    % step back to where we encounter allbeach
    while (AllBeach(y,x) == 'n')
        x = x-1;
    end

    % if on side of shape, need to average
    if (PercentFull(y,x+2) > 0)

        xtop = x+1;

        while(PercentFull(y,xtop) > 0)
            xtop = xtop + 1;
        end

        xsave = x;

        for i = (x+1):xtop
            xsave = xsave + PercentFull(y,i);
        end

    % otherwise Regular Beach Condition
    else
        xsave = x + PercentFull(y,x+1);
    end

    Coastline(index) = xsave - InitBeach + 0.5;

    index = index + 1;

end


function [GroinRightOfBeach,GroinLeftOfBeach] =
CEM_FindGroinBeachCells(X,Y,GroinCells,TotalBeachCells,Ymax
,debug40)

% CEM Function: Determines which beach cells have a
terminal groin on the
% right (downdrift) border.

%% Initialize
```

```
GroinRightOfBeach = zeros(1,TotalBeachCells);
GroinLeftOfBeach = zeros(1,TotalBeachCells);


%% Run loop
for i = 1:TotalBeachCells

    % Account for periodic boundary condtions
    if (Y(i)-1 == 0)
        YMinus = 2*Ymax;
    else
        YMinus = Y(i)-1;
    end

    % Check to see if there is a groin on the right border
of the ith beach
    % cell
    if (GroinCells(Y(i),X(i)) == 1)
        % You've found a beach cell with a terminal groin
on the right
        % border

        if (debug40)
            fprintf(['Groin found right of beach x:
',num2str(X(i)),' ,y: ',num2str(Y(i)),'\n']);
        end

        GroinRightOfBeach(i) = 1;
        GroinLeftOfBeach(i+1) = 1;

    end

    % Check to see if there is a groin on the left border
of the ith beach
    % cell
    if (GroinCells(YMinus,X(i)) == 1)
        % You've found a beach cell with a terminal groin
on the left
        % border

        if (debug40)
            fprintf(['Groin found left of beach x:
',num2str(X(i)),' ,y: ',num2str(Y(i)),'\n']);
        end
```

```
        GroinLeftOfBeach(i) = 1;

    end

end


function [IsInShadow] = CEM_FindIfInShadow(icheck, ShadMax,
AllBeach, PercentFull,
WaveAngle,X,Y,TotalBeachCells,CurrentTimeStep,Ymax)

% CEM Function: Function to determine if particular cell
xin,yin is in
% shadow.  Returns a character 'y' if yes 'n' if no.

% Initialize
%degtorad = pi/180;          % Transform degrees to radians
radtodeg = 180/pi;          % Transform radians to degrees
debug2a = 0;             % local debugger
%debug2b = 0;             % local debugger


% convert angle to a slope and the direction of steps
% note that for case of shoreline, positive angle will be
minus y direction

if (WaveAngle*radtodeg == 0.0)
    % unlikely, but make sure no div by zero

    slope = 0.00001;

elseif (abs(WaveAngle*radtodeg == 90))

    slope = 999.9;

else

    slope = abs(tan(WaveAngle));

end

if (WaveAngle > 0)
    ysign = -1;
```

```
else
    ysign = 1;
end

if (debug2a)
    fprintf(['\nI: ',num2str(icheck),'----------x:
',num2str(X(icheck)),...
        '  Y: ',num2str(Y(icheck)),' WaveAngle:
',num2str(WaveAngle*radtodeg),...
        '  Slope: ',num2str(slope),' Sign:
',num2str(ysign),'\n']);
end

% depending on local orientations, starting point will
differ
% so go through scenarios
xinint = X(icheck);
yinint = Y(icheck);

% Account for periodic boundary conditions
if (yinint == 1)
    % on left boundary

    yinintMinus = Y(TotalBeachCells);
    yintintPlus = yinint+1;

elseif (yinint == 2*Ymax)
    % on right boundary

    yintintPlus = Y(1);
    yinintMinus = yinint-1;

else
    % regular situation

    yinintMinus = yinint-1;
    yintintPlus = yinint+1;

end


if ((AllBeach(yinint,xinint-1) == 'y') || ...
        ((AllBeach(yinintMinus,xinint) == 'y') && ...
        (AllBeach(yinint+1,xinint) == 'y')))
```

```
    % 'regular condition'
    % plus 'stuck in the middle' situation (unlikely
scenario)

    xin = xinint + PercentFull(yinint,xinint);
    yin = yinint + 0.5;

    if (debug2a)
        fprintf(['-- Regular xin: ',num2str(xin),' yin:
',num2str(yin),'\n']);
    end

elseif (AllBeach(yinintMinus,xinint) == 'y')
    % on right side

    xin = xinint + 0.5;
    yin = yinint + PercentFull(yinint,xinint);

    if (debug2a)
        fprintf(['-- Right xin: ',num2str(xin),' yin:
',num2str(yin),'\n']);
    end

elseif (AllBeach(yintintPlus,xinint) == 'y')
    % on left side

    xin = xinint + 0.5;
    yin = yinint + 1.0 - PercentFull(yinint,xinint);

    if (debug2a)
        fprintf(['-- Left xin: ',num2str(xin),' yin:
',num2str(yin),'\n']);
    end

elseif (AllBeach(yinint,xinint+1) == 'y')
    % gotta be on the bottom now

    xin = xinint + 1 - PercentFull(yinint,xinint);
    yin = yinint + 0.5;
    if (debug2a)
        fprintf(['-- Under xin: ',num2str(xin),' yin:
',num2str(yin),'\n']);
    end
```

```
else
    % debug ain't just an insect

    fprintf(['Shadowstart Broke !!!! x: ',num2str(xinint),'
y: ',num2str(yinint),'\n']);
    return;

end

x = xin;
y = yin;

while ((floor(x) < ShadMax ) && (y > Ymax/2) && (y <
3*Ymax/2))

    NextXInt = floor(x) + 1;

    if (ysign > 0)
        NextYInt = floor(y) + 1;
    else
        NextYInt = ceil(y-1);
    end

    % moving to next whole 'x' position, what is y
position?
    Yup = y + (NextXInt - x)*slope * ysign;
    DistanceUp = ((Yup - y)*(Yup - y) + (NextXInt -
x)*(NextXInt - x));

    % moving to next whole 'y' position, what is x
position?
    Xside = x + abs(NextYInt - y) / slope;
    DistanceSide = ((NextYInt - y)*(NextYInt - y) + (Xside
- x)*(Xside - x));

    if (debug2a)
       %fprintf([
    end

    if (DistanceUp < DistanceSide)
        % next cell is the up cell

        x = NextXInt;
        y = Yup;
```

```
        xtestint = NextXInt;
        ytestint = floor(y);
        if (debug2a)
            fprintf(' up ');
        end

    else
        % next cell is the side cell

        x = Xside;
        y = NextYInt;
        xtestint = floor(x);
        ytestint = y + (ysign-1)/2;
        if (debug2a)
            fprintf(' side ');
        end

    end

    % if (debug2a)

    % Now Test
    % If AllBeach is along the way, will we pass through
'diamond'?
    % Trick - if crossing through the diamond, will change
quadrants
    % Probably won't get to this one, though

    if (AllBeach(ytestint,xtestint) == 'y')
        % use same approach to find exit (could make this
modular)
        % don't change 'x' or 'y' and this will be ok

        NextXInt = floor(x) + 1;

        if (ysign > 0)
            NextYInt = floor(y) + 1;
        else
            NextYInt = ceil(y-1);
        end

        Yup = y + (NextXInt-x)*slope * ysign;
        DistanceUp = ((Yup - y)*(Yup - y) + (NextXInt -
x)*(NextXInt - x));
```

```
        Xside = x + abs(NextYInt - y) / slope;
        DistanceSide = ((NextYInt - y)*(NextYInt - y) +
(Xside - x)*(Xside - x));

        if (DistanceUp < DistanceSide)
            % next cell is the up cell

            xout = NextXInt;
            yout = Yup;

        else
            % next cell is the side cell

            xout = Xside;
            yout = NextYInt;

        end


        if(( (xout-xtestint-0.5) * (x-xtestint-0.5) < 0 )
|| ...
                ((yout-ytestint-0.5) * (y-ytestint-0.5) <
0))
            if (debug2a)
                fprintf('   Shaddowed   ');
            end
            IsInShadow = 'y';
            return;
        end

    % Compare a partially full cell's x - distance to a
line projected      */
    % from the starting beach cell's x-distance
*/
    % This assumes that beach projection is in x-direction
(not too bad)    */

    elseif (PercentFull(ytestint,xtestint) > 0)

        if ((AllBeach(ytestint,xtestint-1) == 'y') || ...
                ((AllBeach(ytestint-1,xtestint) == 'y') &&
...
                (AllBeach(ytestint+1,xtestint) == 'y')))
            % 'regular' condition
```

```
            % plus 'stuck in the middle' situation
(unlikely scenario)

            xtest = xtestint +
PercentFull(ytestint,xtestint);
            ytest = ytestint + 0.5;

            if (xtest > (xin  + abs(ytest-yin)/slope) )
                IsInShadow = 'y';
                return;
            end
        elseif (AllBeach(ytestint-1,xtestint) == 'y')
            % on right side

            xtest = xtestint + 0.5;
            ytest = ytestint +
PercentFull(ytestint,xtestint);

            if (ytest > (yin + (xtest-xin) * slope))
                IsInShadow = 'y';
                return;
            end
        elseif (AllBeach(ytestint+1,xtestint) == 'y')
            % on left side

            xtest = xtestint + 0.5;
            ytest = ytestint + 1.0 -
PercentFull(ytestint,xtestint);

            if (ytest < (yin + (xtest-xin) * slope))
                IsInShadow = 'y';
                return;
            end
        elseif (AllBeach(ytestint,xtestint+1) == 'y');
            % gotta be on the bottom now

            xtest = xtestint + 1 -
PercentFull(ytestint,xtestint);
            ytest = ytestint + 0.5;

            if (xtest < (xin  + abs(ytest-yin)/slope) )
                 IsInShadow = 'y';
                return;
            end
```

```
        else
            % debug ain't just an insect

            fprintf(['Shadows not responding in xin:
',num2str(xin),' yin: ',...
                num2str(yin),' xtestint:
',num2str(xtestint),' ytestint: ',num2str(ytestint),'
Timestep:',num2str(CurrentTimeStep),'\n']);
        end

    end

end

IsInShadow = 'n';
return;



function WaveAngle = CEM_FindWaveAngle

% CEM Function: calculates wave angle for given time step

% Initialize
flag = 1;
% i = 0;
index = 1;
% index2 = 0;

if (WaveIn)

% Choose a random PDF bin
RandBin = rand(1);

while(flag)
    if (RandBin <=  BinProbabilitiy(index))
        AngleBin = WaveAngleBin_max(index);
        flag = 0;
        break
    end
    index = index + 1;

end
```

```
% Create a random angle fluctuation
AngleFluct = rand(1);

% Calculate the offshore wave angle
WaveAngle = (AngleFluct * BinSize + (AngleBin - BinSize));

else

end


function WaveAngle =
CEM_FindWaveAngleWaveIn(WaveMax,BinProbabilitiy,BinSize)

% CEM Function: calculates wave angle for given time step
if using an
% imported wave PDF.

% Constants
% radtodeg = 180/pi;          % Transform radians to
degrees
degtorad = pi/180;          % Transform degrees to radians

% Initialize
flag = 1;
index = 1;

% Choose a random PDF bin
RandBin = rand(1);

while(flag)
    if (RandBin <=  BinProbabilitiy(index))
        AngleBin = WaveMax(index);
        flag = 0;
        break
    end
    index = index + 1;

end

% Create a random angle fluctuation
AngleFluct = rand(1);

% Calculate the offshore wave angle
```

```
Angle = (AngleFluct * BinSize + (AngleBin - BinSize));
WaveAngle = Angle*degtorad;



function [PercentFull, AllBeach, CellDepth, FixBeachFlag] =
CEM_FixBeach(PercentFull, AllBeach,
CellDepth,GroinCells,ShadowXMax,DepthShoreface,ShorefaceSlo
pe,CellWidth,Xmax,Ymax,LandHeight,debug8,debug9)

% For CEM: Looks at entire data set, finds unattached
pieces of sand, and
% moves them back to the shore.  This should take care of
floating bits of
% sand as well as over/under filled beach pieces.
%
% Required function files:
%   CEM_OopsImFull.m
%   CEM_OopsImFull.m

%fillcells3 = 0;
%sweepsign = '';
% debug9 = 0;

FixBeachFlag = 0;
RandNumber = rand(1);

if (RandNumber*2 > 1)
    sweepsign = 1;
    if (debug9)
        fprintf('fixL  ');
    end
else
    sweepsign = 0;
    if (debug9)
        fprintf('fixR  ');
    end
end

FixXMax = ShadowXMax +
ceil(DepthShoreface/CellWidth/ShorefaceSlope) + 3;

if (FixXMax > Xmax)
    FixXMax = Xmax-1;
```

```
end

for x = FixXMax:-1:2
    for i = 2:2*Ymax-1
        if (sweepsign == 1)
            y = i;
        else
            y = (2*Ymax)-i;
        end

        % Account for periodic boundary conditions
        if (y == 1)
            yMinus = 2*Ymax;
        else
            yMinus = y-1;
        end
        if (y == 2*Ymax)
            yPlus = 1;
        else
            yPlus = y+1;
        end

        % ye old depth fix
        if ((PercentFull(y,x) <= 0) && (CellDepth(y,x) >
DepthShoreface)...
                && (CellDepth(y,x-1) == DepthShoreface))
            if ((CellDepth(y,x+1) == DepthShoreface) && ...
                    (CellDepth(y,x-1) == DepthShoreface) &&
...
                    (CellDepth(yPlus,x) == DepthShoreface))

                % Fill hole
                CellDepth(y,x) = DepthShoreface;
            end
        end

        % Fix beaches that are too full
        if (PercentFull(y,x) > 1)
            if (debug9)
                fprintf('too full (%i,%i) PF:
%G\n',x,y,PercentFull(y,x));
            end
            PercentFull(y,x+1) = PercentFull(y,x) - 1;
            AllBeach(y,x+1) = 'n';
```

```
            PercentFull(y,x) = 1;
            AllBeach(y,x) = 'y';
        end

        % Take care of situations that shouldn't exist

        % Empty cell (percent full < 0)
        if (PercentFull(y,x) < 0)
            AllBeach(y,x) = 'n';
            if (debug9 && (y ~= 0))
                fprintf(['\nUnder 0 Percent X:
',num2str(x),' Y: ',...
                    num2str(y),' Per:
',num2str(PercentFull(y,x)),'\n']);
            end

            [PercentFull, AllBeach, CellDepth] =
CEM_OopsImEmpty(PercentFull,...

AllBeach,CellDepth,GroinCells,x,y,DepthShoreface,Ymax,debug
8);

            if (debug9)

fprintf(['Underzerofill(',num2str(x),',',num2str(y),') ']);
            end
        end

        % Overfull cell (percent full > 1)
        if (PercentFull(y,x) > 1)
            AllBeach(y,x) = 'y';
            CellDepth(y,x) = -LandHeight;

            if (debug9)
                fprintf(['\nOver 100 Percent X:
',num2str(x),' Y: ',...
                    num2str(y),' Per:
',num2str(PercentFull(y,x)),'\n']);
            end

                [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,...

AllBeach,CellDepth,GroinCells,x,y,LandHeight,Ymax,debug8);
```

```
        end

        if (((PercentFull(y,x) >= 0) && (PercentFull(y,x) <
1)) && (AllBeach(y,x) == 'y'))
            AllBeach(y,x) = 'n';
            CellDepth(y,x) = -LandHeight;
            if (debug9 && y ~= 0)
                fprintf(['\nALLBeachProb X: ',num2str(x),'
Y: ',num2str(y),'\n']);
            end
        end



        % Take care of 'loose' bits of sand

        fillcells3 = 0;

        if ((PercentFull(y,x) ~= 0) && (PercentFull(y,x-1)
< 1) && ...
                (PercentFull(y,x+1) < 1) &&
(PercentFull(yPlus,x) < 1) &&...
                (PercentFull(yMinus,x) < 1) &&
(AllBeach(y,x) == 'n'))
            % Beach in cell, but bottom, top, right, and
left neighbors not all full
            if (debug9 && y ~= 0)
                fprintf(['\nFB Moved loose bit of sand,  X:
',num2str(x),...
                    ' Y: ',num2str(y),' Per:
',num2str(PercentFull(y,x))]);
            end

            % distribute to partially full neighbors

            if ((PercentFull(y,x-1) < 1) &&
(PercentFull(y,x-1) > 0))
                fillcells3 = fillcells3 + 1;
            end
            if ((PercentFull(y,x+1) < 1) &&
(PercentFull(y,x+1) > 0))
                fillcells3 = fillcells3 + 1;
            end
```

```
            if ((PercentFull(yMinus,x) < 1) &&
(PercentFull(yMinus,x) > 0))
                fillcells3 = fillcells3 + 1;
            end
            if ((PercentFull(yPlus,x) < 1) &&
(PercentFull(yPlus,x) > 0))
                fillcells3 = fillcells3 + 1;
            end

            if (fillcells3 > 0)

                if ((PercentFull(y,x-1) < 1) &&
(PercentFull(y,x-1) > 0))
                    PercentFull(y,x-1) = PercentFull(y,x-1)
+ (PercentFull(y,x)/fillcells3);
                    if (debug9)
                        fprintf('   MOVEDBACK');
                    end
                end
                if ((PercentFull(y,x+1) < 1) &&
(PercentFull(y,x+1) > 0))
                    PercentFull(y,x+1) = PercentFull(y,x+1)
+ (PercentFull(y,x)/fillcells3);
                    if (debug9)
                        fprintf('   MOVEDUP');
                    end
                end
                if ((PercentFull(yMinus,x) < 1) &&
(PercentFull(yMinus,x) > 0))
                    PercentFull(yMinus,x) =
PercentFull(yMinus,x) + (PercentFull(y,x)/fillcells3);
                    if (debug9)
                        fprintf('   MOVEDLEFT');
                    end
                end
                if ((PercentFull(yPlus,x) < 1) &&
(PercentFull(yPlus,x) > 0))
                    PercentFull(yPlus,x) =
PercentFull(yPlus,x) + (PercentFull(y,x)/fillcells3);
                    if (debug9)
                        fprintf('   MOVEDRIGHT');
                    end
                end
```

```
            else

                fprintf(['Loner fixbeach breakdown - mass
disintegrated x: '...
                    ,num2str(x),' y: ',num2str(y),'\n']);

            end

            PercentFull(y,x) = 0;
            AllBeach(y,x) = 'n';
            CellDepth(y,x) = DepthShoreface;

            if (debug9)
                fprintf('\n');
            end

            % If we have overfilled any of the cells in
this loop, need to OopsImFull()

            if (PercentFull(y,x-1) > 1)
                [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,...
                    AllBeach,CellDepth,GroinCells,(x-
1),y,LandHeight,Ymax,debug8);
                if (debug9)
                    fprintf('   Below Overfilled\n');
                end
            end
            if (PercentFull(yMinus,x) > 1)
                [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,...

AllBeach,CellDepth,GroinCells,(x),(yMinus),LandHeight,Ymax,
debug8);
                if (debug9)
                    fprintf('   Left side Overfilled\n');
                end
            end
            if (PercentFull(yPlus,x) > 1)
                [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,...

AllBeach,CellDepth,GroinCells,(x),(yPlus),LandHeight,Ymax,d
ebug8);
```

```
                if (debug9)
                    fprintf('   Right Side Overfilled\n');
                end
            end
            if (PercentFull(yPlus,x+1) > 1)
                [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,...

AllBeach,CellDepth,GroinCells,(x+1),(yPlus),LandHeight,Ymax
,debug8);
                if (debug9)
                    fprintf('   Top Overfilled\n');
                end
            end

        end


        % Fix situations that break shadow sweep
        % Added for v2.15.1

        if ((PercentFull(y,x) > 0) && (PercentFull(y,x) <
1))
            % We have a beach cell

            if ((AllBeach(y,x-1) == 'n') &&
(PercentFull(y,x+1) == 0))
                % We have stacked beach cells

                if ((PercentFull(yMinus,x) == 0) &&
(PercentFull(yPlus,x) == 0)...
                        && ((PercentFull(yMinus,x-1) == 0)
&& (PercentFull(yMinus,x-1) == 0)))
                    % We have stacked beach cells sticking
out

                    % Move the beach back to correct
                    PercentFull(y,x-1) = PercentFull(y,x-1) +
PercentFull(y,x);
                    AllBeach(y,x) = 'n';
                    CellDepth(y,x) = DepthShoreface;

                    if (debug9)
                        fprintf('   MOVEDBACK');
```

```
                end

            end

        end

        FixBeachFlag = 1;  % Run FixBeach again just to
be sure

    end % end added fix for v2.15.1

end

end



function
[InDiffShadow,GroinTipX,GroinTipY,DiffPhi_b,DiffK_b] =
CEM_GroinShadowing(X,...

WaveAngle,OffShoreWvHt,Period,MaxDepth,GroinData,GroinStart
,TotalBeachCells,...

DeanProfileA,RefractStep,PeriodicCopyGroins,Ymax,UseGroinDi
ffraction,...

UseGroinShadowBlock,UseGeometricDiffAng,PosMWD_0,NegMWD_0,C
ellWidth,KBreak,debug6)

% CEM Function: Creates shadow zones from groins to
simulate
% diffraction of waves

% Initialize
degtorad = pi/180;                          % Transform
degrees to radians
InDiffShadow = zeros(1,TotalBeachCells);    % Indicator if
cell is in diffraction shadow (0/1)
DiffPhi_b = zeros(1,TotalBeachCells);       % Breaking wave
angle under diffraction (angle from right boundary of cell
to groin tip)
DiffK_b = zeros(1,TotalBeachCells);         % Diffraction
coefficient at the right boundary all beach cells
```

```
% Make the new groin data matrix (that accounts for
periodic bounaries)
if (PeriodicCopyGroins == 0)
    NewGroinData =
zeros(length(GroinData(:,1)),length(GroinData(1,:)));
    NewGroinData(:,2) = GroinData(:,2);
    NewGroinData(:,1) = GroinData(:,1) + (Ymax/2);
elseif (PeriodicCopyGroins)
    beep
    fprintf('ERROR: Periodic Boundary Copying of Groins NOT
YET FUNCTIONAL\n');
    pause

    %     NewGroinData =
zeros(length(GroinData(:,1))*2,length(GroinData(1,:)));
    %     NewGroinData((Ymax/2)+1:(3*Ymax/2),1) =
GroinData(:,1) + (Ymax/2);
    %     NewGroinData((Ymax/2)+1:(3*Ymax/2),2) =
GroinData(:,2);
    %     NewGroinData(:,1) =
GroinY(1:length(GroinData(:,1))*2);
    %     NewGroinData(1:length(GroinData(:,1)),2) =
GroinData(:,2);
    %     NewGroinData(length(GroinData(:,1))+1:end,2) =
GroinData(:,2);
end


% Find the coordinates (x,y relative to PercentFull) for
tips of all groins
% Qualifier: make sure there is no beach above it (for the
moment, I'm
% going to assume that CEM_FindIfInShadow already takes
care of this)

NumberOfGroins = length(NewGroinData(:,1));
GroinTipX = zeros(1,NumberOfGroins);    % X coordinate of
tips of groins
GroinTipY = zeros(1,NumberOfGroins);    % Y corodinate of
tips of groins

for GroinNo = 1:NumberOfGroins
    GroinTipY(GroinNo) = NewGroinData(GroinNo,1);
```

```
    GroinTipX(GroinNo) = NewGroinData(GroinNo,2) +
GroinStart;
end



if (UseGroinDiffraction)

    if (WaveAngle >= 0) % Positive angle
        % Wave moving left to right (transport should be
right)

        for i = 1:NumberOfGroins

            % Calculate the cross-shore distance from
shoreline to groin tip
            CSGroinLength = GroinTipX(i) - X(GroinTipY(i));
% Cross-shore distance (in cells) from groin tip to
shoreline

            if (CSGroinLength > 0)
                % Groin is past the shorline; it causes
wave diffraction

                % Calcalute the water depth at the groin
tips using the mean cross-shore length of the groins
                % Assume that the bathymetry follows the
Dean profile
                DepthStop = DeanProfileA * ((CellWidth *
CSGroinLength) ^ (2/3));
                ShoreAngle = 0;     % Assume that
refraction occurs over contours parallel to y-axis

                % Determine the wave angle (WvAngle) at the
groin tip
                [WvHeight,WaveAngleAtGroinTip] =
CEM_AWTRefractToDepth(WaveAngle,OffShoreWvHt,Period,MaxDept
h,DepthStop,ShoreAngle,RefractStep,debug6);

                % Determine the mean wave angle at the
groin tip
                [WvHeight,PosMWD] =
CEM_AWTRefractToDepth(PosMWD_0,OffShoreWvHt,Period,MaxDepth
,DepthStop,ShoreAngle,RefractStep,debug6);
```

```
                % Calculate the longshore length of wave
shadow
                LSShadow = floor(CSGroinLength *
tan(abs(WaveAngleAtGroinTip))); % Longshore extent of
diffraction shadow (in cells, down up)

                % Find the beginning and ending cells of
shadow
                ShadowStart = GroinTipY(i) + 1; % Beginning
cell of shadow (left to right)
                ShadowEnd = ShadowStart + LSShadow - 1; %
Ending cell of shadow (left to right)

                % Declare which cells in are the shadow
                InDiffShadow(ShadowStart:ShadowEnd) = 1;

                % Determine the diffraction coefficient
(DiffK_b) and
                % breaking wave angle (DiffPhi_b) for every
cell in shadow
                for cell = ShadowStart:ShadowEnd

                    % If UseGroinShadowBlock, block wave
energy in shadow
                    if (UseGroinShadowBlock)

                         DiffPhi_b(cell) = 0;
                         DiffK_b(cell) = 0;

                    else

                        % Determine the longshore distance
from break point to groin
                        % Right boundary of cell used as
break point (LST right)
                        LSDist = cell - GroinTipY(i);


                        % Find the diffraction coefficient
                        % First, we need theta (I believe
this is the angle between the mean wave direction and the
wave ray
```

```
                        % at the groin tip). Theta should
be negative if wave angle is less than MWD
                        theta = WaveAngleAtGroinTip -
PosMWD;


                        % Calculate the diffraction
coefficient
                        % Use Kamphius (2010)
                        if (theta <= 0)
                            DiffK_b(cell) = 0.71 -
(0.0093*theta) + (0.000025*(theta^2));
                        elseif (theta <= 40*degtorad)
                            DiffK_b(cell) = 0.71 +
0.37*sin(-theta);
                        else
                            DiffK_b(cell) = 0.83 +
0.17*sin(-theta);
                        end

                        % Find the breaking wave angle
                        if (UseGeometricDiffAng)
                            % Gemetrically determine the
breaking wave angle

                            DiffPhi_b(cell) =
atan(LSDist/CSGroinLength);

                        else
                            % Use Eq. 15.11 Kamphius (2010)

                            % First, we need the breaking
wave angle without diffraction
                            % Refract assuming zero wave
angle (using
                            % another wave angle in the
shadow zone can get
                            % messy)
                            [WvHeight,Alpha_b] =
CEM_AWTRefractToBreak(WaveAngle,OffShoreWvHt,Period,MaxDept
h,KBreak,0,RefractStep,debug6);

                            % Calculate the breaking wave
angle
```

```
                                DiffPhi_b(cell) = Alpha_b *
(DiffK_b(cell).^0.375) *...
                                    ( (2*LSDist) /
(CSGroinLength) * ((tan(PosMWD)) + (tan(0.88*Alpha_b))) );

                            end


                        end % End if UseGroinShadowBlock

                    end % end loop for shadowed cells

                end % end if groin length > 0

            end % End loop for all groins

        elseif (WaveAngle < 0) % Negative angle
            % Wave moving right to left (transport should be
left)

            for i = NumberOfGroins:-1:1

                % Calculate the cross-shore distance from
shoreline to groin tip
                CSGroinLength = GroinTipX(i) - X(GroinTipY(i));
% Cross-shore distance (in cells) from groin tip to
shoreline

                if (CSGroinLength > 0)
                    % Groin is past the shorline; it causes
wave diffraction

                    % Calcalute the water depth at the groin
tips using the mean cross-shore
                    % length of the groins
                    % Assume that the bathymetry follows the
Dean profile
                    DepthStop = DeanProfileA * ((CellWidth *
CSGroinLength) ^ (2/3));
                    ShoreAngle = 0;     % Assume that
refraction occurs over contours parallel to y-axis

                    % Determine the wave angle (WvAngle) at the
groin tip
```

```
                [WvHeight,WvAngle] =
CEM_AWTRefractToDepth(WaveAngle,OffShoreWvHt,Period,MaxDept
h,DepthStop,ShoreAngle,RefractStep,debug6);

                % Determine the mean wave angle at the
groin tip
                [WvHeight,NegMWD] =
CEM_AWTRefractToDepth(NegMWD_0,OffShoreWvHt,Period,MaxDepth
,DepthStop,ShoreAngle,RefractStep,debug6);

                % Calculate the longshore length of wave
shadow
                LSShadow = floor(CSGroinLength *
tan(abs(WvAngle))); % Longshore extent of diffraction
shadow (in cells, rounded up)

                % Find the beginning and ending cells of
shadow
                ShadowEnd = GroinTipY(i); % Ending cell of
shadow (left to right)
                ShadowStart = GroinTipY(i) - LSShadow + 1;
% Beginning cell of shadow (left to right)

                % Declare which cells in are the shadow
                InDiffShadow(ShadowStart:ShadowEnd) = 1;

                % Geometrically determine the breaking wave
angle for every
                % beach cell in shadow
                for cell = ShadowStart:ShadowEnd

                    % If UseGroinShadowBlock, block wave
energy in shadow
                    if (UseGroinShadowBlock)

                        DiffPhi_b(cell) = 0;
                        DiffK_b(cell) = 0;

                    else

                        % Determine the longshore distance
from break point to groin
                        % Right boundary of cell used as
break point (LST right)
```

```
                        LSDist = GroinTipY(i) - cell + 1;


                        % Find the diffraction coefficient
                        % First, we need theta (I believe
this is the angle
                        % between the mean wave direction
and the wave ray
                        % at the groin tip.  You'll need to
refract the
                        % wave to the groin tip.
                        theta = abs(DiffPhi_b(cell)) -
abs(NegMWD);
                        % Theta should be negative if wave
angle is less
                        % than MWD


                        % Calculate the diffraction
coefficient
                        % Use Kamphius (2010)
                        if (theta <= 0)
                            DiffK_b(cell) = 0.71 -
(0.0093*theta) + (0.000025*(theta^2));
                        elseif (theta <= 40*degtorad)
                            DiffK_b(cell) = 0.71 +
0.37*sin(-theta);
                        else
                            DiffK_b(cell) = 0.83 +
0.17*sin(-theta);
                        end

                        % Find the breaking wave angle
                        if (UseGeometricDiffAng)

                            DiffPhi_b(cell) =
atan(LSDist/CSGroinLength);

                        else
                            % Use Eq. 15.11 Kamphius (2010)

                            % First, we need the breaking
wave angle without diffraction
```

```
                                % Refract assuming zero wave
angle (using
                                % another wave angle in the
shadow zone can get
                                % messy)
                                [WvHeight,Alpha_b] =
CEM_AWTRefractToBreak(WaveAngle,OffShoreWvHt,Period,MaxDept
h,KBreak,0,RefractStep,debug6);
                                Alpha_b = abs(Alpha_b);

                                % Calculate the breaking wave
angle
                                DiffPhi_b(cell) = -Alpha_b *
(DiffK_b(cell).^0.375) * ( (2*LSDist) / (CSGroinLength) *
((tan(abs(NegMWD))) + (tan(0.88*abs(Alpha_b)))) );


                            end

                        end % End if UseGroinShadowBlock

                    end % end loop for shadowed cells

                end % end if groin length > 0

            end % end loop for all groins

        else
            % wave angle is neither positive nor negative (how
did this happen?!?!)
            fprintf('GroinShadowing Broke!!!!\n')
            beep
            pause
        end

else
    % Not using groin diffraction

    % All pertinent matrcies should already be zero

end
```

```
function [PercentFull, CellDepth, AllBeach, Age] =
CEM_InitConds(PercentFull,...

CellDepth,AllBeach,Age,InitialDepth,InitBeach,InitBWidth,Ce
llWidth,...

ShelfSlope,LandHeight,DepthShoreface,InitialSmooth,InitialP
ert,...

InitCType,Xmax,Ymax,StartWithGaussian,GaussMult,GaussSigma,
GaussCenter)

% Function that makes the initial conditions for the MATLAB
version of the
% Coastline Evolution Model (CEM)

fprintf('Condition Initial \n');

if (InitCType == 0)

    % Create regular initial conditions - beach backed by
sandy island

    for y = 1:2*Ymax
        for x = 1:Xmax
            CellDepth(y,x) = InitialDepth + ((x-InitBeach)
* CellWidth * ShelfSlope);

            if (x < InitBeach)

                PercentFull(y,x) = 1;
                AllBeach(y,x) = 'y';
                CellDepth(y,x) = -LandHeight;

            elseif (x == InitBeach)
                if (InitialSmooth || StartWithGaussian)
                    PercentFull(y,x) = .5;
                else
                    PercentFull(y,x) = rand(1);
                end
                AllBeach(y,x) = 'n';
                CellDepth(y,x) = -LandHeight;
            elseif (x > InitBeach)
                PercentFull(y,x) = 0;
```

```
                AllBeach(y,x) = 'n';

                if (CellDepth(y,x) < DepthShoreface)
                    CellDepth(y,x) = DepthShoreface;
                end
            else
                fprintf(['WTF!?! X: ',num2str(x),' Y:
',num2str(y),' Per: ',PercentFull(y,x)]);
                beep
                return
            end

            Age(y,x) = 0;
        end
    end

elseif (InitCType == 1)

    % Create a simple barrier type initial condition -
island backed by
    % lagoon at slope of shelf

    for y = 1:2*Ymax
        for x= 1:Xmax

            CellDepth(y,x) = InitialDepth + ((x-InitBeach)
* CellWidth * ShelfSlope);

            if (CellDepth(y,x) <= 0)
                % This must be land due to the continental
shelf
                % intersection

                PercentFull(y,x) = 1;
                AllBeach(y,x) = 'y';
                CellDepth(y,x) = -LandHeight;

            elseif (x > InitBeach)
                % Shoreward of beach - enforce
ShorefaceDepth if necessary

                PercentFull(y,x) = 0;
                AllBeach(y,x) = 'n';
```

```
                if (CellDepth(y,x) < DepthShoreface)
                    CellDepth(y,x) = DepthShoreface;
                end

            elseif (x == InitBeach)
                % Beach cells

                if (InitialSmooth || StartWithGaussian)
                    PercentFull(y,x) = 0.5;
                else
                    PercentFull(y,x) = rand(1);
                end
                AllBeach(y,x) = 'n';
                CellDepth(y,x) = -LandHeight;

            elseif ((x < InitBeach) && (x > InitBeach -
InitBWidth - 1))
                % Island area

                PercentFull(y,x) = 1;
                AllBeach(y,x) = 'y';
                CellDepth(y,x) = -LandHeight;

            elseif (x == InitBeach - InitBWidth - 1)
                % Back of barrier

                if (InitialSmooth || StartWithGaussian)
                    PercentFull(y,x) = 0.5;
                else
                    PercentFull(y,x) = rand(1);
                end
                AllBeach(y,x) = 'n';
                CellDepth(y,x) = -LandHeight;

            elseif (x < InitBeach - InitBWidth - 1)
                % Lagoon at depth of shelf slope

                PercentFull(y,x) = 0;
                AllBeach(y,x) = 'n';
            end

            if (PercentFull(y,x) > 1)
```

```
                fpritnf(['Yo,
PercentFull(',num2str(y),',',num2str(x),') is greater than
1!!!']);
            end

            Age(y,x) = 0;

        end

    end

end


% Make the initial perturbation
if (InitialPert)

    % Establish perturbation parameters
    PWidth = 20;
    PHeight = 20;
    PYstart = 40;

    if (InitialPert == 1)
        % Square perturbation

        % Fill all beach areas
        for x = InitBeach:(InitBeach + PHeight)
            for y = PYstart:(PYstart + PWidth)
                PercentFull(y,x) = 1;
                AllBeach(y,x) = 'y';
            end
        end

        % PercenFull Top
        for y = (PYstart -1):(PYstart + PWidth +1)
            PercentFull(y,(InitBeach + PHeight + 1)) =
rand(1);
        end

        % PercentFull Sides
        for x = InitBeach:(InitBeach + PHeight)
            PercentFull(PYstart-1,x) = rand(1);
            PercentFull((PYstart + PWidth + 1),x) =
rand(1);
```

```
        end

    elseif (InitialPert == 2)
        % Steep point perturbation

        x = InitBeach;

        PercentFull(17,x) = 0.8;
        PercentFull(18,x) = 1.0;
        AllBeach(18,x) = 'y';
        PercentFull(19,x) = 0.8;

        x = InitBeach + 1;

        PercentFull(17,x) = 0.6;
        PercentFull(18,x) = 1.0;
        AllBeach(18,x) = 'y';
        PercentFull(19,x) = 0.6;

        x = InitBeach + 2;

        PercentFull(17,x) = 0.2;
        PercentFull(18,x) = 1.0;
        AllBeach(18,x) = 'y';
        PercentFull(19,x) = 0.2;

        x = InitBeach + 3;

        PercentFull(18,x) = 0.3;
    end

end

% Make the Gaussian coastline
if (StartWithGaussian)

    % Create the Gaussian Function
    y = 0:.1:Ymax;
    GaussianFunction = GaussMult.*(1/GaussSigma*sqrt(2*pi))
.* exp(-.5 .* ((y - GaussCenter)/GaussSigma).^2);
    GaussianCoastline = zeros(1,Ymax);

    for i = 1:Ymax
        GaussianCoastline(i) = GaussianFunction(y==i);
```

```
    end

    % Translate the function into computational arrays for
CEM
    ii = 1;     % Placekeep for GaussianCoastline

    for y = (Ymax/2)+1:(3*Ymax/2)
        xBeachPos = InitBeach + 0.5 +
GaussianCoastline(ii);
        xCell = floor(xBeachPos);

        % Create computational cells for beach cell
location
        PercentFull(y,xCell) = xBeachPos - xCell;
        AllBeach(y,xCell) = 'n';
        CellDepth(y,xCell) = -LandHeight;

        % Create computational cells for cells behind beach
        % Do this just to be sure
        PercentFull(y,1:(xCell-1)) = 1;
        AllBeach(y,1:(xCell-1)) = 'y';
        CellDepth(y,1:(xCell-1)) = -LandHeight;

        % Prepare for next iteration
        ii = ii+1;
    end

end


function [PercentFull, CellDepth, AllBeach, Age] =
CEM_InitCondsFromLine(InputLine,...

InitialDepth,InitBeach,CellWidth,ShelfSlope,LandHeight,Dept
hShoreface,...

UsePeriodicBoundaries,InitialStraigthBoundaries,Xmax,Ymax)

% CEM Function: Makes the initial conditions for CEM if
starting from an
% input coastline

% Initialize local variables
loc_debug = 0;
```

```
% Initialize overall shoreface configuation arrays
AllBeach = char(ones(2*Ymax,Xmax) * 'n');       % Flag
indicating of cell is entirely beach ('y'/'n')
PercentFull = zeros(2*Ymax,Xmax);   % Fractional amount of
shore cell full of sediment
Age = zeros(2*Ymax,Xmax);           % Age since cell was
deposited
CellDepth = zeros(2*Ymax,Xmax);     % Depth array (m)

% Set up CelLDepth first (assuming that the average beach
position is x =
% InitBeach
for y1 = (Ymax/2+1):(3*Ymax/2);
    for x1 = 1:Xmax
        CellDepth(y1,x1) = InitialDepth + ((x1-InitBeach) *
CellWidth * ShelfSlope);
    end
end

% Loop for all coastline positions
for LineCell = 1:length(InputLine)

    % Find the y (longshore) coordinate of the coastline
    y = LineCell + (Ymax/2);

    % Find the x (cross-shore) coordinate of the coastline
    x = floor(InputLine(LineCell) + 0.5) + InitBeach;

    % Fill in the configuration arrays
    % Fill in the coastline cells first
    PercentFull(y,x) = InputLine(LineCell) - x + InitBeach
+ 0.5;
    AllBeach(y,x) = 'n';
    CellDepth(y,x) = -LandHeight;

    % Fill in the beach cells
    PercentFull(y,1:x-1) = 1;
    AllBeach(y,1:x-1) = 'y';
    CellDepth(y,1:x-1) = -LandHeight;

    % Fill in the ocean cells
    PercentFull(y,x+1:Xmax) = 0;
    AllBeach(y,x+1:Xmax) = 'n';
```

```
end

% Run CellDepth Correction
for y2 = (Ymax/2+1):(3*Ymax/2);
    for x2 = 1:Xmax

        if ((CellDepth(y2,x2) < DepthShoreface) &&
(AllBeach(y2,x2) == 'n') && (AllBeach(y2,x2-1) == 'n'))
            CellDepth(y2,x2) = DepthShoreface;
        end
    end
end

% Periodic Boundary Copy
if (InitialStraigthBoundaries)

    [PercentFull, CellDepth, AllBeach, Age] =
CEM_StraightBoundaries(PercentFull,CellDepth,AllBeach,Age,Y
max);

elseif (UsePeriodicBoundaries)

    [PercentFull, CellDepth, AllBeach, Age] =
CEM_PeriodicBoundaryCopy(PercentFull,...
        CellDepth,AllBeach,Age,Xmax,Ymax);

end

% Debug plots
if (loc_debug)
    figure;
    imagesc(PercentFull);
    figure;
    imagesc(CellDepth);
end


function
CEM_InitialPert(PercentFull,AllBeach,InitBeach,InitialPert)

% Function that makes the initial perturbation in the
coastline for the
% MATLAB version of the Coastline Evolution Model (CEM)
```

```
% Establish perturbation parameters
PWidth = 4;
PHeight = 4;
PYstart = 20;

if (InitialPert == 1)
    % Square perturbation

    % Fill all beach areas
    for x = InitBeach:(InitBeach + PHeight)
        for y = PYstart:(PYstart + PWidth)
            PercentFull(y,x) = 1;
            AllBeach(y,x) = 'y';
        end
    end

    % PercenFull Top
    for y = (PYstart -1):(PYstart + PWidth +1)
        PercentFull(y,(InitBeach + PHeight + 1)) = rand(1);
    end

    % PercentFull Sides
    for x = InitBeach:(InitBeach + PHeight)
        PercentFull(PYstart-1,x) = rand(1);
        PercentFull((PYstart + PWidth + 1),x) = rand(1);
    end

elseif (InitialPert == 2)
    % Steep point perturbation

    x = InitBeach;

    PercentFull(17,x) = 0.8;
    PercentFull(18,x) = 1.0;
    AllBeach(18,x) = 'y';
    PercentFull(19,x) = 0.8;

    x = InitBeach + 1;

    PercentFull(17,x) = 0.6;
    PercentFull(18,x) = 1.0;
    AllBeach(18,x) = 'y';
    PercentFull(19,x) = 0.6;
```

```
    x = InitBeach + 2;

    PercentFull(17,x) = 0.2;
    PercentFull(18,x) = 1.0;
    AllBeach(18,x) = 'y';
    PercentFull(19,x) = 0.2;

    x = InitBeach + 3;

    PercentFull(18,x) = 0.3;
end


function [MassCurrent] =
CEM_MassCount(PercentFull,Xmax,Ymax,UsePeriodicBoundaries)

% CEM Function: Counts the total volume occupied by beach
cells;
% Uses same algorhythm as AdjustShore and returns the total
sum

% Initialize
Mass = 0;

% Determine limits
if (UsePeriodicBoundaries)
    yLow = (Ymax/2)+1;
    yHigh = (3*Ymax/2);
else
    yLow = 1;
    yHigh = 2*Ymax;
end

% Run through all cells and sum up the mass
 for x = 1:Xmax
     for y = (yLow:yHigh)

         Mass = Mass + PercentFull(y,x);

     end
 end

 MassCurrent = Mass;
```

```
function [MassBetweenPoints] =
CEM_MassCountBetweenPoints(MassCountPoints,PercentFull,Xmax
,Ymax)

% CEM Function: Counts the total volume occupied by beach
cells between
% two specified longshore points
% Uses same algorhythm as AdjustShore and returns the total
sum

% Initialize
NumRuns = length(MassCountPoints(:,1));
MassBetweenPoints = zeros(1,NumRuns);

% Run through every count point
for i = 1:NumRuns

    % Reset the mass count
    Mass = 0;

    % Determine limits
    yLow = MassCountPoints(i,1)+(Ymax/2);
    yHigh = MassCountPoints(i,2)+(Ymax/2);


    % Run through all cells and sum up the mass
    for x = 1:Xmax
        for y = (yLow:yHigh)

            Mass = Mass + PercentFull(y,x);

        end
    end

    MassBetweenPoints(i) = Mass;

end



end
```

```
function [X, Y, TotalBeachCells] =
CEM_MATLABFindBeachCells(PercentFull,AllBeach,InitBeach,Xma
x,Ymax)

% CEM Function: Finds the location of beach cells from
PercentFull.  This
% function is made specfically for the MATLAB version of
CEM.


%% Begin Function

% Find which PercentFull cells are beach cells
[row,col] = find((PercentFull > 0) & (PercentFull < 1));

% Sort the beach cells by row (y/cross-shore position)
[row_sorted,row_order] = sort(row);
col_sorted = col(row_order,:);

% Save output variables
X = col_sorted;
Y = row_sorted;

TotalBeachCells = length(Y);


% Make sure that X and Y coordinates go from left to right
in domain
% Determine the coastline
% Initialize
ShoreLength = 2*Ymax;
Coastline = zeros(1,ShoreLength);
index = 1;

% Determine the shoreline
for y = 1:2*Ymax
    x = Xmax-1;
    xtop = Xmax;

    % step back to where we encounter allbeach
    while (AllBeach(y,x) == 'n')
        x = x-1;
```

```
    end

    % if on side of shape, need to average
    if (PercentFull(y,x+2) > 0)

        xtop = x+1;

        while(PercentFull(y,xtop) > 0)
            xtop = xtop + 1;
        end

        xsave = x;

        for i = (x+1):xtop
            xsave = xsave + PercentFull(y,i);
        end

    % otherwise Regular Beach Condition
    else
        xsave = x + PercentFull(y,x+1);
    end

    Coastline(index) = xsave - InitBeach + 0.5;

    index = index + 1;

end % end shoreline determination

% Determine the derivative of the coastline wrt x
DerivCoastline = diff(Coastline);

% Go through all X & Y coordinates to determine the
location of stacked
% beaches
for i = 1:length(DerivCoastline)-1

    if (Y(i) == Y(i+1))
        % We have a stacked beach (2 beach cells with same
y coordinates)

        % Determine the direction of shoreline change in
this location
        if (DerivCoastline(Y(i)) > 0)
            % Positive derivative
```

```
            DerivPositive = 1;
        elseif (DerivCoastline(Y(i)) < 0)
            DerivPositive = 0;
        else
            fprintf('WTF?!?!  Zero derivative at a stacked
beach?!?\n');
            beep
            pause
            return
        end

        if ((DerivPositive) && (col_sorted(i) >
col_sorted(i+1)))
           % Coastline going up but X coordinates going
down

           X(i) = col_sorted(i+1);
           X(i+1) = col_sorted(i);

        elseif ((DerivPositive == 0) && (col_sorted(i) <
col_sorted(i+1)))
            % Coastline going down but X coordinates going
up

            X(i) = col_sorted(i+1);
            X(i+1) = col_sorted(i);

        else
            % Stacked coastline, but it should be in the
right order
            % Do nothing

        end

    end
end


function [PercentFull, AllBeach, CellDepth] =
CEM_OopsImEmpty(PercentFull,AllBeach,CellDepth,GroinCells,x
,y,DepthShoreface,Ymax,debug8)

% CEM function: If a cell is under-full, this will find
source for
```

```
% desparity and move brach in

% Initialize
emptycells = 0;
emptycells2 = 0;
GroinRight = 0;  % Flag indicating groin to the right of
PercentFull(y,x)
GroinLeft = 0;   % Flag indicating groin to the left of
PercentFull(y,x)

if (debug8)
    fprintf(['\n        OOPS I AM EMPTY! X: ',num2str(x),'
Y: ',...
        num2str(y),' Per: ',num2str(PercentFull(y,x))]);
end

% Account for periodic boundary conditions
if (y == 1)
    yMinus = 2*Ymax;
else
    yMinus = y-1;
end
if (y == 2*Ymax)
    yPlus = 1;
else
    yPlus = y+1;
end

% Figure out the position of groins so that no sediment is
redistrubted
% over them
if (GroinCells(y,x) == 1)
    GroinRight = 1;
end
if (GroinCells(yMinus,x) == 1)
    GroinLeft = 1;
end


% find out how many AllBeaches to take from

if (AllBeach(y,x-1) == 'y')
    emptycells = emptycells + 1;
end
```

```
if (AllBeach(y,x+1) == 'y')
    emptycells = emptycells + 1;
end
if ((AllBeach(yMinus,x) == 'y') && (GroinLeft == 0))
    emptycells = emptycells + 1;
end
if ((AllBeach(yPlus,x) == 'y') && (GroinRight == 0))
    emptycells = emptycells + 1;
end

if (emptycells > 0)

    % Move sediment

    if (AllBeach(y,x-1) == 'y')
        PercentFull(y,x-1) = PercentFull(y,x-1) +
(PercentFull(y,x)/emptycells);
        AllBeach(y,x-1) = 'n';
        if (debug8)
            fprintf(' MOVEDBACK');
        end
    end
    if (AllBeach(y,x+1) == 'y')
        PercentFull(y,x+1) = PercentFull(y,x+1) +
(PercentFull(y,x)/emptycells);
        AllBeach(y,x-1) = 'n';
        if (debug8)
            fprintf(' MOVEDUP');
        end
    end
    if ((AllBeach(yMinus,x) == 'y') && (GroinLeft == 0))
        PercentFull(yMinus,x) = PercentFull(yMinus,x) +
(PercentFull(y,x)/emptycells);
        AllBeach(yMinus,x) = 'n';
        if (debug8)
            fprintf(' MOVEDLEFT');
        end
    end
    if ((AllBeach(yPlus,x) == 'y') && (GroinRight == 0))
        PercentFull(yPlus,x) = PercentFull(yPlus,x) +
(PercentFull(y,x)/emptycells);
        AllBeach(yPlus,x) = 'n';
        if (debug8)
            fprintf(' MOVEDRIGHT');
```

```
        end
    end

    % end if (emptycells > 0)
else

    % No full neighbors, so take away from partially full
neighbors

    if (PercentFull(y,x-1) > 0)
        emptycells2 = emptycells2 + 1;
    end
    if (PercentFull(y,x+1) > 0)
        emptycells2 = emptycells2 + 1;
    end
    if ((PercentFull(yMinus,x) > 0) && (GroinLeft == 0))
        emptycells2 = emptycells2 + 1;
    end
    if ((PercentFull(yPlus,x) > 0) && (GroinRight == 0))
        emptycells2 = emptycells2 + 1;
    end

    if (emptycells2 > 0)

        if (PercentFull(y,x-1) > 0)
            PercentFull(y,x-1) = PercentFull(y,x-1) +
(PercentFull(y,x)/emptycells2);
            if (debug8)
                fprintf('   NOTFULL MOVEDDOWN');
            end
        end
        if (PercentFull(y,x+1) > 0)
            PercentFull(y,x+1) = PercentFull(y,x+1) +
(PercentFull(y,x)/emptycells2);
            if (debug8)
                fprintf('   NOTFULL MOVEDUP');
            end
        end
        if ((PercentFull(yMinus,x) > 0) && (GroinLeft ==
0))
            PercentFull(yMinus,x) = PercentFull(yMinus,x) +
(PercentFull(y,x)/emptycells2);
            if (debug8)
                fprintf('   NOTFULL MOVEDLEFT');
```

```
            end
        end
        if ((PercentFull(yPlus,x) > 0) && (GroinLeft == 0))
            PercentFull(yPlus,x) = PercentFull(yPlus,x) +
(PercentFull(y,x)/emptycells2);
            if (debug8)
                fprintf('   NOTFULL MOVEDRIGHT');
            end
        end

    else

        fprintf(['@@@ Did not find anywhere to steal sand
from!! x: ',num2str(x),' y: ',num2str(y)]);

    end

end % end else after if (emptycells > 0)


% Set the current cell to 0 percent full
AllBeach(y,x) = 'n';
PercentFull(y,x) = 0;
CellDepth(y,x) = DepthShoreface;

if (debug8)
    fprintf('\n');
end




function [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,AllBeach,CellDepth,GroinCells,x,
y,LandHeight,Ymax,debug8)

% CEM function: If a cell is overfull, push beach out in
new direction

% Set up matrcies
%%% Note: this may be unncessary, but I want to make
certain that the
%%% entire matrix is exported from the function
% PercentFull = PercentFull;
```

```
% AllBeach = AllBeach;
% CellDepth = CellDepth;

% Initialize function
fillcells = 0;
fillcells2 = 0;
GroinRight = 0;  % Flag indicating groin to the right of
PercentFull(y,x)
GroinLeft = 0;   % Flag indicating groin to the left of
PercentFull(y,x)

if (debug8)
    fprintf(['\n        OOPS I AM FULL!! X: ',num2str(x),'
Y: ',...
        num2str(y),' Per: ',num2str(PercentFull(y,x))]);
end

% Account for periodic boundary conditions
if (y == 1)
    yMinus = 2*Ymax;
else
    yMinus = y-1;
end
if (y == 2*Ymax)
    yPlus = 1;
else
    yPlus = y+1;
end

% Figure out the position of groins so that no sediment is
redistrubted
% over them
if (GroinCells(y,x) == 1)
    GroinRight = 1;
end
if (GroinCells(yMinus,x) == 1)
    GroinLeft = 1;
end

% find out how many cells will be filled up
if (PercentFull(y,x-1) == 0)
    fillcells = fillcells + 1;
end
if (PercentFull(y,x+1) == 0)
```

```
    fillcells = fillcells + 1;
end
if ((PercentFull(yMinus,x) == 0) && (GroinLeft == 0))
    fillcells = fillcells + 1;
end
if ((PercentFull(yPlus,x) == 0) && (GroinRight == 0))
    fillcells = fillcells + 1;
end

if (fillcells > 0)

    % Now Move Sediment

    if (PercentFull(y,x-1) == 0)
        PercentFull(y,x-1) = PercentFull(y,x-1) +
((PercentFull(y,x)-1)/fillcells);
        CellDepth(y,x-1);
        if (debug8)
            fprintf('  MOVEDBACK');
        end
    end
    if (PercentFull(y,x+1) == 0)
        PercentFull(y,x+1) = PercentFull(y,x+1) +
((PercentFull(y,x)-1)/fillcells);
        CellDepth(y,x+1);
        if (debug8)
            fprintf('  MOVEDUP');
        end
    end
    if ((PercentFull(yMinus,x) == 0) && (GroinLeft == 0))
        PercentFull(yMinus,x) = PercentFull(yMinus,x) +
((PercentFull(y,x)-1)/fillcells);
        CellDepth(yMinus,x);
        if (debug8)
            fprintf('  MOVEDLEFT');
        end
    end
    if ((PercentFull(yPlus,x) == 0) && (GroinRight == 0))
        PercentFull(yPlus,x) = PercentFull(yPlus,x) +
((PercentFull(y,x)-1)/fillcells);
        CellDepth(yPlus,x);
        if (debug8)
            fprintf('  MOVEDRIGHT');
        end
```

```
    end

else

    % No fully empty neighbors, so distribute to partially
full neighbors

    if (PercentFull(y,x-1) < 1)
        fillcells2 = fillcells2 + 1;
    end
    if (PercentFull(y,x+1) < 1)
        fillcells2 = fillcells2 + 1;
    end
    if ((PercentFull(yMinus,x) < 1) && (GroinLeft == 0))
        fillcells2 = fillcells2 + 1;
    end
    if ((PercentFull(yPlus,x) < 1) && (GroinRight == 0))
        fillcells2 = fillcells2 + 1;
    end

    if (fillcells2 > 0)

        if (PercentFull(y,x-1) < 1)
            PercentFull(y,x-1) =  PercentFull(y,x-1) +
((PercentFull(y,x)-1)/fillcells2);
            if (debug8)
            fprintf('  MOVEDBACK');
            end
        end
        if (PercentFull(y,x+1) < 1)
            PercentFull(y,x+1) =  PercentFull(y,x+1) +
((PercentFull(y,x)-1)/fillcells2);
            if (debug8)
            fprintf('  MOVEDUP');
            end
        end
        if ((PercentFull(yMinus,x) < 1) && (GroinLeft ==
0))
            PercentFull(yMinus,x) =  PercentFull(yMinus,x)
+ ((PercentFull(y,x)-1)/fillcells2);
            if (debug8)
            fprintf('  MOVEDLEFT');
            end
        end
```

```
        if ((PercentFull(yPlus,x) < 1) && (GroinRight ==
0))
            PercentFull(yPlus,x) =  PercentFull(yPlus,x) +
((PercentFull(y,x)-1)/fillcells2);
            if (debug8)
            fprintf('  MOVEDRIGHT');
            end
        end

    else

        if (debug8)
            fpritnf(['Nobody wants our sand!!! x:
',num2str(x),' y: ',...
                num2str(y),' Per:
',num2str(PercentFull(y,x)),'\n']);
        end
    end

end

AllBeach(y,x) = 'y';
PercentFull(y,x) = 1;
CellDepth(y,x) = -LandHeight;

if (debug8)
    fprintf('\n');
end



function [PercentFull, CellDepth, AllBeach, Age] =
CEM_PeriodicBoundaryCopy(PercentFull,CellDepth,AllBeach,Age
,Xmax,Ymax)

% Simulates periodic boundary conditions by copying middle
section to front
% and end of arrays for CEM

for y = (Ymax+1):(3*Ymax/2)
    for x = 1:Xmax

        AllBeach(y-Ymax,x) = AllBeach(y,x);
        PercentFull(y-Ymax,x)  = PercentFull(y,x);
```

```
        Age(y-Ymax,x) = Age(y,x);
        CellDepth(y-Ymax,x)  = CellDepth(y,x);

    end
end

for y = (Ymax/2)+1:Ymax
    for x = 1:Xmax

        AllBeach(y+Ymax,x) = AllBeach(y,x);
        PercentFull(y+Ymax,x) = PercentFull(y,x);
        Age(y+Ymax,x) = Age(y,x);
        CellDepth(y+Ymax,x) = CellDepth(y,x);

    end
end
```

```
function [Shoreline] =
CEM_SaveLineToFile(PercentFull,AllBeach,InitBeach,Xmax,Ymax
,CurrentTimeStep,savelinename,SaveLineASCII)

% CEM Function: Saves data line of shoreline position
rather than entire array

fprintf('\n Saving Shoreline\n');

% Determine the shoreline
[Shoreline] =
CEM_FindCoastline(PercentFull,AllBeach,InitBeach,Xmax,Ymax)
;

% Save the coastline
if (SaveLineASCII)
    SaveName = [savelinename,num2str(CurrentTimeStep)];
    save(SaveName,'Shoreline','-ascii')
else
    SaveName =
[savelinename,num2str(CurrentTimeStep),'.mat'];
    save(SaveName,'Shoreline')
end
```

```
fprintf(['Line file saved! TimeStep:
',num2str(CurrentTimeStep),'\n\n']);



function
CEM_SaveSandToFile(PercentFull,Age,CellDepth,AllBeach,Curre
ntTimeStep,SaveAge,savefilename)

% CEM Function: Saves the model data in output files

fprintf('\n saving\n');

SaveName = [savefilename,num2str(CurrentTimeStep),'.mat'];

fprintf(['Saving as :',SaveName]);

if (SaveAge)

save(SaveName,'PercentFull','Age','CellDepth','AllBeach')
else
    save(SaveName,'PercentFull','CellDepth','AllBeach')
end

fprintf('--- regular file saved! ----\n\n');



function [VolumeIn, VolumeOut] = CEM_SedTrans(From, To,
ShoreAngle, ...
    MaxT, Y, WaveAngle, Period, OffShoreWvHt, MaxDepth, ...
    RefractStep, KBreak, TimeStep, VolumeIn, VolumeOut, ...
    UseVariableCERC,K,rho,rho_s,porosity,GroinTransport,...

GroinPermeability,GroinTipDepth,DiffShadowed,DiffPhi,DiffKb
,MaxVol,...
    A_w,UseBreakDepthAsD_LT,debug6,debug40a, debug41a)

% CEM Function: This central function will calcualte the
sediment
```

```
% transported from the cell at From to the cell at To,
using the input
% ShoreAngle.  This function also contains the Ashton-
Murray Wave
% Transformation (AWT) to determine breaking wave
information.  This
% information is used to determine transport.

% Initialize
BYPDebug = 0;       % Local debugger for groin bypassing

% Coefficients
g = 9.80665;                % Acceleration due to gravity
(m/s^2)
%rho = 1020;        % kg/m3 - density of water and
dissolved matter (I will likely put this in main later)
PerSecondToPerDay = 86400;  % Multiplying a flux by this
converts a flux per second to a flux per day

% Initialize Variables
counter = 0;
Broken = 0;
radtodeg = 180/pi;          % Transform radians to degrees
degtorad = pi/180;          % Transform degrees to radians


% Use the Ashton-Murray Wave Transformation to determine
wave breaking data
% Primary assumption is that waves refract over shore-
parallel contours
% New algorithm 6/02 iteratively takes wave onshore until
they break, then computes Qs

% Initialize offshore wave conditions
StartDepth = MaxDepth;
AngleDeep = WaveAngle - ShoreAngle;
StartHeight = OffShoreWvHt;
Depth = StartDepth;

if (debug6)
    fprintf(['Wave Angle: ',num2str(WaveAngle*radtodeg),'
Shore Angle: ',...
        num2str(ShoreAngle*radtodeg),'   ']);
end
```

```
if (MaxT == 'y')
    AngleDeep = 42.0*degtorad;
end

if (debug6)
    fprintf(['Deep Transport Angle:
',num2str(AngleDeep*radtodeg),'\n\n']);
end

% Don't do calculations if over 90 degrees, should be in
shadow

if ((AngleDeep > 0.995*pi/2) || (AngleDeep < -0.995*pi/2))
    return;
else

    y_coord = Y(From);

    % Begin refraction
    % Calculate Deep Water Celerity & Length, Komar 5.11 c
= gT / pi, L = CT

    CDeep = g * Period / (2.0 * pi);
    LDeep = CDeep * Period;

    if (debug6)
        fprintf(['CDeep = ',num2str(CDeep),' LDeep =
',num2str(LDeep),'\n']);
    end

    while (Broken == 0)

        % non-iterative eqn for L, from Fenton & McKee

        Wavelength = LDeep * (tanh(
(((2*pi/Period)^2)*(Depth/g))^(.75))) ^(2/3);
        C = Wavelength / Period;

        if (debug6)
            fprintf(['DEPTH: ',num2str(Depth),' Wavelength
= ',...
                num2str(Wavelength),' C = ',num2str(C),'
']);
```

```
        end

        % Determine n = 1/2(1+2kh/tanh(kh)) Komar 5.21
        % First Calculate kh = 2 pi Depth/L  from k = 2
pi/L

        kh = pi*Depth/Wavelength;
        n = 0.5 * (1 + 2 * kh / sinh(2*kh));

        if (debug6)
            fprintf(['kh: ',num2str(kh),' n: ',num2str(n),'
']);
        end

        % Calculate angle, assuming shore parallel contours
and no conv/div
        % of rays from Komar 5.47

        Angle = asin(C/CDeep * sin(AngleDeep));

        if (debug6)
            fprintf(['Angle: ',num2str(Angle*radtodeg)]);
        end

        % Determine Wave height from refract calcs - Komar
5.49

        WvHeight = StartHeight * sqrt(abs(CDeep *
cos(AngleDeep) / (C * 2 * n * cos(Angle))));

        if (debug6)
            fprintf([' WvHeight:
',num2str(WvHeight),'\n']);
        end

        if (WvHeight > Depth*KBreak)
            Broken = 1;
            counter = 0;
            DepthBreak = Depth;
        elseif (Depth == RefractStep)
            Broken = 1;
            Depth = Depth - RefractStep;
            counter = 0;
            DepthBreak = Depth;
```

```
        else
            Depth = Depth - RefractStep;
            counter = counter + counter;
        end

    end % end while wave not broken loop

    % Account for wave diffraction
    if (DiffShadowed)
        WvHeightNoDiff = WvHeight;
        Angle = DiffPhi - ShoreAngle;
        WvHeight = DiffKb * WvHeightNoDiff;


        if (debug41a)
            fprintf(['From: ',num2str(From),' in
diffraction shadow; a_b: ',...
                num2str(Angle*radtodeg),' phi_b:
',num2str(DiffPhi*radtodeg),' SLAng: ',...
                num2str(ShoreAngle*radtodeg),' Hb:
',num2str(WvHeight),'\n']);
        end


    end

    % Now Determine Transport
    if (UseVariableCERC)
        % Use the CERC Formula

        if (debug6)
            fprintf('Using CERC formula for SedTrans\n');
        end

        VolumeAcrossBorder = abs( K * (rho * sqrt(g)) / (16
* sqrt(KBreak) * ...
            (rho_s - rho) * (1 - porosity)) *
(WvHeight^2.5)*...
            sin(2*Angle)*PerSecondToPerDay*TimeStep); %
This is in m^3 per time step

    else
        % Use default CEM equation for transport
```

```
        if (debug6)
            fprintf('Using default CEM sediment transport
formula\n');
        end

        VolumeAcrossBorder = abs(1.1 * rho * (g^(3/2)) *
(WvHeight^(5/2))...
            * cos(Angle) * sin(Angle) * TimeStep);
    end

    % Make sure that not too much volume is transported
    if (VolumeAcrossBorder > MaxVol)
        fprintf('Volume (%G) exceeded maximum limits from
%i to %i\n',VolumeAcrossBorder,From,To);
        VolumeAcrossBorder = MaxVol;
    end


    % Account for groin permeability and bypassing
    if (GroinTransport)
        % Transport is occuring over a groin

        % Find the Bypassing Coefficient
        % Calculate the depth of LST
        if (UseBreakDepthAsD_LT)
            % Use depth of wave break as D_LT
            D_LT = DepthBreak;
        else
            % Use D_LT Equation from GENESIS
            D_LT = (A_w/KBreak) * WvHeight;
        end

        % Calculate the bypassing coefficient
        if (GroinTipDepth >= D_LT)
            BypCoeff = 0;
        else
            BypCoeff = 1 - (GroinTipDepth/D_LT);
        end

        if (BYPDebug || debug40a)
           fprintf('From:%i, To:%i, D_G = %G, D_LT = %G,
BypCoeff = %G \n',From,To,GroinTipDepth,D_LT,BypCoeff);
        end
```

```
        % Calculate the fraction of sediment passing
over/through groin
        GroinF = GroinPermeability*(1-BypCoeff) + BypCoeff;

        if (BYPDebug || debug40a)
           fprintf('From:%i, To:%i, GroinF:%G
\n',From,To,GroinF);
        end

        if (GroinF >= 1)
            % All volume is either bypassed or transmitted
via permeability

            VolumeOut(From) = VolumeOut(From) +
VolumeAcrossBorder;

            VolumeIn(To) = VolumeIn(To) +
VolumeAcrossBorder;

            if (debug6)
                fprintf('All volume is either bypassed or
transmitted via permeability\n');
            end

        else
            % Transport accounting for permeability and
bypassing

            VolumeOut(From) = VolumeOut(From) +
GroinF*VolumeAcrossBorder;

            VolumeIn(To) = VolumeIn(To) +
GroinF*VolumeAcrossBorder;

            if (debug40a)
                fprintf('VolumeAcrossBorder:
%G\n',GroinF*VolumeAcrossBorder)
            end

        end

    else
        % Transport as normal
```

```
        VolumeOut(From) = VolumeOut(From) +
VolumeAcrossBorder;

        VolumeIn(To) = VolumeIn(To) + VolumeAcrossBorder;

    end


end



function [InShadow, ShadowXMax] = CEM_ShadowSweep
(InShadow,ShadowXMax,TotalBeachCells,AllBeach,PercentFull,X
max,Ymax,WaveAngle,X,Y,CurrentTimeStep,debug2)

% CEM Function: Moves along beach and tests to see if cells
are in shadow

% Find maximum extent of beach to use as a limit for shadow
searching

[ShadowXMax] = CEM_XMaxBeach(ShadowXMax,AllBeach,Xmax,Ymax)
+ 3;

if (debug2)
    fprintf(['ShadowXMax: ',num2str(ShadowXMax),'
XMaxBeach: ',...
        num2str(ShadowXMax - 3),'\n']);
end

% Determine if beach cells are in shadow
for i = 1:TotalBeachCells

    [IsInShadow] = CEM_FindIfInShadow(i, ShadowXMax,
AllBeach, PercentFull,
WaveAngle,X,Y,TotalBeachCells,CurrentTimeStep,Ymax);
    InShadow(i) = IsInShadow;

end
```

```
function [PercentFull, CellDepth, AllBeach, Age] =
CEM_StraightBoundaries(PercentFull,CellDepth,AllBeach,Age,Y
max)

% CEM Function: creates a straight coastline in the
extremities; this is
% done in lieu of a periodic boundary copy

for y = 1:(Ymax/2)

    AllBeach(y,:) = AllBeach((Ymax/2+1),:);
    PercentFull(y,:) = PercentFull((Ymax/2+1),:);
    Age(y,:) = Age((Ymax/2+1),:);
    CellDepth(y,:) = CellDepth((Ymax/2+1),:);

end

for y = (3*Ymax/2)+1:2*Ymax

    AllBeach(y,:) = AllBeach((3*Ymax/2),:);
    PercentFull(y,:) = PercentFull((3*Ymax/2),:);
    Age(y,:) = Age((3*Ymax/2),:);
    CellDepth(y,:) = CellDepth((3*Ymax/2),:);

end


% CEM_TransportSedimentSweep.m
%
% CEM Function: Sweep through cells to place transported
sediment
% Call function CEM_AdjustShore.m to move sediment.
% If cell full or overempty, call CEM_OopsImFull or
CEM_OopsImEmpty
% sweepsign added to ensure that direction of actuating
changes does not
%    produce unwanted artifacts (e.g. make sure
symmetrical)

% pause

% Determine direction of sediment trasnport sweep
if (rand(1)*2 > 1)
    sweepsign = 1;
```

```
    if (debug7)
        fprintf('L   ');
    end
else
    sweepsign = 0;
    if (debug7)
        fprintf('R   ');
    end
end

if (debug7)
    fprintf(['\n\nTransSedSweep  Ang:
',num2str(WaveAngle*radtodeg),...
        ' TimeStep: ',num2str(CurrentTimeStep),'\n']);
end

%% Run loop
for i = 1:TotalBeachCells;

    if (sweepsign == 1)
        ii = i;
    else
        ii = TotalBeachCells;
    end

    if (debug7)
        fprintf(['ii: ',num2str(ii),' sweepsign:
',num2str(sweepsign),' X: ',...
            num2str(X(i)),' Y: ',num2str(Y(i)),' VolIn:
',num2str(VolumeIn(i)),...
            ' VolOut: ',num2str(VolumeOut(i)),'\n']);
    end

%     [MassCurrent] = CEM_MassCount(PercentFull,Xmax,Ymax);
%     fprintf('i:%i, Mass before
Adjust:%G\n',i,MassCurrent/MassInitial);

    % Run Adjust Shore
    CEM_AdjustShore

%     [MassCurrent] = CEM_MassCount(PercentFull,Xmax,Ymax);
%     fprintf('i:%i, Mass After
Adjust:%G\n',i,MassCurrent/MassInitial);
```

```
    % Take care of any infinities that may arise
    %PercentFull(isinf(PercentFull)) = 1;


    x = X(ii);
    y = Y(ii);

    if (PercentFull(Y(ii),X(ii)) < 0)

        [PercentFull, AllBeach, CellDepth] =
CEM_OopsImEmpty(PercentFull,AllBeach,CellDepth,GroinCells,x
,y,DepthShoreface,Ymax,debug8);

    elseif (PercentFull(Y(ii),X(ii)) > 1)

        [PercentFull, AllBeach, CellDepth] =
CEM_OopsImFull(PercentFull,...

AllBeach,CellDepth,GroinCells,X(ii),Y(ii),LandHeight,Ymax,d
ebug8);

    end

end


function [ShadowXMax] =
CEM_XMaxBeach(ShadowXMax,AllBeach,Xmax,Ymax)

% CEM Function: Finds extent of beach in x direction.
Starts searching at
% a point 3 rows higher than input Max.  Function returns
integer value
% equal to max extent of 'allbeach'

% Initialize
xtest = ShadowXMax + 2;
ytest = 1;

while (xtest > 0)

    while (ytest < 2 *Ymax)
```

```
        if (AllBeach(ytest,xtest) == 'y')

            ShadowXMax = xtest;
            return;

        end

        ytest = ytest + 1;

    end

    ytest = 1;
    xtest = xtest - 1;

end

fprintf(['***** Should have found Xmax for shadow:
',num2str(xtest),', ',...
    num2str(ytest),'*****\n']);

ShadowXMax = Xmax;

end
```