Salt marsh modelling
Implemented on a GPU

L. Peeters

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

Salt marsh modelling

Implemented on a GPU

by

L. Peeters

in partial fulfillment of the requirements for the degree of

Master of Science
in Applied Mathematics

at the Delft University of Technology,
to be defended publicly on Friday June 29, 2018 at 11:00 AM.

Student number:	4237064	
Project duration:	September 1, 2017 – June 29, 2018	
Supervisor:	Prof. dr. ir. C. Vuik,	TU Delft
	Prof. dr. J. van de Koppel,	NIOZ
Thesis committee:	Prof. dr. ir. C. Vuik,	TU Delft
	Prof. dr. J. van de Koppel,	NIOZ
	Dr. H.M. Schuttelaars,	TU Delft

TU Delft Delft University of Technology

NIOZ Royal Netherlands Institute for Sea Research

Abstract

In this thesis, a coupled hydrodynamic, morphodynamic and plant growth model is reviewed. The goal of this model is to simulate plant colonization and channel formation on an initially flat or ascending substrate in a tidal landscape. To be able to simulate these biophysical processes on a small enough scale (while still using the same spatial scales), the model is implemented on a graphics processing unit. The focus of this thesis lies on the grid and space-discretisation and the processes involved.

In the simulation presented in this thesis, the finite difference discretisation on a collocated grid is replaced by a finite volume discretisation on a staggered grid. Besides, the current model assumes that there is a constant input of water throughout the whole computational domain and sediment deposition depends on the water height. In order to make the model more realistic, the simulation of tidal action is added. Afterwards, a transport equation is added, allowing to simulate the transport of substances. When implementing these improvements the features of a graphics processing unit need to be taken into account. In the end, the advantages and limitations of our model are discussed extensively.

Keywords- two-dimensional shallow water equations, morphodynamic, vegetation, salt marsh, staggered grid, graphics processing unit

Preface

This thesis is the last requirement to fulfil the graduation requirements for the degree Master of Science in Applied Mathematics at Delft University of Technology. The project has been conducted in cooperation with the Royal Netherlands Institute for Sea Research, also known as NIOZ, in Yerseke.

I would like to thank the people involved in this project, with special thanks to my supervisors Kees Vuik and Johan van de Koppel for guiding me and for always being willing to answer my queries. Furthermore, I want to thank Henk Schuttelaars for his time and willingness to be a member of my graduation committee.

Lotte Peeters
Delft, June 2018

Contents

List of Tables

List of Figures

Nomenclature

Physics constants

f	Coriolis parameter	$1/s$
g	acceleration of gravity	m/s^2

Other symbols

Δt	time step	s
$\Delta x, \Delta y$	grid sizes	m
Δz_b	bottom elevation change	m/s
η	vertical position of water surface	
$\mathbf{n}$	normal coordinate	$-$
$\mathbf{s}$	tangential coordinate	$-$
μ	dynamic viscosity	m^2/s
$\overline{u}$, $\overline{v}$	averaged velocity components	m/s
ρ	density of water	kg/m^3
ρ_s	sediment density	kg/m^3
τ_b	bottom shear stress	N/m^2
τ_d	drag force exerted by plants	N/m^2
A_s	amplitude sea level	m
C	Chézy roughness coefficient	$-$
c	suspended sediment concentration	kg/m^3
C_b	bottom friction coefficient	$-$
C_d	vegetation friction coefficient	$-$
D	diffusion coefficient	m^2/s
Dr	deposition rate	kg/m^2s
Er	erosion rate	kg/m^2s
f_{MOR}	morphological acceleration factor	$-$
h	water depth	m
H_{crit}	threshold value water depth	m
h_{eff}	effective water depth	m
k	vegetation height	m
M_s	mean sea level	m

n	Manning coefficient	$m^{\frac{1}{3}}/s$
n_b	stem density at the bottom	m^{-2}
p	discharge in the $x-$direction	m^2/s
q	discharge in the $y-$direction	m^2/s
R	hydraulic radius	m
T	period	s
t	time	s
u, v, w	velocity components	m/s
w_s	settling velocity	m/s
x, y, z	coordinate system	m
z_b	position of the bottom	m

1

Introduction

A salt marsh is a vegetated coastal ecosystem in the upper intertidal zone between land and open salt water or brackish water that is regularly flooded by the tides [6]. Salt marshes are characterized by an intertidal surface elevated above the unvegetated tidal flat dissected by a dense network of unvegetated tidal creeks, which supply the salt marsh with sediment and nutrients. The intertidal zone is illustrated in Figure 1.1. During flood tide, water first flows in tidal channels and creeks and then spreads over the marsh platform by overtopping channel boundaries or by entering through networks of minor channels. During ebb tide, water first drains from the platforms and then concentrates in the channels. Salt marshes have a large ecological value, as lots of animals inhabit these areas and highly adapted vegetation is found. They have an important value to human societies in that they protect the hind land from wave action and can dampen tidal waves, forming a first line of protection prior to human-build coastal defences such as dikes. For these reasons, salt marshes are conserved, protected and restored in many developed countries.

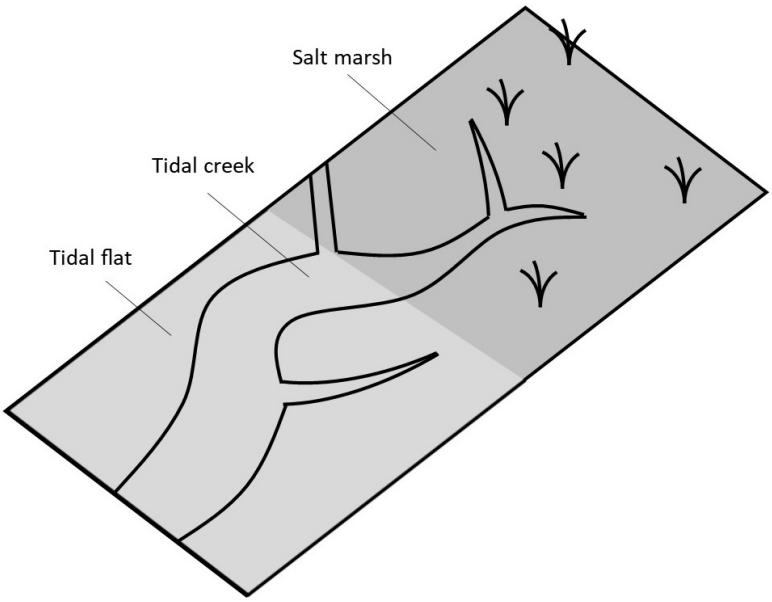


Figure 1.1: Intertidal area.

To help understand the dynamics of salt marshes and aid in their management, biogeomorphological models have been build that can describe salt marsh development. Central in these models are the interactions between vegetation growth, sedimentation-erosion processes and hydrodynamics. Biogeomorphology is a discipline that combines ecology and geomorphology. Geomorphology is the study of landforms and their formation. Ecology is the study of the relationships between biota and their

environment. To build such a model, typically a vegetation growth model is coupled to a pre-existing hydrodynamic model, such as Delft3D or Telemac that provide a detailed description of water flow and sedimentation processes. The advantage of such an approach is that a relatively accurate and well-tested hydrodynamic and morphological model is implemented with little effort, to be linked to the biological component. There are a number of important drawbacks, however. First, models such as Delft3D are computationally heavy, limiting the flexible study of biogeomorphological feedbacks in scientific contexts. Second, they are optimized for applied engineering contexts, to operate at the scales of estuaries, rivers and harbour, and are often not designed for small-scale biological-physical interactions ($< 1m$ scale) on large grids covering extensive areas ($> km$ scale). These drawbacks limit the study of biophysical processes on salt marsh evolution.

In this thesis, we implement a coupled biological-physical model of the interaction between vegetation growth, sedimentation processes and hydrodynamics using graphics processors (GPUs) as a novel, emerging computing technology. GPUs are very efficient computing units specialized in floating point operations. For relatively low costs one can obtain supercomputer performance (1 Teraflop). The specialized nature of GPUs implies that hydrodynamic models need to be programmed from scratch, using a highly parallelized programming approach. However, we aim to use their efficiency to study small-scale biophysical interactions for salt-marsh development at an ecosystem scale.
The corresponding research questions are:

How can we improve the current salt marsh model

- in terms of the grid, discretisation and solver?
- in terms of included processes?

When working on these research questions, we need to take into account the features of the graphics processing unit. When utilizing a GPU, it is for instance very important to perform as many computations in parallel as possible and to send as little data as possible back and forth between the CPU and GPU. As a consequence, not all methods are suitable to be implemented efficiently on a GPU.

1.1. Outline

Chapter 2 will derive the shallow water equations for an incompressible flow and look at the boundary and initial conditions. Chapter 3 focuses on the different model components: hydrodynamics, morphodynamics and vegetation growth. For each component, an overview of the most important processes and corresponding formulations will be given. Thereafter, in Chapter 4 we will look at the mathematical side of our model. Several spatial and time discretisations are discussed. Chapter 5 deals with the modelling on a graphics processing unit, several advantages and disadvantages will be listed. Afterwards, Chapter 6 explains the used formulations and discretisations and outlines the code. Chapter 7 will elaborate more on the specific problem we want to solve in this report. Chapter 8 discusses some computation time experiments. Afterwards, Chapter 9 to 12 will give the results to the research questions. Chapter 13 will show the results to our test problem and Chapter 14 estimates the order of accuracy of our model. In Chapter 15 the report concludes by giving a brief review of this thesis.

2

Shallow Water Equations

To model salt marsh evolution three components need to be modelled, being plant growth, sediment dynamics, and water flow. First, the shallow water equations are derived. These model water flow in areas where the water depth h is much smaller than the characteristic length of the water body, such as a salt marsh. The main characteristics are represented by this two-dimensional model. Chapter 3 discusses the plant growth and sediment dynamics and elaborates more on the hydrodynamic model. To derive the shallow water equations, we need to look at the conservation laws from which we can subsequently derive the incompressible Navier-Stokes equations. It is generally accepted that they describe natural water flows very well. However, without any simplifications they are very difficult to solve. Therefore, we make a number of assumptions, which result in the shallow-water equations.

2.1. Model Equations

A conservation law is a statement that, for any attribute which can be neither created nor destroyed but which may merely move, the total rate of outflow from a certain region must equal the rate of decrease of that attribute located within that region. The derivation of most conservation laws gives first an integral formulation that is then converted to a differential equation [7]. Equations in conservation form take the form

$$\frac{d\phi}{dt} + \nabla \cdot f(\phi) = 0$$

in which ϕ is the conserved variable. Section 4.5 gives more information regarding conservation. The equations of fluid flow are governed by conservation laws for mass, momentum and energy. We are primarily concerned with the first two, from which we can derive the incompressible Navier-Stokes equations. These are the equations of motion for a flow of a viscous fluid, which is a characteristic exhibited by all real fluids. For an inviscid fluid, no shear stresses exist when it is in motion. The Euler equations are the equations of motion for a non-viscous fluid [8].

Now two theorems are introduced, which are needed in Subsections 2.1.1 and 2.1.2. We used $\mathbf{x} = (x, y, z)$ to denote the position vector and $\mathbf{u} = (u, v, w)$ to denote the velocity field.

Theorem 2.1.1 (Reynold's transport theorem) *For any material volume $V(t)$ and differentiable scalar field ϕ, we have*

$$\frac{d}{dt}\int_{V(t)} \phi dV = \int_{V(t)} \left(\frac{\partial \phi}{\partial t} + \nabla \cdot (\phi \mathbf{u}) \right) dV \tag{2.1}$$

Theorem 2.1.2 (Divergence theorem) *Let Ω be a bounded domain in $\mathbb{R}^2$ with piecewise smooth boundary Γ. Let $\mathbf{n}$ be the unit outward normal and $\mathbf{v}$ a continuously differentiable vector field, then*

$$\int_{\Omega} \nabla \cdot \mathbf{v} = \int_{\Gamma} \mathbf{v} \cdot \mathbf{n} d\Gamma \tag{2.2}$$

2.1.1. Conservation of Mass

The mass conservation law, also known as the continuity equation, says the rate of change of mass in an arbitrary volume $V(t)$ equals the rate of mass production in $V(t)$

$$\frac{d}{dt}\int_{V(t)} \rho dV = \int_{V(t)} \sigma dV \tag{2.3}$$

where $\rho(t, \mathbf{x})$ is the density of the material particle at time t and position $\mathbf{x}$ and $\sigma(t, \mathbf{x})$ is the rate of mass production per volume. We assume σ is equal to zero (only for multiphase flows σ is non-zero, in which σ is zero holds for each flow separately) [3]. Using Equation 2.3 and Theorem 2.1 gives

$$\int_{V(t)} \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) \right) dV = 0. \tag{2.4}$$

This holds for every volume $V(t)$ so the mass conservation equation is now given by

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0. \tag{2.5}$$

An incompressible flow is a flow in which the density of each material particle remains the same during the motion

$$\rho(t, \mathbf{x}(t, \mathbf{y})) = \rho(0, \mathbf{x}(0, \mathbf{y})). \tag{2.6}$$

Incompressibility is a property of the flow and not of the fluid. It does not mean the fluid density is constant, but rather that it is independent of pressure p. Hence

$$\frac{D\rho}{Dt} \equiv \frac{\partial \rho}{\partial t} + \mathbf{u} \cdot \nabla \rho = 0. \tag{2.7}$$

We can now derive the following result for incompressible flows

$$\nabla \cdot \mathbf{u} = 0, \tag{2.8}$$

because $\nabla \cdot (\rho \mathbf{u}) = \rho \nabla \cdot \mathbf{u} + \mathbf{u} \cdot \nabla \rho$ and Equations 2.5 and 2.7 hold.

2.1.2. Conservation of Momentum

Momentum is the product of the mass and velocity of an object. Newton's law of conservation of momentum implies the rate of change of momentum and material volume equals the total force on that volume. The total force consist of the body forces ($\mathbf{f}^b$) and surface forces ($\mathbf{f}^s$).

$$\frac{d}{dt}\int_{V(t)} \rho \mathbf{u} dV = \int_{V(t)} \rho \mathbf{f}^b dV + \int_{S(t)} \mathbf{f}^s dS. \tag{2.9}$$

A body force acts on a material particle and is proportional to its mass (for example gravity, centrifugal and Coriolis forces). A surface force works on the surface of $V(t)$ and is proportional to its area. Surface forces consist of forces normal to the surface (pressure) and forces tangential to the surface (shear stresses). Substituting $\phi = \rho \mathbf{u}$ into Theorem 2.1 gives

$$\int_{V(t)} \left[\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) \right] dV = \int_{V(t)} \rho \mathbf{f}^b dV + \int_{S(t)} \mathbf{f}^s dS. \tag{2.10}$$

The surface forces $\mathbf{f}^s$ can now be substituted as follows

$$\mathbf{f}^s = T \cdot \mathbf{n} \tag{2.11}$$

where T is given by Equation 2.14 and $\mathbf{n}$ is the outward unit normal on dS [3]. Applying Theorem 2.2 to Equation 2.10 results in

$$\int_{V(t)} \left[\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) \right] dV = \int_{V(t)} \rho \mathbf{f}^b + \nabla \cdot T dV. \tag{2.12}$$

Since this holds for every $V(t)$, the momentum conservation law is given by

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \nabla \cdot T + \rho \mathbf{f}^b. \tag{2.13}$$

2.2. Assumptions

This section describes several assumptions that are made in order to derive the Navier-Stokes equations. A relation between the stress tensor and the motion of fluid is defined. Such relation is needed to complete the system of equations. Otherwise, we have fewer equations than dependent variables (underdetermined system [9]) [3]. Newtonian fluids are the simplest mathematical models of fluids that account for viscosity. While no real fluid fits the definition perfectly, many common liquids and gasses, such as water and air, can be assumed to be Newtonian. From now on we will assume water is a Newtonian fluid and thus the stress tensor T is written as [10]

$$T = -\left(p + \frac{2}{3}\mu\nabla\cdot\mathbf{u}\right)I + 2\mu D, \quad T_{ij} = -\left(p + \frac{2}{3}\mu\nabla\cdot u\right)\delta_{ij} + 2\mu D_{ij} \tag{2.14}$$

where μ is the dynamic viscosity, I is the unit tensor, p is the static pressure, δ_{ij} is Kronecker delta ($\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise) and D is the rate of strain given by

$$D = \frac{1}{2}[\nabla\mathbf{u} + (\nabla\mathbf{u})^T], \quad D_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{2.15}$$

The nine components (T_{ij}) fully describe the stress for each fluid element in motion [8]. Its diagonal components correspond to normal stresses, while the non-diagonal ones correspond to shear stresses [11]. Shear stresses arise in viscous fluids (all real fluids) as a result of the relative motion between the fluid and its boundaries or between adjacent layers of fluid. The viscous part of the stress tensor, τ, is obtained by leaving out the pressure p in Equation 2.14

$$\tau_{ij} = 2\mu D_{ij} - \frac{2}{3}\mu\delta_{ij}\text{div } \mathbf{v}. \tag{2.16}$$

Another assumption we make has to do with the density. For realistic temperature and salinity variations only small variations in density occur. Such small variations have no important consequences in most terms, so that we can just take a constant density: $\rho = \rho_0$. The only part where the density variations are important is in the gravity term ρg in the momentum equation in the z-direction. In this term, we will use the actual density. The approach of taking density variations into account only in the gravity term is termed the Boussinesq approximation and is commonly made in almost all kinds of geophysical flows [12].

Furthermore, we assume the water is well mixed. Stratification is the formation of water layers based on salinity and temperature, which influence the density. These layers are normally arranged according to density, with the least dense water masses resting above the more dense layers. In this report one layer is used. Furthermore we will assume we have an incompressible flow under which Equation 2.8 holds. Also we will assume gravity is our only body force and thus no Coriolis forces appear in the momentum equations. The Coriolis effect is the effect of the Earth's rotation which induces a force known as the Coriolis force. This force is determined by the location of the model area on the Earth's globe (the angle of latitude).

2.3. Navier-Stokes Equations

The Navier-Stokes equations are derived from the mass, momentum and energy conservation laws. As mentioned in Section 2.2 the only body force which is taken into account is the gravitational force and we assumed $T_{ij} = -p\delta_{ij} + \tau_{ij}$. Thus system 2.13 becomes

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial}{\partial x}(\rho u^2) + \frac{\partial}{\partial y}(\rho uv) + \frac{\partial}{\partial z}(\rho uw) + \frac{\partial p}{\partial x} - \frac{\partial \tau_{xx}}{\partial x} - \frac{\partial \tau_{xy}}{\partial y} - \frac{\partial \tau_{xz}}{\partial z} = 0 \tag{2.17a}$$

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial}{\partial x}(\rho uv) + \frac{\partial}{\partial y}(\rho v^2) + \frac{\partial}{\partial z}(\rho vw) + \frac{\partial p}{\partial y} - \frac{\partial \tau_{xy}}{\partial x} - \frac{\partial \tau_{yy}}{\partial y} - \frac{\partial \tau_{yz}}{\partial z} = 0 \tag{2.17b}$$

$$\frac{\partial(\rho w)}{\partial t} + \frac{\partial}{\partial x}(\rho uw) + \frac{\partial}{\partial y}(\rho vw) + \frac{\partial}{\partial z}(\rho w^2) + \rho g + \frac{\partial p}{\partial z} - \frac{\partial \tau_{xz}}{\partial x} - \frac{\partial \tau_{yz}}{\partial y} - \frac{\partial \tau_{zz}}{\partial z} = 0 \tag{2.17c}$$

where g is the acceleration due to gravity and ρ the density. The viscous stresses τ_{ij} are given by Equation 2.16. As mentioned in Section 2.2 this report focuses on incompressible flows, for which effectively the energy equation is replaced by the condition of incompressibility (Equation 2.8) For an incompressible flow, we can use Equation 2.7 to get the following momentum equations

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} + \frac{1}{\rho}\left(\frac{\partial p}{\partial x} - \frac{\partial \tau_{xx}}{\partial x} - \frac{\partial \tau_{xy}}{\partial y} - \frac{\partial \tau_{xz}}{\partial z}\right) = 0 \tag{2.18a}$$

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial v^2}{\partial y} + \frac{\partial vw}{\partial z} + \frac{1}{\rho}\left(\frac{\partial p}{\partial y} - \frac{\partial \tau_{xy}}{\partial x} - \frac{\partial \tau_{yy}}{\partial y} - \frac{\partial \tau_{yz}}{\partial z}\right) = 0 \tag{2.18b}$$

$$\frac{\partial w}{\partial t} + \frac{\partial uw}{\partial x} + \frac{\partial vw}{\partial y} + \frac{\partial w^2}{\partial z} + g + \frac{1}{\rho}\left(\frac{\partial p}{\partial z} - \frac{\partial \tau_{xz}}{\partial x} - \frac{\partial \tau_{yz}}{\partial y} - \frac{\partial \tau_{zz}}{\partial z}\right) = 0 \tag{2.18c}$$

in which the viscous stresses τ_{ij}, using Equations 2.8 and 2.16, are given by

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{2.19}$$

Thus for an incompressible flow (together with the assumptions we made in Section 2.2) the Navier-Stokes equations are given by Equations 2.18a-2.18c in combination with 2.19 and 2.8.

Reynolds Time-averaging Procedure

Although the Navier-Stokes equations are generally believed to describe turbulence, our interest will be in the large-scale features only [12]. In order to isolate those, we need to average the equations in some way. Here we assume that each variable can be split into a slowly varying ”mean” value and ”random” value

$$u = \overline{u} + u'. \tag{2.20}$$

If we substitute this splitting into the Navier-Stokes equations and take the average, we obtain the Reynolds equations for the statistical average of a turbulent flow. These have the same form as the original equations; the difference is that additional stresses, referred to as Reynolds stresses appear (turbulent part) [12], [8]. If we combine them with the viscous stresses given in Equation 2.19 we get

$$\frac{\tau_{ij}}{\rho} = \nu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \overline{u_i' u_j'}. \tag{2.21}$$

2.4. Shallow Water Equations

In a lot of areas, for example in a tidal landscape or river, the water depth is much smaller than the horizontal length scales (includes physical dimensions such as width of an estuary, horizontal scales of bottom topography, wavelength). How small the ratio of the scales should be is not easy to say [12]. Under this shallowness assumption, the Navier-Stokes equations can be simplified to the 3D shallow water equations (SWE). This derivation will be explained in Subsection 2.4.2 The 2DH SWE are derived from the 3D SWE in Subsection 2.4.3 by integrating over the depth $h(x, y, t) = \eta(x, y, t) - z_b(x, y, t)$, where η represents the vertical position of the surface water and z_b the position of the bottom as shown in Figure 2.1. For this derivation, surface and bottom boundary conditions are need.

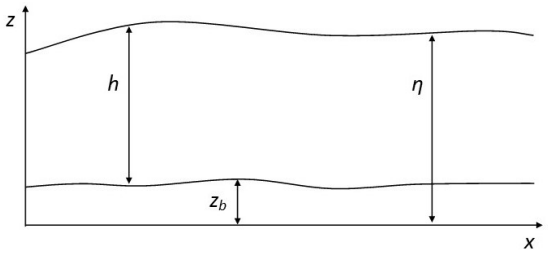


Figure 2.1: Sketch of the relevant variables.

2.4.1. Boundary Conditions

Surface and bottom conditions come in two kinds, we have kinematic conditions, which say that normal water particles will not cross the boundary (impermeability), and dynamic conditions, which say something about the forces acting at the boundaries. The necessary boundary conditions are:

- We have a no-slip condition at the bottom

$$u_{z_b} = v_{z_b} = 0 \tag{2.22}$$

  because we may assume the viscous fluid "sticks" to the bottom.

- Water particles will not cross the boundary at the bottom (no relative normal flow at the bottom)

$$\frac{\partial z_b}{\partial t} + u_{z_b}\frac{\partial z_b}{\partial x} + v_{z_b}\frac{\partial z_b}{\partial y} - w_{z_b} = 0. \tag{2.23}$$

  Using the no-slip condition at the bottom (Equation 2.22) this gives

$$\frac{\partial z_b}{\partial t} - w_{z_b} = 0. \tag{2.24}$$

- Water particles will not cross the boundary at the surface (no relative normal flow at the surface)

$$\frac{\partial \eta}{\partial t} + u_\eta\frac{\partial \eta}{\partial x} + v_\eta\frac{\partial \eta}{\partial y} - w_\eta = 0. \tag{2.25}$$

- The pressure at the surface is equal to the atmospheric pressure p_a

$$p = p_a \text{ at } z = \eta. \tag{2.26}$$

  The atmospheric pressure is the pressure within the atmosphere of the Earth.

- The surface shear stress (τ_{sx}, τ_{sy}) tangent to the water surface is defined as

$$\tau_{sx} = -\tau_{xx}\frac{\partial \eta}{\partial x} - \tau_{xy}\frac{\partial \eta}{\partial y} + \tau_{xz} \text{ at } z = \eta \tag{2.27}$$

  and similarly for the y direction.

- The bottom shear stress (τ_{bx}, τ_{by}) is defined as

$$\tau_{bx} = -\tau_{xx}\frac{\partial z_b}{\partial x} - \tau_{xy}\frac{\partial z_b}{\partial y} + \tau_{xz} \text{ at } z = z_b \tag{2.28}$$

  and similarly for the y direction.

2.4.2. Three-dimensional Shallow Water Equations

The central property in shallow water theory is that Equation 2.18c simplifies to the hydrostatic pressure distribution

$$\frac{\partial p}{\partial z} = -\rho g, \tag{2.29}$$

because all terms in Equation 2.18c are small relative to the gravitational acceleration and only the pressure gradient remains to balance it [12]. By using that at the free surface the pressure is equal to the atmospheric pressure, p_a, we find

$$p(z) = g\int_z^\eta \rho dz + p_a. \tag{2.30}$$

We focus on depth-averaged SWE and thus we assume density to be constant over the depth, which gives

$$p = \rho g(\eta - z) + p_a. \tag{2.31}$$

in which $z = 0$ represents a certain reference plane. This equation implies the pressure is larger closer to the bottom. We can use it to determine $\frac{\partial p}{\partial x}$ and $\frac{\partial p}{\partial y}$ in order to remove pressure from the momentum equations

$$\frac{\partial p}{\partial x} = \rho g \frac{\partial h}{\partial x} + g(\eta - z)\frac{\partial \rho}{\partial x} + \frac{\partial p_a}{\partial x} \tag{2.32a}$$

$$\frac{\partial p}{\partial y} = \rho g \frac{\partial h}{\partial y} + g(\eta - z)\frac{\partial \rho}{\partial y} + \frac{\partial p_a}{\partial y} \tag{2.32b}$$

The atmospheric pressure gradient $\frac{\partial p_a}{\partial x_i}$ may be important for the simulation of storm surges (tsunami-like phenomenon). This is not a goal of our model, so we will disregard this term. From now on we will use the Boussinesq assumption and take a constant density for all terms. Collecting the results we get the following 3D shallow-water equations

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{2.33a}$$

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} + g\frac{\partial h}{\partial x} + \frac{g(\eta - z)}{\rho_0}\frac{\partial \rho}{\partial x} - \frac{1}{\rho_0}\left(\frac{\partial \tau_{x,x}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right) = 0 \tag{2.33b}$$

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial v^2}{\partial y} + \frac{\partial vw}{\partial z} + g\frac{\partial \eta}{\partial y} + \frac{g(\eta - z)}{\rho_0}\frac{\partial \rho}{\partial y} - \frac{1}{\rho_0}\left(\frac{\partial \tau_{y,x}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}\right) = 0 \tag{2.33c}$$

2.4.3. Two-dimensional Depth-averaged Shallow Water Equations

This subsection derives the two-dimensional depth-averaged (2DH) shallow water equations. Therefore we need to integrate the 3D SWE (Equations 2.33a-2.33c) over the water depth $h = \eta - z_b$. It is a logical step, because under the shallowness condition, conservation of mass implies that the vertical velocity of the fluid is small [12]. Vertical integrating allows the vertical velocity to be removed from the equations. To be able to perform this derivation we thus assume η is greater than z_b. The depth-averaged velocities are given by

$$\overline{u} = \frac{1}{h}\int_{z_b}^{\eta} u dz \tag{2.34a}$$

$$\overline{v} = \frac{1}{h}\int_{z_b}^{\eta} v dz \tag{2.34b}$$

These velocities are independent of z. We will first derive the averaged continuity equation and secondly the averaged momentum equations. To integrate both we need to use Leibniz integration rule, which states

$$\frac{\partial}{\partial x}\int_{a(x)}^{b(x)} f(x,t)dt = f(x,b(x))\frac{d}{dx}b(x) - f(x,a(x))\frac{d}{dx}a(x) + \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x,t)dt. \tag{2.35}$$

Averaged Continuity Equation

Using Equation 2.35 on the first term in the continuity equation results in

$$\int_{z_b}^{\eta} \frac{\partial u}{\partial x} dz = \frac{\partial}{\partial x}\int_{z_b}^{h} u dz - u_\eta \frac{d\eta}{dx} + u_{z_b}\frac{dz_b}{dx} \tag{2.36}$$

in which u_η, v_η denote the velocities at the surface (η) and u_{z_b}, v_{z_b} at the bottom (z_b). Thus Using Leibniz integration rule (Equation 2.35) and the kinematic boundary conditions on the fluid surface and bottom (Equations 2.23 and 2.25) gives

$$\begin{aligned} 0 &= \int_{z_b}^{\eta}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right)dz \\ &= \frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(h\overline{u}) + \frac{\partial}{\partial y}(h\overline{v}) = 0 \end{aligned} \tag{2.37}$$

Averaged Momentum Equations

As a final step we need to integrate the horizontal momentum equations. By again applying Leibniz integration rule (Equation 2.35) for the first three terms we get

$$\begin{aligned}\int_{z_b}^{\eta} \frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} dz = \frac{\partial}{\partial t}\int_{z_b}^{\eta} u dz + \frac{\partial}{\partial x}\int_{z_b}^{\eta} u^2 dz + \frac{\partial}{\partial y}\int_{z_b}^{\eta} uv dz \\ + u_\eta\left(-\frac{\partial \eta}{\partial t} - u_\eta\frac{\partial \eta}{\partial x} - v_\eta\frac{\partial \eta}{\partial y} + w_\eta\right) \\ + u_{z_b}\left(\frac{\partial z_b}{\partial t} + u_{z_b}\frac{\partial z_b}{\partial x} + v_{z_b}\frac{\partial h}{\partial y} - w_{z_b}\right)\end{aligned} \tag{2.38}$$

The first term in brackets disappears due to Equation 2.25 and the second term in brackets due to Equation 2.23. The first integral is by definition equal to $\frac{\partial}{\partial t}(h\overline{u})$. In the second and third integral we get multiple nonlinear terms. For instance for the third we have

$$\int_{z_b}^{\eta} uv dz = h\overline{uv} + \int_{z_b}^{\eta} (u-\overline{u})(v-\overline{v}) dz. \tag{2.39}$$

And thus Equation 2.38 reduces to

$$\frac{\partial}{\partial t}(h\overline{u}) + \frac{\partial}{\partial x}(h\overline{u^2}) + \frac{\partial}{\partial y}(h\overline{uv}) + \int_{z_b}^{\eta} (u-\overline{u})(u-\overline{u}) dz + \int_{z_b}^{\eta} (u-\overline{u})(v-\overline{v}) dz. \tag{2.40}$$

We will now move to the averaging of the terms which are the contribution of the gradient of the pressure

$$\int_{z_b}^{\eta} g\frac{\partial \eta}{\partial x} + \frac{g}{\rho}(\eta - z)\frac{\partial \rho}{\partial z} dz = gh\frac{\partial \eta}{\partial x} + \frac{gh^2}{2\rho}\frac{\partial \rho}{\partial x} \tag{2.41}$$

Finally, the stresses are averaged. Applying Leibniz integration rule results in

$$\begin{aligned}&\int_{z_b}^{\eta}\left(\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right) dz \\ &= \frac{\partial}{\partial x}\int_{z_b}^{\eta} \tau_{xx} dz + \frac{\partial}{\partial y}\int_{z_b}^{\eta} \tau_{xy} dz - \left[\tau_{xx}\frac{\partial \eta}{\partial x} + \tau_{xy}\frac{\partial \eta}{\partial y} - \tau_{xz}\right]_{z=\eta} + \left[\tau_{xx}\frac{\partial z_b}{\partial x} + \tau_{xy}\frac{\partial z_b}{\partial y} - \tau_{xz}\right]_{z=z_b}\end{aligned}$$

The terms in brackets can be substituted by τ_{sx} and τ_{bx} due to Equation 2.27 and 2.28. The 2DH SWE are now given by

$$\frac{\partial}{\partial t}(hu) + \frac{\partial}{\partial x}(hu^2) + \frac{\partial}{\partial y}(huv) + gh\frac{\partial \eta}{\partial x} + \frac{gh^2}{2\rho}\frac{\partial \rho}{\partial x} + \frac{1}{\rho}(\tau_{bx} - \tau_{sx}) - \frac{\partial}{\partial x}(\overline{T}_{xx}) - \frac{\partial}{\partial y}(\overline{T}_{xy}) = 0 \tag{2.42a}$$

$$\frac{\partial}{\partial t}(hv) + \frac{\partial}{\partial x}(huv) + \frac{\partial}{\partial y}(hv^2) + gh\frac{\partial \eta}{\partial y} + \frac{gh^2}{2\rho}\frac{\partial \rho}{\partial y} + \frac{1}{\rho}(\tau_{by} - \tau_{sy}) - \frac{\partial}{\partial x}(\overline{T}_{xy}) - \frac{\partial}{\partial y}(\overline{T}_{yy}) = 0 \tag{2.42b}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = 0 \tag{2.42c}$$

We have omitted the overbars indicating the depth-averaged values and used $\overline{T}_{ij}$ to denote

$$\begin{aligned}\overline{T}_{ij} &= \frac{1}{\rho_0}\int_{z_b}^{\eta} \tau_{ij} dz + \int_{z_b}^{\eta} (u_i - \overline{u}_i)(u_j - \overline{u}_j) dz \\ &= \int_{z_b}^{\eta} \nu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \overline{u_i' u_j'} + (u_i - \overline{u}_i)(u_j - \overline{u}_j) dz\end{aligned} \tag{2.43}$$

The terms $\frac{1}{\rho}\tau_{sx}$ and $\frac{1}{\rho}\tau_{sy}$ are contributions of the wind stress, the shear stress exerted by the wind on the surface water [12]. In this report the winds stress is not taken into account, so we disregard these

terms. In the case of 2DH SWE the influence of the density gradient is usually small [12], so we will also disregard this term. We have now obtained

$$\frac{\partial}{\partial t}(hu) + \frac{\partial}{\partial x}(hu^2) + \frac{\partial}{\partial y}(huv) + gh\frac{\partial \eta}{\partial x} + \frac{1}{\rho}\tau_{bx} - \frac{\partial}{\partial x}(\overline{T}_{xx}) - \frac{\partial}{\partial y}(\overline{T}_{xy}) = 0 \tag{2.44a}$$

$$\frac{\partial}{\partial t}(hv) + \frac{\partial}{\partial x}(huv) + \frac{\partial}{\partial y}(hv^2) + gh\frac{\partial \eta}{\partial y} + \frac{1}{\rho}\tau_{by} - \frac{\partial}{\partial x}(\overline{T}_{xy}) - \frac{\partial}{\partial y}(\overline{T}_{yy}) = 0 \tag{2.44b}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = 0 \tag{2.44c}$$

Usually the terms belonging to $\overline{T}_{xx}, \overline{T}_{xy}, \overline{T}_{yx}$ and $\overline{T}_{yy}$ are related to local velocity gradients as follows

$$-\frac{\partial}{\partial x}(\overline{T}_{xx}) - \frac{\partial}{\partial y}(\overline{T}_{xy}) = -K_x\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.45}$$

$$-\frac{\partial}{\partial x}(\overline{T}_{xy}) - \frac{\partial}{\partial y}(\overline{T}_{yy}) = -K_y\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{2.46}$$

in which K_x and K_y represent some effective dispersion coefficients [8]. To simplify the above system, we now assume the bottom friction term relates to a sink term S and we take K_x and K_y equal to hD and substitute Equations 2.45 and 2.46

$$\frac{\partial}{\partial t}(hu) + \frac{\partial}{\partial x}(hu^2) + \frac{\partial}{\partial y}(huv) + gh\frac{\partial \eta}{\partial x} - Dh\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + Shu = 0 \tag{2.47a}$$

$$\frac{\partial}{\partial t}(hv) + \frac{\partial}{\partial x}(huv) + \frac{\partial}{\partial y}(hv^2) + gh\frac{\partial \eta}{\partial y} - Dh\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + Shv = 0 \tag{2.47b}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = 0 \tag{2.47c}$$

in which D is a diffusion term and S denotes a friction term corresponding to the bottom friction terms τ_{bx} and τ_{by}. The time-derivative is referred to as the local acceleration, while the second and third term are the convective accelerations. These three terms together are the inertia term. The fourth term is the water slope term, the fifth term is the viscosity term and the last term represents friction. To account for the influence of bottom friction we could for instance take

$$S = \frac{g}{C^2 h}\|\mathbf{u}\| \tag{2.48}$$

in which C represents the Chézy coefficient, which depends on the bottom roughness [3]. By using the averaged continuity equation given in Equation 2.47c and expanding the derivatives several terms are removed form the equations, and we obtain

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + g\frac{\partial \eta}{\partial x} - D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + Su = 0 \tag{2.49a}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + g\frac{\partial \eta}{\partial y} - D\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + Sv = 0 \tag{2.49b}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = 0 \tag{2.49c}$$

The momentum equations are now written in non-conservative form. This set of equations is equivalent to the previous set for smooth solutions, but it is important to note that the manipulations performed depend on smoothness [7]. In Section 4.5 conservation will be discussed.
Often the viscosity term (related to the lateral friction terms ($\overline{T}_{ij}$)) is also disregarded. Then the SWE are of hyperbolic type. This implies that we can think of solutions as combinations of waves [12] and the number of boundary conditions (Subsection 2.5.2) is closely related to the behaviour of characteristics. If we do not disregard the viscous terms the SWE are parabolic [13].

2.5. Boundary and Initial Conditions

To get a well-posed mathematical problem with a unique solution, a set of initial and boundary conditions need to be prescribed.

2.5.1. Initial Conditions

The plane at time t equal to zero is also a boundary of the region in the x, y, t space. For the shallow water equations the initial water levels (h), bottom elevations (z_b) and horizontal velocities (u, v) must be specified on the entire domain. The problem is that we do not know the precise initial conditions, so we have to make assumptions. Fortunately, the influence of wrong initial data gradually fades out due to wave damping by bottom friction. And if we have open boundaries that are not fully reflective we also have wave radiation into the "outside world" [12].

2.5.2. Boundary Conditions

A set of differential equations does not mean anything unless appropriate boundary conditions are specified. The number of boundary conditions is closely related to the behaviour of characteristics if the system is hyperbolic. The SWE are hyperbolic if we do not take lateral friction into account and are otherwise parabolic [13]. First, the difference between open and closed boundaries is discussed. Subsequently, the number of boundary conditions for the hyperbolic and parabolic SWE is given.

Open and Closed Boundaries

The contour of the model domain consists of parts along "land-water" lines (river banks, coastlines) which are termed closed boundaries and parts across the flow field which are termed open boundaries. Closed boundaries are natural boundaries and are situated at the transition between land and water. Open boundaries are always artificial "water-water" boundaries. They are introduced to obtain a limited computational area and so to reduce the computational effort. In nature, waves can cross these boundaries unhampered and without reflection, but any boundary condition gives a certain amount of reflection, so an open boundary may not be as open as we would wish. We would want to define boundary conditions which do not suffer from reflection, but this is in general not possible [12]. To reduce the reflections at the open boundaries a so-called weakly reflecting boundary condition may be applied. Next, the number and type of boundary conditions to get a well-posed problem are mentioned. However, we might need some additional conditions, because the number of boundary conditions is usually less than the number of unknowns.

Hyperbolic

For a hyperbolic system, the number of boundary conditions specified at any particular point of the boundary should be equal to the number of characteristics entering the region at that point [12]. The cases of zero or three boundary conditions are concerned with supercritical flow, which does not occur very often, while subcritical flow indicates one or two boundary conditions. Whether a fluid is subcritical or supercritical depends on whether the velocity is less or greater than the propagation velocity of an elementary surface wave. In other words, whether the Froude number $Fr = \frac{|U|}{\sqrt{gh}}$ is smaller or larger than unity.

We will assume that flow at the boundaries is subcritical (also done in [13]). For subcritical flow, we distinguish two situations, namely inflow and outflow. At inflow, we have to specify two boundary conditions and at outflow, we have to specify one boundary condition [13]. In a tidal flow, this situation will probably change in time between ebb and flood [12]. Summarizing, we need exactly one condition for closed boundaries, which is that the normal velocity is zero [12]. For open boundaries, the first boundary condition can be an external forcing by the water level, the normal velocity, the discharge rate (depth times velocity) or for instance a weakly reflective boundary condition [13]. Delft3D-FlOW takes as an additional condition for inflow, namely that the velocity component along the open boundary is set to zero In [13]. However, they mention it would be better to specify the tangential velocity component.

Parabolic

If we do not disregard the viscosity term, the system becomes parabolic and the theory of characteristics does not apply any more. We should provide additional boundary conditions to those mentioned before

[12]. For a closed boundary, we should now also specify whether we have a no-slip boundary for which the tangential velocity is zero

$$u_s = 0 \tag{2.50}$$

here s represents the tangential coordinate. Or a free-slip boundary, where the shear stress is equal to zero

$$\frac{\partial u_s}{\partial n} = 0 \tag{2.51}$$

here n represents the normal coordinate. On an open boundary, the additional boundary condition is less clear. In [12] the authors propose

$$\frac{\partial u_n}{\partial n} = 0 \text{ on inflow and } \frac{\partial u_s}{\partial n} = 0 \text{ on outflow}$$

3

Model Components

The processes involved in a salt marsh can be divided into three main domains: a hydrodynamic, morphodynamic and vegetation part. These parts will have to exchange information. The hydrodynamic model determines among other things the water depth h and the depth-averaged velocities u and v in the x and y direction. The morphodynamic part is concerned with calculating the bed elevation and the vegetation part with the calculation of the plant density. As illustrated in Figure 3.1 the hydrodynamics will be reviewed in Section 3.1, the morphodynamics in Section 3.2 and the plant growth model in Section 3.3.

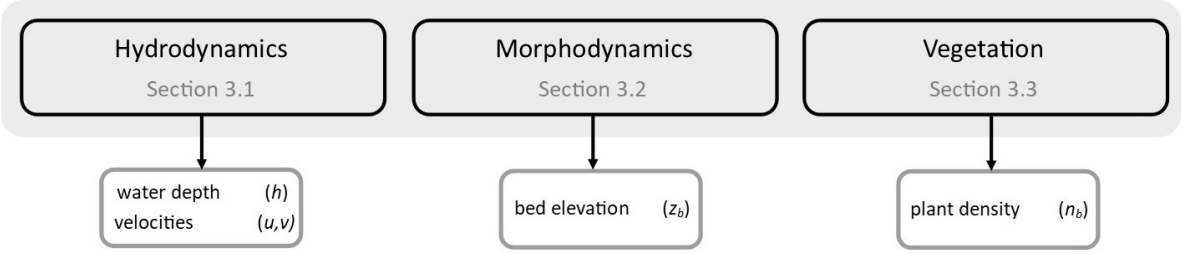


Figure 3.1: Model components.

3.1. Hydrodynamic Model

The main component of the hydrodynamic model will be the shallow water equations (Subsection 2.4), which model water flow in areas where the water depth h is much smaller than the characteristic length of the water body. In the derivation of the SWE in Subsection 2.4, we have already decided to disregard Coriolis forces, stratification and the wind stress. In this section, multiple ways to deal with the bottom friction are explained. Afterwards several wetting-drying schemes are reviewed. Wetting-drying methods are needed, because the SWE cannot automatically deal with dry areas. Additionally, the modelling of tidal action is discussed.

3.1.1. Bottom Friction

The terrain and vegetation exert shear stresses on the passing flow. The bottom stress is an unknown in the SWE and it has to be expressed in terms of the other variables. In general it is assumed that the bottom stress depends quadratically on the depth-averaged velocities [12]. We could for instance use Equation 3.1.

$$\boldsymbol{\tau}_b = \rho g \|\mathbf{u}\| \frac{1}{C^2} \tag{3.1}$$

in which C represents the Chézy coefficient. Equation 3.1 indeed implies the bottom stress has the same direction as the depth-averaged velocities and depends quadratically on its magnitude [12]. To

use the bottom stress for the sink term in the SWE (Equations 2.49a - 2.49c) given in Subsection 2.4.3 we have to divide by the water depth and density

$$S = \frac{\boldsymbol{\tau}_b}{\rho h} \tag{3.2}$$

Different Chézy coefficients exist for different kinds of flows (for instance laminar or turbulent flow). Besides, the presence of vegetation has a large impact on the bed shear stress.

No Vegetation

As mentioned before, various expressions for the Chézy coefficient exist. The Chézy formulation just takes C equal to a constant [13], while the White–Colebrook formulation is given by

$$C = 18 \log\left(\frac{12h}{k}\right) \tag{3.3}$$

in which k is the Nikuradse roughness length and h the total water depth [13]. Or we could use Manning's formulation

$$C = \frac{h^{\frac{1}{6}}}{n} \tag{3.4}$$

in which n is the Gauckler-Manning coefficient (empirically derived parameter), which is dependent on many factors including the bed roughness [13]. Often the hydraulic radius R is used in these formulations instead of the water depth h (for instance in [2]). Note that for all these formulations the smaller the water depth, the higher the bottom stress in Equation 3.1.

Vegetation

In Subsection 3.3.2 paper [2] will be discussed. This paper derives the Chézy coefficients for submergent and emergent vegetation. In case of present vegetation, the Chézy coefficients given in Equations 3.21 and 3.23 can be used in Equation 3.1 to determine the bottom stress.

3.1.2. Wetting-drying

Salt marshes are tidal areas and so they are subject to alternated wetting and drying. A major issue for the shallow water equations in coastal modeling is their inability to deal with dry areas, where the water height is theoretically zero. The role of any wetting–drying method is to facilitate the appearance and disappearance of dry areas. This section discusses ways to deal with this.

Wetting-drying methods can be classified into two main categories: the deformed mesh (Lagrangian) methods and the fixed mesh (Eulerian) methods. The deformed mesh strategy is to let the nodes on the boundary between wet and dry zones move following the front. A drawback of this method is that an important part of the model is devoted to mesh adaption. The Eulerian methods can again be divided into two main approaches: flux-limiting and modified equation methods. With the flux-limiting strategy, only the discrete algebraic form of the hydrodynamic equations is modified, while with the modified equation strategy it is the original continuous form of the partial differential equations that is modified [14]. Examples of both methods are given, again divided into four categories according to the work of [1]:

- specifying a thin film of fluid over the entire domain
- checking to see if an element or node is wet, dry or potentially one of the two, and subsequently adding or removing elements from the computational domain
- linearly extrapolating the fluid depth onto a dry element and its nodes from nearby wet elements and computing the velocities
- allowing the water surface to extend below the topographic ground surface

In Figure 3.2 a representation of each category is shown. The first three methods are all flux-limiting methods. The porosity method follows the modified equation approach. We will discuss these categories in more detail.

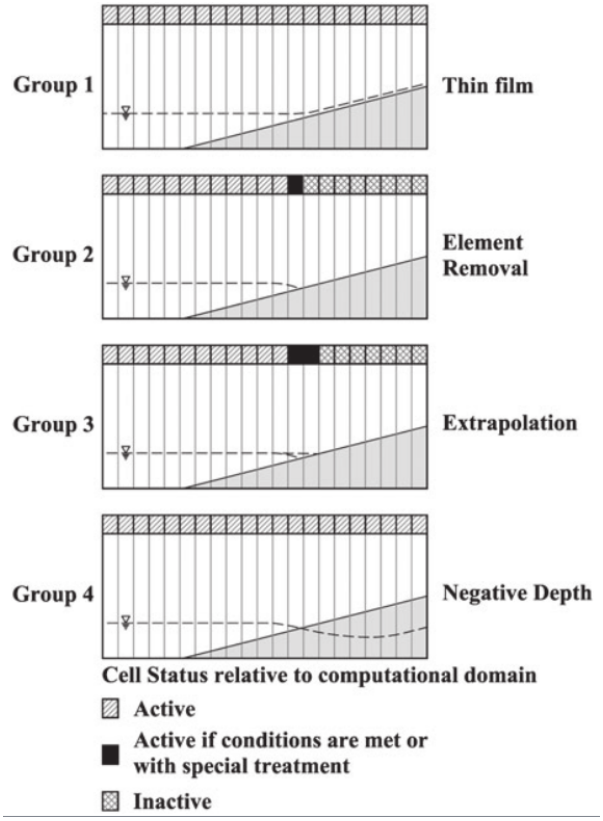


Figure 3.2: Four categories of the wetting-drying algorithm. Image taken from [1].

Thin Film Algorithms

Thin film algorithms specify a viscous sublayer of fluid over the entire computational domain. This allows all nodes to be included in the computational domain at each time step. There is typically a minimum threshold depth that defines the categories of wet or dry in the model, even though there is some fluid present over the entire domain. There are now two possibilities. Each time-step, we could update the velocities according to the SWE, after taking the maximum of a certain threshold, H_{crit}, and the water height

$$h = \max(h, H_{crit}). \tag{3.5}$$

and thus the water height will never fall below this threshold. Another possibility is to check each cells water depth, and if it drops below H_{dry} we set the velocity to zero and update the cell state to dry. Both methods require a nonzero depth in each cell.

Element Removing Algorithms

The idea now is to turn off/on mesh cells when the water thickness rises below/above a threshold value. We could also include partially wet elements besides dry and wet ones. The wet elements are included in the computational domain and the dry ones are not. However, in the case of partially wet elements, further consideration is necessary to determine whether the flow conditions at the wetting front are capable of fully wetting a partially wet element.

Depth Extrapolating Algorithms

For dpeth extrapolating algorithms, the conditions at the wetting front are given special consideration and play a vital role in advancing the water's edge in the model. In most cases, the depth is extrapolated from wet cells onto dry cells if the conditions warrant that.

Negative Depth Algorithms

The negative depth method is also known as the porosity method in which the hydrodynamic equations are modified to allow water to flow in a porous layer below the bed. The water depth can therefore be

negative and areas with negative depths are considered to be dry. The wetting of dry cells is simulated, when the flow depth increases and eventually becomes positive. This method avoids handling dry or wet cells separately, the governing equations are computed over the entire domain. However it allows unphysical water fluxes through dry zones [14].

3.1.3. Tides

The salt marsh is strongly dependent on the inundation frequency. Tides are the rise and fall of sea levels caused by the combined effects of the gravitational forces exerted by the Moon and the Sun and the rotation of the Earth. The influence of other celestial bodies is negligibly small. The most important motions for the tide are the earth's rotation around its axis (1 day), the moon's orbit around the earth ($27,32$ days), and the earth's orbit around the sun ($365,25$ days) [15]. The tidal current (horizontal tide) and the water level (vertical tide) are two appearances of the same tidal phenomenon [15].

The Netherlands experiences approximately two high and low tides each day (semi-diurnal cycle). The semi-diurnal range (the difference in height between high and low waters over about half a day) varies in a two-week cycle. A spring tides result in water levels that are higher than average and low water levels that are lower than average. Neap tides result in less extreme tidal conditions. There is about a seven-day interval between spring and neap tides [16]. Of course the sea level rise also influences the sea level. Between 1890 and 2014 the sea level along the Dutch coast steadily rose by about $1.9mm$ per year [17]. While for different places in the Netherlands the mean tidal difference approximately varies between $169cm$ and $382cm$ [18]. Thus it is probably not essential to take into account the sea level rise.

Comparing the two, the ebb surge is commonly stronger than the flood surge as the outflow of the upstream drainage area will be concentrated in the channels whereas the inflow takes place through the channels and over the entire mudflat [19].

3.2. Morphodynamic Model

The morphodynamic model will mainly be concerned with the simulation of erosion, sedimentation, bed elevation and sediment concentrations. First general information regarding sediment transport and sediment types will be given. Afterwards several formulas to model erosion, sedimentation, bed elevation and transport are explained.

3.2.1. Sediment Transport

Sediment transport is the movement of solid particles, typically due to a combination of gravity acting on the sediment, and/or the movement of the fluid in which the sediment is entrained. Three layers can be distinguished: the bed load, the suspended load and the wash load. The bed load consists of the larger sediment which is transported by saltation, rolling, and dragging on the bed. The suspended load is the middle layer that consists of the smaller sediment that is suspended. The wash load is the uppermost layer which consists of the smallest sediment that can be seen with the naked eye; however, the wash load gets easily mixed with the suspended load [20]. Most morphodynamic models simulate the transport of both the bedload and suspended load, or one of the two. The bed load is more difficult to model, because the dependence on the velocity is small.

3.2.2. Sediment Types

We can also distinguish between cohesive and non-cohesive sediment. Sediment smaller than $63\ \mu m$ is generally defined as cohesive sediment. Cohesive sediments in water form aggregated larger particles (often called flocs) through binding together (aggregation). The degree of flocculation depends on the salinity of the water [13]. These flocs are much larger than the individual sediment particles and settle at a faster rate [13]. Particles of non-cohesive sediment move individually. Mud belongs to cohesive transport, but sand and bedload belong to non-cohesive transport [13].

3.2.3. Transport Equation

A well-known transport equation is the convection-diffusion equation. Here two transport mechanisms can be distinguished. Convection is the transport due to the motion of the medium and diffusion is the

transport due to differences in concentration. The general convection-diffusion equation is given by

$$\frac{\partial c}{\partial t} + \nabla \cdot (\mathbf{u}c) - \nabla \cdot (D\nabla c) = S \tag{3.6}$$

where c is the concentration of the transported quantity, D the diffusivity, and S describes the sources and sinks, which accounts for erosion and sedimentation. The second term at the left-hand side represents convection and the third term diffusion. Suspended sediments are transported across the shallow areas adjacent to channel networks mainly by advection, even though dispersion processes driven by concentration gradients become important as flow velocities decrease [21]. Assumed is S is given by the erosion rate (Er) minus the deposition ration (Dr). These will be explained in Subsection 3.2.4.

In this report, the focus is on the two-dimensional SWE and therefore do not determine the velocity in the z direction. That is why we are interested in a two-dimensional sediment transport model. The two-dimensional sediment transport equation for the depth-averaged suspended-load concentration $c = c(x, y, t)$ is obtained by integrating the 3D sediment transport equation over the suspended-load zone.

$$\frac{\partial (hc)}{\partial t} + \nabla \cdot (h\mathbf{u}c) - \nabla \cdot (hD\nabla c) = Er - Dr \tag{3.7}$$

in which h represents the water depth, D the diffusion coefficient and S equal to E_r - D_r the source sink term [20], which again accounts for erosion and sedimentation.

3.2.4. Sedimentation and Erosion

The amount of erosion and sedimentation determines the bed elevation. Erosion is the action of surface processes (such as water flow or wind) that removes soil, rock or dissolved material from one location, and then transport it away to another location. Sedimentation is the tendency for particles in suspension to settle out of the fluid in which they are entrained and come to rest against a barrier.

In general, the Partheniades-Krone formulation is used for cohesive sediments [20]. In this formulation the erosion rate and sedimentation rate are modeled as functions of the bottom shear stress

$$Dr = w_s C_b \left(1 - \frac{\tau}{\tau_{cr,d}}\right) \quad \text{when } \tau < \tau_{cr,d} \tag{3.8}$$

$$Er = M \left(\frac{\tau}{\tau_{cr,e}} - 1\right) \quad \text{when } \tau > \tau_{cr,e} \tag{3.9}$$

where C_b is the suspended sediment concentration at the bottom, τ the bottom shear stress, $\tau_{cr,d}$ the critical shear stress for sedimentation, $\tau_{cr,e}$ the critical shear stress for erosion, M the erosion parameter and w_s the settling velocity [22]. At a vegetated marsh platform, we often have the case that τ is smaller than $\tau_{cr,e}$, so that erosion may be neglected [22]. This is caused by an increase of shear strength of the sediment bed, due to the roots of salt marsh vegetation. The bed elevation change as a result of erosion and sedimentation is now given by

$$\Delta z_b = \frac{Dr - Er}{\rho_s} \tag{3.10}$$

where Δz_b is bottom elevation change (ms^{-1}) and ρ_s is the bulk density of bottom sediment. In [23] also the bed porosity n (fraction of the volume of voids over the total volume) was taken into account, such that Equation 3.10 transforms to Equation 3.11

$$(1 - n)\Delta z_b = \frac{Dr - Er}{\rho_s} \tag{3.11}$$

3.3. Vegetation

The vegetation distribution of a salt marsh is patchy at the edges to get more dense and uniform closer to land [24]. The presence of vegetation generally results in an enhanced sediment trapping and enhanced accretion rates. The sediment is deposited along the channels forming a somewhat elevated edge [24]. This section presents a vegetation-growth model. Afterwards, the hydrodynamic roughness caused by the vegetation is discussed.

3.3.1. Plant Growth Model

We will review the vegetation-growth model based on the method developed by [5] which simulates spatial-temporal changes of the Spartina anglica, a species of cordgrass which is frequently found in coastal salt marshes. This method, describing the vegetation density, can be classified as a population dynamics model. These type of models consider the change of a population per unit of time. The growth model is the sum of five factors: the initial plant establishment, the lateral expansion, the growth, mortality due to flow stress and mortality due to inundation height. The variable n_b is used to denote the stem density at the bottom. Each factor is now discussed in detail.

- The colonization of vegetation on bare tidal flats starts with the establishment of vegetation stems which could, under the right conditions, develop into tussocks or patches of vegetation. The initial plant establishment is given by simple stochastical formulation

$$P_{est} n_{b,est} \tag{3.12}$$

  where P_{est} is the chance of plant establishment and $n_{b,est}$ the stem density of a new established tussock.

- The lateral expansion of plants to neighbouring grid cells is given by

$$D\left(\frac{\partial^2 n_b}{\partial x^2} + \frac{\partial^2 n_b}{\partial y^2}\right) \tag{3.13}$$

  where D is the plant diffusion coefficient. The magnitude of the expansion is related to the diffusion coefficient, D, and the difference between the stem densities in the tussock and the adjacent area.

- The stem density growth or population growth is the process that describes the vegetation density increase within a patch of vegetation in time. The growth of stem density within a cell up to its maximum carrying capacity K is given by a logistic growth function

$$r\left(1 - \frac{n_b}{K}\right) n_b \tag{3.14}$$

  where r is the intrinsic growth rate of stem density. A logistic growth function is characterised by an initial and final period where the stem density increase is more gradual, and a middle period with a more rapid increase.

- Calm flow conditions are needed in order for vegetation to successfully establish or expand. The plant mortality by tidal flow stress is given by

$$PE_\tau(\tau - \tau_{\sigma,p}) \text{ when } \tau > \tau_{cr,p} \tag{3.15}$$

  where PE_τ is the plant mortality coefficient related to flow stress, τ is the bottom shear stress and $\tau_{cr,p}$ is the critical shear stress for plant mortality.

- Vegetation needs sufficient time for photosynthesis. This requirement can be represented by a vegetation density decay function that considers vegetation loss due to inundation. The plant mortality caused by tidal inundation stress is given by

$$PE_H(H - H_{cr,p}) \text{ when } H > H_{cr,p} \tag{3.16}$$

  where PE_H is the plant mortality coefficient related to inundation stress, H the inundation height at high tide and $H_{cr,p}$ critical inundation height for plant mortality.

The two mortality functions can cause mortality even if there is little or no vegetation present, which can result in a negative vegetation density. So after updating the vegetation, the maximum between the vegetation density and zero should be taken. Of course, there are other options to model the plant growth.

3.3.2. Friction

This subsection looks at the relationship between flow resistance and the presence of vegetation. The terrain and vegetation exert shear stresses on the passing flow. The magnitude of the shear stress of the bed is often characterised by means of roughness coefficient. Still, much research is done to define the relationship between flow resistance and the presence and spatial distribution of vegetation. What we want is a relation between the vegetation characteristics, bed resistance, water depth and equivalent resistance coefficient.

Background

Early measurements (18th century) of flow velocities in channels revealed that the depth-averaged flow velocity $\overline{u}$ was a function of the water level slope i and the hydraulic radius R. The well-known Chézy formula is

$$\overline{u} = C\sqrt{Ri}. \tag{3.17}$$

Here C expresses the hydraulic roughness (Chézy roughness coefficient), a measure of the amount of the frictional resistance water experiences from bed and banks [2]. This parameter was first thought to be a constant. Note that the higher the Chézy value, the lower the roughness or resistance to flow is. This traditional approach of using a single resistance coefficient fails to correctly describe the physics of the phenomenon. One way of improving upon this description is to update the equivalent resistance coefficient based on the computed water depth. Further investigations revealed the following formulas for the Chézy value: those of Strickler, Manning or White-Colebrook. These were explained in Subsection 3.1.1.

But still, in all these single roughness equations, vegetation is treated as large bed structures with a logarithmic flow profile above them [2]. While in reality there is also flow through the submerged vegetation. Actually, four distinct zones can be identified as can be seen in Figure 3.3. In the first

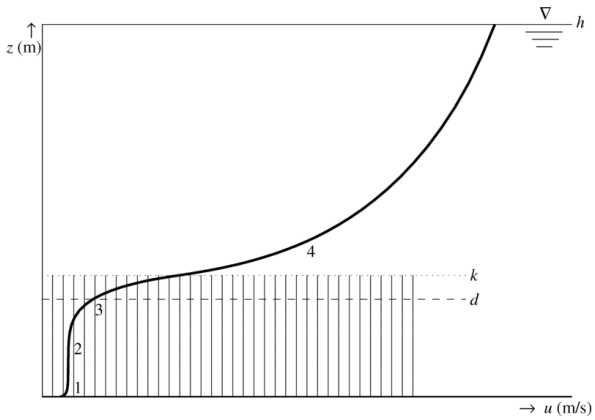


Figure 3.3: Four zones in the vertical profile for horizontal velocity $u(z)$ through and over vegetation. Image taken from [2].

zone, the velocity is highly influenced by the bed and its vertical profile joins the logarithmic boundary layer profile. The second zone, in the vegetation, corresponds to a uniform velocity. While in the third zone there is a transitional profile between the zones two and four, where also a logarithmic profile is observed. The White-Colebrook, Strickler and Manning formulas are not correct and another type of resistance formula is needed. In paper [2] analytical expressions for C are found, which depend on the vegetation height k and vertical density. Two cases are considered: fully submerged vegetation or non-submerged (emergent) vegetation. In these expressions, the solidity A_p (fraction of horizontal area taken by cylinders) which can be used to correct for the available volume, or available horizontal area in the calculation of fluid stress, is not take into account, because experimental evidence has shown it can be disregarded [2].

Emergent Vegetation

For emergent vegetation the total fluid shear stress ($\boldsymbol{\tau}_t$) is equal to the sum of the bottom shear stress ($\boldsymbol{\tau}_b$) and the drag force exerted by plants ($\boldsymbol{\tau}_d$) [2]

$$\boldsymbol{\tau}_t = \boldsymbol{\tau}_b + \boldsymbol{\tau}_d. \tag{3.18}$$

The total fluid shear stress is defined by

$$\boldsymbol{\tau}_t = \rho g h i \tag{3.19}$$

in which i is the water level slope, g the gravitational acceleration, h the water depth, ρ the fluid density ($1000\ kgm^{-3}$), $\mathbf{u}$ the depth averaged velocities and C the Chézy coefficient [2], [20]. The bottom shear stress $\boldsymbol{\tau}_b$ is given in Equation 3.1 and $\boldsymbol{\tau}_d$ is given by Baptist formula

$$\boldsymbol{\tau}_d = \frac{1}{2}\rho C_D m D h \|\mathbf{u}\| \tag{3.20}$$

here m represents the number of cylinders per m^2 horizontal area, D the cylinder diameter, h the water depth and C_D is the dimensionless bulk drag coefficient for flow through vegetation. According to [2], the bulk drag coefficient C_D can be seen as a function of, among other things, the spatial arrangement of the rigid cylinders. This formula considers plants as rigid cylinders with uniform properties and therefore mDh represents the surface of the cylinders. From Equation 3.18 the representative Chézy value for non-submerged vegetation can be deduced

$$C = \sqrt{\frac{1}{\frac{1}{C_b^2} + \frac{C_D m D h}{2g}}} \tag{3.21}$$

in which C_b represents the Chézy coefficient of the bed. Note that if we do not have any vegetation (m is equal to zero) the Chézy coefficient is equal to the Chézy coefficient of the bed. When the bed resistance is negligible with respect to the drag force of the vegetation (when we have tall and dense enough vegetation or a very small bed roughness), Equation 3.21 reduces to

$$C = \sqrt{\frac{2g}{C_D m D h}} \tag{3.22}$$

These formulas can be used in Equation 3.1 to determine the bottom friction when taking into account both the vegetation drag and bed shear stress.

Submergent Vegetation

For submerged conditions, the derivation of an analytical equation for vegetation resistance proves to be more difficult. The four zones in the velocity profile can be reduced to a model of the two most important flow zones. This model, as illustrated in Figure 3.4, consists of a uniform flow velocity, u_c, inside the vegetation and a logarithmic flow profile, u_u, above the vegetation. The Chézy value for

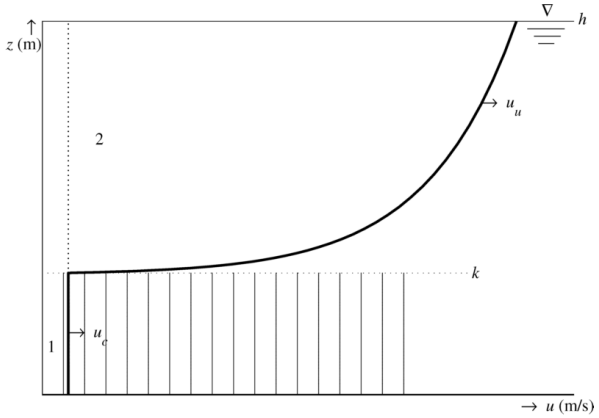


Figure 3.4: Two zones in the vertical profile for horizontal velocity $u(z)$ through and over vegetation. Image taken from [2].

submerged vegetation is then given by

$$C = \sqrt{\frac{1}{\frac{1}{C_b^2} + \frac{C_D m D k}{2g}}} + \sqrt{\frac{g}{k_0}} \ln\left(\frac{h}{k}\right) \tag{3.23}$$

in which k denotes the vegetation height and k_0 represents the Von Karman constant (0.4) [25]. Note that the first term on the right-hand side equals the representative roughness of emergent vegetation when the vegetation height is equal to the water depth (h is equal to k) and the second term goes to zero at the transition from submerged to emerged vegetation.

These models only consider rigid vegetation elements, when in reality vegetation is flexible and thus gives rise to complex interactions between flow and vegetation structures [22]. Recent studies showed that the underlying assumption of Equation 3.23, the squared relationship between drag force and flow velocity derived for rigid stems, does not hold for flexible vegetation [25]. Furthermore, the complex vertical structure of real vegetation is neglected. In fact, marsh vegetation may be less dense at low height, where the main stem is located, and denser where the plant structure branches out [22].

The plant growth model of Section 3.3.1 determines the vegetation density n_b. For rigid cylinders, the area occupied by stems in an area of one square meter, in other words the stem density, is given by $\frac{1}{4}mD^2\pi$. Thus we can use

$$mD = \frac{4n_b}{\pi D} \tag{3.24}$$

to determine mD, which can be used in the derived friction coefficients (given in Equations 3.21 and 3.23).

4

Grid and Discretisation

This chapter is concerned with the possible grids, space discretisations and time-integration methods. Additionally, multi-(time-)scale models and the concept of conservation are discussed.

4.1. Grid

Discretisation always entails subdivision of the domain into small cells. The resulting set of cells is called the computational grid. There are basically three types of grids: Cartesian, structured boundary fitted and unstructured. The different grids are illustrated in Figure 4.1. In a Cartesian grid, the cells are rectangular. A grid is referred to as structured if all interior cell vertices belong to the same number of cells. Cartesian and structured boundary-fitted grids are therefore structured. If the boundary consists of cell faces then the grid is boundary-fitted [3]. In Cartesian grids the boundary usually does not coincide with the cell faces. The advantage of structured grids is that data structures are easier,

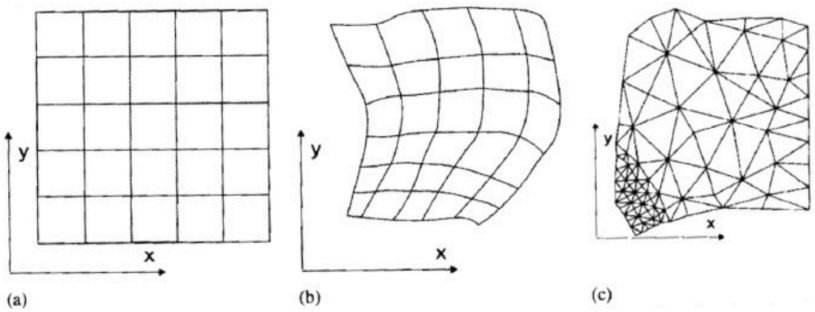


Figure 4.1: Three types of grids: (a) Cartesian; (b) structured boundary fitted; (c) unstructured. Image taken from [3].

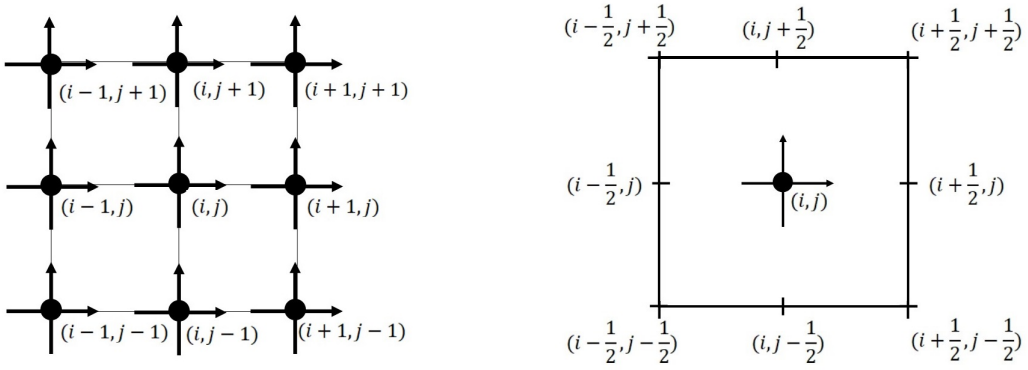
leading to smaller computing times and suitability for GPUs. On the other hand in complicated domains, the construction of unstructured grids is easier and requires far less computation time [3]. Another disadvantage of structured grids is that it may be difficult to control the distribution of the grid points: concentration of points in one region for reasons of accuracy produces unnecessarily small spacing in other parts of the solution domain and a waste of resources [10].

4.1.1. Collocated and Staggered Grids

For the placement of dependent variables we can use a collocated or staggered approach. In the 2DH SWE given in Equations 2.49a- 2.49c the dependent variables are the depth-averaged velocities u, v and the water depth h. When all variables are stored in the same positions, we have a collocated grid arrangement. In a staggered grid, not all quantities are defined at the same location in the numerical grid.

The Arakawa A, B, C, D and E grids are well-known grids. The collocated grid (Arakawa A) is illustrated in Figure 4.2. Here i and j are the indices of the grid points in the $x-$ and in the $y-$ direction. All

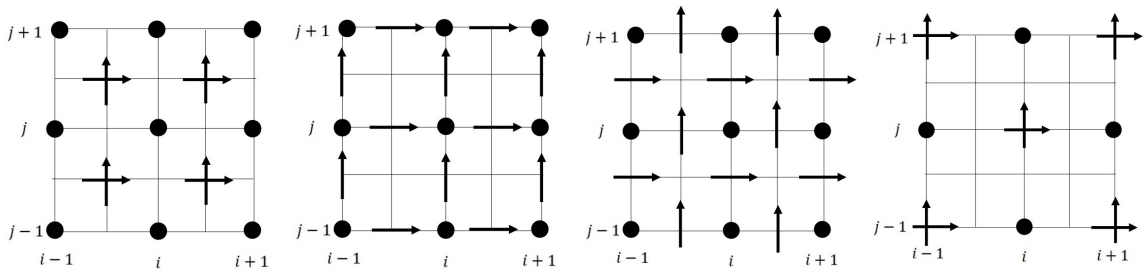
dependent variables are stored at the same grid points. The Arakawa B, C, D and E grids are



(a) Part of grid. (b) Single grid cell.

Figure 4.2: Collocated placement of unknowns; →,↑: velocity components; •: water depth.

illustrated in Figure 4.3. The Arakawa B grid is obtained by taking the flow velocities u and v in the corner points and the water depth h in the cell centres of the grid cell (i,j). The Arakawa C grid is obtained by taking the flow velocities u at grid points $(i \pm \frac{1}{2}, j)$, v at $(i, j \pm \frac{1}{2})$ and water depth h at (i,j), etc. Here we explain the standard Arakawa grids.



(a) Arakawa B (b) Arakawa C (c) Arakawa D (d) Arakawa E

Figure 4.3: Staggered placement of unknowns; →,↑: velocity components; •: water depth.

An advantage of a staggered grid is that for some cases it is possible to use a smaller number of discrete state variables in comparison to discretisations on non-staggered grids and obtain the same accuracy. Because for for certain equations and discretisations, one can choose just one of the grids (the four possible grids are all equivalent), while the numerical solution that belongs to that grid is just as accurate as the numerical solution that belongs to the original grid. However, the number of computational values is then decreased by a factor of 4. This will result in a smaller computation time, although compared to collocated grids, we probably need extra averaging operations to determine the variables in grid points where they are not defined [12]).
A second advantage is that staggered grids for shallow water solvers prevent spatial oscillations in the water levels [13]. Odd-even decoupling is a discretization error that can occur on collocated grids and which leads to checkerboard patterns in the solutions. The odd-even ordering of grid points is also called the red-black ordering. A grid point $(x_i,\ y_j)$ is odd or even if the sum of i and j is odd or even. Odd-even decoupling means that a variable in an odd point is only coupled with variables in even points and vice versa. Using a staggered grid is a simple way to avoid odd-even decoupling between the water depth and the velocity. But staggered grids also have disadvantages. The dependent variables are stored at different places and this makes it more difficult to handle the control volumes for these variables.

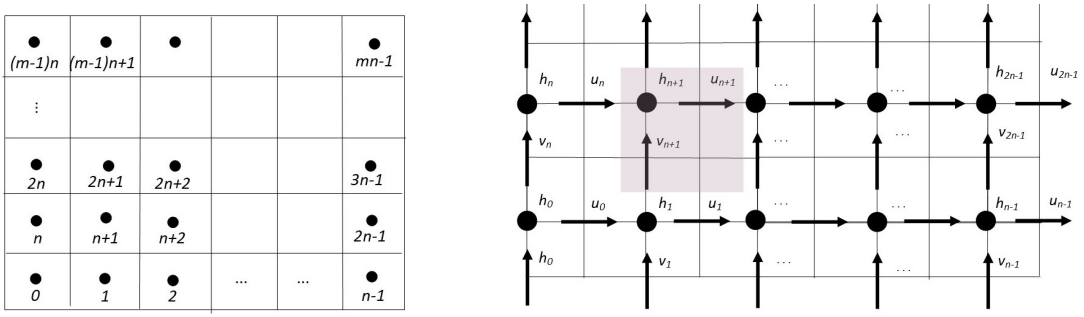
Ordering of Unknowns

We denote the approximation of the variable u at grid point (x_i, y_j) by $u_{i,j}$. This notation is used for all variables. We will now convert the double index (i,j) into a single index k. Then the approximation

of the variable u at grid point x_k is denoted by by u_k. This converting can be done in a number of ways: horizontal numbering, vertical numbering and oblique numbering [26]. For the horizontal numbering the nodes are numbered sequentially in the horizontal direction, the vertical numbering numbers the nodes sequentially in the vertical direction and for oblique numbering, the nodes are numbered sequentially along the lines $i + j = k$. In this report a horizontal numbering (Equation 4.1) is used

$$k = i + (j - 1)n \tag{4.1}$$

in which $i \in \{0, \dots, n-1\}$, $j \in \{0, \dots, m-1\}$. Here n represents the number of grid points in the horizontal direction and m the number of grid points in the vertical direction. An example is shown in Figure 4.4a. The dependent variables of the SWE located in control volume k are now denoted by: u_k, v_k and h_k.



(a) Collocated grid in which • represents the location of all unknowns.

(b) Staggered grid (Arakawa C) in which →, ↑: velocity components; •: water depth.

Figure 4.4: Horizontal numbering of unknowns.

For the staggered grid we have to pay more attention to the ordering. In Figure 4.4b an example of a staggered grid and the corresponding numbering is given. The grey box indicates which velocity components in x- and y-direction and depth have the same (array) number in the computational grid. The variables located at →, ↑ and • have different control volumes (CV). These are illustrated in Figure 4.5.

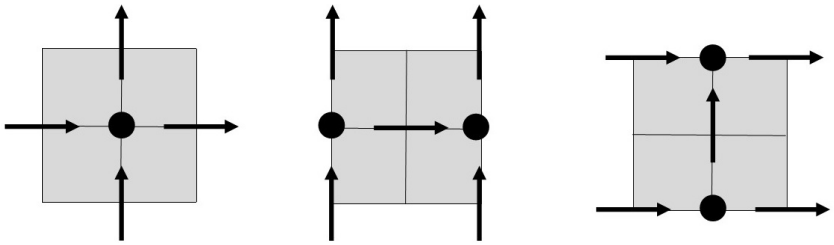


Figure 4.5: Control volume for • (left), → (middle) and ↑ (right) in which →, ↑: velocity components; •: water depth.

4.2. Discretisation

The most used mesh methods are the finite difference method (FDM), the finite volume method (FVM) and the finite element method (FEM). We will briefly explain them in this section.

Finite Difference Method

Finite difference methods (FDM) approximate spatial derivatives by means of finite differences. Taylor series expansion is used to obtain these approximations. The result is one algebraic equation per grid node, in which the variable value at that grid point and a certain number of neighbour nodes appear as unknowns. Two major disadvantages of this method are that it is not clear how to proceed with non-equidistant grids and natural boundary conditions are hard to implement [26].

Finite Volume method

The finite volume method (FVM) does not possess the major disadvantages of the finite difference method. The FVM can accommodate any type of grid, so it is suitable for complex geometries [10]. The solution domain is subdivided into a finite number of contiguous control volumes (CVs), and it calculates the values of the conserved variables averaged across the volume. Therefore the volume integrals in a partial differential equation that contain a divergence term are converted to surface integrals, using the divergence theorem. These terms are then evaluated as fluxes at the surfaces of each finite volume. Because the flux entering a given volume is identical to that leaving the adjacent volume, these methods are conservative. Compared to the FDM and FEM only the FVM has an advantage of guaranteeing mass conservation.

Finite Element Method

The FEM is similar to the FVM in many ways. The domain is broken into a finite set of non-overlapping volumes or finite elements that are generally unstructured; in 2D, they are usually triangles or quadrilaterals, while in 3D tetrahedra or hexahedra are most often used [10]. The distinguishing feature of FEMs is that the equations are multiplied by a weight function before they are integrated over the entire domain. In the simplest FEMs, the solution is approximated by a linear shape function within each element in a way that guarantees continuity of the solution across element boundaries [10]. An important advantage of FEM is that it is well suited for unstructured grids. Besides, all information in one element is used without considering neighbours. This makes the method very attractive for computer implementation. Another advantage of FEM is that the treatment of boundaries is almost always very natural [26]. A disadvantage is that this method is not conservative and not easy to make conservative. The FDM is also not conservative, but easier to adapt.

4.3. Time Integration

When using these spatial discretisation methods time is kept continuous, we get a semi-discrete system of ordinary differential equations

$$M\frac{d\mathbf{u}}{dt} + A\mathbf{u} = 0 \tag{4.2}$$

where $\mathbf{u}$ contains all unknowns. Several standard methods exist to integrate these systems. In this section, the difference between explicit and implicit discretisations is discussed.

4.3.1. Explicit

When a direct computation of the dependent variables can be made in terms of known quantities, the computation is said to be explicit. A well known explicit one-step method is the forward Euler method

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t f(t^n, \mathbf{u}^n) = 0 \tag{4.3}$$

in which f is a function dependent of the solution $\mathbf{u}$ at time n. In general, for these methods, the time step Δt and step size Δx can not be chosen independently. There is a criterion for the time step, called the Courant, Freidrichs and Lewy (CFL) condition, which represents a condition for stability.

CFL condition

The CFL condition is a necessary condition that must be satisfied by any finite volume or finite difference method if we expect it to be stable and converge to the solution of the differential equation as the grid is refined. It simply states that the method must be used in such a way that information has a chance to propagate at the correct physical speeds. In other words that the numerical solution is determined only by the point sources that physically have an influence on the solution [26]. The numerical domain of dependence can be defined as the set of points where the initial data can possibly affect the numerical solution at a certain point. If we refine both Δt and Δx, the numerical approximation to a certain point, now depends on more points, but it is the same interval as before.

For a parabolic equation such as the diffusion equation the CFL condition places more severe constraints on an explicit method. A better way to satisfy the CFL condition, in this case, is to use an implicit method. Then the numerical domain of dependence is the entire domain, since all grid points are coupled together. The next subsection will discuss implicit time-integration methods.

4.3.2. Implicit

All explicit methods suffer from the CFL condition, which requires relatively small time-steps to be taken. To avoid such restrictions, implicit methods such as the Crank-Nicolson method can be used. When the dependent variables are defined by coupled sets of equations the numerical method is said to be implicit. Implicit methods tend to sidestep the CFL condition by using the entire numerical domain, which obviously includes the true limited domain-of-dependence.

4.3.3. Semi-implicit

An example of a semi-implicit method is the semi-implicit Euler. This methods only treats some of the terms implicitly. This method remains effectively explicit (we do not need to solve systems of equations at each time-step), but the values of the new time-level are used in subsequent equations. So the system 4.3 then changes to

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t f(t^n, \mathbf{u}^n, \mathbf{u}^{n+1}) = 0 \tag{4.4}$$

If the updated or old value of u is used, depends on the fact if that value was already updated. Regarding the CFL condition, semi-implicit methods have a larger domain of dependence, and thus the CFL condition might be less restrictive.

4.3.4. Alternating Direction Implicit

Fully implicit methods have the disadvantage that a large system of algebraic equations has to be solved at each time-step. Another method to avoid this is the alternating direction implicit (ADI) method. In order to reduce computing time, the implicit scheme may be approximated by stages that are implicit in one direction only. The time step is divided in two time stages, where each stage consists of half a time step. In each part, only tridiagonal systems have to be solved, along horizontal or vertical grid lines, which is considerably cheap. These methods are widely used for the approximation of shallow water equations [3]. Powerful modelling software, like Delft3D-FLOW and Mike ([27]) still use this time-integration method. Nowadays computers have more computing power and thus it is not as useful anymore to use ADI methods.

4.4. Multi-(time-)scale

Solving all processes on the same time and spatial scales could lead to an incredibly large computation time. Therefore different scales can be used to solve certain processes. Afterwards, the coupling of those processes needs to be taken into account, because the exchange of information will probably be an issue.

4.4.1. Time

We could decide to use a certain acceleration factor in the morphological and vegetation equations to speed up these processes. The morphological acceleration factor (morfac), f_{MOR}, is used to assist in dealing with the difference in time-scales between hydrodynamic and morphological developments. This approach allows decoupling both timescales by upscaling the bottom elevation changes by a constant factor f_{MOR}, thereby effectively extending the morphological time step

$$\Delta t_{morphology} = f_{MOR} \Delta t_{hydrodynamic}. \tag{4.5}$$

Using this technique, a hydrodynamic simulation of period T corresponds to a geomorphic simulation of period $f_{MOR}T$, allowing for longer time geomorphic simulations with no extra computational cost. However, this approach assumes the bottom elevation changes are linear, which is only valid when the morfac is not too high. The selection of a suitable morphological acceleration factor remains a matter of judgement. For a fixed hydrodynamic timestep, larger morfac values reduce the simulation time, but at the same time, a large morfac may lead to an unrealistic morphological change of the model. It is important to find a balance between the low computation costs (large morfac value) and acceptable error. If we want a less complex method, we could just upscale the bottom elevation changes each iteration, without extending the timestep. This method will not result in a reduction of computation time.

There is also a big difference in the time-scales between the hydrodynamic and vegetation models. The hydrodynamic processes have a much finer temporal resolution (typically seconds) than the vegetation dynamics (from months to years). Therefore, it could be beneficial to run the hydrodynamic, geomorphic and vegetation models computations all in separate time domains, and again exchange information at specific moments.

4.4.2. Spatial

The integration of processes operating at very different spatial scales (from m^2 to km^2) is a big challenge. Combining, on a single grid, the simulation of all hydrodynamic, sediment and vegetation processes requires tremendous computational capacities, even challenging for state-of-the-art supercomputers. So it would be beneficial to define the vegetation model on a fine grid, but the hydrodynamic and geomorphological parts on a coarser grid. This approach will result in a lower computation cost compared to solving all processes on a fine grid. A disadvantage, however, is that this multi-space-scale approach will increase the complexity of the implementation and the exchange of information will be a challenge.

4.5. Conservation

Since the equations to be solved are conservation laws, the numerical scheme should also respect these laws. We can distinguish between the strong and weak form. The weak form of the conservation law is usually written as

$$\frac{d}{dt}\int_{\Omega}\phi dV + \int_{\partial\Omega}\mathbf{f}(\phi)dS = \int_{\Omega}s(\phi)dV$$

Here ϕ represents the conserved quantity and $\mathbf{f}$ the flux. The strong form (or differential form) can be derived from the weak form by taking an infinitesimally small control volume and applying the divergence theorem. The equation is then written as

$$\phi_t + \nabla\cdot\mathbf{f}(\phi) = s(\phi)$$

The strong form may not always hold, as it requires that $\nabla\cdot\mathbf{f}(\phi)$ exist. The strong form is not valid when there is a shock, contact discontinuity, or when the function is not smooth.

Local momentum conservation states that in absence of sources and sinks the change in momentum within the control volume is only a result of fluxes through the control volume surfaces. If the summation is performed over all the faces in a mesh (including the boundary faces) then the interior fluxes cancel out and we have global conservation, which states that the change in momentum is due only to fluxes through the boundary of the domain [10]. The treatment of sources or sink terms should be consistent so that the total source or sink in the domain is equal to the net flux of the conserved quantity through the boundaries. The strong conservation form of the equations, when used together with a finite volume method, automatically ensures global momentum conservation in the calculation. This is an important property of the conservation equations. Other discretization methods can be made conservative when care is taken in the choice of approximations.

5

Graphics Processing Units for Scientific Computing

Graphics processing units (GPUs) initially were designed to support computer graphics. Their specific architecture provides high processing power and massive parallelism, which allows for efficient solving of typical graphics-related tasks. However, soon it was realised that such architectures are also suitable for many other computational tasks, such as scientific computing [28]. For relatively low costs one can obtain supercomputer performance (1 Teraflop). It appears however that some work has to be done to make an ordinary program suitable for use on the GPU. The development of the tools CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language) helped to reduce this work [4]. But still algorithms must be specifically adapted (in a way which is possibly sub-optimal for a CPU (central processing unit)) in order to realize acceleration on the GPU. In this chapter background information about the GPU is given, such as information about threads, blocks and grids and different types of memory on the GPU. Afterwards we will focus on the advantages and disadvantages of a GPU.

5.1. GPU Architecture

A GPU takes over tasks of a CPU. Calculations are performed much faster on a GPU, because of the fact that it consists of thousands of small, efficient cores, while a CPU consists only of a few cores. The most intensive functions can be computed on a GPU, while other calculations can still be performed on a CPU. The GPU is a SIMD (Single Instruction Multiple Data) processor. A single SIMD instruction encapsulates a request that the same operation is performed on multiple data elements in parallel. A GPU has a fixed number of processors within, these are called multiprocessors (MPs). Each multiprocessor further has several scalar processors (SPs). A scalar processor is classified as a SISD processor (Single Instruction Single Data) [4]. A general GPU architecture is shown in Figure 5.1a (M is equal to eight). We will first explain what threads, blocks and grids are and subsequently explain the different types of memory on a GPU.

5.1.1. Threads, Blocks and Grids

To run a simulation model on the GPU, it is crucial to understand how parallel processing is organized. We can distinguish between threads, blocks and grids. Threads calculate the same parallel computations that have to be performed, but on different parts of an array (so threads can run concurrently). A group of threads together is called a block and a group of blocks together is called a grid. Each thread or block is identified by an index that has a x, y and z coordinate.

Each thread is processed by a specific processor core which has its own local memory to be used in calculations within the thread. However, grids can easily have over 10.000 grid points, and hence even with the most extensive GPUs, each core has to handle multiple threads before the entire grid is updated. Haphazardly assigning processing cores to threads would be very inefficient. For this reason, the grid is split up in blocks, which is handled by what is called a multiprocessor; a group of for instance 8 processor cores. This block has its own "shared" memory in which the variables needed to process the threads within this block are copied. Hence, the grid values used within this block are loaded into

the multiprocessor's shared memory, allowing it to efficiently process all threads without additional access to the global GPU memory outside the multiprocessor, on which the entire grid is stored. The block structure ensures that the transfer of data from the graphics card's global memory to the graphics processor register memory is handled in an efficient way, preventing separate memory transfers for each individual thread. For best performance, it is important that the information needed by each thread is local (for example from neighbouring nodes), so that it does not originate from the other side of the grid, and it can all be loaded in the multiprocessor's memory before the block is executed.

The programmer has to specify how many of such threads exist in each block and how many blocks exist in each grid. When developing a model to run on the GPU, it is essential to optimize the size and number of the blocks, so that all processing cores are used efficiently. Obviously, a block size is limited by the size of the multiprocessor's memory, and hence large block sizes are not optimal. Also, it is essential that the number of blocks is adjusted to the number of multiprocessors. For instance, if a GPU has 6 multiprocessors, it is inefficient to split the grid up into 8 blocks. Six blocks would then be first processed, after which two blocks remain. These would subsequently be processed by two multiprocessors, leaving the others waiting until those two have finished. It would be more efficient to divide the grid into 12 smaller blocks, so that after two rounds, all blocks have been processed and no multiprocessors remained idle for part of the time. The number of multiprocessors and the size of their internal memory are specific to the type of GPU and graphics card; more expensive graphics cards have more multiprocessors and more memory. This means that in order to maximize the acceleration, the model needs to be geared to the GPU in the computer and recompiled before the model is run. However, a thread block size of 16×16 (256 threads) is a common choice [29].

Summarizing one entire grid is handled by a single GPU chip. This chip is organized as a collection of MPs and those MPs are responsible for handling one or more blocks. It never happens that one block is divided over multiple MPs. Each MP is subdivided into multiple SPs and those are responsible for one or more threads. An illustration of threads, blocks and grids is given in Figure 5.1b. In this figure, the term kernel is used. Compute-intensive and data-intensive portions of a given application, called kernels, may be offloaded to the GPU, trying to achieve significant performance while the host CPU continues to execute non-kernel tasks [30].

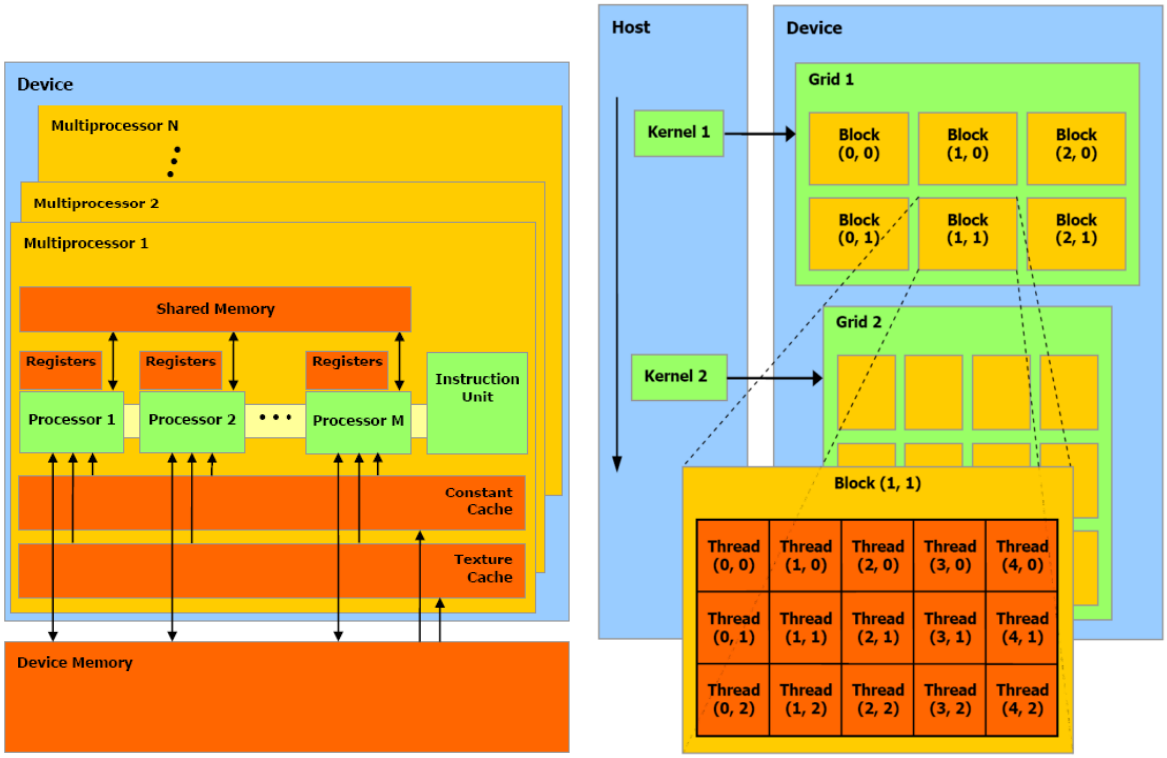
Execution of threads

Threads inside a block are grouped into warps. The scheduler that picks up threads for execution, does so in granularity of a warp. So, if the warp size is say, 32, it will pick 32 threads with consecutive thread Ids and schedule them for execution in the next cycle. Each thread executes on one of the SPs. The MPs are capable of executing a number of warps simultaneously. This number can vary from 512 up to 1024 on a GPU depending on the type of card. At the time of issue from the scheduler the MPs are handed over a number of blocks to execute. These can vary on the requirement that each thread imposes in terms of registers and shared memory since they are limited on each multiprocessor. For example, when the maximum number of threads that can be scheduled on a multiprocessor is 768 and each warp is composed of 32 threads then we can have a maximum of 24 warps. We can choose the following division schemes: 256 threads per block (3 blocks), 128 threads per block (6 blocks) and 16 threads per block (48 blocks). Now each MP has a restriction on the number of blocks that can simultaneously run on it. So if the maximum number of blocks is say 8 then only the first and second schemes could form a valid execution configuration. By choosing the first two schemes instead of the third we also take into account that threads per block should be a multiple of warp size to avoid wasting computation on underpopulated warps and to facilitate coalescing. Coalescing is the act of merging two adjacent free blocks of memory [31].

5.1.2. Memory Model

Each multiprocessor has a set of memories associated with it. These memories have different access times. It must be noted that these memories are on the device (GPU) and are different from the dynamic random-access memory (DRAM) available with the CPU. The different memories are

- Register Memory
  There are a fixed number of registers (per block) that must be divided amongst the number



(a) GPU architecture. (b) Grids of Blocks with threads.

Figure 5.1: Overview of GPU architecture. Images taken from [4].

of threads (in a block) that are configured (by the previously discussed allocation of threads in blocks and grids). Registers are exclusive to each thread.

- Shared Memory
  Shared memory is accessible to all the threads within a block. It is the next best thing after registers since accessing it is cheaper than the global memory.

- Texture Memory
  Texture memory is read-only and could be read by all the threads across blocks on a single multiprocessor. It also has a local cache on the SM.

- Global (device) Memory
  This memory is the biggest in size and is placed farthest from the threads executing on the multiprocessors. Its access times compared to the shared memory (access) latency might be up to 200 times more

The amount of time required to move n data items depends on the latency or start-up time α and the incremental time per data item moved β, which is related to bandwidth. The bandwidth is the rate at which information can be transferred to or from the memory system. Therefore a simple formula for the time it takes to move n data items is: $\alpha + \beta n$. When a computer has a relatively high latency, it is useful to combine communications.

5.1.3. Single and Double Precision

Both single and double precision floating-point formats can be used on GPUs. Single precision numbers occupy four bytes in computer memory, while double precision numbers occupy eight bytes. A byte consists of eight bits. The double precision format is preferred to have a lower error, but is also slower compared to the single precision format. Based on computation time considerations, only single precision calculations are used in this report.

5.2. Application Programming Interfaces

The most popular software to use for a GPU are CUDA and OpenCL. CUDA (Compute Unified Device Architecture) is a software toolkit by Nvidia to ease the use of (Nvidia) graphics cards for scientific programming. OpenCL (Open Computing Language) is a restricted version of the C99 language with extensions appropriate for executing data-parallel code on a variety of heterogeneous devices. OpenCL can run on several types of GPUs (AMD, Nvidia) in contrast to CUDA which can only run on Nvidia graphics cards.

CUDA and OpenCL use different terminology. The most important differences are that in OpenCL a thread is called a work-item, a thread block a work-group and a grid is called a NDRange. Regarding the memory, the register memory is addressed by private memory and shared memory by local memory. In this report, both terminologies are used interchangeably.

5.3. When to use GPUs

The GPU is specialized for compute-intensive, highly parallel computations and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control (the order in which instructions are executed). More specifically, the GPU is especially well suited to address problems that can be expressed as data-parallel computations (the same program is executed on many data elements in parallel) with high arithmetic intensity (ratio of arithmetic operations to memory operations). Because the same program is executed for each data element, there is a lower requirement for sophisticated flow control. Besides, big caches are not necessary, because the memory access latency can be hidden with calculations. A GPU also has very high memory bandwidth compared to a CPU.

However, it could happen that algorithms only become slower when running them on GPUs. One of the possible causes is the fact that sending information from the CPU to the GPU, and the other way around, is time-consuming. Since the GPU is a separate device, data cannot be read from host memory. Instead, GPUs have their own memory. Therefore, data needs to be copied from host to device memory, which is much slower than reading from memory directly. This could be the case if one iteration on the GPU is not fast enough compared to one iteration on the CPU, to catch up the time lost for sending information to the GPU. To achieve a faster calculation on a GPU compared to a CPU we need to perform more calculations on a GPU. And whenever possible we need to limit the amount of data transfers. Another point which forms the core of all optimization on the GPU is that minimal or no thread divergence should be used. Because on the GPU, all threads scheduled together (in a warp) must execute the same code. As a consequence, when executing a conditional, all threads execute both branches, with their output disabled when they are in the wrong branch. It is best to avoid conditionals or use built-in functions [32]. To speed-up our GPU program, coalesced memory access and/or shared memory should be used wherever possible. Besides, as mentioned in Subsection 5.1.1, enough block should be used, to keep all multiprocessors at work and the block size should always be a multiple of 32 (64 in AMD hardware) to achieve maximum warp utilization.

Other downsides of GPUs we need to keep in mind are: limited memory is available, no error correcting memory (ECC), debugging can be complicated and fast double precision could still be quite expensive (depending on the specific GPU).

5.3.1. Cluster

The cost at which supercomputer performance is available is a couple of hundred euros for a GPU. This makes it an attractive option already compared to setting up or sharing a cluster that might be hard to get access to or costlier to put up in the first place [4]. A computer cluster consists of a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system. They were designed to get more computing power, but communication is much slower than computation [4].

5.4. GPU and CPU Specifications

For this thesis, our program is tested on two different machines. The GPU of machine A is used and the CPU and GPU of machine B. The CPU is only used in Section 6.6 to determine and compare the wall-clock times one the CPU and GPU of machine B. For the results in Chapters 9 - 14 both GPUs are used. Machine A contains an HD Graphics 4000 GPU. The CPU of machine B is an Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz, which consists of six cores running at 3.5GHz and each core can run two threads. It was released November 2013 and has a memory bandwidth of 59.7 GB/s. The GPU residing in this machine is an AMD Radeon HD - FirePro D700 Compute Engine. Specifications of both GPUs are shown in Table 5.1. A GPU's performance is given by the number of floating point operations it can perform in a second, noted as GFLOPS.

	HD Graphics 4000	AMD Radeon HD - FirePro D700
Released	May 2012	January 2014
Processing elements	128	2048
Multiprocessors	16	32
Single precision $(GFLOPS)$	332.8	3482
Memory bandwidth (GB/s)	25.6	263.0
Max global memory (MB)	1536	6144
Max work-item sizes	512 512 512	256 256 256

Table 5.1: Specifications of both GPUs.

6

Basic Framework

As mentioned in Chapter 3, the processes involved in a salt marsh can be divided into three main domains: a hydrodynamic, morphodynamic and vegetation part. This chapter shows which formulations are used in our model. In Section 6.1 the used formulations for the hydrodynamic, morphodynamic and vegetation part are revealed. Subsequently, the domain, the initial- and boundary conditions, the discretisation and the pseudo-code are discussed. In the end, the results will be shown.

6.1. Model Components

This section discusses the used formulations for our model components.

6.1.1. Hydrodynamics

The main equations of our hydrodynamic model are the shallow water equations,

$$\frac{\partial}{\partial t}(hu) + \frac{\partial}{\partial x}(hu^2) + \frac{\partial}{\partial y}(huv) + gh\frac{\partial \eta}{\partial x} - Dh\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + Shu = 0 \tag{6.1a}$$

$$\frac{\partial}{\partial t}(hv) + \frac{\partial}{\partial x}(huv) + \frac{\partial}{\partial y}(hv^2) + gh\frac{\partial \eta}{\partial y} - Dh\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + Shv = 0 \tag{6.1b}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = H_{in} \tag{6.1c}$$

which were derived in Chapter 2. A source term, H_{in}, is added to the continuity equation, which results in a small input of water at every grid cell. Now, the chosen bottom friction term and wetting-drying method are explained.

Bottom Friction

To convert the bottom friction to the friction term S we use Equation 2.48

$$S = \frac{g}{C^2 h}\sqrt{u^2 + v^2} = \frac{g}{C^2 h}\|\mathbf{u}\| \tag{6.2}$$

in which C represents the the Chézy coefficient. The Baptist equation is used to determine this coefficient

$$C = \sqrt{\frac{1}{\frac{1}{c_b^2} + \frac{c_D n_b k}{2g}}} + \sqrt{\frac{g}{k_0}}\ln\left(\max\left(\frac{h}{k}, 1.0\right)\right). \tag{6.3}$$

The vegetation density is used to approximate mD (Equation 3.24). For emergent vegetation ($h <= k$, k vegetation height), Equation 6.3 reduces to

$$C = \sqrt{\frac{1}{\frac{1}{c_b^2} + \frac{c_D n_b k}{2g}}}, \tag{6.4}$$

since $\ln(1)$ is equal to zero. For submergent vegetation ($h > k$) Equation 6.3 transforms to

$$C = \sqrt{\frac{1}{\frac{1}{C_b^2} + \frac{C_D n_b k}{2g}}} + \sqrt{\frac{g}{k_0}} \log\left(\frac{h}{k}\right). \tag{6.5}$$

Since $h > k$, the second term is positive. The friction term S is now determined by

$$\boldsymbol{\tau}_b = \rho_0 g \|\mathbf{u}\| \frac{1}{C^2}. \tag{6.6}$$

Wetting-drying

As mentioned in Subsection 3.1.2, the SWE are not able to deal with dry areas. A thin film algorithm is used. After each time-step the maximum of a certain threshold, H_{crit}, and the water height is taken

$$h = \max(h, H_{crit}) \tag{6.7}$$

and thus the water height will never fall below this threshold. In Figure 6.1 this method is illustrated. Afterwards the velocities are updated according to the SWE. This wetting-drying method is not mass

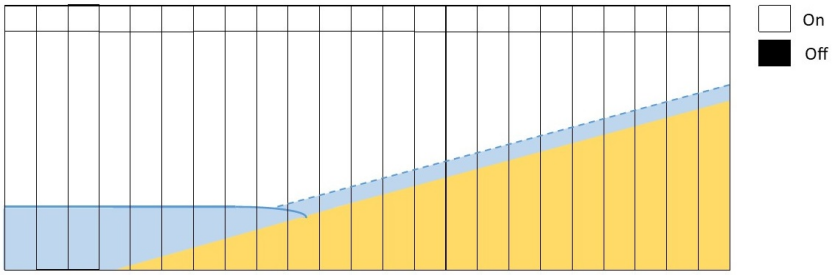


Figure 6.1: Thin film algorithm.

conservative, since mass is added if the water height falls below the chosen threshold. In Section 4.5 conservation is discussed.

6.1.2. Morphodynamics

Our morphodynamic model consists of three factors which contribute to the sediment loss and gain

$$\frac{\partial z_b}{\partial t} = S_{in}\left(\frac{h_{eff}}{h_{eff} + Q_s}\right) - E_0\left(1 - p_E \frac{n_b}{K}\right)(u^2 + v^2) z_b + D\left(\frac{\partial^2 z_b}{\partial x^2} + \frac{\partial^2 z_b}{\partial y^2}\right). \tag{6.8}$$

The first factor is the deposition term, in which the effective water height is defined as

$$h_{eff} = h - H_{crit}.$$

and S_{in} is the sediment input. The term in brackets ensures that there is an initial period in which the increase of sedimentation is rapid for an increasing effective water depth and the larger h_{eff} gets, the more gradual the increase will be. This relation depends on the constant Q_s. There will be no gain of sediment at the grid cells with a water depth of H_{crit}. The second term accounts for the sediment erosion. The parameter K is the carrying capacity, E_0 the background erosion rate and p_E is the fraction by which sediment erosion is reduced if the vegetation is at carrying capacity. The first term in brackets ensures that there is less erosion for a higher plant density. The second terms in brackets guarantees there is more erosion for a larger net speed. The last term is a straightforward diffusion term.

6.1.3. Vegetation

In this part, the vegetation density, n_b , is calculated. Compared to the model in Subsection 3.3.1, we use no plant establishment function. Instead we prescribe an initial vegetation density ((Subsection 6.3)) and assume no vegetation establishes afterwards. In addition, one decay function related to the

flow stress is used. Instead of two decay functions, due to inundation stress and flow stress. Our plant growth model is now given by

$$\frac{\partial n_b}{\partial t} = r\left(1 - \frac{n_b}{K}\right) n_b \left(\frac{K_p}{K_p + h}\right) - E_p n_b \sqrt{u^2 + v^2} + D\left(\frac{\partial^2 n_b}{\partial x^2} + \frac{\partial^2 n_b}{\partial y^2}\right). \tag{6.9}$$

The first term accounts for the growth, the second for the mortality and the last term for the dispersal of vegetation. Here D is the plant diffusion coefficient, K is the carrying capacity, K_p is the value of the water level where plant growth is approximately half the maximum and E_p represents the plant loss due to flow. The growth factor is logistic, namely, the growth rate gets smaller as the vegetation density approaches the carrying capacity. Compared to the growth function in Subsection 3.3.1 the term $\frac{K_p}{K_p+h}$ is added, which ensures the plant growth is less for a higher water depth. The second term assumes that there is a higher mortality rate for a higher density and net speed. The dispersal factor is again straightforward.

6.2. Domain

Discretisation always entails subdivision of the domain into small cells. The resulting set of cells is called the computational grid. Our domain is a simple square. So a Cartesian grid (explained in Section 4.1), as illustrated in Figure 6.2, is chosen. In addition, a horizontal numbering of the unknowns is used. The variables L_1 and L_2 represent the lengths, n and m the number of grid points and Δx and

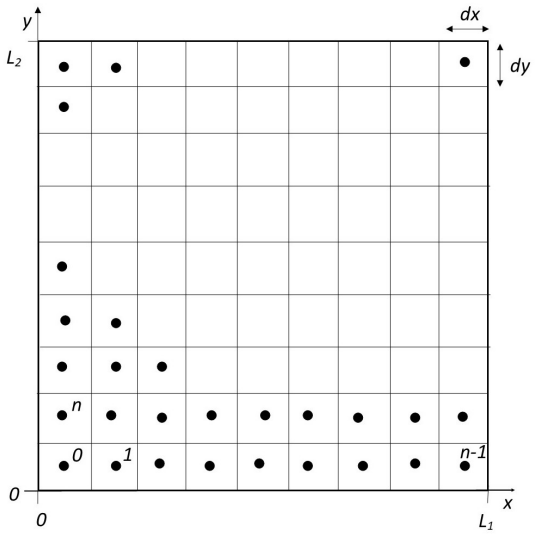


Figure 6.2: Cartesian collocated grid with horizontal numbering.

Δy the grid size in the x- and in the y-direction. For the sake of simplicity, a simple square, collocated grid is used in such a way that $n = m$, $\Delta x = \Delta y$ and $L_1 = L_2$.

6.3. Initial and Boundary Conditions

The boundary and initial conditions are shown in Table 6.1. We have three closed boundaries and one open boundary, through which we have outflow (the left boundary in Figure 6.2).

First, the boundary conditions are discussed. To prevent the tidal creeks from going through the upper, right and bottom boundaries, a vegetation density of one is taken at these boundaries. At the left boundary, the normal derivative of the density is set equal to zero (Neumann condition). For u and v also homogeneous Neumann and Dirichlet conditions are used. For the water depth the normal derivative is set equal to zero at all boundaries. We use also homogeneous Neumann conditions for the bottom elevation except at the outflow boundary. There it is prescribed that the sediment level is equal to zero.

Variable	Boundary Conditions				Initial Condition
	Bottom	Top	Left	Right	
u	$\frac{\partial u}{\partial n} = 0$	$\frac{\partial u}{\partial n} = 0$	$\frac{\partial u}{\partial n} = 0$	$u = 0$	$u = -u_{homo}$
v	$v = 0$	$v = 0$	$\frac{\partial v}{\partial n} = 0$	$\frac{\partial v}{\partial n} = 0$	$v = 0$
h	$\frac{\partial h}{\partial n} = 0$	$\frac{\partial h}{\partial n} = 0$	$\frac{\partial h}{\partial n} = 0$	$\frac{\partial h}{\partial n} = 0$	$h = h_{homo}$
n_b	$n_b = 1$	$\frac{\partial n_b}{\partial n} = 0$	$n_b = 1$	$n_b = 1$	$n_b \in \{1,\ 0\}$
z_b	$\frac{\partial z_b}{\partial n} = 0$	$\frac{\partial z_b}{\partial n} = 0$	$z_b = 0$	$\frac{\partial z_b}{\partial n} = 0$	$z_b = S_{homo}$

Table 6.1: Boundary and initial conditions of dependent variables.

Now the initial conditions are reviewed. For the vegetation randomly $0,2$ percent of the grid points are assigned an initial vegetation density of one, the other $99,8$ percent a value of zero. The depth-averaged velocity v is set equal to zero. For u, h and z_b the following equations are used

$$h_{homo} = H_0$$

$$u_{homo} = \sqrt{slope}\left(\frac{h_{homo}^{\frac{2}{3}}}{nn}\right)$$

$$S_{homo} = S_{in}u_{homo}^2 \frac{H_0 - H_{Crit}}{Q_s + \frac{H_0 - H_{Crit}}{E_0}}$$

A slope of $0.002\ m/m$ is applied to the underlying bathymetry. So it ranges between zero at the left boundary (sea side), and $0,2$ at the right boundary.

6.4. Discretisations

The method of lines is used. Hence, we choose time integration independent from space discretisation. In the current model a semi-implicit time-integration method is used in combination with the finite difference method (central differences in space) for all equations. The notation as illustrated in Figure 6.3 is used. Here C represents the grid point $(i,j) = k$, E the point $(i+1,j) = k+1$, W the grid point $(i-1,j) = k-1$, S the point $(i,j-1) = k-n$ and N the point $(i,j+1) = k+n$. Applying these methods

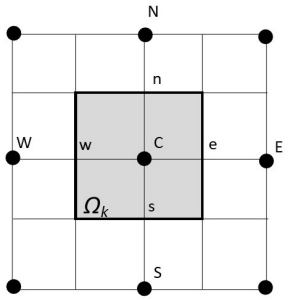


Figure 6.3: Cartesian collocated grid (Arakawa C) in which • represents the location of all variables.

to the SWE given in Equations 6.1a-6.1c and Equations 6.8 and 6.9 gives us the following system to solve for each grid point

$$h_C^{m+1} = h_C^m + \Delta t\left(-\frac{h_E^{m^*}u_E^{m^*} - h_W^{m^*}u_W^{m^*}}{2\Delta x} - \frac{h_N^{m^*}v_N^{m^*} - h_S^{m^*}v_S^{m^*}}{2\Delta y}\right) \tag{6.10}$$

$$u_C^{m+1} = u_C^m + \Delta t[-u_C^{m^*}\frac{u_E^{m^*} - u_W^{m^*}}{2\Delta x} - v_C^{m^*}\frac{u_N^{m^*} - u_S^{m^*}}{2\Delta y} - g\frac{\eta_E^{m^*} - \eta_W^{m^*}}{2\Delta x} + D\left(\frac{u_W^{m^*} - 2u_C^{m^*} + u_E^{m^*}}{\Delta x^2} + \frac{u_S^{m^*} - 2u_C^{m^*} + u_N^{m^*}}{\Delta y^2}\right) - S_C^{m^*}u_C^{m^*}] \tag{6.11}$$

$$v_C^{m+1} = v_C^m + \Delta t[-u_C^{m^*}\frac{v_E^{m^*} - v_W^{m^*}}{2\Delta x} - v_C^{m^*}\frac{v_N^{m^*} - v_S^{m^*}}{2\Delta y} - g\frac{\eta_N^{m^*} - \eta_S^{m^*}}{2\Delta y} + D\left(\frac{v_W^{m^*} - 2v_C^{m^*} + v_E^{m^*}}{\Delta x^2} + \frac{v_S^{m^*} - 2v^{m^*_C} + v_N^{m^*}}{\Delta y^2}\right) - S_C^{m^*}v_C^{m^*}] \tag{6.12}$$

$$n_{b_C}^{m+1} = n_{b_C}^n + \Delta t[D\left(\frac{n_{b_W}^{m^*} - 2n_{b_C}^{m^*} + n_{b_E}^{m^*}}{\Delta x^2} + \frac{n_{b_S}^{m^*} - 2n_{b_C}^{m^*} + n_{b_N}^n}{\Delta y^2}\right) + r\left(1 - \frac{n_{b_C}^n}{K}\right)n_{b_C}^{m^*}\left(\frac{K_p}{K_p + a_C^n}\right) - E_p n_{b_C}^{m^*}\sqrt{(u_C^{m_2})^2 + (v_C^{m_2})^2}] \tag{6.13}$$

$$z_{b_C}^{m+1} = z_{b_C}^m + \Delta t[S_{in}h_{eff_C}^{m^*} - E_0\left(1 - p_E\frac{n_{b_C}^{m^*}}{K}\right)((u_C^{m^*})^2 + (v_C^{m^*})^2)z_{b_C}^{m^*} + D\left(\frac{z_{b_W}^{m^*} - 2z_{b_C}^{m^*} + z_{b_E}^{m^*}}{\Delta x^2} + \frac{z_{b_S}^{m^*} - 2z_{b_C}^{m^*} + z_{b_N}^{m^*}}{\Delta y^2}\right)] \tag{6.14}$$

Since a semi-implicit time-integration method is used, the variable m^* has a value of m or $m+1$. The value depends on the fact if we have already updated the corresponding variable for the new time step. More information about the derivation can be found in the Appendix A.1.

6.5. Code

First the code that describes how to implement the kernel that calculates the rate of change of several variables (u, v, h, z_b, n_b) is explained and illustrated in Listing 6.1. The kernel itself will be explained afterwards.

The OpenCL implementation would, after including the standard input/output and header files and defining the model's constants, first define any specific functions that are used within the GPU kernel. We left out this part in Listing 6.1. First, memory is allocated on the host (lines 3-7), after which it is initialized (line 10). Here $Grid\_Width$ is the number of threads in the x-direction and $Grid\_Height$ is the number of threads in the y-direction. The total number of threats is then given by $Grid\_Width$ times $Grid\_Height$ and the variable $NumFrames$ specifies how often we want to save our data. Afterwards, the context is set up (line 14). The context includes a set of devices (line 13) and the memory accessible to those devices. A context is needed to share memory objects between devices. Next, the OpenCL command queues, which are used for submitting work to a device, are created (lines 17 - 18). They order the execution of kernels on a device and manipulate memory objects. OpenCL executes the commands in the order that we enqueue them. The clCreateBuffer command creates a buffer object of the appropriate size (line 21). Lines 23-25 enqueues a command to write data from host memory (h_V1) to a buffer (d_V1). This function is used to provide data for processing by a kernel executing on the device. Lines 28-29 create the OpenCL program, which is a set of OpenCL kernels, auxiliary functions called by the kernels, and constants used by the kernels. Afterwards, the kernel object is specified (line 32), which encapsulates a specific kernel declared in a program, along with the argument values to use when executing this kernel (line 35). One dimension is used in lines 38-47 to specify the global work-items and work-items in the work-group. Since we still want to access

the grid based on coordinates a simple $2D \leftrightarrow 1D$ mapping is used

$$(i,j) \rightarrow thread\_id : thread\_id = i + (j-1)Grid\_Width \tag{6.15}$$

$$thread\_id \rightarrow (i,j) : j = floor\left(\frac{thread\_id}{Grid\_Width}\right) \tag{6.16}$$

$$i = thread\_id\%Grid\_Width; \tag{6.17}$$

in which floor function rounds down the fraction and the % operator finds the remainder after division. Notice that the mapping is chosen based on memory access in $C/C++$ style (row-wise). The explicitly specified $local\_item\_size$ will be used to determine how to break the global work-items specified by $global\_item\_size$ into appropriate work-group instances. If $local\_item\_size$ is specified, $global\_item\_size$ must be evenly divisible (leaving no remainder) by $local\_item\_size$. Another option is to set the $local\_item\_size$ to a NULL value in which case the OpenCL implementation will determine how to break the global work-items into appropriate work-group instances.

In a kernel, we must explicitly tell OpenCL to pass the specified buffer object as an argument to the specific kernel function that is defined in the OpenCL program source code (clSetKernelArg). A for-loop is constructed that calls the GPU kernel for each timestep (note: a loop in time, not in space) and another for-loop is constructed that determines how often the output of the kernel is read back from the device to the host application (lines 42-55). After the for-loops have finished, we open the output file (line 58) and write some important parameters (line 61) together with the results, which were read back to the host, to this file. To finalize, all gridded memory blocks are freed from memory (lines 68–69). After the host application no longer requires the various objects associated with the OpenCL runtime and context, it should free these resources. You can use the following functions to release your OpenCL objects: clReleaseMemObject, clReleaseKernel, clReleaseProgram, clReleaseCommandQueue, clReleaseContext.

Listing 6.1: Main Programme

```
1  int main(){

3  // Defining and allocating the memory blocks
4  float * h_V1 = (float *) malloc(Grid_Width * Grid_Height* sizeof(float));

6  // Defining and allocating storage blocks
7  float * h_store_V1=(float*) malloc(Grid_Width * Grid_Height* NumFrames);

9  // Initialize variables
10 randomInit(h_V1, Grid_Width, Grid_Height, u_Flow);

12 //- - - Setting up the device - - -
13 cl_device_id* devices;
14 cl_context context = CreateGPUcontext(devices);

16 // Create a command queue on the device
17 cl_command_queue command_queue = clCreateCommandQueue(context, devices[Device_No],
18 0, &err);

20 // Create Buffer Objects on the device
21 cl_mem d_V1 = clCreateBuffer(context, CL_MEM_READ_WRITE, Grid_Memory, NULL, &err);

23 // Copy input data to the memory buffer
24 clEnqueueWriteBuffer(command_queue, d_V1, CL_TRUE, 0, mem_size_storegrid2,
25 h_V1, 0, NULL, NULL);

27 // Building the PDE kernel
28 cl_program program = BuildKernelFile("Computing_Kernel.cl", context,
```

```
29 &devices[Device\_No], &err);

31 // Create OpenCL kernel
32 cl_kernel kernel = clCreateKernel(program, "SimulationKernel", &err);

34 // Set OpenCL kernel bindings
35 clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&d_V1);


38  //  Kernel parameterization
39 size_t global_item_size = Grid_Width*Grid_Height;
40 size_t local_item_size = Block_Size_X*Block_Size_Y;

42 for (int Counter=1;Counter<(NumFrames+1);Counter++){
43     for (int Runtime=0;Runtime<floor((float)EndTime/NumFrames/dT);Runtime++){
44         // Execute OpenCL kernel as data parallel
45         clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL, &global_item_size,
46         &local_item_size, 0, NULL, NULL);
47     }

49     // Transfer result to host
50     clEnqueueReadBuffer(command_queue, d_V1, CL_TRUE, 0, Grid_Memory, h_V1, 0,
51     NULL, NULL);

53     //Store values at this frame.
54     memcpy(h_store_V1+(Counter*Grid_Size),h_V1,Grid_Memory);
55 }

57  // - - - Write to file - - -
58  FILE * fp=fopen(DataPath.c_str(),"wb");

60 // Storing parameters
61 fwrite(&width_matrix,sizeof(int),1,fp);

63 for(int store_i=0;store_i<(NumFrames+1);store_i++){
64     fwrite(&h_store_V1[store_i*Grid_Size],sizeof(float),Grid_Size,fp);
65 }

67 // Freeing host space
68 free(h_V1);
69 free(h_store_V1);

71 // Freeing kernel and block space
72 clFlush(command_queue);
73 clFinish(command_queue);
74 clReleaseKernel(kernel);
75 clReleaseProgram(program);

77 clReleaseMemObject(d_V1);

79 clReleaseCommandQueue(command_queue);
80 clReleaseContext(context);
81 free(devices);
82 }
```

Note in Listing 6.1 no error handling is shown. For example the error checking for the enqueueRead-

Buffer command could look as shown in Listing 6.2.

Listing 6.2: Error Checking

```
2  err  = clEnqueueReadBuffer(command_queue, d_V1, CL_TRUE, 0, Grid_Memory,
3       h_V1, 0, NULL, NULL);

5  if (err!=0) { printf("Read Buffer Error: %d\n\n", err); exit(-10);};
```

A value for err of zero (CL_SUCCESS) corresponds to a successful execution of clEnqueueReadBuffer. More information about errors which could be encountered in OpenCL can be found at [33].

Please note this code is specially designed for readability. We took into account the number of host-device memory transfers, by initializing all dependent variables as global variables and sending them to the device only once. Afterwards, we only send them back to the host at certain times. However, at the moment no coalesced memory access or shared memory is used.

The basic pseudo-code of the GPU kernel is shown in Algorithm 1. Here the variables h, u, v, n_b, z_b again represent the water depth, velocity in the x- and in the y-direction and the vegetation density and bottom elevation. H_{crit} is the critical water depth value.

input : dT, H_{crit}
global input: h, u, v, n_b, z_b
1 Initialize $t = 0$;
2 Update $t = t + dT$;
3 $k = thread_{id}$;
4 **if** $k ==$ *internal grid point* **then**
5 | Update $h[k]$ by Eq. 6.10;
6 **end**
7 **else**
8 | Update $h[k]$ by boundary condition;
9 **end**
10 Update $h[k] = max(h[k], H_{crit})$;
11 **if** $k ==$ *internal grid point* **then**
12 | Update $u[k]$ by Eq. 6.11;
13 | Update $v[k]$ by Eq. 6.12;
14 | Update $n_b[k]$ by Eq. 6.13;
15 | Update $z_b[k]$ by Eq. 6.14;
16 **end**
17 **else**
18 | Update $u[k]$ by boundary condition;
19 | Update $v[k]$ by boundary condition;
20 | Update $n_b[k]$ by boundary condition;
21 | Update $z_b[k]$ by boundary condition;
22 **end**

Algorithm 1: Pseudo-code kernel.

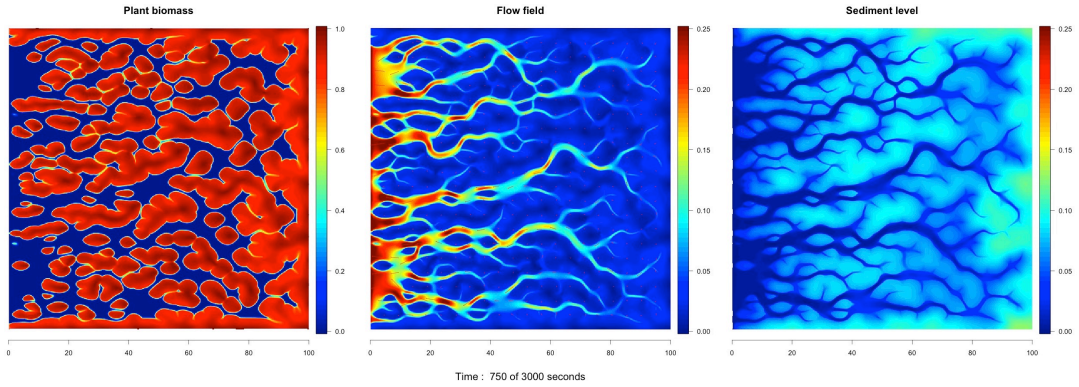
We want to avoid divergent execution of threads, but if and else statements are used to separate the boundary nodes from the internal nodes.

6.6. Results

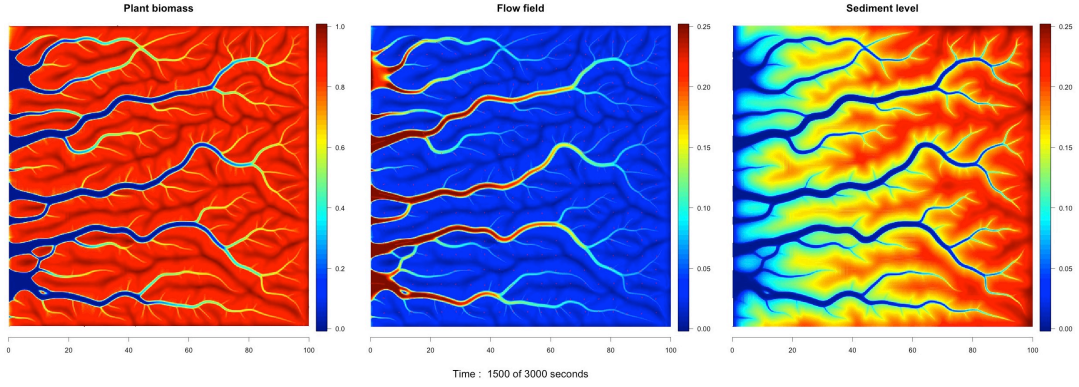
The vegetation density (n_b), the net speed ($\sqrt{u^2 + v^2}$) and the sediment level (z_b) are shown after 750, 1500 and 3000 seconds in Figure 6.4. Since the same seed is used in the random number generator (used to determine the location of the initial vegetation patches), every simulation has the same outcome. The left boundary represents the seaside boundary. Looking at the plant biomass we notice the vegetation forms large compact areas. At the start of the simulation, the flow field shows

multiple wide tidal creeks. The number of tidal creeks decreases in time. However, the ones that remain, have a larger net speed and depth. Besides they have become more straight. Please note the meandering of the tidal creeks in this model is a result of the placement of the initial vegetation patches.

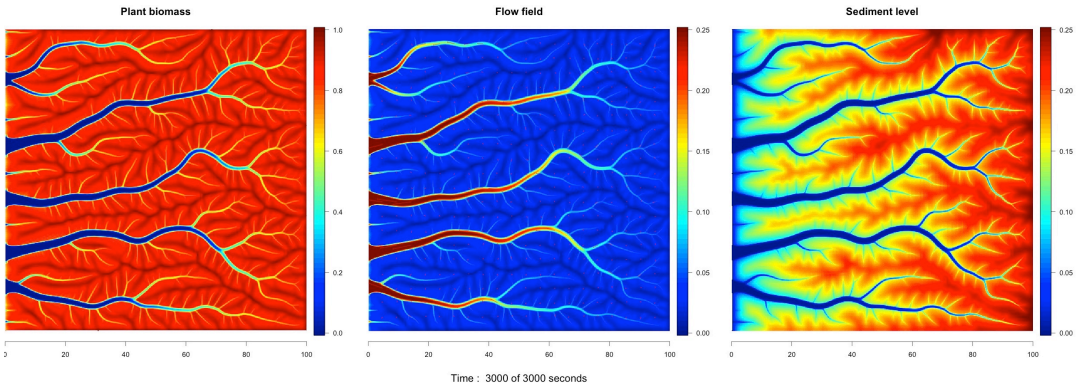
In Figure 6.7 the cross-sections through both axes are shown. This figure also shows the tidal creeks become smaller in time. More visible is the rising of the areas next to the tidal creeks in time due to sediment deposition.



(a) After 750 seconds.



(b) After 1500 seconds.

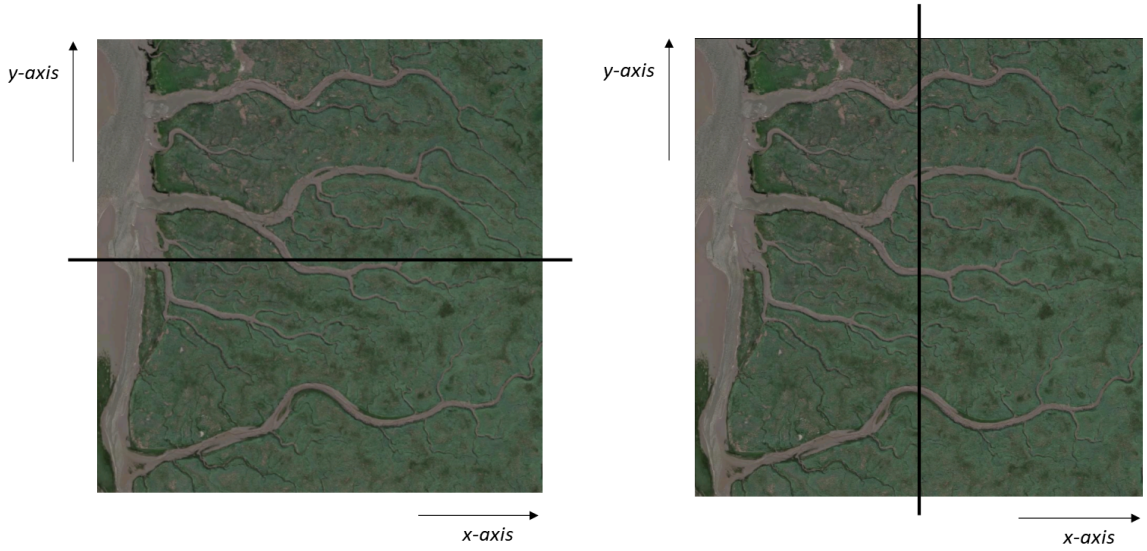


(c) After 3000 seconds.

Figure 6.4: The vegetation density, net speed and sediment level.

The cross-sections through the x- and y- axes are illustrated in Figure 6.5. The influence of the water input H_{in} becomes clear in Figure 6.6. It shows the net speed for different values of H_{in}: $0,00001$, $0,00004$ and $0,00009$. We conclude that the higher H_{in} the more bifurcations and the wider the tidal creeks.

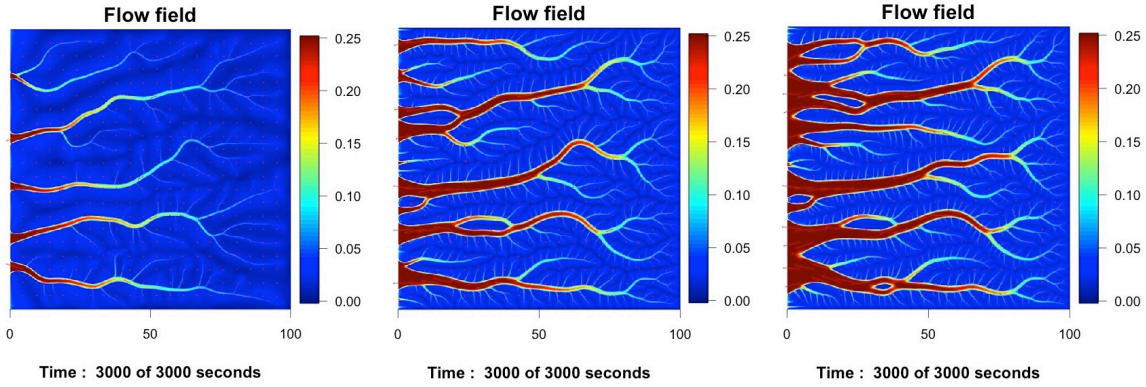
A grown salt marsh is simulated in 3000 seconds. No 'acceleration factor', as explained in Section 4.4,



(a) Cross section along the x-axis. (b) Cross section along the y-axis.

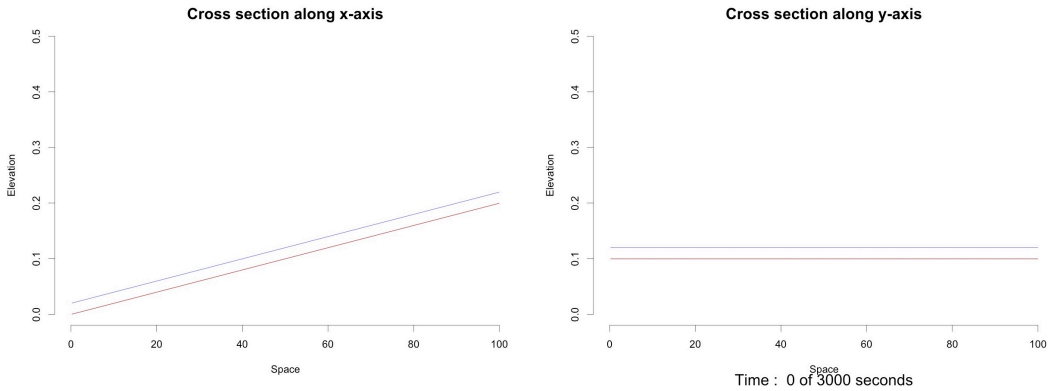
Figure 6.5: The thick black line indicates the cross-sections along the x-axis and y-axis.

is used. However the parameters are chosen in such a way to simulate speed-up vegetation growth, erosion and deposition. The results show the model is a good representation of the meandering and bifurcations of tidal creeks. However, there are points of improvement, which are discussed in detail in the next chapter.

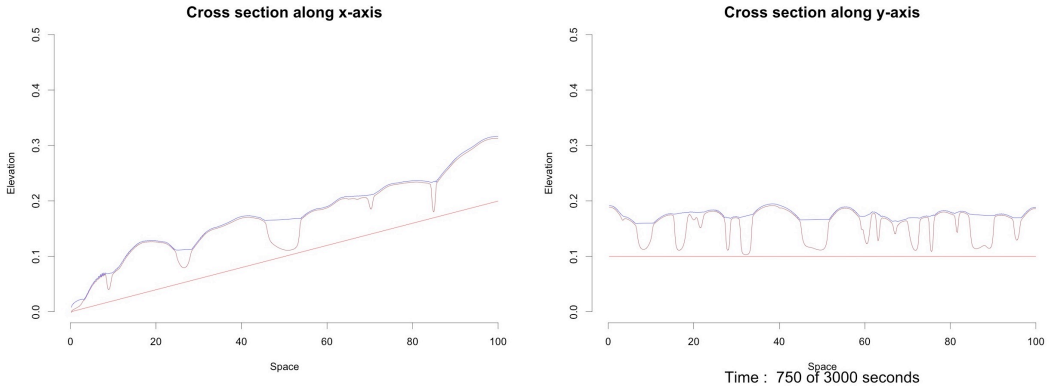


(a) H_{in} equal to 0.00001. (b) H_{in} equal to 0.00006. (c) H_{in} equal to 0.00009.

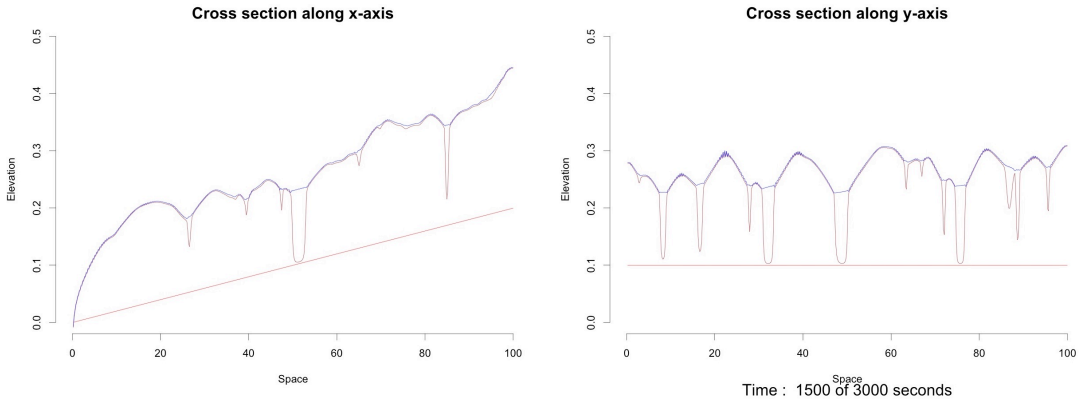
Figure 6.6: The net speed for different values of H_{in}.



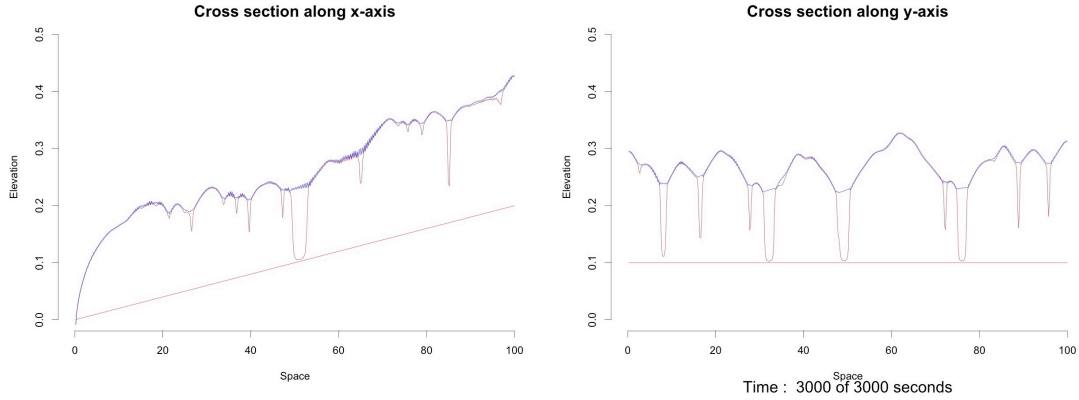
(a) After 0 seconds.



(b) After 750 seconds.



(c) After 1500 seconds.



(d) After 3000 seconds.

Figure 6.7: Cross section along the x- and y- axes.

7

Research

The two main aims of my thesis are stated below.

1. How can we improve the current salt marsh model in terms of the used grid, discretisation and solver?
2. How can we improve the current salt marsh model in terms of included processes?

When working on these research questions, we need to take into account the features of the graphics processing unit. Section 5.3 discussed some relevant features. For instance, it was mentioned that when utilizing a GPU, it is for instance very important to perform as many computations in parallel as possible and to send as little data as possible back and forth between the CPU and GPU. As a result, not all methods are suitable to be implemented efficiently on a GPU.

7.1. Research Question 1

This research question is concerned with the grid, the discretisation and the time-integration method.

7.1.1. Grid

As mentioned in Chapter 6 a collocated grid is used . We have a collocated grid arrangement, when all variables are stored in the same positions (Chapter 4). Instead, we could implement a staggered grid in an attempt to prevent spatial oscillations, as can be noticed in Figure 6.7d. In a staggered grid, not all quantities are defined at the same location in the numerical grid. The Arakawa C grid (shown in Figure 4.3b) is often used in computational fluid dynamics. In this grid, the velocity in the y direction is located on the top and bottom boundaries, the velocity in the x direction on the right and left boundaries and the water depth in the centre. Of course, there are other options and it depends on the prescribed boundary conditions which one we should choose.

7.1.2. Space Discretisation

In Chapter 4 it was mentioned that there are several numerical methods that can be used to solve partial differential equations (PDEs), such as finite differences methods and finite volume methods. Currently, the finite difference method is used. Finite difference methods solve partial differential equations directly by approximating the derivatives using local Taylor expansion, while finite volume methods solve the differential equations after integration over a control volume. Since the finite element method is especially well-suited for unstructured grids, and we have a structured grid we will not use this method. We shall implement the finite volume method. The advantage of using the finite volume method instead of the finite difference method is the conservation property associated with it.

7.1.3. Time-integration

The semi-implicit Euler method is used. This method is effectively explicit and thus well-suited for a GPU. Compared to the forward Euler method, a larger time-step can be used without the method

becoming unstable. Besides we only need to save each variable once, instead of twice (the old value and new value), which reduces the memory needed. Moreover, when implementing the explicit method, we need to wait for all threads to finish updating the new values, before setting the old values equal to the new values and also the other way around. Therefore we would, for instance, need to use two kernels, with a clFinish command after both, which would slow down our computations. In this thesis, we will not focus on other time-integration methods.

7.2. Research Question 2

For this research question, we have to find out if we can improve our model by adding or changing the used processes. At the moment there are two main downsides: there is a constant input of water in the whole computational domain and we would like small dikes to arise next to the tidal creeks.

7.2.1. Water Input

The model explained in Chapter 6 simulates no tidal action. Instead, a constant input of water is assumed everywhere, by adding a source H_{in} to the continuity equation

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = H_{in}.$$

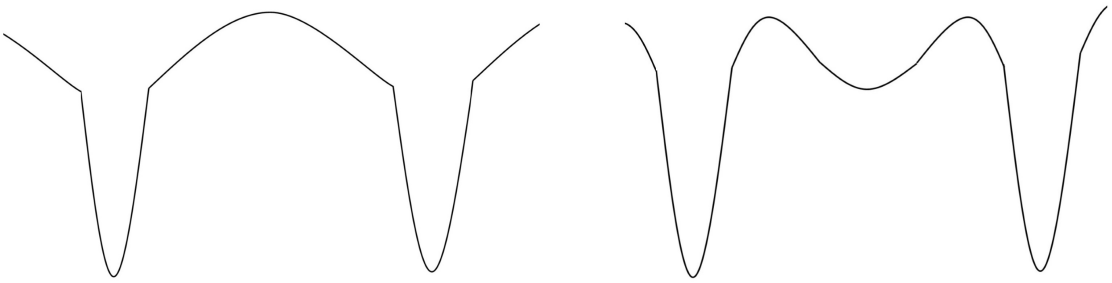
The consequence of this approach is that the whole computational domain is reached by the water at all times. In reality, though, it could be the case that the highest parts of the salt marsh are not reached by water during a flood. Besides, the water always flows to the seaside boundary. Thus we more or less have ebb tide all the time. A real salt marsh experiences flood and ebb. So in order to make our model more realistic, we could incorporate tides in our model. Therefore, the water level at the seaside boundary could be prescribed. For instance by taking

$$h(t) = M_S + A_S \sin\left(\frac{2\pi t}{T}\right) \tag{7.1}$$

in which M_S represents the mean sea level, A_S the tidal range and T the time it takes for ebb and flood to occur. Equation 7.1 does not take into account any sea level rise or increase of tidal range.

7.2.2. Deposition

In between tidal creeks, we would like most of the sediment to fall down as soon as it floods the area between the tidal creek network (mentioned in Section 3.3). This would result in the situation shown in Figure 7.1b. Here two tidal creeks are drawn with small hills of sediment at their boundaries. In our model, we have the situation which is shown in Figure 7.1a. The bottom elevation is higher for areas located further away from a tidal creek. This results could be related to how we define the sedimentation and the absence of tidal action. We could try to obtain this situation by redefining the



(a) Observed situation. (b) Wanted situation.

Figure 7.1: The area in between two tidal creeks.

sedimentation formula. At the moment it is assumed that there is more sedimentation for a larger effective water depth. Instead we could let it depend on the sediment concentration and determine this concentration by using a transport equation.

8

Computation Time Experiments

The simulation speed of our model is very important because it determines how big and how many problems can be solved in a given amount of time. We have measured the wall clock times of one or multiple calls of clEnqueueNDRangeKernel by using the command 'gettimeofday', which gives the number of seconds and microseconds since the Epoch, $1970-01-01\ 00:00:00(UTC)$. The timings, as presented later in this chapter, are the result of five runs. In this chapter the mean value is given, but all results can be found in Appendix C. The code for one call of EnqueueNDRangeKernel is shown in Listing 8.1.

Listing 8.1: Time Function

```
1  gettimeofday(&Time_Measured, NULL);
2  double Time_Begin=Time_Measured.tv_sec+(Time_Measured.tv_usec/1000000.0);

4  clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,&global_item_size,
5  NULL, 0, NULL, NULL);

7  clFinish(command_queue);

9  gettimeofday(&Time_Measured, NULL);
10 double Time_End=Time_Measured.tv_sec+(Time_Measured.tv_usec/1000000.0);

12 printf(" Processing time: %4.4f (s) \n", Time_End-Time_Begin);
```

The clFinish command blocks until all previously queued OpenCL commands in the command queue are issued to the associated device and have completed. For the results in this Chapter the Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz and AMD Radeon HD - FirePro D700 GPU are used. The most important specifications are listed in Section 5.4.

8.1. Results One Work Dimension

For different problem sizes of the model problem (finite difference method on a collocated grid), the wall-clock times for one iteration on the CPU and one iteration on the GPU are computed. The device was "warmed-up" by running several kernels in advance before we begin our experiments. The OpenCL implementation has determined how to break the global work-items into appropriate work-group instances (set the value of local_item_size to NULL). The results can be seen in Table 8.1. Grids of $16{,}384$ (128×128), $65{,}536$ (256×256), $262{,}144$ (512×512) etc. unknowns are used. The second column shows the CPU time. For a CPU the computation time is equal to the calculation time, because of the fact that data does not have to be sent to the GPU. Sending information to the GPU has to be done only once, while we send back information once or more, depending on how often we want to check our solution. For the results in Table 8.1, we send back the data once. The third column shows the computation time on a GPU, and the fourth and the fifth column the time it takes to once send

Unknowns	Time CPU (s)	Time GPU (s)	CPU → GPU (s)	GPU → CPU (s)	Total GPU (s)	Accel-eration	Max Accel-eration
16,384	0.0006	0.0003	0.0094	0.0006	0.0069	0.09	2.0
65,536	0.0019	0.0004	0.0060	0.0007	0.0105	0.18	4.8
262.144	0.0075	0.0006	0.0100	0.0016	0.0123	0.62	12.6
1,048,576	0.0303	0.0020	0.0157	0.0036	0.0211	1.45	15.1
4,194,304	0.0866	0.0065	0.0369	0.0121	0.0555	1.56	13.3
16,777,216	0.3644	0.0237	0.0894	0.0421	0.1552	2.35	15.4

Table 8.1: Single precision results in seconds on CPU and GPU for a varying number of unknowns.

the data from the CPU to the GPU and the other way around. The sixth column shows the sum of the computation time on the GPU and the time for the two data transfers. The seventh column shows the actual acceleration by dividing the CPU time by the total GPU time. We notice that for only one iteration and small problem sizes (until 65,535 unknowns), the CPU has a smaller computation time than the GPU. These results are also illustrated in Figure 8.1. Please note the CPU time increases almost by the

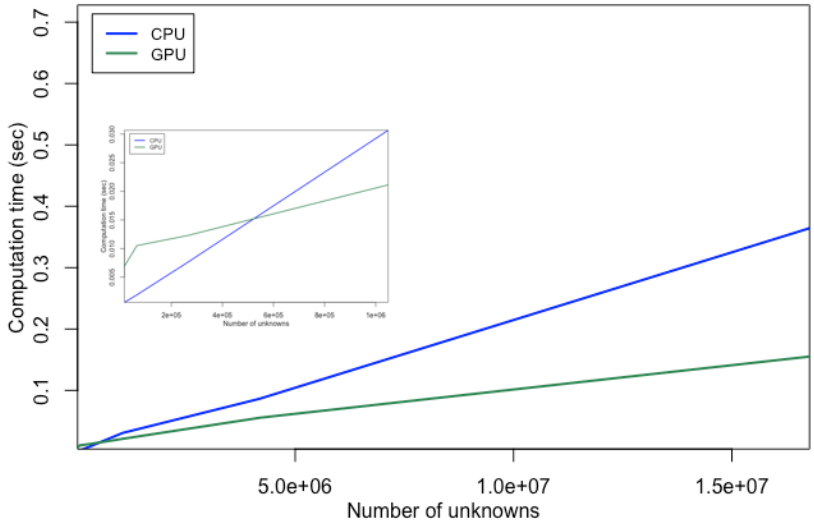


Figure 8.1: CPU and GPU computation times in seconds for varying number of unknowns.

same factor as the increase of unknowns (a factor of 4). For the GPU this is not the case. At first the increase is about 1.3 and it increases to 3.6 when the number of unknowns is 16,777,216. The eight column in Table 8.1 shows the maximum acceleration, which can be computed by dividing the GPU time by the CPU time. It can be obtained if enough iterations are performed on the GPU, so that the time needed to send information back and forth (the transferring of data) is negligibly small. Figure 8.2 shows the maximum acceleration for different problem sizes. We performed our experiments five times and took the mean, but there is still some variation in our results. We believe this causes the maximum acceleration to first increase to 15.1 and afterwards decrease again to 13.3.

We will now show that the maximum speedup can be obtained by performing enough iterations. Table 8.1 shows that for a grid size of 16,777,216, the CPU takes around 0.364 seconds for one iteration and the GPU around 0.024 seconds. This means that the maximum speedup is around 15 if enough iterations are performed. In Table 8.2, the wall-clock times for a varying number of iterations are shown. We expect the acceleration to increase for an increasing number of iterations, when the total GPU time is used, because the time needed for transferring data becomes smaller if more iterations are performed. The results are also illustrated in Figure 8.3. We first estimated the speed-up to be about a factor of 15.4. But the results in Table 8.2, show a maximum speedup of around 14.5. This discrepancy is probably a result of performing our experiments a limited number of times.

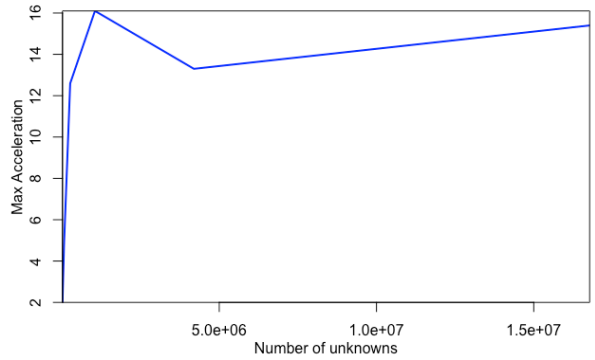


Figure 8.2: Maximum acceleration for varying number of unknowns.

Iterations	Time CPU (s)	Total Time GPU (s)	Acceleration
10	3.90104	0.48684	8.0
100	38.43702	2.87662	13.4
250	96.77444	6.64258	14.6
500	194.5518	13.40305	14.5

Table 8.2: Single precision results in seconds on CPU and GPU for a varying number of iterations for $16{,}777{,}216$ unknowns.

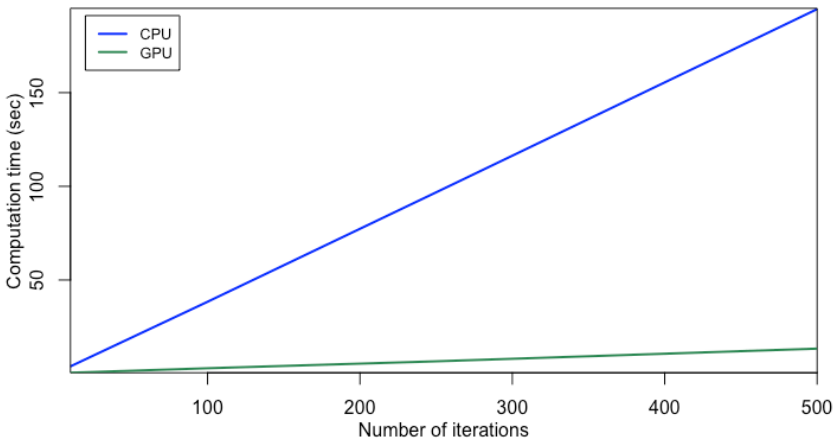


Figure 8.3: CPU and GPU computation times in seconds for varying number of iterations.

9

Results Discretisations

In Chapter 6 we have shown the results of the finite difference discretisation on a collocated grid. This chapter gives the results of the finite volume method on a staggered grid. The staggered grid (Arakawa C) as shown in Figure 9.1 is used. This specific grid is chosen, since the left boundary is

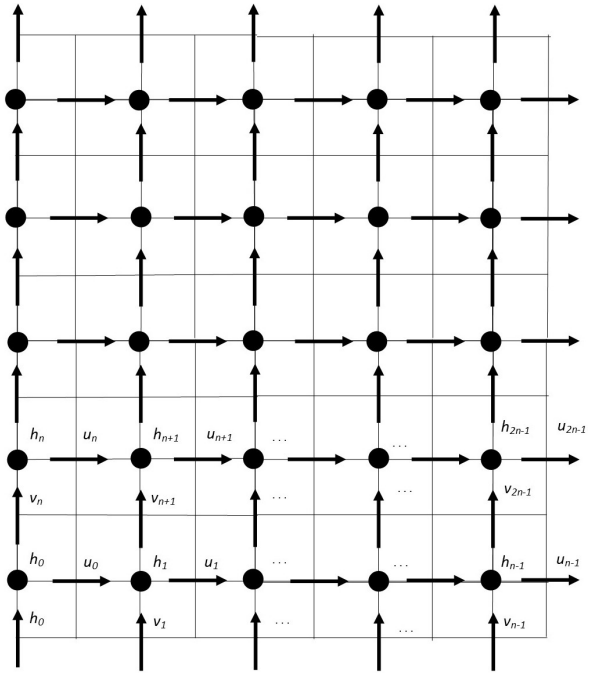
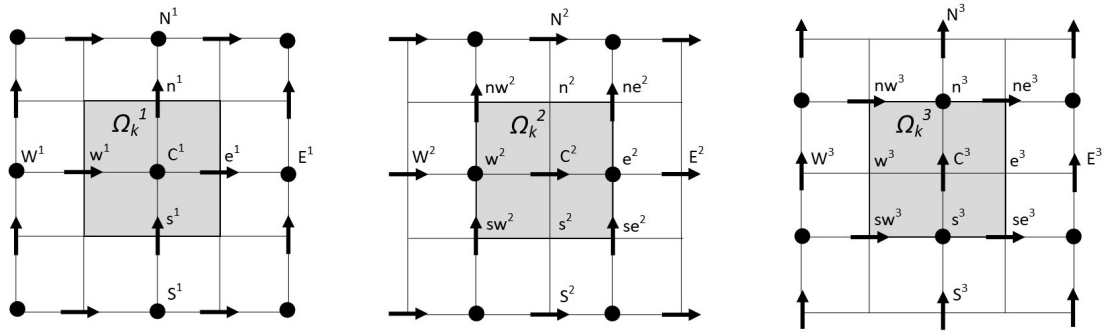


Figure 9.1: Staggered grid (Arakawa C).

an open boundary and the other boundaries are closed. The bottom elevation z_b, vegetation density n_b and water depth h are located at the • grid points, the velocity in the x-direction u (or discharge hu) is located at the → points and the velocity in the $y-$direction v (or discharge hv) is located at the ↑ points. The corresponding control volumes and used notation are shown in Figure 9.2. Here c stands for central, n for north, e east, s south and w west. We will first derive the finite volume discretisations on a staggered grid, after which the results are given.

9.1. Discretisations

Two approaches are used. In order to show the difference between the two approaches, the discretisations of the continuity equation and momentum equation in the x-direction are given.



(a) Control volume Ω_k^1. (b) Control volume Ω_k^2. (c) Control volume Ω_k^3.

Figure 9.2: Different control volumes.

Approach 1

The continuity equation is given by

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\mathbf{u}) = H_{in}.$$

Applying the finite volume method to a volume element Ω_k^1 and using the divergence theorem and midpoint rule gives

$$\frac{dh_{C^1}}{dt}\Delta x\Delta y + \int_{e^1} hudy - \int_{w^1} hudy + \int_{n^1} hvdx - \int_{s^1} hvdx = H_{in}\Delta x\Delta y.$$

Using linear interpolation results in

$$\frac{dh_{C^1}}{dt}\Delta x\Delta y + \left(u_{e^1}\frac{h_{E^1}+h_{C^1}}{2} - u_{w^1}\frac{h_{C^1}+h_{W^1}}{2}\right)\Delta y + \left(v_{n^1}\frac{h_{N^1}+h_{C^1}}{2} - v_{s^1}\frac{h_{C^1}+h_{S^1}}{2}\right)\Delta x = 0. \quad (9.1)$$

For the momentum equation in the x-direction

$$\frac{\partial}{\partial t}(hu) + \nabla \cdot (hu\mathbf{u}) + gh\frac{\partial h}{\partial x} - Dh(\nabla \cdot \nabla u) + Shu = 0.$$

we get by integrating over Ω_k^2

$$\frac{d(h_{C^2}u_{C^2})}{dt}\Delta x\Delta y + \int_{e^2} hu^2dy - \int_{w^2} hu^2dy + \int_{n^2} huvdx - \int_{s^2} huvdx + gh_{C^2}\left(\int_{e^2}\eta dy - \int_{w^2}\eta dy\right)$$
$$- Dh_{C^2}\left(\int_{e^2}\frac{\partial u}{\partial x}dy - \int_{w^2}\frac{\partial u}{\partial x}dy + \int_{n^2}\frac{\partial u}{\partial y}dx - \int_{s^2}\frac{\partial u}{\partial y}dx\right) + S_{C^2}h_{C^2}u_{C^2}\Delta x\Delta y = 0.$$

Approximating the integrals and using the semi-implicit Euler method results in

$$h_{C^2}^{m^*}u_{C^2}^{m+1} = h_{C^2}^{m^*}u_{C^2}^{m} - \frac{\Delta t}{\Delta x\Delta y}[h_{e^2}^{m^*}(u^{m^*})_{e^2}^2\Delta y - h_{w^2}(u^{m^*})_{w^2}^2\Delta y + (huv)_{n^2}^{m^*}\Delta x - (huv)_{s^2}^{m^*}\Delta x$$
$$+ gh_{C^2}^{m}\left(\eta_{e^2}^{m^*} - \eta_{w^2}^{m^*}\right)\Delta y - Dh_{C^2}^{m}\left(\frac{u_{E^2}^{m^*} - 2u_{C^2}^{m^*} + u_{W^2}^{m^*}}{\Delta x}\Delta y + \frac{u_{N^2}^{m^*} - 2u_{C^2}^{m^*} + u_{S^2}^{*}}{\Delta y}\Delta x\right)$$
$$+ S_{C^2}^{m}h_{C^2}^{m}u_{C^2}^{m^*}\Delta x\Delta y]. \quad (9.2)$$

In which $m^* \in \{m, m+1\}$ depending on if that variable has already been updated (for more information on semi-implicit time-integration we refer to Subsection 4.3.3). We need averaging operations for u in e^2 and w^2 and for h, u and v in n^2 and s^2.

Approach 2

We will again use three different control volumes as shown in Figure 9.2. We define $p = hu$ and $q = hv$ and $\mathbf{p} = [p\ q]^T$. If needed we can determine u and v by dividing p and q by the average value of h in the grid point of p or q. The continuity equation can now be written as

$$\frac{\partial h}{\partial t} + \frac{\partial p}{\partial x} + \frac{\partial q}{\partial y} = H_{in}$$

Integrating over volume Ω_k^1 and applying the divergence theorem and midpoint rule gives

$$\frac{dh_{C^1}}{dt}\Delta x\Delta y + \int_{e^1} p dy - \int_{w^1} p dy + \int_{n^1} q dx - \int_{s^1} q dx = H_{in}\Delta x\Delta y$$

This gives

$$\frac{dh_{C^1}}{dt}\Delta x\Delta y + p_{e^1}\Delta y - p_{w^1}\Delta y + q_{n^1}\Delta x - q_{s^1}\Delta x = H_{in}\Delta x\Delta y \tag{9.3}$$

For the momentum equation in the x-direction we have

$$\frac{\partial p}{\partial t} + \nabla \cdot (p\mathbf{u}) + gh\frac{\partial \eta}{\partial x} - Dh(\nabla \cdot \nabla u) + Sp = 0$$

Integrating over volume Ω_k^2

$$\int_{\Omega_k^2} \frac{\partial p}{\partial t} d\Omega + \int_{\Omega_k^2} \nabla \cdot (p\mathbf{u}) + gh\frac{\partial \eta}{\partial x} - Dh(\nabla \cdot \nabla u) + Spd\Omega = 0$$

Using the divergence theorem and the midpoint rule

$$\frac{dp_{C^2}}{dt}\Delta x\Delta y + \int_{e^2} pudy - \int_{w^2} pudy + \int_{n^2} pvdx - \int_{s^2} pvdx + gh_{C^2}\left(\int_{e^2} \eta dy - \int_{w^2} \eta dy\right)$$
$$- Dh_{C^2}\left(\int_{e^2} \frac{\partial u}{\partial x} dy - \int_{w^2} \frac{\partial u}{\partial x} dy + \int_{n^2} \frac{\partial u}{\partial y} dx - \int_{s^2} \frac{\partial u}{\partial y} dx\right) + S_{C^2} p_{C^2}\Delta x\Delta y = 0$$

Using again central differences gives and rewrite gives

$$\begin{aligned}
&\frac{dp_{C^2}}{dt}\Delta x\Delta y + p_{e^2}u_{e^2}\Delta y - p_{w^2}u_{w^2}\Delta y + p_{n^2}v_{n^2}\Delta x - p_{s^2}v_{s^2}\Delta x \\
&+ gh_{C^2}(h_{e^2} - h_{w^2})\Delta y - Dh_{C^2}\left(\frac{u_{E^2} - 2u_{C^2} + u_{W^2}}{\Delta x}\Delta y + \frac{u_{N^2} - 2u_{C^2} + u_{S^2}}{\Delta y}\Delta x\right) \\
&+ S_{C^2} p_{C^2}\Delta x\Delta y = 0
\end{aligned} \tag{9.4}$$

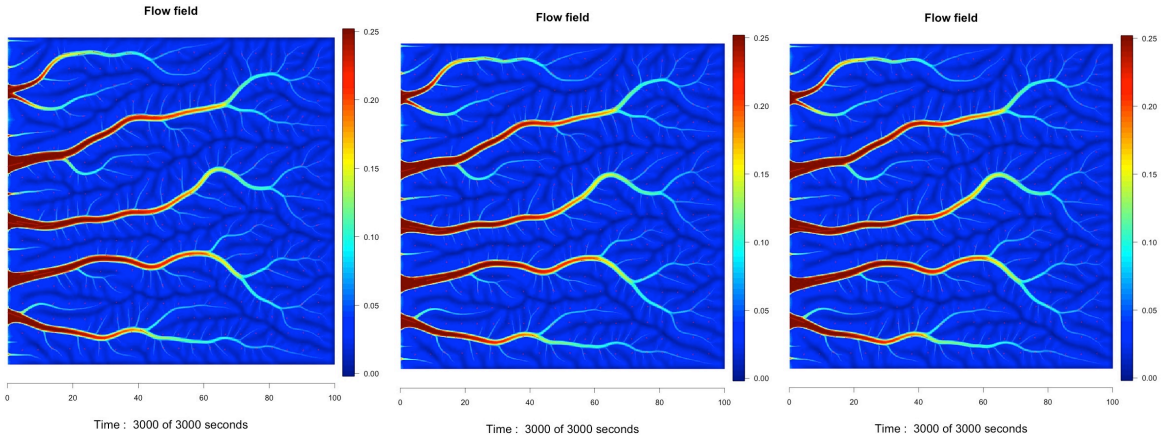
The unknown h_{C^2} can be determined by averaging over the neighbouring grid points: $h_{C^2} = \frac{h_{e^2} + h_{w^2}}{2}$. The velocity in the x-direction u can be determined in grid points belonging to p nodes by dividing by h_{C^2}.

9.2. Results

In Table 9.1 the parameters for three different cases can be found. First the parameters belonging

Case	$T\ (s)$	$dT\ (s)$	NXY	$L\ (m)$	$dX, dY\ (m)$
1	3000	0.025	512	100	$\pm$ 0.2
2	3000	0.025	256	100	$\pm$ 0.4

Table 9.1: Three cases with varying grid sizes and length.

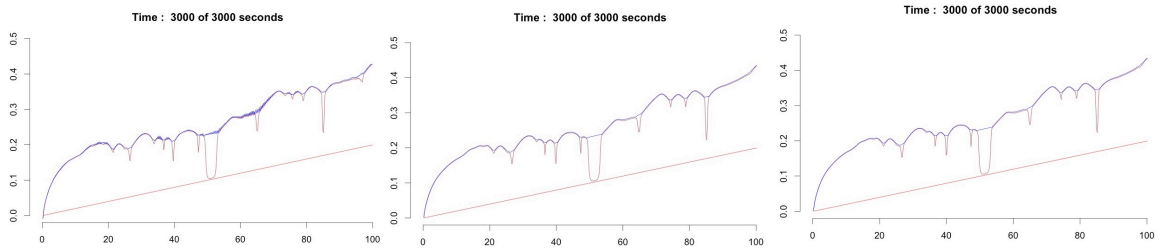


(a) Finite difference method on collocated grid.

(b) Finite volume method on staggered grid - approach 1.

(c) Finite volume method on staggered grid - approach 2.

Figure 9.3: Flow field salt marsh case 1



(a) Finite difference method on collocated grid.

(b) Finite volume method on staggered grid - approach 1.

(c) Finite volume method on staggered grid - approach 2.

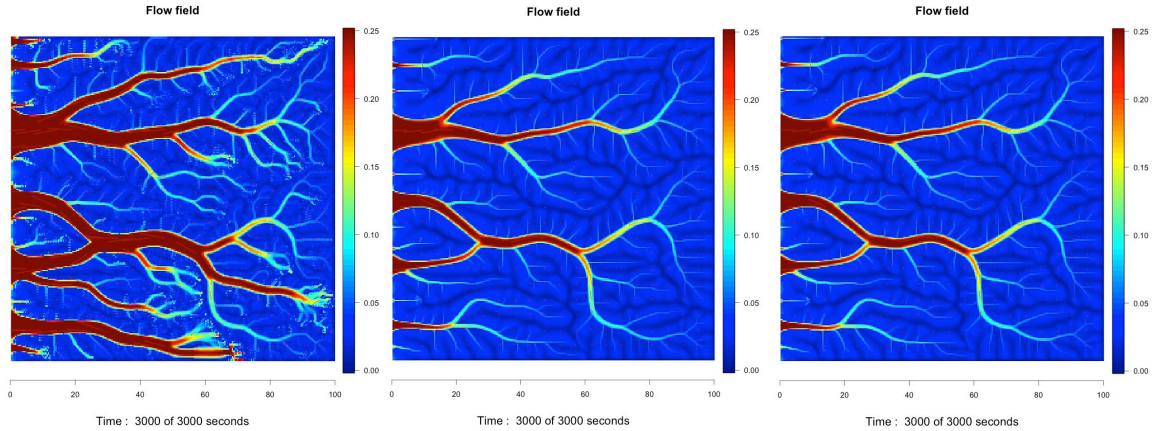
Figure 9.4: Cross-section along the x-axis for case 1.

to case 1 are used. In Figure 9.3 the flow field ($\sqrt{u^2 + v^2}$) after 3000 seconds for the finite difference method on a collocated grid and the finite volume method on a staggered grid is shown (both approaches). Figure 9.3 shows no noticeable differences between the three approaches. If we plot the cross section of the bathymetry (red line), sediment (brown line) and water depth (blue line) (Figure 9.4) we do notice a difference. For the staggered variants (Figures 9.4b and 9.4c) we do not experience any wiggles on the areas with a small water depth (almost equal or equal to H_{crit}). While for the collocated case we do experience wiggles in Figure 9.4a. If we use a twice as large grid size (case 2) the wiggles start to affect the flow field when using the finite difference method on a collocated grid (Figure 9.5a). However the staggered variants work fine (Figures 9.5b and 9.5c). In Figure 9.6 the size of the wiggles is illustrated by showing the cross-sections for the finite difference case on a collocated grid is shown.

Does the staggered implementations also have any advantages on the time step? As can be seen in table 9.2 both models crash for a time step of 0.075 seconds and work fine for time steps of 0.025 and 0.05 seconds. Thus there seems to be no significant difference in the timestep that can be used. In 6.6

Case	$T\ (s)$	$dT\ (s)$	NXY	$L\ (m)$	FDM Col	FVM Stag 1	FVM Stag 2
3	3000	0.025	256	50	✓	✓	✓
4	3000	0.050	256	50	✓	✓	✓
5	3000	0.075	256	50	✗	✗	✗

Table 9.2: Three cases with varying time steps.

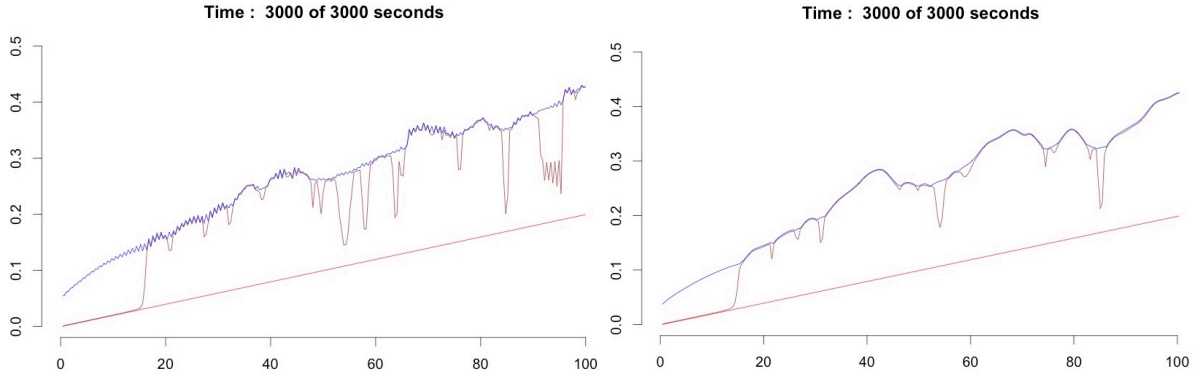


(a) Finite difference method on collocated grid.

(b) Finite volume method on staggered grid approach 1.

(c) Finite volume method on staggered grid approach 2.

Figure 9.5: Flow field salt marsh: case 2



(a) Finite difference method on a collocated grid.

(b) Finite volume method on staggered grid approach 2.

Figure 9.6: Cross-section along the x-axis for case 2.

we showed the computation times of $100{,}000$ iterations on a GPU and CPU. We have used a timestep of $0.05s$ and and end time of $3000s$. Consequently a grown salt marsh can be simulated in around $60{,}000$ iterations (takes approximately 24 seconds). While in reality it could take multiple decades.

9.3. Conclusion

The finite volume method on a staggered grid results in a smoother solution. Therefore, larger grid sizes still lead to good results. As a result, a reduction of computation time can be obtained for the finite volume methods on a staggered grid. For this specific case we did not notice any differences between approach 2 and 3. However, the third approach is more common. So from now on, we will use approach 3 for our implementations.

10

Results Tidal Case

For the results in Chapter 9 we have used a constant input of water on the whole computational domain. This results in water flowing from the higher elevated areas to the sea side boundary. In reality, salt marshes are regularly flooded by tides. This chapter is going to simulate tidal action. First a one-dimensional example is examined without any vegetation and a constant bottom elevation. Thereafter, a two-dimensional case, with equations for the vegetation density and bottom elevation.

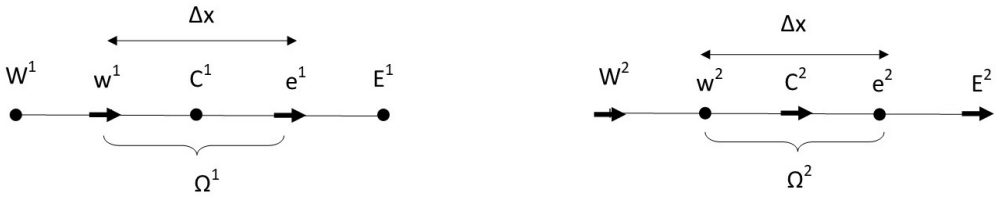
10.1. One-dimensional Shallow Water Equations.

The one-dimensional shallow water equations are given by

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x} p = 0 \tag{10.1a}$$

$$\frac{\partial}{\partial t} p + \frac{\partial}{\partial x}(pu) + gh\frac{\partial \eta}{\partial x} - Dh\left(\frac{\partial^2 u}{\partial x^2}\right) + \frac{g}{C^2}\|u\|u = 0 \tag{10.1b}$$

The control volumes for the velocity u and discharge p (equal to hu) are shown in Figure 10.1. For the results in this section the finite volume method (approach 2, explained in Chapter 9) has been used in combination with a staggered grid. Using the finite volume method gives us the following



(a) Control volume Ω_k^1 water depth h.

(b) Control volume Ω_k^2 discharge p.

Figure 10.1: Control volumes staggered grid.

discretisation

$$\frac{dh_{C^1}}{dt}\Delta x + p_{e^1} - p_{w^1} = 0 \tag{10.2a}$$

$$\frac{dp_{C^2}}{dt}\Delta x + p_{e^2}u_{e^2} - p_{w^2}u_{w^2} + gh_{C^2}\left(\eta_{e^2} - \eta_{w^2}\right) - Dh_{C^2}\left(\frac{u_{E^2} - u_{C^2}}{\Delta x} - \frac{u_{C^2} - u_{W^2}}{\Delta x}\right) + |u|_{C^2}u_{C^2}\Delta x\Delta y = 0 \tag{10.2b}$$

The variables p_{e^2}, p_{w^2} and h_{C^2} are determined by taking the average of the neighbouring nodes.

Boundary and Initial Conditions

The water depth is located at the boundaries of the domain. The boundary and initial conditions are shown in Table 10.1. For our simulations we assume the water does not go through the right boundary. Thus, u is set equal to zero and so we do not need a condition for h. On the left boundary the sinusoidal

Variable	Boundary Conditions		Initial Condition
	Left	Right	
u	$\frac{\partial u}{\partial x} = 0$	$u = 0$	$u = 0$
h	$h = h_{tide}$	(-)	$h = H_{crit}$

Table 10.1: Boundary and initial conditions of dependent variables.

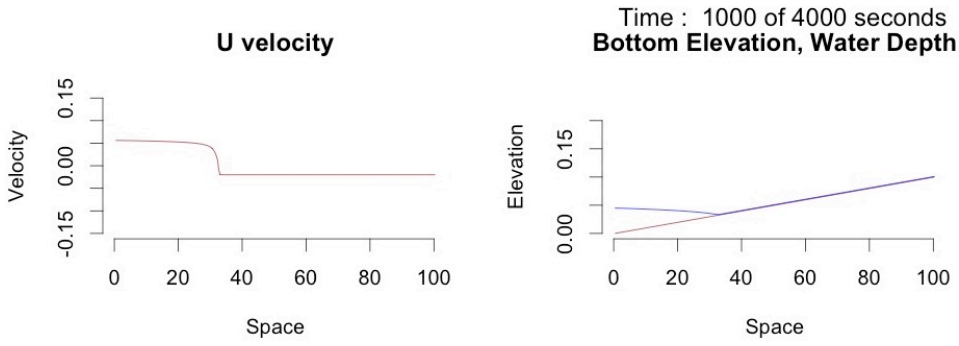
function given in Equation 10.3

$$h_{tide}(t) = \max(M_S + A_S \sin\left(\frac{2\pi t}{T} - \frac{\pi}{2}\right), H_{crit}) \tag{10.3}$$

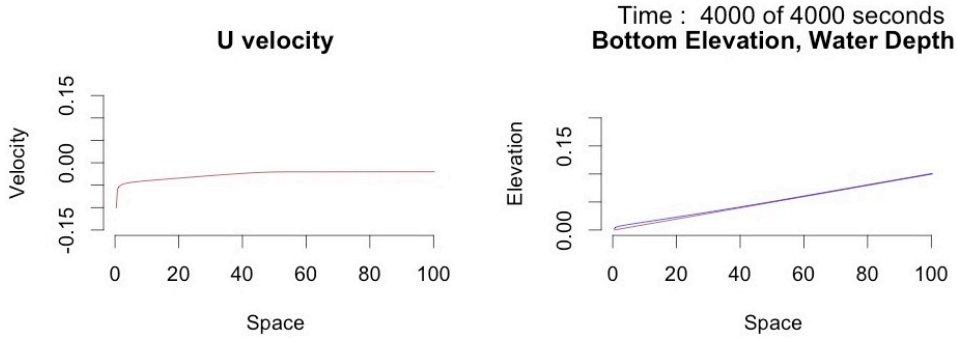
is imposed, which has an amplitude of A_S, a period of T seconds and fluctuates around M_S. By taking the maximum between the sinusoidal function and H_{crit}, we ensure the imposed water depth is always larger than or equal to the critical value. By subtracting $\frac{\pi}{2}$ from the time t, the tide starts at its lowest level. So the velocity is taken equal to zero and the water depth equal to our critical value on the whole computational domain. Differences in the tidal range like spring tides, neap tides and sea level rise are disregarded.

Results

The water depth and velocity during one tidal cycle at 1000 and 4000 seconds are shown in Figure 10.2. Figure 10.2b shows the flow speed at the left boundary unexpectedly increases a lot at 4000 seconds



(a) After $1000s$.



(b) After $4000s$.

Figure 10.2: One-dimensional tidal cycle with period $4000s$, with mean and amplitude of $0.045m$.

(minus sign, flow to the left). Besides, due to our wetting-drying algorithm and the imposed slope,

water always keeps flowing from the higher elevated parts to the lower elevated parts. If equations to account for vegetation and morphodynamic changes are included, erosion and plant mortality could take place in ”dry” areas. In the next two subsections, the left boundary and different approaches to obstruct the water from flowing from right to left will be investigated.

Water Depth Left Boundary

Figure 10.2b showed a sudden increase in velocity at the left boundary. At that moment the water depth at this boundary is equal to H_{crit}. However not all water has left our area. This results in a large negative u-velocity at the left boundary. By choosing a smaller mean and amplitude, for instance 0.02, the water has the time to leave the area. For larger amplitudes however the mean sea level could be chosen larger than the amplitude, or the the boundary could be moved further away. The water depth at the left boundary is then always substantial. For the result in Figure 10.3 a mean sea level of $0.055m$ and amplitude of $0.045m$ is chosen. So h_{tide} is always larger or equal to 0.01. Figure 10.3 shows no sudden increase of the velocity.

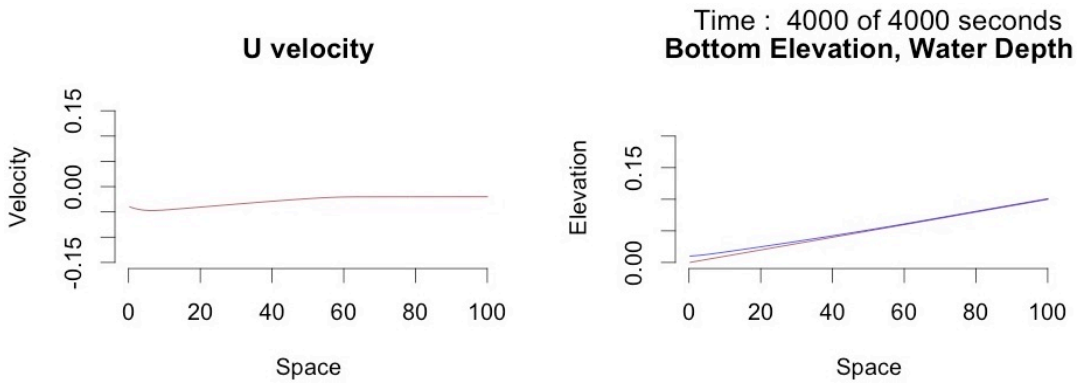


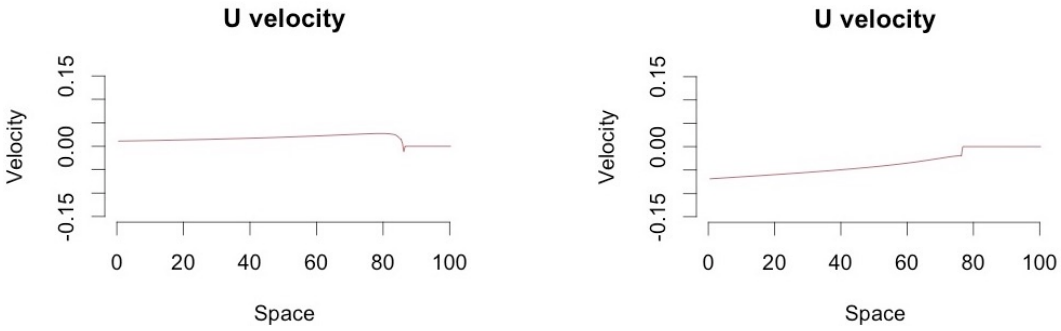
Figure 10.3: One-dimensional tidal cycle with period $4000s$ with mean $0.055m$ and amplitude of $0.045m$.

Obstruct Flow Dry Grid Points

First the flow is obstructed by setting the velocity in the x-direction equal to zero, if the water depth in both neighbouring nodes is equal to H_{crit}:

$$u_{C^2} = \begin{cases} 0 & \text{if } h_{w^2}, h_{e^2} \leq H_{crit} \\ \text{determined by Eq. } 10.2b & \text{else} \end{cases} \tag{10.4}$$

In Figure 10.4 the velocity after 2000 and 3000 seconds are shown. Figure 10.4 shows that imposing



(a) After $2000s$.

(b) After $3000s$.

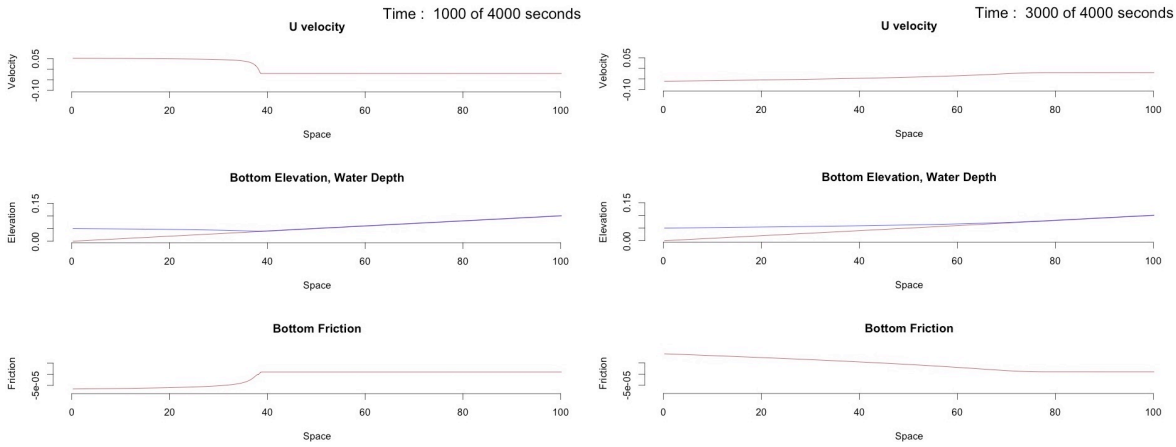
Figure 10.4: One-dimensional tidal cycle with period $4000s$ with mean and amplitude of $0.045m$ with no flow condition.

Equation 10.4 causes a jump in velocity at the front of the tide.
In order to obstruct the water from flowing from right to left, a certain no flow condition was used. Another option is to increase the bottom friction. This solution is not uncommon. Since our model does

not contain any vegetation, the friction term, C, is chosen equal to the the Chézy friction coefficient for the bottom, $C_b = 20$. For these results we used a mean sea level of $0.05m$ and the amplitude to 0.04. For the bottom friction, bf,we used

$$bf = -\frac{g}{C^2}\sqrt{u^2}u$$

The results are shown in Figure 10.5. The bottom friction has the opposite sign of u and since g and



(a) After $1000s$. (b) After $3000s$.

Figure 10.5: One-dimensional tidal cycle with period $4000s$, mean of $0.05m$ and amplitude of $0.04m$.

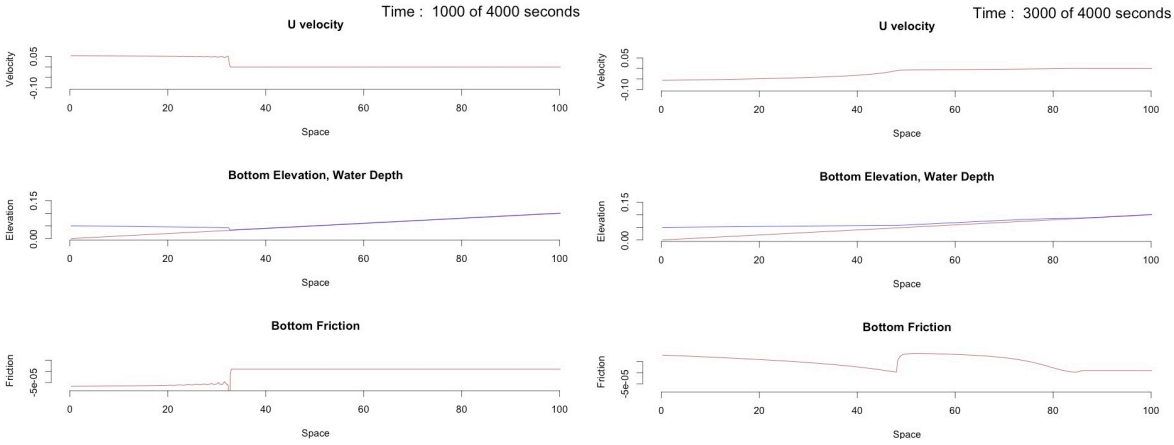
C are taken constant, it has the same shape as u. In order to increase the bottom friction, we now used the following friction term

$$bf = \begin{cases} \frac{g}{C_b^2}\sqrt{u^2}u\left(\frac{10.01H_{crit} - \frac{h_{e^2}+h_{w^2}}{2}}{0.01H_{crit}}\right) & \text{if } h_{e^2}, h_{w^2} \leq 10H_{crit} \\ \frac{g}{C_b^2}\sqrt{u^2}u & \text{else} \end{cases}$$

This new friction term will enlarge the bottom friction when both neighbouring nodes are smaller than $10H_{crit}$. When both neighbouring nodes correspond to a water depth of $10H_{crit}$, the term in brackets is equal to one. The smaller the water depth in the neighbouring nodes, the larger the bottom friction term. When the neighbouring nodes have a water depth of H_{crit} the term in brackets increases the bottom friction by approximately a factor 900. The results are shown in Figure 10.6 Figure 10.6a shows the increased bottom friction influences the front of the tide, it limits the flood. For this simple one-dimensional model, we can make sure the bottom friction does not influence the front of the tide by adding the constraint that the velocity has to be smaller than zero. The bottom friction in Figure 10.6b is not increasing to the right, because the velocity has a squared influence. Our new friction term indeed retards the flow. For a thin layer of water, with a depth of approximately H_{crit}, the water now flows down with a velocity of approximately $0.00067ms^{-1}$ instead of $0.02ms^{-1}$. However another consequence of this larger bottom friction is that water stays behind.

Conclusion

This section showed the results of the one-dimensional shallow water equations during one tidal cycle. We have shown that a water depth of H_{crit}, almost equal to zero, at the left boundary can cause weird phenomena. Thus from now on, a substantial water depth is ensured at the left boundary. We also noted that in case of a (small) slope in our bathymetry, water flows from higher elevated areas to lower elevated areas, since our wetting-drying algorithm retains a small layer of water. Two solutions were implemented. First a 'no flow condition', which sets the velocity equal to zero, if the water depth in the neighbouring nodes is equal to H_{crit} was used. Thereafter the bottom friction was increased for (almost) 'dry' grid points. Both solutions had their downsides. A third option is to just set the erosion equal to zero and make sure no vegetation dies due to flow stress if the water depth is equal to H_{crit}.



(a) After $1000s$. (b) After $3000s$.

Figure 10.6: One-dimensional tidal cycle with period $4000s$, mean of $0.05m$ and amplitude of $0.04m$ and increased bottom friction.

10.2. Two-dimensional Shallow Water Equations

This section models the two-dimensional shallow water equations in combination with equations for the bottom elevation and vegetation, as given in Equations 6.1a - 6.1c, 6.8, 6.9. The staggered grid is shown in Figure 9.1.

Boundary and Initial Conditions

The boundary and initial conditions are shown in Table 10.2. An underlying constant bathymetry of

Variable	Boundary Conditions				Initial Condition
	Bottom	Top	Left	Right	
u	$\frac{\partial u}{\partial n} = 0$	$\frac{\partial u}{\partial n} = 0$	$\frac{\partial u}{\partial n} = 0$	$u = 0$	$u = 0$
v	$v = 0$	$v = 0$	$\frac{\partial v}{\partial n} = 0$	$\frac{\partial v}{\partial n} = 0$	$v = 0$
h	–	–	$h = h_{tide}$	–	$h = 0.05 - 2z_b$
n_b	$\frac{\partial n_b}{\partial n} = 0$	$\frac{\partial n_b}{\partial n} = 0$	$\frac{\partial n_b}{\partial n} = 0$	$\frac{\partial n_b}{\partial n} = 0$	$n_b \in \{1,\ 0\}$
z_b	$\frac{\partial z_b}{\partial n} = 0$	$\frac{\partial z_b}{\partial n} = 0$	$z_b = 0$	$\frac{\partial z_b}{\partial n} = 0$	$z_b = 0.5 \cdot slope \cdot x$

Table 10.2: Boundary and initial conditions of dependent variables.

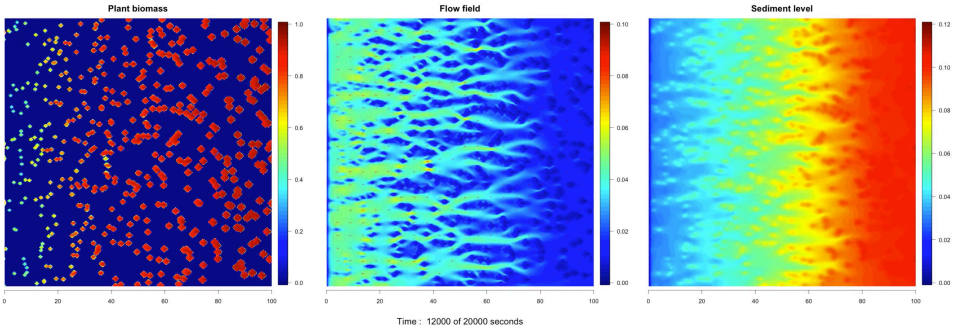
height z_b is used. Besides a mean sea level of $0.13m$ is taken, an amplitude of $0.08m$ and a period of $4{,}000$ seconds. Thus a water depth of $0.05m$ is always retained at the left boundary.

10.2.1. Results

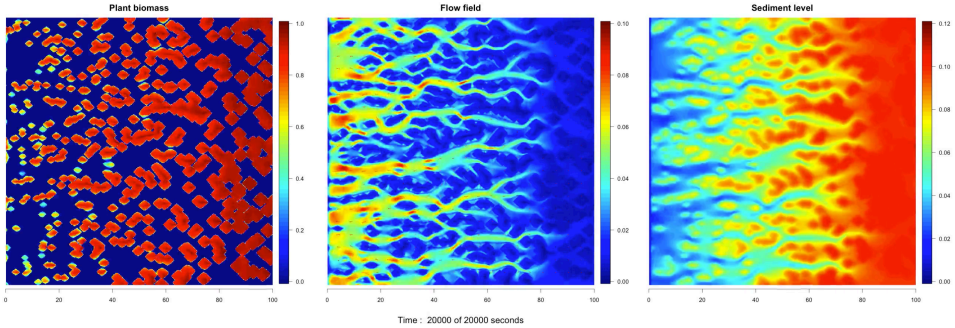
A time step of $0.025s$ and grid size of approximately $0.2m$ is used. The results of one special case (at $12{,}000$ and $20{,}000$ seconds) are presented in Figure 10.7. The parameters are chosen in such a way, that tidal creeks develop in a short period of time ($20{,}000$ seconds). But that does not necessarily mean that we have a network of meandering tidal creeks in the long-term. In Figure 10.8 the result, after $40{,}000s$ is shown. It is noticed that large hills of sediment have arisen and as a result, the vegetation has overgrown a large part of the domain. Thus the number of tidal creeks has decreased and the water only reaches the first 75 meters of our domain. The resulting cross-sections are shown in Figure 10.9. The bottom elevation is increasing with an increasing distance of the tidal creek. This represents the case shown in Figure 7.1a.

Dry Parts

In Section 10.1 it was mentioned that without any adjustments to the bottom friction or velocities, water keeps flowing from the higher elevated areas to the lower elevated areas. For the results in



(a) After $12{,}000\ s$.



(b) After $20{,}000\ s$.

Figure 10.7: Two-dimensional tidal cycle with period $4000s$.

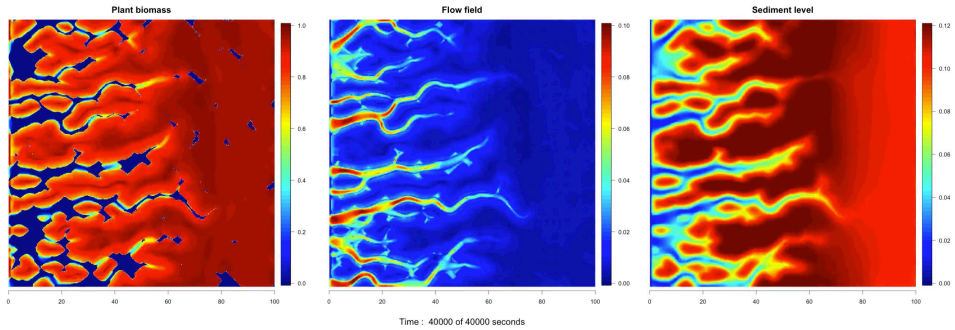
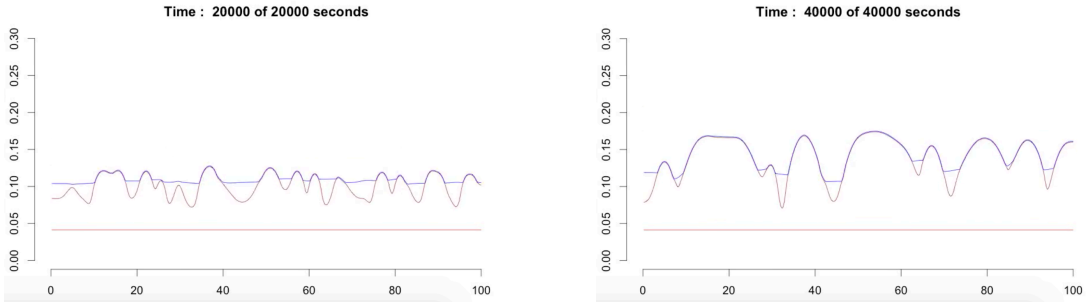


Figure 10.8: Two-dimensional tidal cycle with period $4000s$ - after $40{,}000$ seconds.

Subsection 10.2.1 nothing was done to obstruct the flow. This subsection will investigate the consequences because a large part of our area is often dry.

We have investigated two cases: with and without vegetation. The water depth at the left boundary is taken equal to the initial condition. Figure 10.10a shows the result after 4000 seconds without any vegetation present. The water depth is equal to H_{crit} for most of the computational domain and there is a constant flow to the seaside boundary. Figures 10.10b and 10.10c show the water depth and flow field after 2000 and 4000 seconds, when vegetation is present. The flow field shown in Figures 10.10b and 10.10c show an increase and decrease of the flow at certain locations. Looking at the water depth, there are areas with an increased water depth. We would expect that an increase in water depth would cause a decrease of water in other parts. However, this is not the case since we always set the water depth equal to H_{crit}.

In Figure 10.11 we have used a large square tussock to make the influence of the vegetation patches on the water depth and flow velocity more clear. The water depth increases the most at the vegetation patches, but there is also a small increase behind them. Regarding the flow field, the larger bed friction causes the water to slow down at the vegetation patch. But to the left and right, we notice an increase



(a) After $20{,}000\ s$. (b) After $40{,}000\ s$.

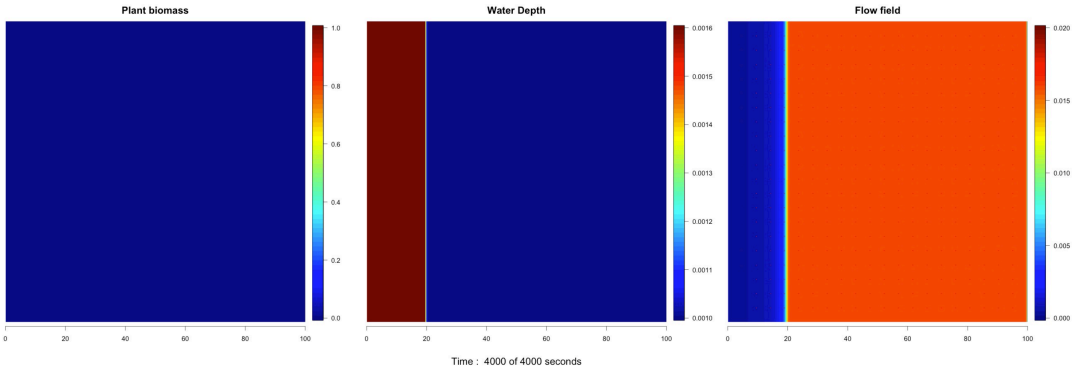
Figure 10.9: Two-dimensional tidal cycle with period $4000s$ - cross-section through the y-axis.

in flow speed.

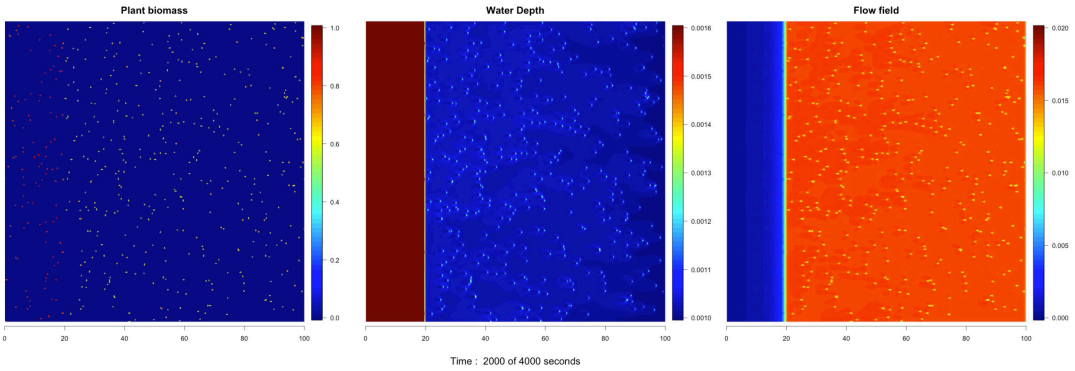
10.2.2. Conclusion

The results for a special tidal case were shown. Afterwards, the magnitude of the flow at dry areas was investigated. The vegetation had a significant effect on the flow field and water depth at dry areas. Which should be taken into account in the next chapters.

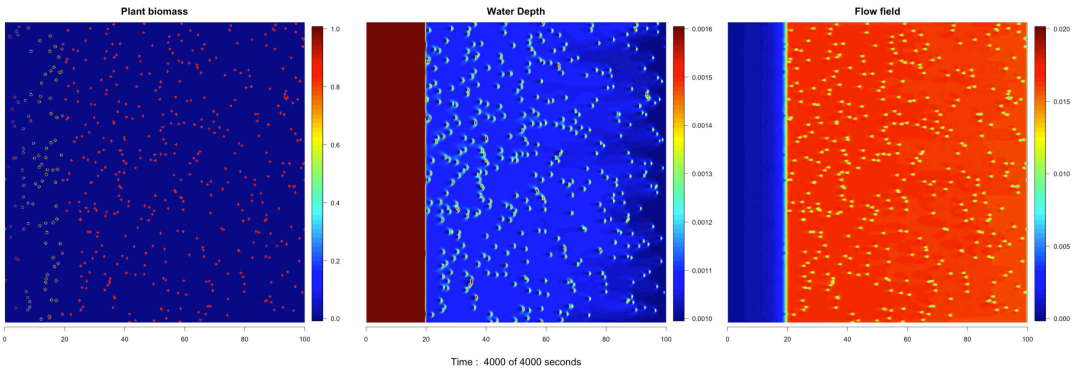
A very small timestep of 0.025 seconds was taken, which is comparable to the time step used in Chapter 9. However, for these results, it was not a real problem since a grown salt marsh could be simulated in about 3000 seconds. But for the results in this chapter the simulation time is already six times larger. Besides, the M2 tide (principal lunar semi-diurnal tide) takes 12 hours and 25.2 minutes (approximately 44700 seconds). As a consequence, if we want to simulate tidal action of a longer period for a longer time, a time step of 0.025 has a huge influence on our computation time. So in the next chapter, the time step is investigated.



(a) No vegetation, after $4000s$.



(b) Included vegetation, after $2000s$.



(c) Present vegetation, after $4000s$.

Figure 10.10: Two-dimensional system of equations.

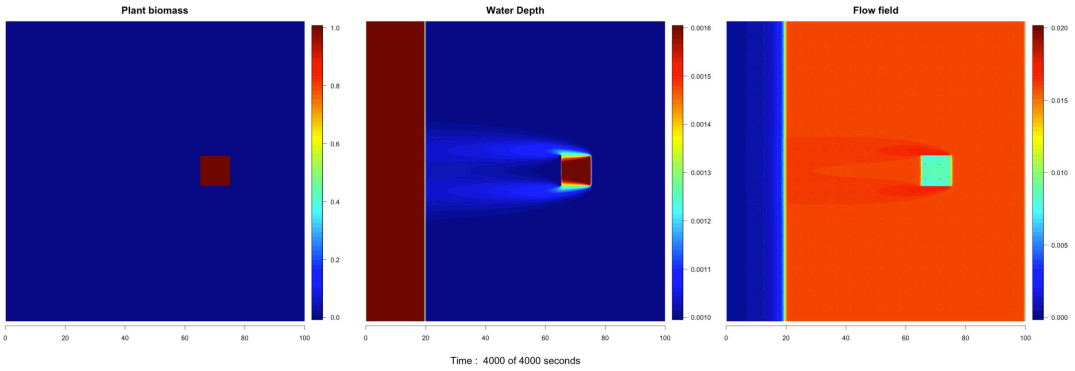


Figure 10.11: Two-dimensional system of equations.

11

Time Step Experiments

The time step has a huge influence on the accuracy, stability and of course the computation time. Chapter 9 simulated a grown salt marsh in about 3000 seconds by accelerating the vegetation and morphological processes. As a consequence, using a small time step was not of huge importance. However for the results in Chapter 10 tidal action is simulated. As mentioned in Subsection 10.2.2 the M2 tide takes approximately 44700 seconds and because of the otherwise long computation time, the processes were again accelerated. But it was noticed our time step had to be taken really small. This chapter is going to experiment how large we can take the time step for five special cases without our model becoming unstable and compares this number to the Courant–Friedrichs–Lewy number.

11.1. CFL condition

Explicit time integration of the shallow water equations on a rectangular grid is subject to a time step condition based on the Courant number for wave propagation [13]:

$$2\Delta t\sqrt{gh}\sqrt{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}} \leq 1 \tag{11.1}$$

where Δt is the time step (in seconds), g is the acceleration of gravity, h is the water depth, and $\Delta x, \Delta y$ represent the grid size. When Δx is equal to Δy Equation 11.1 reduces to

$$2\sqrt{2}\sqrt{gh}\frac{\Delta t}{\Delta x} \leq 1 \tag{11.2}$$

For one dimension the Courant number for wave propagation is given by

$$2\sqrt{gh}\frac{\Delta t}{\Delta x} \leq 1 \tag{11.3}$$

So the two-dimensional case is more restrictive by a factor $\sqrt{2}$.

11.2. Results

The first three cases simulate a tidal cycle for the one-dimensional shallow water equations, the last two cases simulate two-dimensional flow around a square vegetation area.

11.2.1. Case 1a

The one-dimensional shallow water equations are used. An area of $100m$, 256 grid points and thus a grid size of $0.39m$ is taken. A sinusoidal function (Equation 10.3) with mean $0.04m$, amplitude $0.03m$ and a period of 4000 second is used. We have varied the timestep and used an end time of $20{,}000s$. An explicit (forward Euler) and semi-implicit (Euler) scheme will be used. In Subsection 7.1.3 it was explained how to implement the forward Euler method in OpenCL. The results can be found in Table 11.1.

Explicit Method

The explicit method already crashes for a time step of 0.025 seconds after a simulation period of 800 seconds. A time step of 0.0025 works fine. Note that from Equation 11.3 a timestep restriction of $0.24s$ follows. We used that the water depth is at most $0.07m$ at the left boundary. However, our method only results in a stable solution for dT equal to $0.0025s$.

Semi-implicit Method

The results are again illustrated in Table 11.1. A time step of $0.5s$ is too large. However, for the explicit time-integration, we could not use a timestep of $0.01s$ while for the semi-implicit method already a timestep of $0.25s$ could be used for this specific case. Besides this time-integration method uses less memory resources (Subsection 7.1.3).

Semi-implicit: Larger Period

For case 1a a period of $4000s$ was used. The M2 tide in the Netherlands has a period of 12.25 hours (approximately $44{,}100$ seconds). Using this "slower" tide might enable us to take a larger timestep because the changes each iteration are smaller. A period of $50{,}000s$ was used. Unfortunately, using a considerably larger period, does not enable us to use a timestep of $0.5s$.

11.2.2. Case 1b

This case will investigate if our small time step is the result of our wetting-drying method. Because after each iteration the solution to the shallow water equations is adjusted if the water depth falls below H_{crit}. The focus of this report is not on implementing and reviewing wetting-drying schemes. However, the time step needs to be taken so small, that it has a huge impact on our computation time. So one other wetting-drying approach is implemented for a one-dimensional example. Each time step we performed Algorithm 2 to determine the water depth h. It checks if all the resulting depths are larger than or equal to H_{crit}. If not, it finds a grid point with a smaller depth and sets the velocity in the neighbouring nodes equal to zero. Then we again update h, and we do this until there is no grid point with a smaller water depth than H_{crit}. Algorithm 2 was implemented in Matlab.

```
Update t = t + dT;
counter = 0;
for (boundary grid points) do
    Update h by boundary condition;
end
if (counter == 0 || min(h) < H_crit) then
    if (counter ≠ 0) then
        ∃ i ∈ {i : h(i) < HCrit}
        u_old[i] = 0;
        u_old[i − 1] = 0;
        p_old[i] = 0;
        p_old[i − 1] = 0;
    end
    Update h by explicit Eq. 6.1c;
    counter = counter+1;
end
```

Algorithm 2: Pseudo-code wetting drying approach.

Unfortunately, the results in Table 11.1 show we can still not use a time step of 0.1 seconds. Besides, even if we could use a larger time step, the computation time could be almost the same or larger. Since each iteration, the water depth and velocity might has to be updated several times, until the water depth in all grid points is above a certain threshold.

11.2.3. Case 2a

Cases 1 simulated a one-dimensional tidal cycle, in which the wetting-drying method played a huge role since a large part of our area is often "dry". Two different wetting-drying approaches were used, but no signification difference was found. Now two-dimensional flow (without tidal action) is simulated.

In this example, the water depth does not fall below the critical threshold value. A constant vegetation density and bottom elevation are taken. A square tussock of vegetation density one and a vegetation density of zero elsewhere will be used, like in Figure 10.11. An area of $100 \times 100m^2$, 256 grid points in each direction and thus again a grid size of $0.39m$ is taken. We have varied the time step and used an end time of $20{,}000\ s$.

Semi-implicit Method

The Courant number (Equation 11.1) for explicit time-integration of the shallow water equations gives us a time step restriction of 0.0254 seconds (maximum water depth of approximately $3m$). The results are shown in Table 11.1. Note that the timestep we could take is now close to the Courant number.

11.2.4. Case 2b

For this case we used the same equations as for case 2a, but we used a larger grid size. Instead of a grid size of $0.39m$ we used a grid size of $3.9m$. Equation 11.2 now results in a time step restriction of $0.254s$. In Table 11.1, it is shown that we can now use a time step of 0.25 seconds, which is indeed a factor ten higher.

It should be realized that halving the grid size means that we get four times the number of grid points. Moreover, usually the time step needs to be halved as well, so the amount of work is 8 times that of the basic run.

11.3. Conclusion

In this chapter, we wanted to find out if the small time step was a result of our wetting-drying approach or the simulation of tidal action. Case 1 used the one-dimensional shallow water equations and case 2 the two-dimensional shallow water equations. Case 1a showed a that for the semi-implicit time-integration a much larger time step could be taken than for the explicit time-integration. For the restricted set of time steps that we tested, it was a factor of 100. We can conclude the time step has to be chosen much smaller than the CFL number. Besides, for this specific case, slowing down the tide, by enlarging the period, does not enable us to use a larger timestep. The limiting factor might be our current wetting-drying approach, so case 1c investigated another approach. However, no significant increase in the time step was noticeable. Cases 2a and 2b did not simulate tidal action nor did they use a wetting-drying method. However, for the explicit method, we could not use a 100 times smaller time step. Case 2b showed that by lowering the grid size by a factor ten, indeed a ten times larger time step could be used. So the wetting-drying method and simulation of tidal action do not cause the small time step. In [7] it is mentioned that for a parabolic equation (such as the shallow water equations including diffusion) the CFL condition places more severe constraints on an explicit method. More information about the CFL condition is given in Section 4.3. Unfortunately, leaving out the diffusion term did not result in a larger time step, since it made our solution more unstable.

Time step (s)	Case 1a		Case 1b	Case 2a		Case 2b
	Explicit	Semi-implicit	Explicit	Explicit	Semi-implicit	Semi-implicit
0.0025	✓	✓	✓	✗	✓	✓
0.01	✗	✓	✗	✗	✓	✓
0.025	✗	✓	✗	✗	✓	✓
0.1	✗	✓	✗	✗	✗	✓
0.25	✗	✓	✗	✗	✗	✓
0.5	✗	✗	✗	✗	✗	✗

Table 11.1: One-dimensional and two dimensional cases (explicit CFL condition, 1D: $0.24s$, 2D: $\pm 0.025s$ for case 2a and $\pm 0.25s$ for case 2b)

12

Sediment Transport

Until now we have let sediment deposition depend on a constant sediment input S_{in}. To make our model more realistic sediment transport is added in this chapter. General information about sediment transport has already been given in Subsection 3.2.1. In this chapter, the focus is on the modelling of cohesive suspended sediment. First, a one-dimensional transport equation is examined. Afterwards, we will use the two-dimensional transport equation with sources and sinks and couple it to the bed elevation. As mentioned in Section 7.2, we want to obtain the situation shown in Figure 7.1b instead of the result in Figure 10.9, which is similar to the situation in Figure 7.1a.

12.1. One-dimensional Transport Equation

A one-dimensional transport equation is used in combination with the one-dimensional shallow water equations (Equations 10.1a and 10.1b) to simulate the inflow of sediments at the left boundary. The one-dimensional depth-averaged transport equation is given by

$$\frac{\partial d}{\partial t} + \frac{\partial}{\partial x}(ud) + \frac{\partial}{\partial x}\left(hD\frac{\partial c}{\partial x}\right) = 0, \tag{12.1}$$

where c is the depth-averaged mean sediment concentration, the variable d is used to denote hc. The control volume of d is the same as the control volume of the water depth h, as shown in Figure 10.1a. which is discretised as follows

$$\frac{dd_{C^1}}{dt}\Delta x + (ud)_{e^1} - (ud)_{w^1} - \left(\left(h_{e^1}D\frac{c_{E^1} - c_{C^1}}{\Delta x}\right) - \left(h_{w^1}D\frac{c_{C^1} - c_{W^1}}{\Delta x}\right)\right) = 0 \tag{12.2}$$

We can determine d in the grid points e^1 and w^1 by averaging between the neighbouring nodes. More information about this derivation can be found in Appendix A.

Initial and Boundary Conditions

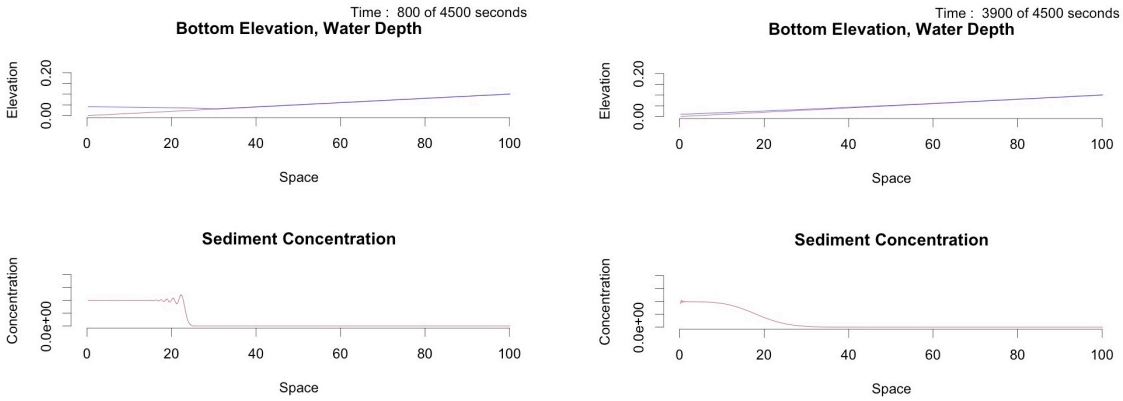
For u and h the same boundary and initial conditions as in Section 10.1 are chosen. For c concentration of zero at the beginning of our simulation is assumed. A Dirichlet condition is imposed at both boundaries.

Variable	Boundary Conditions		Initial Condition
	Left	Right	
u	$\frac{\partial u}{\partial n} = 0$	$u = 0$	$u = 0$
h	$h = h_{tide}$	-	$h = H_{crit}$
c	$c = C_1$	$c = 0$	$c = 0$

Table 12.1: Boundary and initial conditions of dependent variables.

Results

We have plotted the resulting sediment concentration (left boundary) with and without diffusion in Figure 12.1. For the case without diffusion, we already experience wiggles during inflow, but also when we use a small diffusion coefficient we experience wiggles during outflow.



(a) Without diffusion. (b) With diffusion.

Figure 12.1: One-dimensional tidal cycle with period $4000s$.

Above we used a centered method, symmetric about the point where we are updating the solution, to determine $(ud)_{e^1}$ and $(ud)_{w^1}$. However the horizontal transport of dissolved substances in rivers, estuaries, and coastal seas is dominated by advection. The horizontal diffusion in flow direction is of secondary importance. The consequence is that the transport equation is advection dominated, has a wave character, and from a mathematical point of view is of hyperbolic type [13]. For hyperbolic problems, we expect information to propagate as waves moving along characteristics [7]. This idea gives rise to upwind methods in which the information for each characteristic variable is obtained by looking in the direction from which the this information should be coming. So instead of averaging to determine d_{e^1} and d_{w^1}, a first order upwind scheme (Equation 12.3) can be used for the discretisation of the convective terms in order to remove the oscillations.

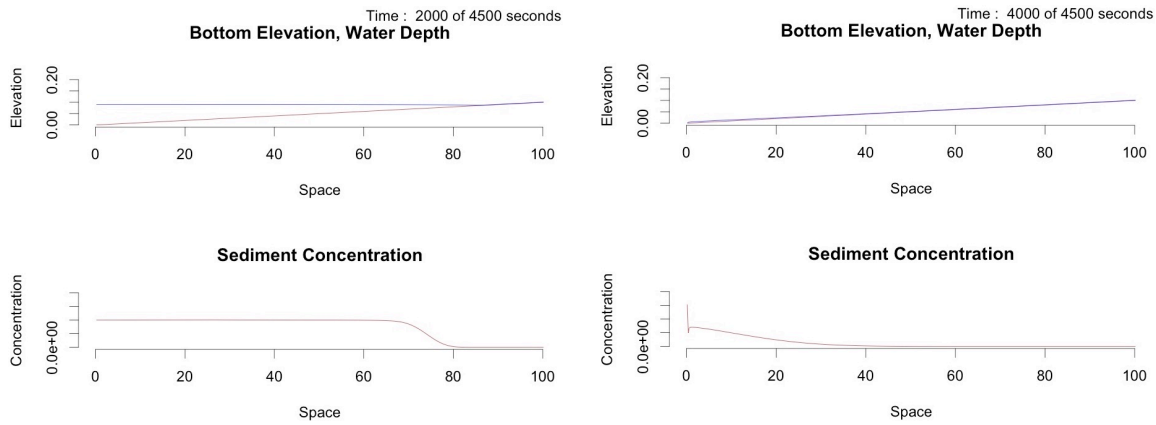
$$d_{e^1} = \begin{cases} d_{C^1} & \text{if } u_{e_1} > 0 \\ d_{E^1} & \text{else} \end{cases}, \quad d_{w^1} = \begin{cases} d_{W^1} & \text{if } u_{w_1} > 0 \\ d_{C^1} & \text{else} \end{cases} \tag{12.3}$$

As mentioned in [10], for first order upwind schemesthe solution may correspond to a larger diffusion coefficient, which is sometimes much larger than the actual diffusivity.

Again an incoming tide from the left boundary was simulated. In Figure 12.2 the bottom elevation, water depth and sediment concentration is shown. We notice that when the water depth reaches H_{crit} at the left side of the boundary the concentration increases a lot. So as was recommended in Section 10.1, the boundary will be placed further away, by increasing the mean sea level M_s. The results are illustrated in Figure 12.3.

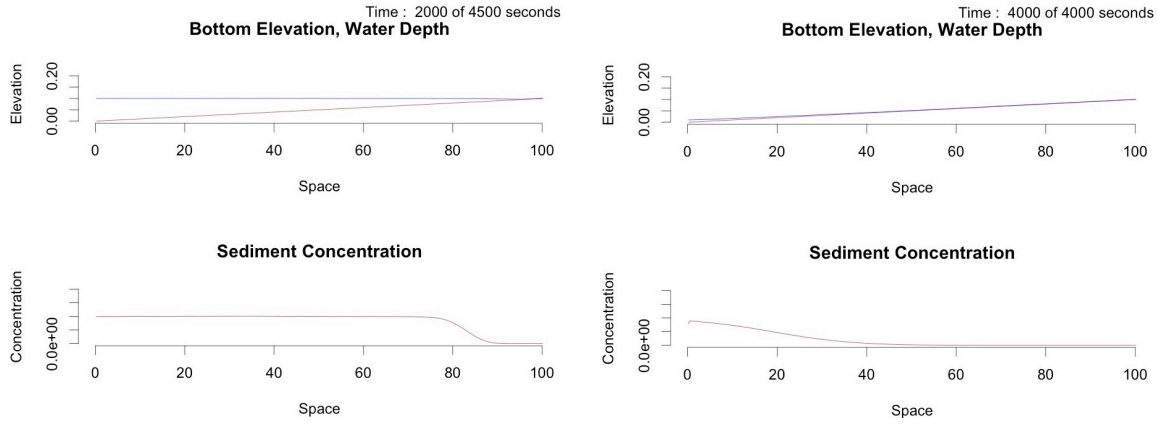
Conclusion

It was shown that when the diffusion is small, using an upwind method prevents getting an oscillating solution. So from now on the upwind scheme in Equation 12.3 is used for the convective terms.



(a) After $2000s$. (b) After $4000s$.

Figure 12.2: One-dimensional tidal cycle with period $4000s$ and M_s equal to A_s.



(a) After $2000s$. (b) After $4000s$.

Figure 12.3: One-dimensional tidal cycle with period $4000s$ and M_s larger than A_s.

12.2. Two-dimensional Tides - Coupled

We have now shown the transport of sediment for a one-dimensional case. In this subsection, we will move to two dimensions and couple the sediment concentration to the bed elevation. We will use the depth-averaged transport equation

$$\frac{\partial(hc)}{\partial t} + \nabla \cdot (h\mathbf{u}c) - \nabla \cdot (hD\nabla c) = Er - Dr \tag{12.4}$$

Here c is the depth-averaged mean sediment concentration, Er is the flux of the sediment erosion and Dr the flux of the sediment deposition. The bottom elevation is determined by

$$\frac{\partial z_b}{\partial t} = \frac{Dr - Er}{\rho_s} \tag{12.5}$$

in which ρ_s represents the sediment density.

12.2.1. Deposition Formula

In order to let the deposition depend on the sediment concentration, S_{in} is changed into a constant times the sediment concentration

$$Dr = \rho_s S_{in}\left(\frac{h_{eff}}{Q_s + h_{eff}}\right) \rightarrow Dr = \rho_s c C_{sed}\left(\frac{h_{eff}}{Q_s + h_{eff}}\right). \tag{12.6}$$

Initial and Boundary Conditions

Table 12.2 shows the initial and boundary conditions.

Variable	Boundary Conditions				Initial Condition
	Bottom	Top	Left	Right	
u	$\frac{\partial u}{\partial n} = 0$	$\frac{\partial u}{\partial n} = 0$	$\frac{\partial u}{\partial n} = 0$	$u = 0$	$u = 0$
v	$v = 0$	$v = 0$	$\frac{\partial v}{\partial n} = 0$	$\frac{\partial v}{\partial n} = 0$	$v = 0$
h	–	–	$h = h_{tide}$	–	$h = 0.05 - 2z_b$
n_b	$\frac{\partial n_b}{\partial n} = 0$	$\frac{\partial n_b}{\partial n} = 0$	$\frac{\partial n_b}{\partial n} = 0$	$\frac{\partial n_b}{\partial n} = 0$	$n_b \in \{1,\ 0\}$
z_b	$\frac{\partial z_b}{\partial n} = 0$	$\frac{\partial z_b}{\partial n} = 0$	$z_b = 0$	$\frac{\partial z_b}{\partial n} = 0$	$z_b = 0.5 \cdot slope \cdot x$
c	$\frac{\partial c}{\partial n} = 0$	$\frac{\partial c}{\partial n} = 0$	$\frac{\partial c}{\partial n} = 0$	$c = 0$	$c = 0$

Table 12.2: Boundary and initial conditions of dependent variables.

Results

We have again simulated five tidal cycles using a period of $4000s$. The results after $20{,}000$ seconds are shown in Figure 12.4 for a value of 0.1 for C_{sed}. The bed elevation stays quite flat and no deep tidal creeks develop. Therefor also the vegetation is able to diffuse more, compared to Figure 10.7. Also for other values of C_{sed} (1.0 and 0.0001) there was no development of tidal creeks. This is probably because a lower C_{sed} results in a larger concentration and thus still a larger deposition.

The erosion rates, deposition rates and sediment concentration after $200,\ 1000$ and 3000 seconds are shown in Figure 12.5. First, we notice the deposition rate and erosion rate behave quite similar. The deposition formula in Equation 12.6 depends on the sediment concentration and water depth. Since more erosion results in a larger sediment concentration, the deposition rate has the same shape as the erosion rate. So there is almost no difference in sedimentation and erosion rates and little bottom elevation differences arise. We conclude this deposition formula does not give us the desired results. In the next section, a different approach is tried: the Partheniades-Krone formulation.

Secondly Figure 12.5a shows a large sediment amount (kgm^{-3}) in the dry areas. As mentioned in Chapter 10 our wetting-drying method causes a constant flow from the higher elevated to the lower

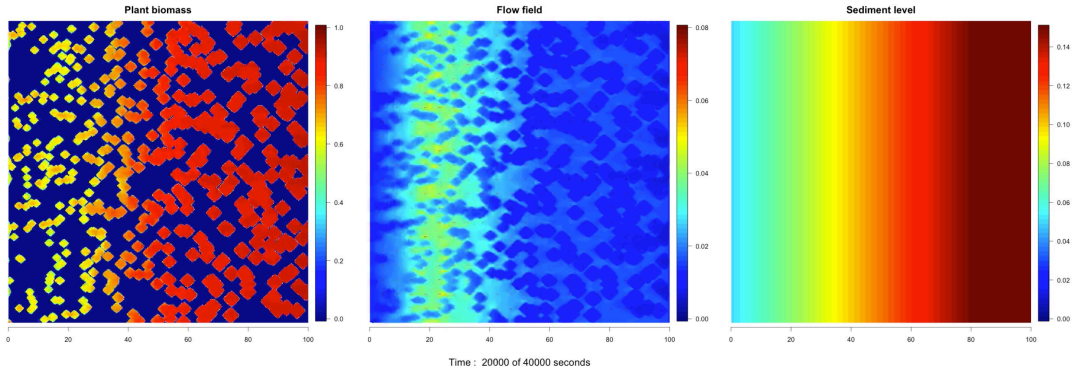
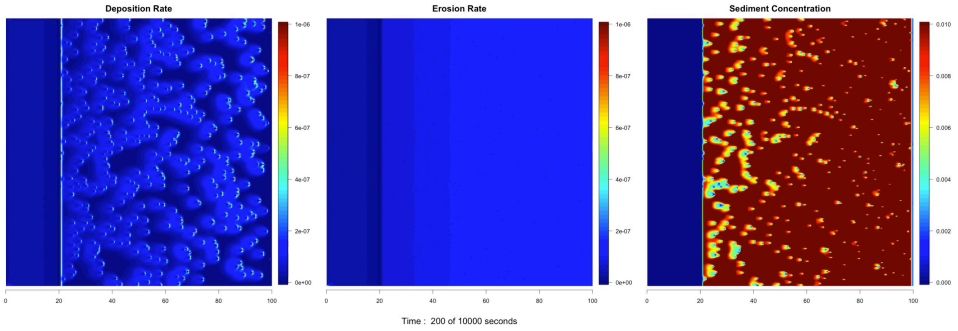


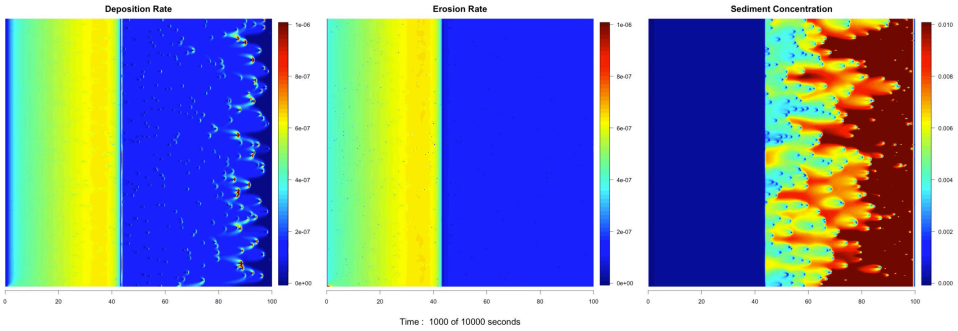
Figure 12.4: Two-dimensional tidal cycle with period $4000s$ for C_{sed} equal to 0.1.

areas, which causes erosion. This flow is increased by vegetation. However, the water depth is (almost) equal to H_{crit} in dry areas, thus the sediment does not fall down. In Figures 12.5b and 12.5c the concentration decreases, because the water depth increases somewhat (because of the vegetation, Subsection 10.2.1) and sediment is able to deposit.

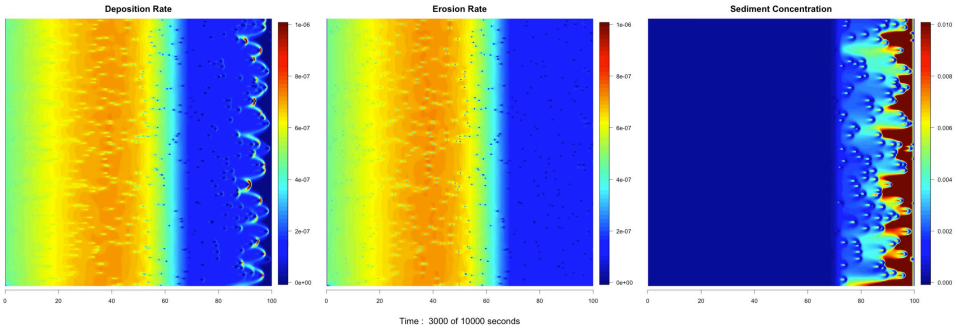
The Partheniades Krone formulation, used in the next section, ensures erosion to take place if the bottom stress is above a certain value. Thus this "artifact" might disappear. For this model, we can circumvent the second problem by ensuring erosion and deposition only occur when the water depth is larger than a certain threshold. We could for instance check if the water depth is larger than 1.1 times H_{crit} for erosion and sedimentation to occur. The result at $200s$ are shown in Figure 12.6



(a) After $200s$.



(b) After $1000s$.



(c) After $3000s$.

Figure 12.5: Two-dimensional tidal cycle with period $4000s$.

12.2.2. Partheniades-Krone

The previous section showed the used deposition approach gives no good results. A well-known method to determine sedimentation and erosion rates is the Partheniades-Krone formulation, it defines

$$Dr = w_s c \left(1 - \frac{\tau}{\tau_{cr,d}}\right) \quad \text{when } \tau < \tau_{cr,d} \tag{12.7}$$

$$Er = M \left(\frac{\tau}{\tau_{cr,e}} - 1\right) \quad \text{when } \tau > \tau_{cr,e} \tag{12.8}$$

where c is the suspended sediment concentration, τ the bottom shear stress, $\tau_{cr,d}$ the critical shear stress for sedimentation, $\tau_{cr,e}$ the critical shear stress for erosion, M the erosion parameter and w_s is the settling velocity of the suspended sediment [22]. τ is given by the magnitude of the bed roughness ($\boldsymbol{\tau}_b$, explained in Section 6.1.1)

$$\tau = |\boldsymbol{\tau}_b| = \rho g \|\mathbf{u}\|^2 \frac{1}{C^2} \tag{12.9}$$

in which C represents the Chézy coefficient. In our simulations we have used the baptist formulations for C, which represents the total effect of the bottom and vegetation on the bed shear stress. In Section

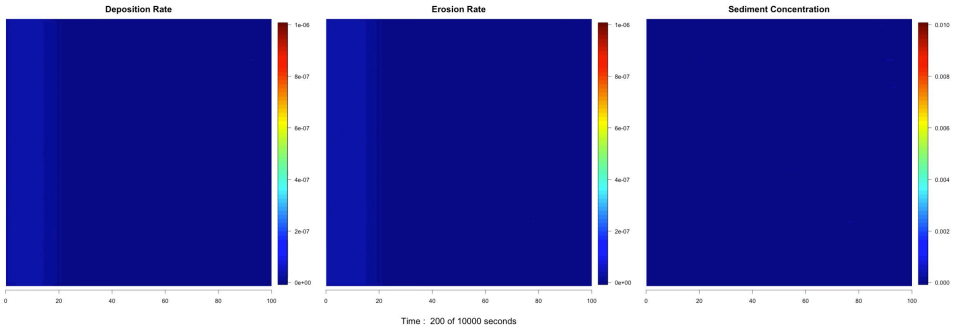


Figure 12.6: Transport equation tide, after $200s$.

6.1.1 the Baptist formulation for the Chézy coefficient is explained. But now we are only interested in the bottom shear stress (not including the drag force exerted by plants). We parametrize the bottom shear stress by substituting $C = C_b$ in Equation 12.9, where C_b is the Chézy coefficient of the bottom. Another option is to use Mannings formulation:

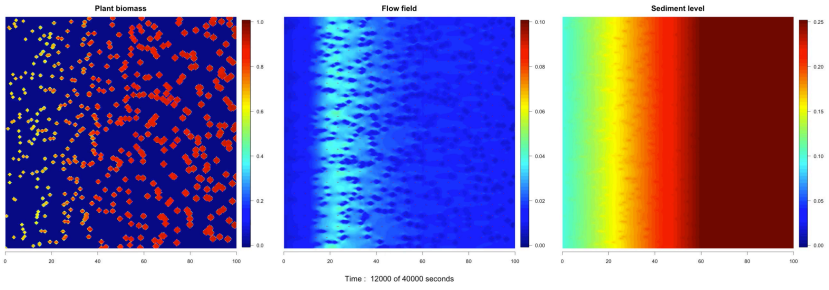
$$C_b = \frac{h^{\frac{1}{6}}}{n} \tag{12.10}$$

Results

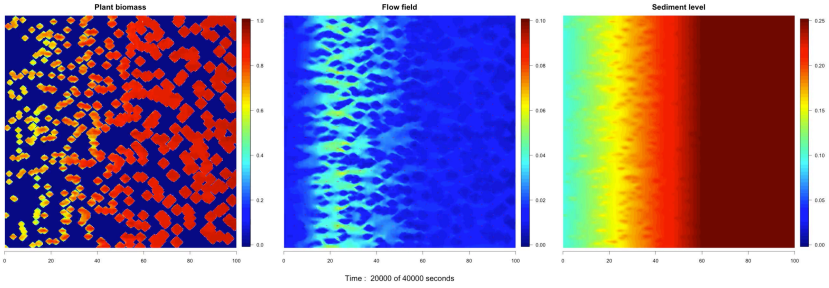
Multiple parameter sets were experimented with. The results of the parameter set in Table 12.3 are shown in this subsection. Figure 12.7 shows the vegetation, flow field and bottom elevation at $12{,}000$ and $20{,}000$ seconds. Shallow tidal creeks are formed.

Parameters			
$M \sim 0.001$	$\tau_{cr,e} \sim 0.16$	$w_s \sim 0.01$	$\tau_{cr,d} \sim 0.16$

Table 12.3: Partheniades-krone parameters.



(a) After $12{,}000s$.

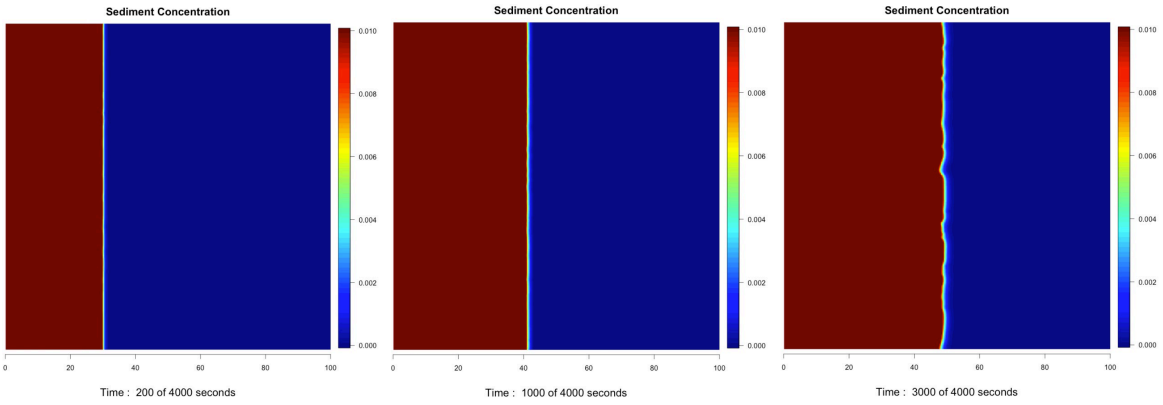


(b) After $20{,}000s$.

Figure 12.7: Two-dimensional tidal cycle with period $4000s$ using the Partheniades Krone formulation.

Dry Areas

In Figure 12.5 it was noticed the flow at the 'dry' grid points caused a large concentration. The concentration at 200, $1{,}000$ and $3{,}000$ seconds is shown in Figure 12.8. The bottom stress is smaller



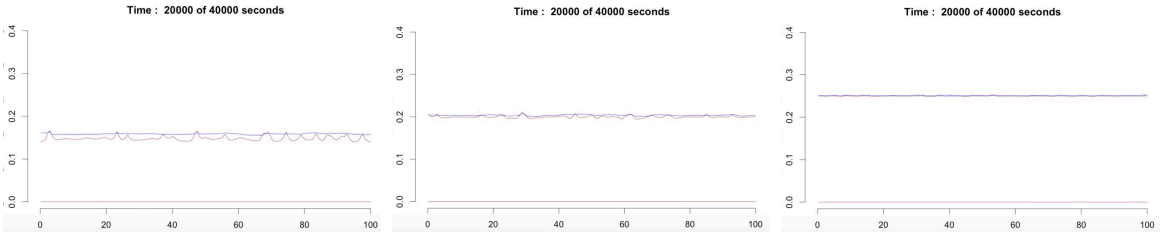
(a) After $200s$. (b) After $1{,}000s$. (c) After $3{,}000s$.

Figure 12.8: The sediment concentration.

than 0.10 at the dry grid points and therefore no erosion takes place, since the critical value for erosion to occur is set to 0.16.

Formation of Levees

As shown in Figure 12.7 small hills of sediment and small creeks have developed. In Figure 12.9 three cross-sections along the y-axis are shown. As expected, the further away we move from sea-side boundary, the smaller the sediment differences are. The bottom stress formulation in Equation 12.9



(a) At one-fifth of the x-axis. (b) At two-fifth of the x-axis. (c) At three-fifth of the x-axis.

Figure 12.9: Cross-section after $20{,}000s$.

is fully dependent on the magnitude of $\mathbf{u}$. In the middle of the vegetation patches, the vegetation density is the highest and thus the water is slowed down the most. As a consequence, the bottom stress is the lowest in the middle of the vegetation patches. This results in the formation of small hills of sediment of the form 7.1a.

We expect levees to form after a longer time frame on the right side of our domain when hills have already developed. As shown in Figure 12.10, the water still floods the whole area at three-fifth of the x-axis, because there are no hills presents.

In this section we were not able, to find a parameter set to show the formation of levees on such a short time frame. Morphological changes would have to happen more rapidly, or maybe we would need to use a longer simulation. Besides, we would need to make sure there is a large enough sediment concentration when the water overtops a hill. Furthermore, a developed vegetation cover is needed, such that the decrease in sedimentation rates with increasing distance from the creeks is rapid enough. In [34] it is indeed mentioned that levees are generally not developed along tidal creeks that are present on unvegetated mudflats.

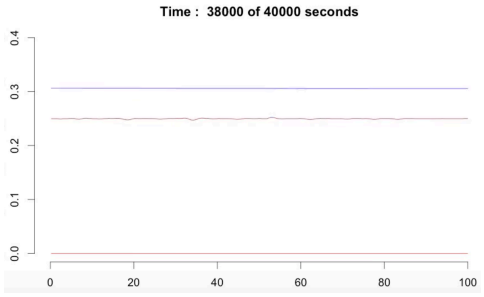


Figure 12.10: Cross-section after $20{,}000s$ at three-fifth of the x-axis.

Conclusion

This section coupled the transport of sediments to the modelling of the bed elevation by using the deposition and erosion rates. Two different deposition and erosion approaches were used. The first approach was not able to model the formation of tidal creeks. The second one was, but the bottom elevation differences were still small after $20{,}000$ seconds and thus no deep tidal creeks and sediment hills developed. As a consequence, we could not observe the situation in Figure 7.1b.

13

Validation

Using computers to simulate the real world is the main task in scientific computing. The complexity of the shallow-water equations impedes us to determine the analytical solution, so an approximation will have to be calculated. Therefore a numerical model is constructed. Whenever using a numerical method to solve a differential equation, we should be concerned about the accuracy and convergence properties [7]. For benchmark problems, this is relatively easy, because accurate results are available to compare with. But when CFD code is used to predict flow properties, which are a priori unknown, there is nothing to compare with. Verification is the process by which the degree of numerical accuracy is assessed [3]. Numerical errors may be divided into three kinds:

- discretisation errors
- iterative convergence errors
- rounding errors

During the construction of the numerical model, truncation errors are made. This discretisation error is due to the replacement of the differential equations by an algebraic system. When solving the numerical model small errors, like rounding errors and iteration errors, are unavoidable. Rounding errors are due to the finite precision arithmetic of computers. For CFD applications this is usually negligible compared to the two other sources of numerical error [7]. Iterative convergence errors arise from an inexact solution of the algebraic system by some iteration method.

We can give an indication of the size of the error, by showing the discrepancy between numerical solutions at two refinement levels (Richardson Extrapolation).

13.1. Richardson Extrapolation

In this section, Richardson is used to determine the order of our error. The variable h is used to denote the grid size. We assume the error $E(h)$ has the following shape

$$E(h) = M - N(h) = K_1 h^{\alpha_1} + K_2 h^{\alpha_2} + \ldots \tag{13.1}$$

in which $K_i \in \mathbb{R} \neq 0$ and for $\alpha_i \in \mathbb{N}$ we have $0 \leq \alpha_1 < \alpha_2 \ldots$ For h small enough we assume

$$M - N(h) = Kh^{\alpha} \tag{13.2}$$

where M, K and α are unknown. Here M represents the exact solution, $N(h)$ the approximation of the solution and Kh^{α} the error (α the order of the error). Given a specific grid size, the approximation $N(h)$ can be computed. But then we still have three unknowns: M, k and α. We will now determine $N(h)$, $N(3h)$ and $N(9h)$ in order to obtain three equations with three unknowns:

$$M - N(h) = Kh^{\alpha} \tag{13.3}$$
$$M - N(3h) = K(3h)^{\alpha} \tag{13.4}$$
$$M - N(9h) = K(9h)^{\alpha} \tag{13.5}$$

In this report a grid refinement factor of 3 is used, in order to let the variables of the coarsest grid coincide with the variables of the finer grids. Now combine equations 13.3 - 13.5 to solve for the

Figure 13.1: Two grids with a refinement factor of 3.

unknowns. Notice

$$\begin{aligned}[M - N(9h) - [M - N(3h)]] &= N(3h) - N(9h) \\ &= K(9h)^\alpha - K(3h)^\alpha \\ &= K(9h)^\alpha \left(1 - \left(\frac{1}{3}\right)^\alpha\right) \end{aligned} \tag{13.6}$$

Similarly we can obtain

$$N(h) - N(3h) = K(3h)^\alpha \left(1 - \left(\frac{1}{3}\right)^\alpha\right). \tag{13.7}$$

By using Equations 13.6 and 13.7 in different ways, we can calculate the unknowns M, K and α.

Determining α

By dividing them we can obtain

$$A = \frac{N(3h) - N(9h)}{N(h) - N(3h)} = \frac{K(9h)^\alpha \left(1 - \left(\frac{1}{3}\right)^\alpha\right)}{K(3h)^\alpha \left(1 - \left(\frac{1}{3}\right)^\alpha\right)} = 3^\alpha. \tag{13.8}$$

The order α is now determined by

$$\alpha = \frac{\log(A)}{\log(3)}. \tag{13.9}$$

Determining K

We also know that

$$N(h) - N(3h) = K(3h)^\alpha \left(1 - \left(\frac{1}{3}\right)^\alpha\right). \tag{13.10}$$

Now K can be determined, because $N(h)$, $N(3h)$ and α are known.

$$K = \frac{N(h) - N(3h)}{(3h)^\alpha \left(1 - \left(\frac{1}{3}\right)^\alpha\right)}. \tag{13.11}$$

The error is now approximated by Kh^α.

13.1.1. Expected Order of Accuracy

As mentioned before the numerical error is composed of discretisation errors, iterative convergence errors and rounding errors and thus we can represent the error by

$$E(\Delta t, \Delta x) = O(\Delta x^{\beta_1}) + O(\Delta t^{\beta_2}) + \text{"noise"} \tag{13.12}$$

here β_1 represents the order of our space discretisation and β_2 the order of our time integration, the noise consists of rounding errors.

Time-integration

The forward Euler method is first order accurate, so is the backward (implicit) Euler method. So we expect our semi-implicit Euler method to be first order accurate as well.

Space Discretisation

In [10] the order of accuracy is given for the approximation of surface integrals and volume integrals. For the approximation of volume integrals, we have used the following second order accurate midpoint rule

$$\int_{\Omega} q d\Omega = \overline{q}\Delta\Omega \approx q_C \Delta\Omega \tag{13.13}$$

where q_C stands for the value of q at the center of the control volume and $\Delta\Omega$ stands for the volume of the control volume. For the approximation of surface integrals, we first used that the net flux through the control volume boundary is the sum of integrals over the four (in 2D) faces

$$\int_S f dS = \sum_k \int_{S_k} f dS, \quad k \in n,\ e,\ s,\ w \tag{13.14}$$

in which f for instance represents the convective or diffusive flux in the direction normal to the control volume face. We will now consider the face e. To determine 13.14, we would need to know f everywhere on the surface S_e. This information is not available, so the following approximations are used.

$$F_e = \int_{S_e} f dS = \overline{f_e} S_e \approx f_e S_e \tag{13.15}$$

$$F_e = \int_{S_e} f dS \approx \frac{S_e}{2}(f_{ne} + f_{se}) \tag{13.16}$$

depending on if the variable is located at the centre of the boundary (Equation 13.15) or at the corners of the control volume (Equation 13.16). Both approximations are second-order accurate [10]. Sometimes we needed the values of variables at locations other than computational nodes in which they are defined. Linearly approximating for instance ϕ_e by interpolating between the neighbouring nodes, C and E

$$\phi_e = \frac{1}{2}(\phi_E + \phi_C)$$

is also second-order accurate. For the diffusion term we have also used a central difference scheme, which is also second order accurate. So in conclusion we expect the order of the space-discretisation, β_1, to be equal to two and the order of the time-integration, β_2, to be equal to one.

13.2. Results

We will apply Richardson to the one-dimensional shallow water equations, given by

$$\frac{\partial}{\partial t}p + \frac{\partial}{\partial x}(pu) + gh\frac{\partial \eta}{\partial x} - Dh\left(\frac{\partial^2 u}{\partial x^2}\right) + \frac{g}{C^2}\|\mathbf{u}\|u = 0 \tag{13.17a}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}p = 0 \tag{13.17b}$$

For all cases we have used the finite volume method on a staggered grid with a semi-implicit or explicit time-integration method. The order is also a function of Δt (Equation 13.12), so we need to choose Δt really small or we need to choose Δt such that we have $O(\Delta t) = O(h)$. For the first case an area of length 10 meters and for the second case an area of length 50 meters is used. As explained in Section 13.1 three grids are needed. We have four grids and will thus use both the three most coarse and the three fine grids to determine the order. The vector $N(h)$ is composed as the vector of the two unknowns u and h.

The implementations are done in Matlab and OpenCL. Since we work with a graphics processing unit with float precision, we obtain 7 decimal digits of precision, while Matlab uses by default 16 digits of precision [35]. First, the expected order of accuracy is discussed. Afterwards, the results are shown.

Grid	Case 1 & 2			Case 3		
	# u	# h	$dX(m)$	# u	# h	$dX(m)$
1	16	15	0.66	15	15	3.33
2	46	45	0.22	45	45	1.11
3	136	135	0.07	135	135	0.37
4	406	405	0.02	405	405	0.12

Table 13.1: Number of grid points for the variables u and h and the corresponding grid size.

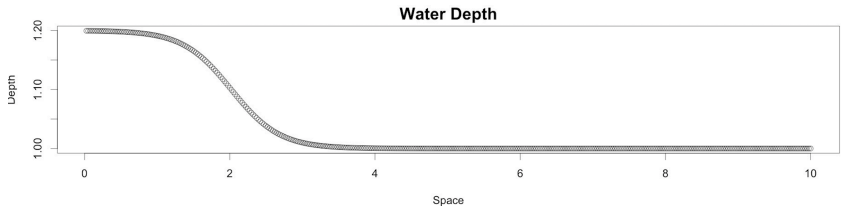


Figure 13.2: Initial water depth: case 1.

13.2.1. Case 1a

Consider a shallow water wave in a well with the horizontal extent of $0 \leq x \leq 10$ meters. The boundary and initial conditions are chosen as in Table 13.2. The initial condition for h is also shown in Figure 13.2. A staggered grid is used in which the velocity is located at the boundaries of our domain. At the edges there is no flow into and out of the well (u is equal to zero at the boundaries). Besides, a time step of $0.00001s$ and an end time of $2s$ are taken. The results are shown in Figure 13.3. It is

Variable	Boundary Conditions		Initial Condition
	Left	Right	
u	$u = 0$	$u = 0$	$u = 0$
h	-	-	$h(x) = 0.2e^{-x} + 1$

Table 13.2: Boundary and initial conditions of dependent variables: case 1a

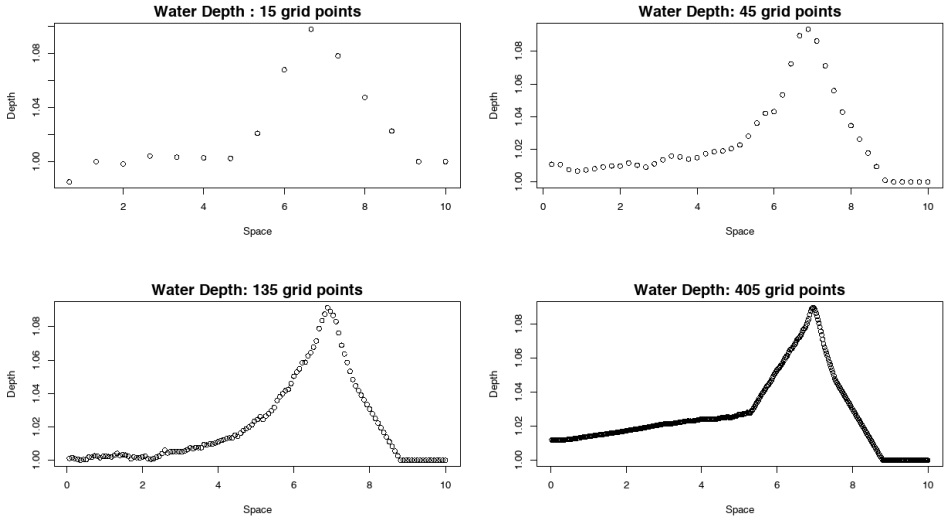
noticed that the water depth on the left side of the wave does not coincide for 135 and 405 grid points. The order α is determined from Equation 13.9. The results for the three coarse and fine grids are shown in Appendix B.1. Most of the determined values had a value of NaN, because the logarithm of a negative number was taken. Since we use single precision numbers, a time step of 0.00001 seconds could be too small. The next case uses a larger time step.

13.2.2. Case 1b

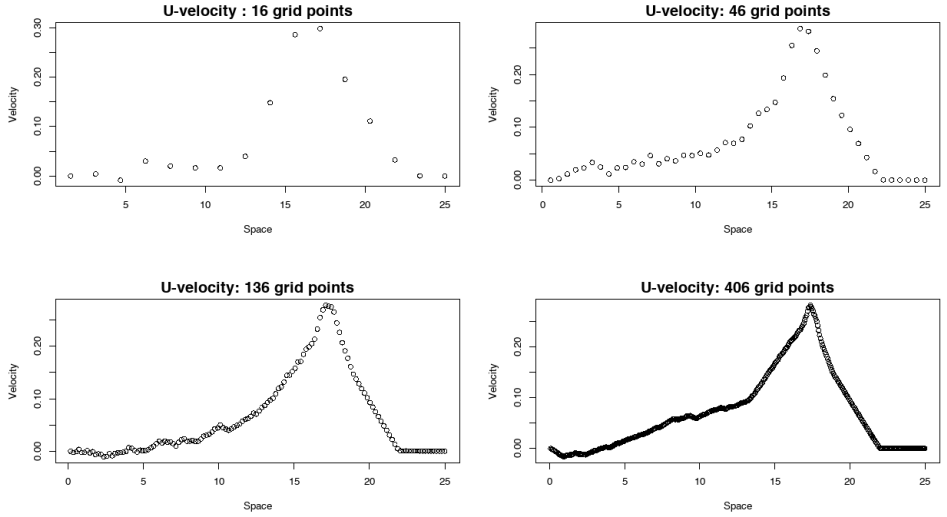
Case 1b is the same as case 1a, but uses a larger time step: $0.001s$. The result of the finest grid is shown in Figure 13.4. In Appendix B.2 the determined orders are found. Most values are around 1.3. The results are more indicative of what we would expect (an order of 2 for the space-discretisation) than for case 1a. So the time step of $0.000001s$ was probably too small for the precision used on the GPU. It could be that the larger time step introduces a larger iterative errors, which impedes us from obtaining an order of 2. Therefore, Matlab is now going to be used in combination with a small time step.

13.2.3. Case 1c

Case 1a is now going to be tested in Matlab. The difference with Case 1a is that an explicit method is used instead of a semi-implicit method. The results are shown in Figure 13.4. It is noticed that the water depth on the left side of the wave now does coincide for 135 and 405 grid points. The order of accuracy is put in the Appendix B.3. Most of the determined values have a value between 1.2 and 2.5. Using a semi-implicit approach resulted in the same indication.



(a) Water Depth.



(b) Velocity.

Figure 13.3: The water depth and velocity at $2s$ for four different grid sizes: case 1a.

13.2.4. Case 2

This case is the same as case 1b, only a different initial condition for h is used, in order to remove the high peek in the water depth and velocity (Figure 13.5). The results are shown in Figure 13.7. Appendix B.4 shows the results are not more indicative of an order 2 than the results of case 1b and 1c.

13.2.5. Case 3

This case simulates tidal action, and thus uses a wetting-drying method (which was not necessary for the first cases). The boundary and initial conditions as shown in Table 13.3 are used. The variable h is located on the left boundary and u on the right boundary. Besides an end time of $1000s$ in combination with a time step of 0.001 seconds is used. The tidal action was imposed using a mean sea level and amplitude of $0.045m$ and a period of 4000 seconds. The resulting water depth, h, and velocity, u, are shown in Figure 13.7. We notice a large jump in velocity at the front of the tide. The order α is determined from Equation 13.9. The results for the three coarse and fine grids are shown in Appendix B.5. We obtain a lot of varied values and thus no order of accuracy can be derived for the space-discretisation.

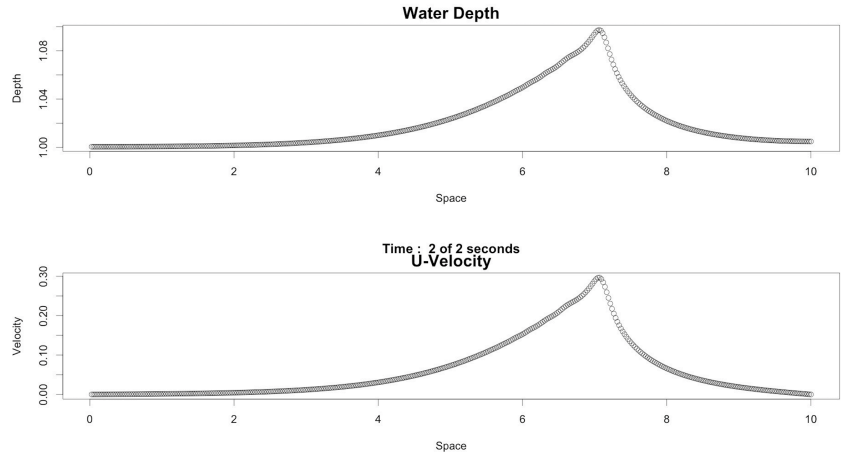


Figure 13.4: The water depth and velocity at $2s$: case 1b.

Variable	Boundary Conditions		Initial Condition
	Left	Right	
u	$\frac{\partial u}{\partial x} = 0$	$u = 0$	$u = 0$
h	$h = h_{tide}$	-	$h = H_{crit}$

Table 13.3: Boundary and initial conditions of dependent variables: case 3

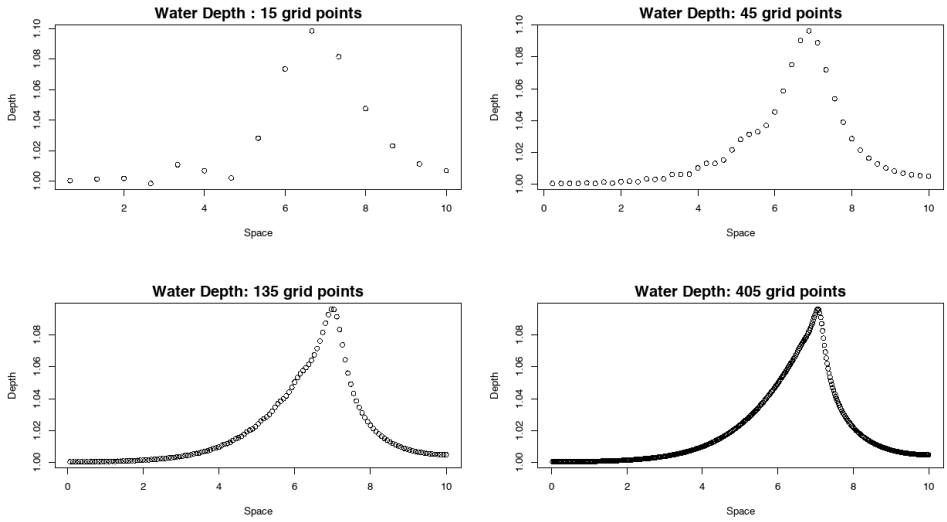
13.2.6. Conclusion

Three cases were experimented with in order to determine the accuracy of the spatial discretisation of the one-dimensional shallow-water equations. The first and second case simulated a wave in a well. The third case simulated tidal action in combination with a wetting-drying approach. The cases were implemented in OpenCL (single precision, semi-implicit time-integration) or Matlab (double precision, explicit time-integration).

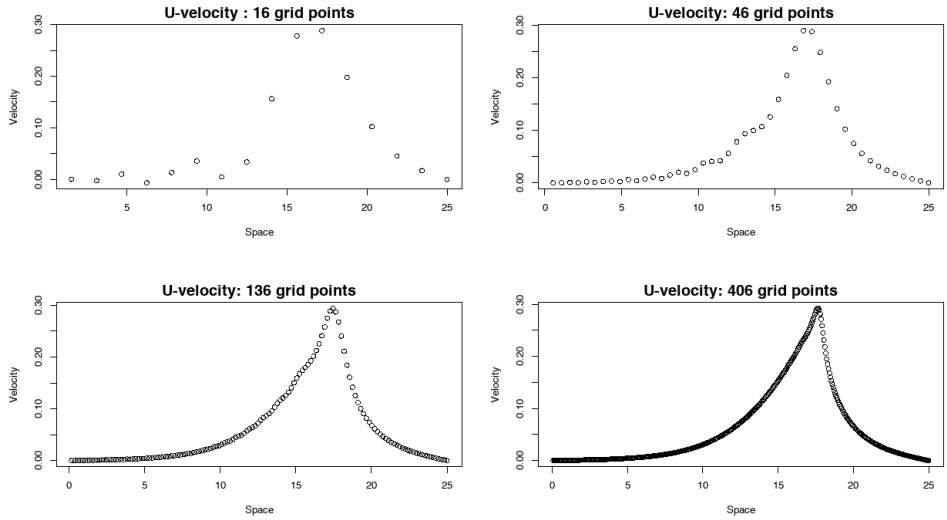
The OpenCl implementation of case 1a did not give us conclusive results. A time step of of $0.000001s$ was used. Since this small time step might induce larger rounding errors for single precision calculations, we also used a time step of 0.001 seconds. These results indicate an order of 1 or 2 for the space-discretisation. It was thought that because of this larger time step, the iteration errors might impede us from obtaining a order of 2. In Matlab double precision numbers are used. So case 1c performed the simulation in Matlab, with again a time step of $0.000001s$. But also the Matlab implementation indicated an order of 1 or 2.

Case 2 used another initial condition than case 1, but obtained no different results. Case 3 simulated tidal action (used a wetting-drying approach). The code is implemented in OpenCL and a time step of $0.001s$ was used. We did not expect to obtain better results for this case.

In conclusion, we expected an order of 2 for the space-discretisation. The three cases were not conclusive (indicated an order of 1 or 2). A pointwise approach has been used, by only using the grid points of the finer grids, which coincide with the grid point of the coarsest grid. An idea could be to also involve the neighbours of these grid points, in order to get more conclusive results.



(a) Water Depth.



(b) Velocity.

Figure 13.5: The water depth and velocity at $2s$ for four different grid sizes: case 1c.

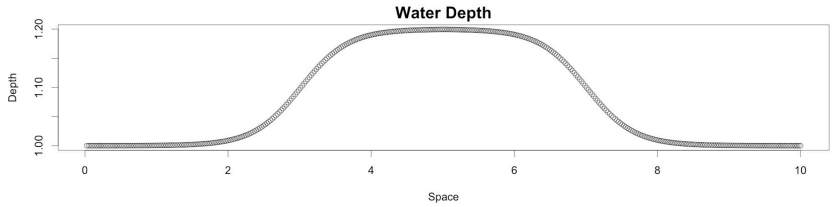
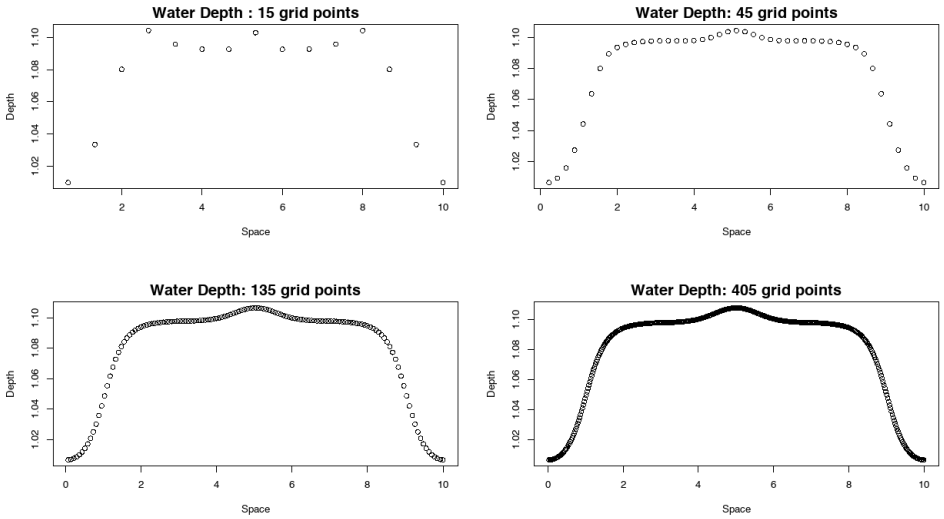
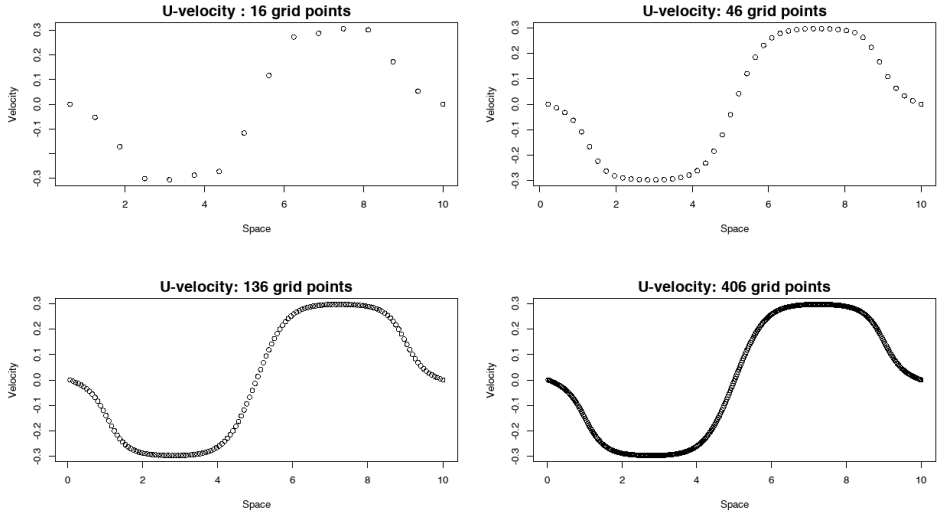


Figure 13.6: Initial water depth: case 2.

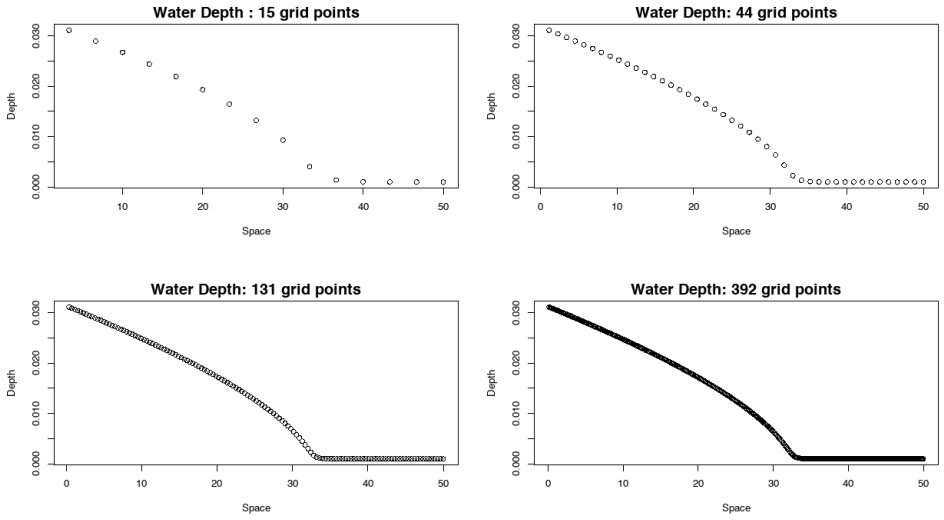


(a) Water Depth.

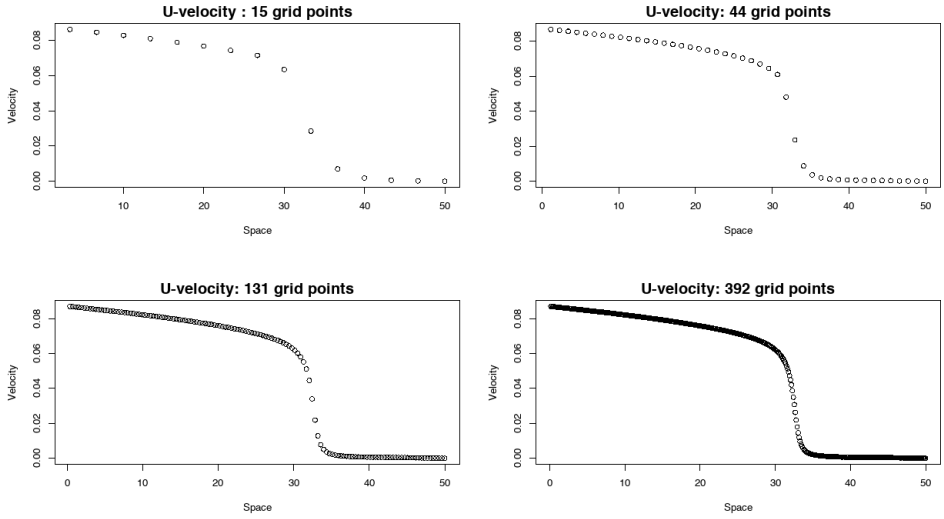


(b) Velocity.

Figure 13.7: The water depth and velocity at $2s$ for four different grid sizes: case 2.



(a) Water Depth.



(b) Velocity.

Figure 13.8: The water depth and velocity at $800s$ for four different grid sizes: case 3.

14

Test Case

The test case used in this chapter is an adjusted version of the test case mentioned in [5]. The objective is to compare the simulation times of Delft3D-FLOW and the model used in Section 12.2. The geomorphic and vegetative developments after 30 years are not compared to the outcome of [5] and the outcome of Delft3D-FLOW, because a simulation of 30 years requires tremendous computational capacities. First, the test case of [5] is discussed. Afterwards, general information about Delft3D-FLOW is given. In the end, the simulation times are compared.

14.1. Test Case

The test case proposed in [5] solves the three-dimensional shallow water equations in combination with a three-dimensional transport equation with 8 layers. The vegetation model as explained in Subsection 3.3.1 is used, consisting of the terms given in Equations 3.12-3.16. The Partheniades-Krone formulation, given in Equations 12.7 and 12.8, is used to determine the exchange of sediment between the bottom and the water (Equation 12.5).

The model was applied to a rectangular grid, representing a horizontal domain of $80m \times 600m$ with a $2m \times 2m$ horizontal resolution and 8 vertical layers. First, the hydrodynamic model is solved for one average tidal cycle with a time step of 3 seconds. Second, topographic changes were computed with the morphodynamic model and multiplied by the total number of tidal inundations per year to obtain the new topography after a coarse time step of one year. Third, the plant growth model was run to compute the new spatial stem density distribution after the same year. The calculated topography and stem density distribution were used then as input for the next hydrodynamic computation. Feedback loops were run until a stable plant density distribution (average change $< 0.1 stems m^{-2} yr^{-1}$) and stable topography (average elevation change $< 0.1 mm yr^{-1}$) were obtained

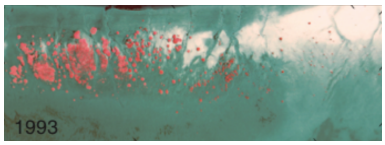
14.1.1. Results

Aerial photographs from the Westerschelde estuary (southwest Netherlands) were used to document the patterns of Spartina Anglica colonization and channel formation on a tidal flat (Figure 14.1). The resulting bottom elevation (for two values of the carrying capacity K) after 30 years is shown in Figure 14.2.

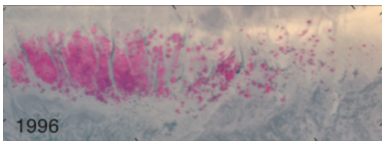
14.1.2. Adjusted Test Case

The test area of Temmerman ([5]) consists of two open boundaries. For this test area we want to ensure a positive water depth at the open boundary and so we will extend the computational domain to the west. Besides we also want to extend the domain to the east (like in [36]) to avoid the influence of reflection against the northern boundary, by enlarging our domain 320 meters to the west. It is assumed the water can not flow through the inland boundary, because of a present dike. By setting u equal to zero, we indirectly prescribe this. Thus instead of a bare flat of $600m$ by $80m$ located $1.6m$

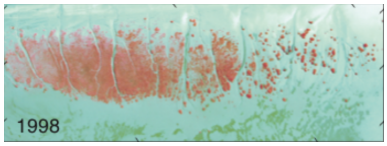
(a) Year 1989.



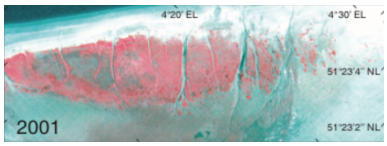
(b) Year 1993.



(c) Year 1996.



(d) Year 1998.



(e) Year 2001.

Figure 14.1: Aerial photographs documenting patterns of plant colonization by Spartina anglica (red color) and channel formation on tidal flat (Plaat van Valkenisse, Scheldt estuary, southwest Netherlands). Tidal flow alternates from top to bottom of photos, and vice versa. All photos are taken at low tide, when area is dry [5].

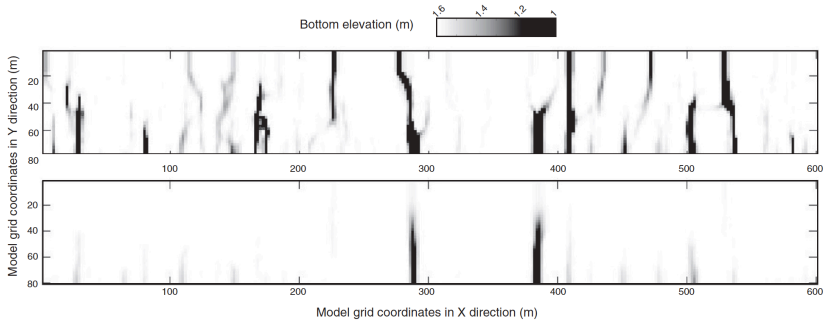
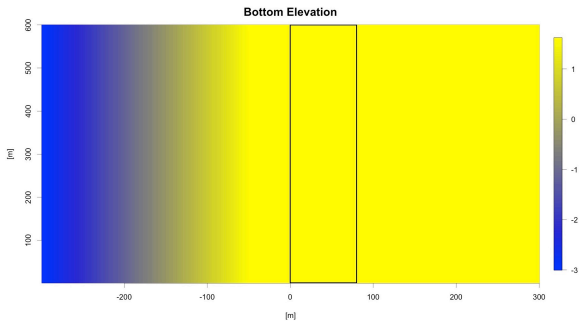
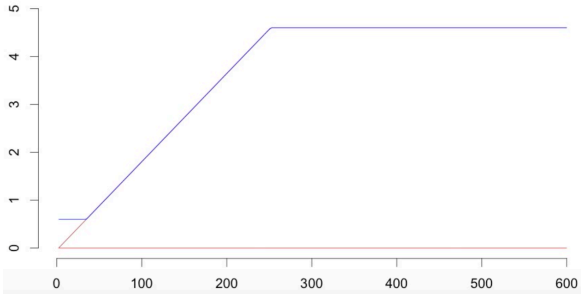


Figure 14.2: Maps of simulated final bottom elevation after 30 years for two scenarios with different maximum vegetation density : upper panel: $K = 1200$ stems m^{-2}; lower panel: $K = 60$ stems m^{-2} [5].

above the mean sea level, our computational domain is now $600m \times 600m$. The an initial bathymetry is given in Figure 14.3a.



(a) Initial bathymetry of the idealized marsh test case; the black box corresponds to the area of interest.



(b) Cross-section initial bathymetry and water depth.

Figure 14.3: Test area.

Even though the computational domain is extended, the area of interest for this test case remains the original computational domain, which is highlighted by the black box in Figure 14.3a. We now use this test case to compare the performance of our model to the performance of the powerful modelling software Delft3D-FLOW.

14.2. Our Model

The depth-averaged shallow water equations with one layer are used. We decided to compare the simulation times of Delft3D-FLOW and our model for a flow simulation only. So no vegetation and sediment transport is included. The boundary conditions for u, v and h as shown in Table 10.2 are used. The initial values for h and z_b are shown in Figure 14.3b.

14.2.1. Results

For the results in this subsection, different problem sizes are used: $1{,}048{,}576$, $65{,}536$, $16{,}384$ and $9{,}216$ grid points which corresponds respectively to 1024, 256, 128 and 96 grid points in both the x- and the y- direction and a grid size of approximately 0.6, 2.3, 4.7 and $6.3m$. The maximum water depth is $5.4m$ at the left boundary. And thus respectively the CFL condition for explicit time-integration is given by 0.0015, 0.055, 0.11 and $0.15s$ (derived from Equation 11.2). All cases use a work-group size of 256 (which may not be optimal). The results of the AMD Radeon HD - FirePro D700 GPU can be found in Table 14.1.

Case	# Grid Points	Time Step	Simulation Time
1	1,048,576	$0.01s$	$1.7h$
2	65,536	$0.05s$	$152s$
3	16,384	$0.1s$	$32s$
4	9,216	$0.1s$	$32s$

Table 14.1: Simulation time 1 day on the AMD Radeon HD - FirePro D700 GPU.

The cases 3 and 4 have approximately the same wall-clock time. As mentioned in 8.1, the increase in wall-clock time for an increase in problem size is small for small problem sizes. Since the GPU is more optimal for larger problem sizes, the increase in simulation time between two cases is smaller than the difference in the number of grid points times the decrease in time step. For instance, case 2 uses 4 times more grid points and a twice as small time step than case 3, but the increase in simulation time is only a factor 4.75. The results of the HD Graphics 4000 GPU can be found in Table 14.2.

Case	# Grid Points	Time Step	Simulation Time
3	16,384	$0.1s$	$1.2h$
4	9,216	$0.1s$	$1.1h$

Table 14.2: Simulation time 1 day on the HD Graphics 4000 GPU.

The HD Graphics 4000 GPU simulated two cases, in order to compare the two GPUs. The difference in simulation time is roughly a factor 130.

14.3. Delft 3D-FLOW

Delft3D-FLOW is a multi-dimensional (2D or 3D) hydrodynamic (and transport) simulation program, it is the hydrodynamic module of Delft3D. The Delft3D suite is composed of several modules, grouped around a mutual interface, while being capable to interact with one another. It can be used for the modelling of water flows, sediment transports, waves, water quality, morphological developments and ecology [13].

An alternating direction implicit time-integration method is used in Delft3D-FLOW. General information about ADI methods is given in Subsection 4.3.4. Generally, the time step can be chosen based on accuracy arguments only. In most cases stability is not an issue [13] due to the implicit character of the ADI, which allows for a large time step. Generally, the Courant number should not exceed a value of ten, but for problems with rather small variations in both space and time, the Courant number can be taken substantially larger.

14.3.1. Results

For the results in this subsection, the adjusted test case is run with Delft3D-FLOW. No sediment transport or vegetation was included. The Delft3D-FLOW module was run on another machine as was used for the results of Section 14.2.1. A machine consisting of a CPU with a clock speed of approximately $3GHz$ was used for the results in this section. Thus, the results are not totally comparable, but they can give us an indication of the difference in run-time. In Table 14.3 the simulation times for one day are shown for two different cases. These cases differ in the used grid size and time step. Problem size of $1{,}000{,}000$ and $10{,}000$ grid poitns are used. The first corresponds to a grid of $1{,}000 \times 1{,}000$ grid points (a grid size of $0.6m$). In combination with a time step of 0.3 seconds, it took Delft3D-FLOW 16 hours ($57{,}600$ seconds) to simulate one day. The second case has a problem size of $10{,}000$ and a grid of 100×100. A ten times larger time step could be used (30 seconds). It then took 40 seconds to simulate one day. The simulation did not include sediment transport. But simulating the transport of one sediment type would increase the computation time with approximately one third, which would result in a simulation time of approximately $21h$ for case 1 and $53s$ for case 2. Table 14.3 shows an

Case	# Grid Points	Time Step	Simulation Time
1	1,000,000	$3s$	$16h$
2	10,000	$30s$	$40s$

Table 14.3: Simulation time 1 day Delft3D-FLOW.

increase of a factor of $1{,}440$ in simulation time between case 1 and 2. Which is roughly what we would expect, because decreasing the grid size by a factor 10 means that we get 100 times the number of grid points. Moreover, the time step is divided by ten as well, so the amount of work is $1{,}000$ times that of the basic run.

14.4. Conclusion

This chapter gave the simulation times of our model and Delft3D-FLOW for a specific test case. Our model is a factor of 9.4 faster for a problem size of around $1{,}000{,}000$ and a factor of 1.25 faster for a problem size of $10{,}000$ on the AMD Radeon HD - FirePro D700 GPU. As expected the GPU performs better for larger problem sizes. Table 14.3 shows an increase of a factor of $1{,}440$ in simulation time between these two problem sizes for Delft3D-FLOW. For the AMD Radeon HD - FirePro D700 GPU the simulation time differs approximately a factor of 190 between case 1 and 4 14.1. However, the HD Graphics 4000 GPU is almost a factor of 130 slower.

We note the huge difference in the used time step between Delft3D-FLOW and our model. Delft3D-FLOW used a time-step of 3 seconds, where we needed to use a time step of around $0.01s$. In Figure 8.3 it was shown that the relation between the number of iterations and the simulation time is almost linear. So being able to use a 300 times larger time step would result in almost a 300 times smaller simulation time.

15

Summary and Conclusion

This chapter concludes this thesis. First, each subject of the thesis is summarized and concluded. Thereafter, the results of the thesis are discussed and suggestions are made for further research.

15.1. Summary

The aim of this thesis was to investigate the implementation of a salt marsh model on a GPU. The corresponding research questions were:

How can we improve the current salt marsh model

- in terms of the grid, discretisation and solver?
- in terms of the included processes?

First, experiments were done to compare the CPU and CPU wall-clock times. A finite volume method on a staggered grid was implemented instead of a finite difference approach on a collocated grid. No adjustments were made to the semi-implicit time-integration method. In order to get a more realistic model, a tidal cycle was added. Afterwards, a transport equation was added to model the transport of substances. A really small time step needed to be used, so several experiments were done in order to observe the time step. Afterwards the Richardson method was used to determine the order of accuracy and in the end, a test problem was used to compare the results and computation times of Delft3D-FLOW and our model.

15.2. Conclusion

It was shown that when the calculations (of the finite difference method on a collocated grid) are performed on a GPU, they take up to 14 times less time in comparison with the same system performed on a CPU.

The finite volume method on a staggered grid showed advantages relative to the finite difference method on a collocated grid. A larger grid size could be used due to the disappearance of water oscillations. Since the semi-implicit Euler method is easily parallelizable, we did not look into other time-integration methods.

Tidal action was added to our model, by imposing the water depth to be a sinusoidal function at our inflow boundary. The results were different than for the model in Chapter 6. The drainage network that emerged consisted of wider tidal creeks and less bifurcations, because the tidal model does not reach a steady-state. Thereafter a transport equation was added to our model to model the transport of sediment. It was coupled by erosion and sedimentation to the bottom elevation changes. At first, the old deposition formula was changed, by replacing the sediment input by the sediment concentration times a constant. However, by using this new deposition formula, we did not obtain the desired results, because no tidal creeks arose. So instead, we changed the erosion and sedimentation formulations

to the Partheniades-Krone formulation, which is widely used. The transport equation worked well in combination with the Partheniades-Krone formulation. Although, we did not find a parameter set yet, to obtain small hills next to our tidal creeks on the short time frame that we used.

For the results in Chapters 9-12 the model simulated accelerated salt marsh development (simulation time of 3000 seconds). In a couple of minutes, a grown salt marsh was established. Simulating real salt marsh evolution ($15 - 30$ years) is computationally heavy. Because as shown in Chapter 11 small time steps need to be used. The time step for explicit time-integration was found to be quite a bit smaller than the time step obtained by the CFL condition. These results need further investigation.

The Richardson method was used to find the order of accuracy of our spatial discretisation. An order of two was expected, but the results do not confirm this.

15.3. Future Work

As mentioned in Section 15.2 the semi-implicit time-integration method works only for a small time-step. This time step becomes a limiting factor when simulating real salt marsh development. In Chapter 14, case 4 simulated a day in 32 seconds, which means a 30 years simulation would take 17.33 hours. Delft3D-FLOW could use a 300 times larger time step. If we could use that time step, then a simulation of 30 years would cost us 3.5 minutes, supposing the implementation would not get more computational heavy. However, the implementation probably gets heavier. Because an advantage of the semi-implicit time-integration method we used, is that the corresponding code is easily parallelizable. It would be interesting to investigate the computing times of, for instance, an implicit method in combination with our model. Then the time step is not limited by the CFL number, but a non-linear system of equations has to be solved. Whether this would result in a speed-up depends on a lot of factors.

At the moment the same time and space discretisations are used for all modules. We have accelerated the morphological and vegetation equations (multi-time-scale method) in Chapters 9-12. This is a simple version of the morphological acceleration factor. Due to the long computation time, we were not able to show the results of the test-case mentioned in Chapter 14. When simulating ”real” salt marsh development a more sophisticated approach of the morphological acceleration factor could be used. For instance we could investigate our results, when extrapolating the bottom elevation changes of one tidal cycle, to a whole year. More information can be found in Section 3.2.

In Sections 10.1, 10.2 and 12.2, several side effects of the thin-film algorithm were mentioned. And more importantly, the method is not mass conservative since additional water is added if the water depth falls below a certain value. In Subsection 11.2.2 another wetting-drying algorithm was implemented for a one-dimensional example. No additional mass needed to be added for this method. However, additional iterations are performed, if the water depth in at least one grid point falls below H_{crit}. Besides, this method is more difficult to parallelize. For implementation on a GPU we would recommend for instance the negative depth algorithms, since they are applicable to the whole domain, easy parallelizable and no additional mass is added.

Furthermore, a lot of improvement can still be made on the GPU implementation. For instance improvements in efficiency can be obtained using shared memory algorithms, where thread blocks that are copied from global memory into faster shared memory are reused efficiently, before global memory is accessed again [29]. Moreover, thread divergence is tolerated by our model, but it is very damaging to the performance and a fully diverged warp can run the warp size slower than a divergence-free warp [37].

Richardson extrapolation did not give us a conclusive result about the order of accuracy. A pointwise approach was used. If another approach also does not give any conclusive results, the code implementation could be checked. This can be addressed through extensive code verification, for instance, by using the method of manufactured solutions.

A

Derivations

The discretisations for two-dimensional equations are derived in this appendix. First the finite difference method is used on a collocated grid. After which the finite volume discretisation is given on a staggered grid. Both have use the semi-implicit Euler method in time.

A.1. Finite Difference Method on a Collocated Grid

The finite difference discretisation on a collocated grid is derived for several equations. We use the notation as given in Figure A.1. The variable C is used to denote the central grid point and the variables E, W, N and S represents it's east west, north and south neighbours.

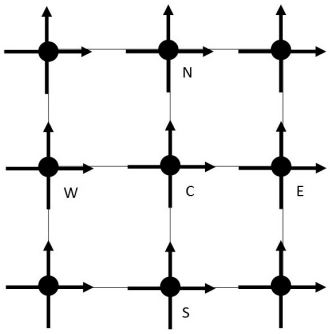


Figure A.1: Collocated grid.

Continuity Equation

The continuity equation is given by

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = 0.$$

Using central differences for the derivatives to x and y gives

$$\frac{dh_C}{dt} + \frac{h_E u_E - h_W u_W}{2\Delta x} + \frac{h_N v_N - h_S v_S}{2\Delta y} = 0.$$

Applying the semi-implicit Euler method gives

$$h_C^{m+1} = h_C^m + \Delta t \left(-\frac{h_E^{m^*} u_E^m - h_W^{m^*} u_W^{m^*}}{2\Delta x} - \frac{h_N^{m^*} v_N^{m^*} - h_S^{m^*} v_S^{m^*}}{2\Delta y} \right) \tag{A.1}$$

in which m^* take a value of m or $m+1$.

Momentum Equations

The momentum equations in the $x-$ and in the y-direction are given by

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + g\frac{\partial \eta}{\partial x} - D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + Su = 0$$
$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + g\frac{\partial \eta}{\partial y} - D\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + Sv = 0$$

Using again central differences on both equations results in

$$\frac{du_C}{dt} + u_C\frac{u_E - u_W}{2\Delta x} + v_C\frac{u_N - u_S}{2\Delta y} + g\frac{\eta_E - \eta_W}{2\Delta x} - D\left(\frac{u_W - 2u_C + u_E}{\Delta x^2} + \frac{u_S - 2u_C + u_N}{\Delta y^2}\right) + S_C u_C = 0$$

$$\frac{dv_C}{dt} + u_C\frac{v_E - v_W}{2\Delta x} + v_C\frac{v_N - v_S}{2\Delta y} + g\frac{\eta_N - \eta_S}{2\Delta y} - D\left(\frac{v_W - 2v_C + v_E}{\Delta x^2} + \frac{v_S - 2v_C + v_N}{\Delta y^2}\right) + S_C v_C = 0$$

The sink term S usually depends on the velocities, u and v and the water depth h. In this term, S_C, we need to use u_C, v_C and h_C. Applying the semi-implicit Euler method gives

$$u_C^{m+1} = u_C^m + \Delta t[-u_C^{m^*}\frac{u_E^{m^*} - u_W^{m^*}}{2\Delta x} - v_C^{m^*}\frac{u_N^{m^*} - u_S^{m^*}}{2\Delta y} - g\frac{\eta_E^{m^*} - \eta_W^{m^*}}{2\Delta x} + D\left(\frac{u_W^{m^*} - 2u_C^{m^*} + u_E^{m^*}}{\Delta x^2} + \frac{u_S^{m^*} - 2u_C^{m^*} + u_N^{m^*}}{\Delta y^2}\right) - S_C^{m^*}u_C^{m^*}] \tag{A.2a}$$

$$v_C^{m+1} = v_C^m + \Delta t[-u_C^{m^*}\frac{v_E^{m^*} - v_W^{m^*}}{2\Delta x} - v_C^{m^*}\frac{v_N^{m^*} - v_S^{m^*}}{2\Delta y} - g\frac{\eta_N^{m^*} - \eta_S^{m^*}}{2\Delta y} + D\left(\frac{v_W^{m^*} - 2v_C^{m^*} + v_E^{m^*}}{\Delta x^2} + \frac{v_S^{m^*} - 2v_C^{m^*} + v_N^{m^*}}{\Delta y^2}\right) - S_C^{m^*}v_C^{m^*}] \tag{A.2b}$$

in which again $m^* \in \{m, m+1\}$.

Vegetation

The equation to model the vegetation density is given by

$$\frac{\partial n_b}{\partial t} = D\left(\frac{\partial^2 n_b}{\partial x^2} + \frac{\partial^2 n_b}{\partial y^2}\right) + r\left(1 - \frac{n_b}{K}\right)n_b\left(\frac{K_p}{K_p + h}\right) - E_p n_b\sqrt{u^2 + v^2}.$$

Using central differences for the diffusion term gives

$$\frac{dn_{b_C}}{dt} = D\left(\frac{n_{b_W} - 2n_{b_C} + n_{b_E}}{\Delta x^2} + \frac{n_{b_S} - 2n_{b_C} + n_{b_N}}{\Delta y^2}\right) + r\left(1 - \frac{n_{b_C}}{K}\right)n_b\left(\frac{K_p}{K_p + h_C}\right) - E_p n_{b_C}\sqrt{u_C^2 + v_C^2}$$

Applying the semi-implicit Euler method gives

$$n_{b_C}^{m+1} = n_{b_C}^m + \Delta t[D\left(\frac{n_{b_W}^{m^*} - 2n_{b_C}^{m^*} + n_{b_E}^{m^*}}{\Delta x^2} + \frac{n_{b_S}^{m^*} - 2n_{b_C}^{m^*} + n_{b_N}^{m^*}}{\Delta y^2}\right) + r\left(1 - \frac{n_{b_C}^{m^*}}{K}\right)n_{b_C}^{m^*}\left(\frac{K_p}{K_p + h_C^{m^*}}\right) - E_p n_{b_C}^{m^*}\sqrt{(u_C^{m^*})^2 + (v_C^{m^*})^2}]. \tag{A.3}$$

in which again $m^* \in \{m, m+1\}$.

Bottom Elevation

The equation to model the behaviour of the bottom elevation is given by

$$\frac{\partial z_b}{\partial t} = S_{in} h_{eff} - E_0 \left(1 - p_E \frac{n_b}{K}\right)(u^2 + v^2) z_b + D\left(\frac{\partial^2 z_b}{\partial x^2} + \frac{\partial^2 z_b}{\partial y^2}\right).$$

Again using central differences results in

$$\frac{dz_{b_C}}{dt} = S_{in} h_{eff_C} - E_0 \left(1 - p_E \frac{n_{b_C}}{K}\right)(u_C^2 + v_C^2) z_{b_C} + D\left(\frac{z_{b_W} - 2z_{b_C} + z_{b_E}}{\Delta x^2} + \frac{z_{b_S} - 2z_{b_C} + z_{b_N}}{\Delta y^2}\right).$$

Applying the semi-implicit Euler method gives

$$z_{b_C}^{m+1} = z_{b_C}^m + \Delta t[S_{in} h_{eff_C}^{m^*} - E_0 \left(1 - p_E \frac{n_{b_C}^{m^*}}{K}\right)((u_C^{m^*})^2 + (v_C^{m^*})^2) z_{b_C}^{m^*} + D\left(\frac{z_{b_W}^{m^*} - 2z_{b_C}^{m^*} + z_{b_E}^{m^*}}{\Delta x^2} + \frac{z_{b_S}^{m^*} - 2z_{b_C}^{m^*} + z_{b_N}^{m^*}}{\Delta y^2}.\right)] \tag{A.4}$$

in which $m^* \in \{m, m+1\}$.

Boundary Conditions

For the finite difference method a cell-centered discretisation is used, the grid points are located in the center of the control volumes. For both we will show how to deal with Dirichlet and Neumann conditions for a one-dimensional grid, illustrated in Figure A.2. Neumann and Dirichlet boundary

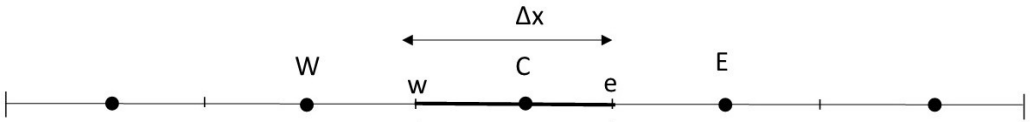


Figure A.2: Collocated grid.

conditions are used for the water depth h and velocity u. We will explain how to use the boundary conditions for the velocity u.
We will look a the right boundary, then u_e is located at the boundary and u_E is the ghost point.

Neumann, assume a Neumann boundary condition of $\frac{\partial u}{\partial x} = C_1$ is presrcibed, in which C_1 represents a constant. Then the ghost point u_E is introduced and it can be determined by using the Neumann boundary condition as follow

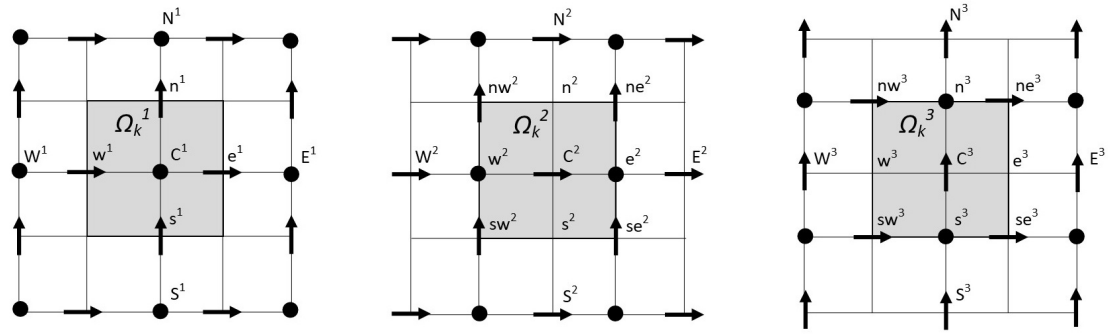
$$C_1 = \frac{du_e}{dx} \approx \frac{u_E - u_C}{\Delta x} \quad \rightarrow \quad u_E \approx u_C + \Delta x C_1 \tag{A.5}$$

Dirichlet, assume a Dirichlet condition of $u = C_1$ is prescribed. Then again the ghost point u_E is introduced and it can be determined by using the Dirichlet condition as follow

$$C_1 = u_e = \frac{u_E + u_C}{2} \quad \rightarrow \quad u_E \approx 2C_1 - u_C \tag{A.6}$$

A.2. Finite Volume on a Staggered Grid Approach 1

The finite volume discretisation on a staggered grid is derived for several equations. The three dependent variables h, u and v of the shallow water equations have their own control volume. The notation as shown in Figure A.3 is used. The bottom elevation z_b and vegetation density n_b are located at the same location as the water depth.



(a) Control volume for the water depth h.

(b) Control volume for the velocity u in the x-direction.

(c) Control volume for the velocity v in the y-direction.

Figure A.3: Different control volumes Ω_k^1, Ω_k^2 and Ω_k^3.

Continuity

The continuity equation can be written as

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\mathbf{u}) = 0.$$

Applying the finite volume method to an volume element Ω_k^1 gives

$$\int_{\Omega_k^1} \frac{\partial h}{\partial t} d\Omega + \int_{\Omega_k^1} \nabla \cdot (h\mathbf{u}) d\Omega = 0.$$

Applying the midpoint rule and the divergence theorem results in

$$\frac{dh_{C^1}}{dt}\Delta x\Delta y + \int_{e^1} hudy - \int_{w^1} hudy + \int_{n^1} hvdx - \int_{s^1} hvdx = 0.$$

Using linear interpolation to determine h in n^1, e^1, s^1 and w^1 results in

$$\begin{aligned}&\frac{dh_{C^1}}{dt}\Delta x\Delta y + u_{e^1}\frac{h_{E^1} + h_{C^1}}{2}\Delta y - u_{w^1}\frac{h_{C^1} + h_{W^1}}{2}\Delta y \\ &+ v_{n^1}\frac{h_{N^1} + h_{C^1}}{2}\Delta x - v_{s^1}\frac{h_{C^1} + h_{S^1}}{2}\Delta x = 0\end{aligned} \tag{A.7}$$

Momentum Equations

The momentum equation in the $x-$ direction can be written as

$$\frac{\partial}{\partial t}(hu) + \nabla \cdot (hu\mathbf{u}) + gh\frac{\partial \eta}{\partial x} - Dh(\nabla \cdot \nabla u) + Shu = 0.$$

Integrating over control volume Ω_k^2 gives

$$\begin{aligned}&\frac{d(h_{C^2}u_{C^2})}{dt}\Delta x\Delta y + \int_{e^2} hu^2 dy - \int_{w^2} hu^2 dy + \int_{n^2} huvdx - \int_{s^2} huvdx + gh_{C^2}\left(\int_{e^2} \eta dy - \int_{w^2} \eta dy\right) \\ &- Dh_{C^2}\left(\int_{e^2} \frac{\partial u}{\partial x} dy - \int_{w^2} \frac{\partial u}{\partial x} dy + \int_{n^2} \frac{\partial u}{\partial y} dx - \int_{s^2} \frac{\partial u}{\partial y} dx\right) + S_{C^2}h_{C^2}u_{C^2}\Delta x\Delta y = 0\end{aligned}$$

This is equal to

$$\begin{aligned}&\frac{d(h_{C^2}u_{C^2})}{dt}\Delta x\Delta y + h_{e^2}u_{e^2}^2\Delta y - h_{w^2}u_{w^2}^2\Delta y + (huv)_{n^2}\Delta x - (huv)_{s^2}\Delta x \\ &+ gh_{C^2}\left(\eta_{e^2} - \eta_{w^2}\right)\Delta y - Dh_{C^2}\left(\frac{u_{E^2} - u_{C^2}}{\Delta x}\Delta y - \frac{u_{C^2} - u_{W^2}}{\Delta x}\Delta y + \frac{u_{N^2} - u_C}{\Delta y}\Delta x - \frac{u_{C^2} - u_{S^2}}{\Delta y}\Delta x\right) \\ &+ S_{C^2}h_{C^2}u_{C^2}\Delta x\Delta y = 0\end{aligned}$$

Rewriting and using the semi-implicit Euler method gives

$$h_{C^2}^{m+1}u_{C^2}^{m+1} = h_{C^2}^{m}u_{C^2}^{m} - \frac{\Delta t}{\Delta x\Delta y}[h_{e^2}^{m^*}(u^{m^*})_{e^2}^2\Delta y - h_{w^2}(u^{m^*})_{w^2}^2\Delta y + (huv)_n^{m^*}\Delta x - (huv)_s^{m^*}\Delta x$$
$$+ gh_{C^2}^{m^*}\left(\eta_e^{m^*} - \eta_w^{m^*}\right)\Delta y - Dh_{C^2}^{m^*}\left(\frac{u_{E^2}^{m^*} - 2u_{C^2}^{m^*} + u_{W^2}^{m^*}}{\Delta x}\Delta y + \frac{u_{N^2}^{m^*} - 2u_{C^2}^{m^*} + u_{S^2}^{m^*}}{\Delta y}\Delta x\right)$$
$$+ S_{C^2}^{m^*}h_{C^2}^{m^*}u_{C^2}^{m^*}\Delta x\Delta y] \tag{A.8}$$

In which $m^* \in \{m, m+1\}$. We need averaging operations for u in e^2 and w^2 and for h, u and v in n^2 and s^2

$$v_{n^2} = \frac{v_{nw^2} + v_{ne^2}}{2}, \quad v_{s^2} = \frac{v_{sw^2} + v_{se^2}}{2}$$
$$(u^2)_{e^2} = \frac{(u^2)_{C^2} + (u^2)_{E^2}}{2}, \quad (u^2)_{w^2} = \frac{(u^2)_{C^2} + (u^2)_{W^2}}{2}, \quad u_{n^2} = \frac{u_{C^2} + u_{N^2}}{2}, \quad u_{s^2} = \frac{u_{C^2} + u_{S^2}}{2}$$
$$h_{n^2} = \frac{h_{Nw^2} + h_{Ne^2} + h_{w^2} + h_{e^2}}{4}, \quad h_{s^2} = \frac{h_{Sw^2} + h_{Se^2} + h_{w^2} + h_{e^2}}{4}, \quad h_{C^2} = \frac{h_{w^2} + h_{c^2}}{2}$$

For the momentum equation in the y-direction

$$\frac{\partial}{\partial t}(hv) + \nabla\cdot(hv\mathbf{u}) + gh\frac{\partial\eta}{\partial y} - Ah(\nabla\cdot\nabla v) + Shv = 0$$

we get in the same way

$$h_{C^3}^{m+1}v_{C^3}^{m+1} = h_{C^3}^{m}v_{C^3}^{m} - \frac{\Delta t}{\Delta x\Delta y}[(huv)_{e^3}^{m^*}\Delta y - (huv)_{w^3}^{m^*}\Delta y + (h_{n^3}^{m^*}(v^{m^*})_{n^3}^2)\Delta x - (h_{s^3}^{m^*}(v^{m^*})_{s^3}^2)\Delta x$$
$$+ gh_{C^3}^{m^*}(\eta_{n^3}^{m^*} - \eta_{s^3}^{m^*})\Delta x - Dh_{C^3}^{m^*}\left(\frac{v_{E^3}^{m^*} - 2v_{C^3}^{m^*} + v_{W^3}^{m^*}}{\Delta x}\Delta y + \frac{v_{N^3}^{m^*} - 2v_{C^3}^{m^*} + v_{S^3}^{m^*}}{\Delta y}\Delta x\right)$$
$$+ S_{C^3}^{m^*}h_{C^3}^{m^*}v_{C^3}^{m^*}\Delta x\Delta y] \tag{A.9}$$

We need averaging operations for h and u in e^3 and w^3, for v in e^3, w^3, n^3 and s^3 and h in C^3

$$v_{e^3} = \frac{v_{E^3} + v_{C^3}}{2}, \quad v_{w^3} = \frac{v_{W^3} + v_{C^3}}{2}, \quad (v^2)_{n^3} = \frac{(v^2)_{N^3} + (v^2)_{C^3}}{2}, \quad (v^2)_{s^3} = \frac{(v^2)_{S^3} + (v^2)_{C^3}}{2}$$
$$u_{e^3} = \frac{u_{ne^3} + u_{se^3}}{2}, u_{w^3} = \frac{u_{nw^3} + u_{sw^3}}{2}$$
$$h_{e^3} = \frac{h_{n^3} + h_{nE^3} + h_{s^3} + h_{sE^3}}{4}, \quad h_{w^3} = \frac{h_{n^3} + h_{nW^3} + h_{s^3} + h_{sW^3}}{4}, \quad h_{C^3} = \frac{h_{s^3} + h_{n^3}}{2}$$

Vegetation

For the vegetation and sedimentation equations does not change much. We only have to determine u and v in the water depth grid points by averaging. The same control volume as for the water depth is used. The equation to model the vegetation density is given by

$$\frac{\partial n_b}{\partial t} = D\nabla\cdot\nabla n_b + r\left(1 - \frac{n_b}{K}\right)n_b\left(\frac{K_p}{K_p + h}\right) - E_p n_b\sqrt{u^2 + v^2}.$$

Integrating over control volume Ω_k^1 gives

$$\frac{dn_{b_{C^1}}}{dt}\Delta x\Delta y = D\left(\int_{e^1}\frac{\partial n_b}{\partial x} - \int_{w^1}\frac{\partial n_b}{\partial x} + \int_{n^1}\frac{\partial n_b}{\partial y} - \int_{s^1}\frac{\partial n_b}{\partial y}\right)$$
$$+ r\left(1 - \frac{n_{b_{C^1}}}{K}\right)n_{b_{C^1}}\left(\frac{K_p}{K_p + h_{C^1}}\right)\Delta x\Delta y - E_p n_b\sqrt{u_{C^1}^2 + v_{C^1}^2}\Delta x\Delta y.$$

This is equal to

$$\frac{dn_{b_{C^1}}}{dt}\Delta x\Delta y = D\left(\frac{n_{b_{E^1}} - n_{b_{C^1}}}{\Delta x}\Delta y - \frac{n_{b_{C^1}} - n_{b_{W^1}}}{\Delta x}\Delta y + \frac{n_{b_{N^1}} - n_{b_{C^1}}}{\Delta y}\Delta x - \frac{n_{b_{C^1}} - n_{b_{S^1}}}{\Delta y}\Delta x\right)$$
$$+ r\left(1 - \frac{n_{b_{C^1}}}{K}\right)n_{b_{C^1}}\left(\frac{K_p}{K_p + h_{C^1}}\right)\Delta x\Delta y - E_p n_b\sqrt{u_{C^1}^2 + v_{C^1}^2}\Delta x\Delta y.$$

Rewriting and using the semi-implicit Euler method gives

$$n_{b_{C^1}}^{m+1} = n_{b_{C^1}}^{m} + \frac{\Delta t}{\Delta x\Delta y}[D\left(\frac{n_{b_{E^1}}^{m^*} - 2n_{b_{C^1}}^{m^*} + n_{b_{W^1}}^{m^*}}{\Delta x}\Delta y + \frac{n_{b_{N^1}}^{m^*} - 2n_{b_{C^1}}^{m^*} + n_{b_{S^1}}^{m^*}}{\Delta y}\Delta x\right)$$
$$+ r\left(1 - \frac{n_{b_{C^1}}^{m^*}}{K}\right)n_{b_{C^1}}^{m^*}\left(\frac{K_p}{K_p + h_{C^1}^{m^*}}\right)\Delta x\Delta y - E_p n_b^{m^*}\sqrt{(u_{C^1}^{m^*})^2(+v_{C^1}^{m^*})^2}\Delta x\Delta y].$$

Bottom Elevation

The equation to model the behaviour of the bottom elevation is given by

$$\frac{\partial z_b}{\partial t} = S_{in}h_{eff} - E_0\left(1 - p_E\frac{n_b}{K}\right)(u^2 + v^2)z_b + D\left(\frac{\partial^2 z_b}{\partial x^2} + \frac{\partial^2 z_b}{\partial y^2}\right).$$

The same control volume as for the water depth is used. We get

$$z_{b_{C^1}}^{m+1} = z_{b_{C^1}}^{m} + \frac{\Delta t}{\Delta x\Delta y}[S_{in}h_{eff_{C^1}}^{m^*} - E_0\left(1 - p_E\frac{n_{b_{C^1}}^{m^*}}{K}\right)((u_{C^1}^{m^*})^2 + (v_{C^1}^{m^*})^2)z_{b_{C^1}}^{m^*}$$
$$+ D\left(\frac{z_{b_{E^1}}^{m^*} - 2z_{b_{C^1}}^{m^*} + z_{b_{W^1}}^{m^*}}{\Delta x}\Delta y + \frac{z_{b_{N^1}}^{m^*} - 2z_{b_{C^1}}^{m^*} + z_{b_{S^1}}^{m^*}}{\Delta y}\Delta x\right)].$$

Transport Equation

We will use for the sediment concentration the same control volume as for the water depth h.

$$\frac{\partial(hc)}{\partial t} + \frac{\partial}{\partial x}(huc) + \frac{\partial}{\partial y}(hvc) - \frac{\partial}{\partial x}\left(hD\frac{\partial c}{\partial x}\right) - \frac{\partial}{\partial y}\left(hD\frac{\partial c}{\partial y}\right) = 0 \tag{A.10}$$

Integrating over Ω_k^1

$$\int_{\Omega_k^1}\frac{\partial(hc)}{\partial t}d\Omega + \int_{\Omega_k^1}\frac{\partial}{\partial x}(huc) + \frac{\partial}{\partial y}(hvc)d\Omega - \int_{\Omega_k^1}\frac{\partial}{\partial x}\left(hD\frac{\partial c}{\partial x}\right) + \frac{\partial}{\partial y}\left(hD\frac{\partial c}{\partial y}\right)d\Omega = 0$$

gives

$$\frac{d(h_{C^1}c_{C^1})}{dt}\Delta x\Delta y + (huc)_{e^1}\Delta y - (huc)_{w^1}\Delta y + (hvc)_{n^1}\Delta x - (hvc)_{s^1}\Delta x$$
$$-\left(\left(hD\frac{\partial c}{\partial x}\right)_{e^1}\Delta y - \left(hD\frac{\partial c}{\partial x}\right)_{w^1}\Delta y\right) - \left(\left(hD\frac{\partial c}{\partial y}\right)_{n^1}\Delta x - \left(hD\frac{\partial c}{\partial y}\right)_{s^1}\Delta x\right) = 0$$

Using central differences results in

$$\frac{d(h_{C^1}c_{C^1})}{dt}\Delta x + (huc)_{e^1}\Delta y - (huc)_{w^1}\Delta y + (hvc)_{n^1}\Delta x - (hvc)_{s^1}\Delta x$$
$$-\left(\left(h_{e^1}D\frac{c_{E^1} - c_{C^1}}{\Delta x}\right) - \left(h_{w^1}D\frac{c_{C^1} - c_{W^1}}{\Delta x}\right)\right)\Delta y - \left(\left(h_{n^1}D\frac{c_{N^1} - c_{C^1}}{\Delta y}\right) - \left(h_{s^1}D\frac{c_{C^1} - c_{S^1}}{\Delta y}\right)\right)\Delta x = 0$$

We can determine h and c in the grid points where they are not defined by averaging.

$$h_{e^1} = \frac{h_{E^1} + h_{C^1}}{2},\ c_{e^1} = \frac{c_{E^1} + c_{C^1}}{2},\ h_{w^1} = \frac{h_{W^1} + h_{C^1}}{2},\ c_{w^1} = \frac{c_{W^1} + c_{C^1}}{2}$$
$$h_{n^1} = \frac{h_{N^1} + h_{C^1}}{2},\ c_{n^1} = \frac{c_{N^1} + c_{C^1}}{2},\ h_{s^1} = \frac{h_{S^1} + h_{C^1}}{2},\ c_{s^1} = \frac{c_{S^1} + c_{C^1}}{2}$$

Transport Equation (only convective) with Upwind

We will now use upwind in the approximation of the convective terms

$$h_{e^1}c_{e^1} = \begin{cases} h_{C^1}c_{C^1} & \text{If } u_{e_1} > 0 \\ h_{E^1}c_{E^1} & \text{Else} \end{cases}, \quad h_{w^1}c_{w^1} = \begin{cases} h_{W^1}c_{W^1} & \text{If } u_{w_1} > 0 \\ h_{C^1}c_{C^1} & \text{Else} \end{cases}$$

$$h_{n^1}c_{n^1} = \begin{cases} h_{C^1}c_{C^1} & \text{If } u_{n_1} > 0 \\ h_{N^1}c_{N^1} & \text{Else} \end{cases}, \quad h_{s^1}c_{s^1} = \begin{cases} h_{S^1}c_{S^1} & \text{If } v_{s_1} > 0 \\ h_{C^1}c_{C^1} & \text{Else} \end{cases}$$

Boundary Conditions

We will distinguish between a vertex- and cell centered discretisation. Since a staggered grid is used, some variables are cell centered and others vertex centered. For both we will show how to deal with Dirichlet and Neumann conditions for a one-dimensional grid. The variable u will be used as unknown.

Vertex-centered Discretisation

In Figure A.4 it is shown that the grid points lie at the vertices of the control volume (one-dimensional grid).

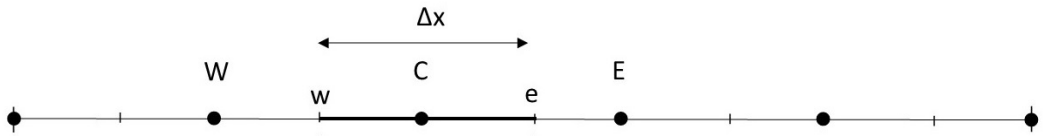


Figure A.4: Vertex centered grid.

Dirichlet boundary condition Simply substitute the value for u_C. This grid point is known and thus we do not need to perform integration over the corresponding control volume.

Neumann boundary condition In case of a Neumann boundary condition, we substitute $\frac{\partial u_C}{\partial x} = C_1$ into the equations and let u_C be.

Cell-centered Discretisation

For this discretisation the grid points are located in the center of the control volumes as is shown in Figure A.5 for a one-dimensional grid.

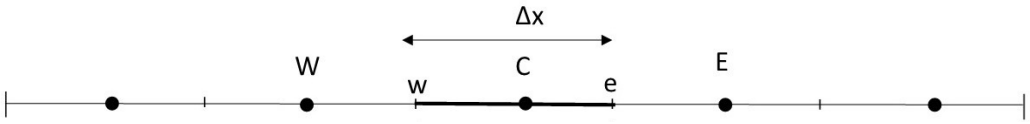


Figure A.5: Cell centered grid.

Dirichlet boundary condition Assume we have the condition $u = C_1$ on the boundary

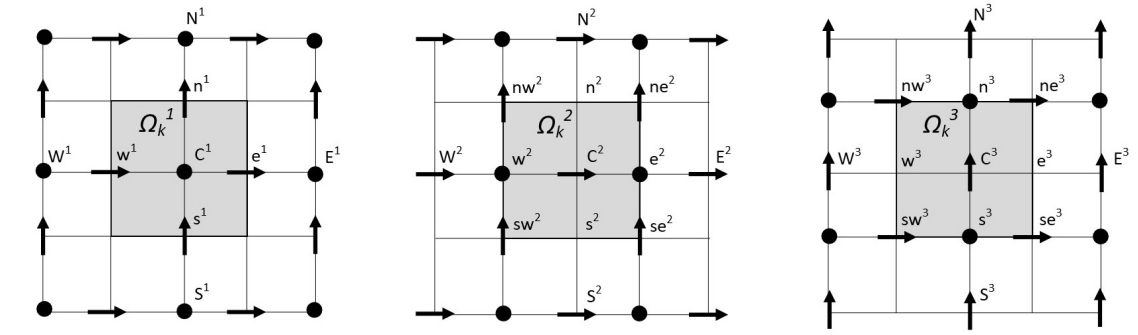
- If we need u on boundary, we can use the boundary value when necessary when integrating over a control volume.
- If we need $\frac{\partial u}{\partial x}$ on the boundary, we introduce a ghost point. For instance on the right boundary we get $u_E = 2C_1 - u_C$ and use again central differences: $\frac{\partial u_e}{\partial x} \approx \frac{u_E - u_C}{\Delta x} = \frac{2(C_1 - u_C)}{\Delta x}$.

Neumann boundary condition Assume we prescribe $\frac{\partial u}{\partial x} = C_1$ at the boundary

- If we need u on the boundary we can average with the ghost point and internal node. We can determine the ghost point by using the Neumann boundary condition. If we have have $\frac{\partial u_C}{\partial n} = 0$ the value on the boundary will be equal to u_C.
- If we need $\frac{\partial u}{\partial n}$ on the boundary, we can just substitute the boundary condition in the equations.

A.3. Finite Volume Staggered Grid Approach 2

We will again use three different control volumes as shown in Figure A.6. We define $p = hu$ and $q = hv$ and $\mathbf{p} = [p\ q]^T$. If needed we can determine u and v by dividing p and q by the average value of h in the grid point of p or q. The discretisation of the vegetation and bottom equations is the same as for approach one. So we will not derive them here again.



(a) Control volume for the water depth h.

(b) Control volume for the variable $p = hu$.

(c) Control volume for the variable $q = hv$.

Figure A.6: Different control volumes Ω_k^1, Ω_k^2 and Ω_k^3.

Continuity Equation

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0$$

Integrating over volume Ω_k^1

$$\int_{\Omega_k^1} \frac{\partial h}{\partial t} d\Omega + \int_{\Omega_k^1} \nabla \mathbf{p} d\Omega = 0$$

Applying the divergence theorem gives

$$\frac{dh_{C^1}}{dt}\Delta x \Delta y + \int_{e^1} p dy - \int_{w^1} p dy + \int_{n^1} q dx - \int_{s^1} q dx = 0$$

This gives

$$\frac{dh_{C^1}}{dt}\Delta x \Delta y + p_{e^1}\Delta y - p_{w^1}\Delta y + q_{n^1}\Delta x - q_{s^1}\Delta x = 0 \tag{A.11}$$

Momentum Equations

$$\frac{\partial p}{\partial t} + \nabla \cdot (p\mathbf{u}) + gh\frac{\partial \eta}{\partial x} - Ah(\nabla \cdot \nabla u) + Sp = 0$$

Integrating over volume Ω_k^2

$$\int_{\Omega_k^2} \frac{\partial p}{\partial t} d\Omega + \int_{\Omega_k^2} \nabla \cdot (p\mathbf{u}) + gh\frac{\partial \eta}{\partial x} - Ah(\nabla \cdot \nabla u) + Sp d\Omega = 0$$

Using the divergence theorem and the midpoint rule

$$\frac{dp_{C^2}}{dt}\Delta x \Delta y + \int_{\partial \Omega_k^2} \begin{pmatrix} pu \\ pv \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \end{pmatrix} d\Gamma + gh_{C^2} \int_{\partial \Omega_k^2} \begin{pmatrix} \eta \\ 0 \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \end{pmatrix} d\Gamma_k$$

$$- Ah_{C^2} \int_{\partial \Omega_k^2} \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \end{pmatrix} d\Gamma + S_{C^2} p_{C^2} \Delta x \Delta y = 0$$

This is equal to

$$\frac{dp_{C^2}}{dt}\Delta x\Delta y + \int_{e^2} pudy - \int_{w^2} pudy + \int_{n^2} pvdx - \int_{s^2} pvdx + gh_{C^2}\left(\int_{e^2}\eta dy - \int_{w^2} hdy\right)$$
$$- Ah_{C^2}\left(\int_{e^2}\frac{\partial u}{\partial x}dy - \int_{w^2}\frac{\partial u}{\partial x}dy + \int_{n^2}\frac{\partial u}{\partial y}dx - \int_{s^2}\frac{\partial u}{\partial y}dx\right) + S_{C^2}p_{C^2}\Delta x\Delta y = 0$$

Using again central differences gives

$$\frac{dp_{C^2}}{dt}\Delta x\Delta y + p_{e^2}u_{e^2}\Delta y - p_{w^2}u_{w^2}\Delta y + p_{n^2}v_{n^2}\Delta x - p_{s^2}v_{s^2}\Delta x$$
$$+ gh_{C^2}\left(\eta_{e^2}\Delta y - \eta_{w^2}\Delta y\right) - Ah_{C^2}\left(\frac{u_{E^2} - u_{C^2}}{\Delta x}\Delta y - \frac{u_{C^2} - u_{W^2}}{\Delta x}\Delta y + \frac{u_{N^2} - u_{C^2}}{\Delta y}\Delta x - \frac{u_{C^2} - u_{S^2}}{\Delta y}\Delta x\right)$$
$$+ S_{C^2}p_{C^2}\Delta x\Delta y = 0$$

Rewriting gives

$$\frac{dp_{C^2}}{dt}\Delta x\Delta y + p_{e^2}u_{e^2}\Delta y - p_{w^2}u_{w^2}\Delta y + p_{n^2}v_{n^2}\Delta x - p_{s^2}v_{s^2}\Delta x$$
$$+ gh_{C^2}(\eta_{e^2} - \eta_{w^2})\Delta y - Ah_{C^2}\left(\frac{u_{E^2} - 2u_{C^2} + u_{W^2}}{\Delta x}\Delta y + \frac{u_{N^2} - 2u_{C^2} + u_{S^2}}{\Delta y}\Delta x\right)$$
$$+ S_{C^2}p_{C^2}\Delta x\Delta y = 0 \tag{A.12}$$

We will determine h_{C^2} by averaging over the neighbouring nodes

$$h_{C^2} = \frac{h_{e^2} + h_{w^2}}{2}$$

We can again determine u in the p nodes by dividing by the mean of h (h_{C^2}). For the y-direction we get

$$\frac{\partial q}{\partial t} + \nabla\cdot(q\mathbf{u}) + gh\frac{\partial \eta}{\partial y} - Ah(\nabla\cdot\nabla v) + Sq = 0$$

Integrating over Ω_k^3

$$\int_{\Omega_k^3}\frac{\partial q}{\partial t}d\Omega + \int_{\Omega_k^3}\nabla\cdot(q\mathbf{u}) + gh\frac{\partial \eta}{\partial y} - Ah(\nabla\cdot\nabla v) + Sqd\Omega = 0$$

Using the divergence theorem and the midpoint rule

$$\frac{dq_{C^3}}{dt}\Delta x\Delta y + \int_{\partial\Omega_k^3}\begin{pmatrix}qu\\ qv\end{pmatrix}\cdot\begin{pmatrix}n_x\\ n_y\end{pmatrix}d\Gamma + gh_{C^3}\int_{\partial\Omega_k^3}\begin{pmatrix}0\\ \eta\end{pmatrix}\cdot\begin{pmatrix}n_x\\ n_y\end{pmatrix}d\Gamma_k$$
$$- Ah_{C^3}\int_{\partial\Omega_k^3}\begin{pmatrix}\frac{\partial v}{\partial x}\\ \frac{\partial v}{\partial y}\end{pmatrix}\cdot\begin{pmatrix}n_x\\ n_y\end{pmatrix}d\Gamma + S_{C^3}q_{C^3}\Delta x\Delta y = 0$$

This is equal to

$$\frac{dq_{C^3}}{dt}\Delta x\Delta y + \int_{e^3} qudy - \int_{w^3} qudy + \int_{n^3} qvdx - \int_{s^3} qvdx + gh_{C^3}\left(\int_{n^3}\eta dx - \int_{s^3}\eta dx\right)$$
$$- Ah_{C^3}\left(\int_{e^3}\frac{\partial v}{\partial x}dy - \int_{w^3}\frac{\partial v}{\partial x}dy + \int_{n^3}\frac{\partial v}{\partial y}dx - \int_{s^3}\frac{\partial u}{\partial y}dx\right) + S_{C^3}q_{C^3}\Delta x\Delta y = 0$$

Using central differences gives

$$\frac{dq_{C^3}}{dt}\Delta x\Delta y + q_{e^3}u_{e^3}\Delta y - q_{w^3}u_{w^3}\Delta y + q_{n^3}v_{n^3}\Delta x - q_{s^3}v_{s^3}\Delta x$$
$$+ gh_{C^3}\left(\eta_n\Delta x - \eta_s\Delta x\right) - Ah_{C^3}\left(\frac{v_{E^3} - v_{C^3}}{\Delta x}\Delta y - \frac{v_{C^3} - v_{W^3}}{\Delta x}\Delta y + \frac{v_{N^3} - v_{C^3}}{\Delta y}\Delta x - \frac{v_{C^3} - v_{S^3}}{\Delta y}\Delta x\right)$$
$$+ S_{C^3}q_{C^3}\Delta x\Delta y = 0$$

Rewriting gives

$$\frac{dq_{C^3}}{dt}\Delta x\Delta y + q_{e^3}u_{e^3}\Delta y - q_{w^3}u_{w^3}\Delta y + q_{n^3}v_{n^3}\Delta x - q_{s^3}v_{s^3}\Delta x + gh_{C^3}(\eta_{n^3} - \eta_{s^3})\Delta x - Ah_{C^3}\left(\frac{v_{E^3} - 2v_{C^3} + v_{W^3}}{\Delta x}\Delta y + \frac{v_{N^3} - 2v_{C^3} + v_{S^3}}{\Delta y}\Delta x\right) + S_{C^3}q_{C^3}\Delta x\Delta y = 0 \tag{A.13}$$

Now h_{C^3} is given by

$$h_{C^3} = \frac{h_{n^3} + h_{s^3}}{2}$$

We can again determine v in the q nodes by dividing by the average of h (h_{C^3}).

Transport Equation

We will use for the sediment concentration the same control volume as for the water depth h. Let us use $d = hc$.

$$\frac{\partial d}{\partial t} + \frac{\partial}{\partial x}(du) + \frac{\partial}{\partial y}(dv) - \frac{\partial}{\partial x}\left(hD\frac{\partial c}{\partial x}\right) - \frac{\partial}{\partial y}\left(hD\frac{\partial c}{\partial y}\right) = 0 \tag{A.14}$$

Integrating over Ω_k^1

$$\int_{\Omega_k^1}\frac{\partial d}{\partial t}d\Omega + \int_{\Omega_k^1}\frac{\partial}{\partial x}(du) + \frac{\partial}{\partial y}(dv)d\Omega - \int_{\Omega_k^1}\frac{\partial}{\partial x}\left(hD\frac{\partial c}{\partial x}\right) + \frac{\partial}{\partial y}\left(hD\frac{\partial c}{\partial y}\right)d\Omega = 0$$

gives

$$\frac{d(d_{C^1})}{dt}\Delta x\Delta y + (du)_{e^1}\Delta y - (du)_{w^1}\Delta y + (dv)_{n^1}\Delta x - (dv)_{s^1}\Delta x - \left(\left(hD\frac{\partial c}{\partial x}\right)_{e^1}\Delta y - \left(hD\frac{\partial c}{\partial x}\right)_{w^1}\Delta y\right) - \left(\left(hD\frac{\partial c}{\partial y}\right)_{n^1}\Delta x - \left(hD\frac{\partial c}{\partial y}\right)_{s^1}\Delta x\right) = 0$$

Using central differences results in

$$\frac{d(d_{C^1})}{dt}\Delta x + (du)_{e^1}\Delta y - (du)_{w^1}\Delta y + (dv)_{n^1}\Delta x - (dv)_{s^1}\Delta x - \left(\left(h_{e^1}D\frac{c_{E^1} - c_{C^1}}{\Delta x}\right) - \left(h_{w^1}D\frac{c_{C^1} - c_{W^1}}{\Delta x}\right)\right)\Delta y - \left(\left(h_{n^1}D\frac{c_{N^1} - c_{C^1}}{\Delta y}\right) - \left(h_{s^1}D\frac{c_{C^1} - c_{S^1}}{\Delta y}\right)\right)\Delta x = 0$$

We can determine h in the grid points where it is not defined by averaging.

$$h_{e^1} = \frac{h_{E^1} + h_{C^1}}{2},\ h_{w^1} = \frac{h_{W^1} + h_{C^1}}{2},\ h_{n^1} = \frac{h_{N^1} + h_{C^1}}{2},\ h_{s^1} = \frac{h_{S^1} + h_{C^1}}{2},$$

Transport Equation (only convective) with Upwind

We will now use upwind in the approximation of the convective terms

$$d_{e^1} = \begin{cases} d_{C^1} & \text{If } u_{e_1} > 0 \\ d_{E^1} & \text{Else} \end{cases},\quad d_{w^1} = \begin{cases} d_{W^1} & \text{If } u_{w_1} > 0 \\ d_{C^1} & \text{Else} \end{cases}$$

$$d_{n^1} = \begin{cases} d_{C^1} & \text{If } u_{n_1} > 0 \\ d_{N^1} & \text{Else} \end{cases},\quad d_{s^1} = \begin{cases} d_{S^1} & \text{If } v_{s_1} > 0 \\ d_{C^1} & \text{Else} \end{cases}$$

B

Richardson Experiments

B.1. Results Case 1a

α_u^{coarse}	α_h^{coarse}	α_u^{fine}	α_h^{fine}
[1,] NaN	[1,] NaN	[1,] NaN	[1,] NaN
[2,] NaN	[2,] NaN	[2,] NaN	[2,] NaN
[3,] NaN	[3,] NaN	[3,] −0.28	[3,] NaN
[4,] −1.24	[4,] NaN	[4,] 2.30	[4,] NaN
[5,] NaN	[5,] NaN	[5,] 3.77	[5,] NaN
[6,] NaN	[6,] NaN	[6,] NaN	[6,] NaN
[7,] NaN	[7,] NaN	[7,] NaN	[7,] NaN
[8,] NaN	[8,]−0.03	[8,] 0.13	[8,] NaN
[9,] 1.91	[9,]2.24	[9,] NaN	[9,] NaN
[10,] NaN	[10,]0.87	[10,] NaN	[10,] NaN
[11,] −0.96	[11,]2.82	[11,] NaN	[11,] NaN
[12,] NaN	[12,]1.38	[12,] NaN	[12,] NaN
[13,] 1.31	[13,]1.39	[13,] NaN	[13,] NaN
[14,] 1.43	[14,]0.92	[14,] NaN	[14,] NaN
[15,] 0.09	[15,]0.96	[15,] NaN	[15,] NaN
[16,] NaN		[16,] NaN	[16,] NaN
		[17,]NaN	[17,] NaN
		[18,] NaN	[18,] NaN
		[19,] NaN	[19,] NaN
		[20,] NaN	[20,] NaN
		[21,] NaN	[21,] NaN
		[22,] NaN	[22,] NaN
		[23,] −1.20	[23,]−0.25
		[24,] −0.03	[24,] NaN
		[25,] NaN	[25,] NaN
		[26,] NaN	[26,] NaN
		[27,] NaN	[27,] −0.18
		[28,] 0.12	[28,] −0.23
		[29,] NaN	[29,] NaN
		[30,] NaN	[30,] NaN
		[31,] 0.36	[31,] −0.17
		[32,] 2.32	[32,] −1.21
		[33,] NaN	[33,] 1.60
		[34,] 1.03	[34,] 2.68
		[35,] NaN	[35,] NaN
		[36,] NaN	[36,] NaN
		[37,] NaN	[37,] NaN
		[38,] NaN	[38,] NaN
		[39,] NaN	[39,] NaN
		[40,] NaN	[40,] NaN
		[41,] NaN	[41,] 1.00
		[42,] 2.83	[42,] 1.10
		[43,] 1.96	[43,] 0.89
		[44,] −1.51	[44,] 1.14
		[45,] NaN	[45,] 1.00
		[46,] NaN	

B.2. Results Case 1b

α_u^{coarse}
[1,] NaN
[2,] 1.49
[3,] 3.28
[4,] NaN
[5,] 0.11
[6,] 1.47
[7,] NaN
[8,] 2.47
[9,] 2.48
[10,] NaN
[11,] NaN
[12,] −0.29
[13,] 1.17
[14,] 1.66
[15,] 1.75
[16,] NaN

α_h^{coarse}
[1,] 3.61
[2,] 0.62
[3,] NaN
[4,] 1.63
[5,] NaN
[6,] NaN
[7,] 2.42
[8,] NaN
[9,] NaN
[10,] 0.47
[11,] −0.01
[12,] 0.60
[13,] 1.52
[14,] 1.76
[15,] 1.83

α_u^{fine}
[1,] NaN
[2,] 3.49
[3,] NaN
[4,] NaN
[5,] NaN
[6,] NaN
[7,] 0.79
[8,] NaN
[9,] 2.05
[10,] 2.88
[11,] NaN
[12,] −1.66
[13,] 1.77
[14,] NaN
[15,] 0.83
[16,] 1.57
[17,] NaN
[18,] NaN
[19,] 4.43
[20,] NaN
[21,] NaN
[22,] NaN
[23,] NaN
[24,] 1.29
[25,] 3.74
[26,] NaN
[27,] 2.27
[28,] NaN
[29,] 2.26
[30,] NaN
[31,] 1.66
[32,] NaN
[33,] 0.18
[34,] −0.02
[35,] 1.02
[36,] 1.31
[37,] 1.36
[38,] 1.33
[39,] 1.32
[40,] 1.28
[41,] 1.26
[42,] 1.28
[43,] 1.27
[44,] 1.19
[45,] 1.25
[46,] NaN

α_h^{fine}
[1,] NaN
[2,] NaN
[3,] NaN
[4,] NaN
[5,] 2.32
[6,] 1.95
[7,] 3.67
[8,] NaN
[9,] 2.28
[10,] NaN
[11,] 1.93
[12,] NaN
[13,] 1.18
[14,] 2.84
[15,] NaN
[16,] NaN
[17,] 2.28
[18,] 2.39
[19,] 1.40
[20,] NaN
[21,] 2.23
[22,] 0.99
[23,] 1.99
[24,] 3.06
[25,] NaN
[26,] NaN
[27,] NaN
[28,] 0.51
[29,] NaN
[30,] NaN
[31,] 0.16
[32,] NaN
[33,] NaN
[34,] 0.67
[35,] 1.22
[36,] 1.35
[37,] 1.35
[38,] 1.32
[39,] 1.31
[40,] 1.28
[41,] 1.31
[42,] 1.31
[43,] 1.33
[44,] 1.38
[45,] 1.36

B.3. Results Case 1c

α_u^{coarse}	α_h^{coarse}	α_u^{fine}	α_h^{fine}
[1,] NaN	[1,] 3.53	[1,] NaN	[1,] NaN
[2,] 1.49	[2,] 0.59	[2,] 3.63	[2,] NaN
[3,] 3.22	[3,] NaN	[3,] NaN	[3,] NaN
[4,] NaN	[4,] 1.62	[4,] NaN	[4,] NaN
[5,] 0.13	[5,] NaN	[5,] NaN	[5,] 2.27
[6,] 1.47	[6,] NaN	[6,] NaN	[6,] 2.00
[7,] NaN	[7,] 2.45	[7,] 0.95	[7,] 3.27
[8,] 2.52	[8,] NaN	[8,] NaN	[8,] NaN
[9,] 2.54	[9,] NaN	[9,] 2.14	[9,] 2.42
[10,] NaN	[10,] 0.49	[10,] 2.73	[10,] NaN
[11,] NaN	[11,] 0.04	[11,] NaN	[11,] 1.98
[12,] −0.27	[12,] 0.59	[12,] −1.24	[12,] NaN
[13,] 1.17	[13,] 1.52	[13,] 1.87	[13,] 1.47
[14,] 1.67	[14,] 1.77	[14,] NaN	[14,] 3.03
[15,] 1.76	[15,] 1.83	[15,] 0.95	[15,] NaN
[16,] NaN		[16,] 1.62	[16,] NaN
		[17,] NaN	[17,] 2.36
		[18,] NaN	[18,] 2.45
		[19,] 3.74	[19,] 1.44
		[20,] NaN	[20,] NaN
		[21,] NaN	[21,] 2.56
		[22,] NaN	[22,] 1.04
		[23,] NaN	[23,] 1.96
		[24,] 1.33	[24,] 2.90
		[25,] 2.78	[25,] NaN
		[26,] NaN	[26,] NaN
		[27,] 2.62	[27,] NaN
		[28,] NaN	[28,] 0.56
		[29,] 2.45	[29,] NaN
		[30,] NaN	[30,] NaN
		[31,] 1.58	[31,] 0.24
		[32,] NaN	[32,] NaN
		[33,] 0.28	[33,] NaN
		[34,] −0.03	[34,] 0.65
		[35,] 1.01	[35,] 1.21
		[36,] 1.31	[36,] 1.35
		[37,] 1.35	[37,] 1.35
		[38,] 1.33	[38,] 1.33
		[39,] 1.31	[39,] 1.30
		[40,] 1.29	[40,] 1.29
		[41,] 1.27	[41,] 1.29
		[42,] 1.26	[42,] 1.30
		[43,] 1.26	[43,] 1.31
		[44,] 1.26	[44,] 1.33
		[45,] 1.26	[45,] 1.33
		[46,] NaN	

B.4. Results Case 2

α_u^{coarse}	α_h^{coarse}	α_u^{fine}	α_h^{fine}
[1,] NaN	[1,] NaN	[1,] NaN	[1,] 0.34
[2,] 0.55	[2,] 0.81	[2,] 0.46	[2,] 0.47
[3,] 1.71	[3,] NaN	[3,] 0.63	[3,] 0.72
[4,] 2.52	[4,] 3.89	[4,] 0.87	[4,] 0.97
[5,] 3.63	[5,] 2.41	[5,] 1.10	[5,] 1.22
[6,] NaN	[6,] 1.60	[6,] 1.38	[6,] 1.50
[7,] 1.69	[7,] 0.96	[7,] 1.55	[7,] 1.28
[8,] NaN	[8,] −0.44	[8,] NaN	[8,] NaN
[9,] NaN	[9,] 0.96	[9,] NaN	[9,] NaN
[10,] 1.69	[10,] 1.60	[10,] NaN	[10,] 4.18
[11,]NaN	[11,] 2.41	[11,] 2.55	[11,] 3.80
[12,] 3.63	[12,] 3.89	[12,] 2.02	[12,] NaN
[13,] 2.52	[13,] NaN	[13,] 1.41	[13,] 0.86
[14,] 1.71	[14,] 0.81	[14,] 3.15	[14,] 1.81
[15,] 0.55	[15,] NaN	[15,] 0.45	[15,] 0.75
[16,] NaN		[16,] 1.58	[16,] 1.35
		[17,] 0.71	[17,] 1.29
		[18,] 0.68	[18,] 1.22
		[19,] 0.97	[19,] 1.16
		[20,] 0.97	[20,] 1.09
		[21,] 0.97	[21,] 0.93
		[22,] 1.00	[22,] 0.77
		[23,] 0.69	[23,] 0.69
		[24,] 0.70	[24,] 0.77
		[25,] 1.00	[25,] 0.93
		[26,] 0.97	[26,] 1.09
		[27,] 0.97	[27,] 1.16
		[28,] 0.96	[28,] 1.22
		[29,] 0.69	[29,] 1.29
		[30,] 0.72	[30,] 1.36
		[31,] 1.58	[31,] 0.73
		[32,] 0.44	[32,] 1.90
		[33,] 3.34	[33,] 0.94
		[34,] 1.39	[34,] NaN
		[35,] 2.11	[35,] 3.79
		[36,] 2.58	[36,] 4.33
		[37,] NaN	[37,] NaN
		[38,] NaN	[38,] NaN
		[39,] NaN	[39,] 1.28
		[40,] 1.55	[40,] 1.50
		[41,] 1.38	[41,] 1.22
		[42,] 1.10	[42,] 0.97
		[43,] 0.87	[43,] 0.72
		[44,] 0.63	[44,] 0.47
		[45,] 0.46	[45,] 0.34
		[46,] NaN	

B.5. Results Case 3

α_u^{coarse}	α_h^{coarse}	α_u^{fine}	α_h^{fine}
[1,] 1.20	[1,] NaN	[1,] NaN	[1,] NaN
[2,] 0.76	[2,] 0.78	[2,] NaN	[2,] 0.77
[3,] 0.91	[3,] 0.81	[3,] NaN	[3,] 0.67
[4,] 1.56	[4,] −1.29	[4,] NaN	[4,] 0.56
[5,] NaN	[5,] NaN	[5,] 2.88	[5,] 0.54
[6,] −0.32	[6,] NaN	[6,] 2.88	[6,] 0.65
[7,] 3.93	[7,] 0.79	[7,] 3.55	[7,] 0.36
[8,] 2.67	[8,] 2.06	[8,] NaN	[8,] 0.19
[9,] NaN	[9,] 2.21	[9,] NaN	[9,] −0.09
[10,] 0.95	[10,] 0.77	[10,] NaN	[10,] −0.40
[11,] 1.88	[11,] 1.85	[11,] 0.45	[11,] −0.44
[12,] 2.77	[12,] 2.39	[12,] 0.65	[12,] 0.20
[13,] NaN	[13,] 2.29	[13,] 0.83	[13,] −0.22
[14,] NaN	[14,] 2.08	[14,] NaN	[14,] −1.42
[15,] NaN	[15,] 1.94	[15,] 2.18	[15,] −1.02
		[16,] 0.87	[16,] −1.11
		[17,] 0.99	[17,] −0.31
		[18,] −0.33	[18,] 0.12
		[19,] NaN	[19,] 0.07
		[20,] −0.99	[20,] 0.06
		[21,] 1.24	[21,] 0.00
		[22,] NaN	[22,] 0.14
		[23,] NaN	[23,] 0.19
		[24,] NaN	[24,] 0.59
		[25,] 2.75	[25,] 1.02
		[26,] 2.87	[26,] 1.20
		[27,] 3.11	[27,] 1.57
		[28,] 3.25	[28,] 1.95
		[29,] NaN	[29,] −0.58
		[30,] 1.59	[30,] 1.61
		[31,] 1.82	[31,] 1.90
		[32,] 2.00	[32,] 2.02
		[33,] 2.03	[33,] 1.94
		[34,] 2.35	[34,] 2.28
		[35,] 1.20	[35,] 1.97
		[36,] 0.68	[36,] 1.81
		[37,]NaN	[37,] 2.22
		[38,] 2.05	[38,] 3.05
		[39,] 0.52	[39,] NaN
		[40,] −1.52	[40,] NaN
		[41,] NaN	[41,] NaN
		[42,] NaN	[42,] NaN
		[43,] NaN	[43,] NaN
		[44,] NaN	[44,] NaN

C

Measured Times

C.1. Varying Number of Iterations

Table C.1 shows the results of five runs and the mean value (two runs for 500 iterations).

Iterations	Time CPU (s)	Total Time GPU (s)	Acceleration
10	3.8679	0.5823	-
	3.9209	0.3881	-
	3.9261	0.3882	-
	3.9365	0.6198	-
	3.8538	0.4558	-
mean	3.90104	0.48684	8.0
100	38.2451	2.9501	-
	38.6472	2.9469	-
	38.3403	2.9349	-
	38.7054	2.677	-
	38.2471	2.8742	-
mean	38.43702	2.87662	13.4
250	96.2493	6.8468	-
	96.7795	6.5081	-
	96.7524	6.5062	-
	97.1534	6.8436	-
	96.9376	6.5082	-
mean	96.77444	6.64258	14.6
500	193.9386	13.3947	-
	195.1650	13.4114	-
mean	194.5518	13.40305	14.5

Table C.1: Single precision results in seconds on CPU and GPU for a varying number of iterations for 16,777,216 unknowns.

C.2. Varying Number of Unknowns

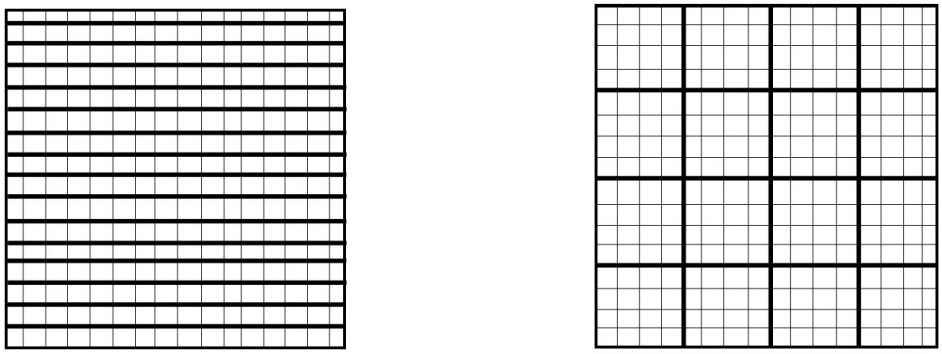
Table C.2 shows the results of five runs and the mean value.

C.3. Different Work-Group Size

Table C.3 shows the results of five runs for three different work-group sizes.

C.4. Two Work Dimensions

Since we use a two-dimensional grid, it could be an obvious choice to use two dimensions to specify the global work-items and work-items in the work-group. Such that threads allocated to the same block correspond to a square (or rectangle) area in our domain. In Figure C.1 we have illustrated a possible one and two-dimensional allocation, in which the total number of work-items and the size of the work-group are the same. This example uses 256 work-items in total and each work-group consists of 16 work-items. However, the way in which each work-group is arranged is different. In order to



(a) One dimension. (b) Two dimensions.

Figure C.1: One and two dimensions to specify the work-group items.

use two dimensions, we need to replace lines 39-47 of Listing 6.1 by Listing C.1.

Listing C.1: Two-dimensional EnqueueNDRangeKernel

```
1 size_t global_item_size[] = {Grid_Width, Grid_Height};
2 size_t local_item_size[] = {Block_Size_X,Block_Size_Y};

4 err = clEnqueueNDRangeKernel(command_queue, kernel, 2, NULL,
5           global_item_size, local_item_size, 0, NULL, NULL);
```

The blocks (work-group) now have dimensions $Block\_Size\_X$ times $Block\_Size\_Y$. We again need to make sure that $Grid\_Width$ is evenly divisible by $Block\_Size\_X$ and $Grid\_Height$ by $Block\_Size\_Y$. This allocation can be beneficial when work-items within a work-group need to communicate because work-items within a work-group have the unique ability to share local memory with one another. The local memory shared by all the work items in a single work-group is faster than the global memory shared by all the work-groups on the device. Private memory, available only to a single work item, is even faster. Besides, work-items within a single work-group can synchronize with one another at programmer-specified barriers.

In Table C.4 we have recalculated one case by specifying two dimensions, instead of one, to specify the global work-items and work-items in the work-group. The time needed for data transfer should be approximately the same. However, we notice the computation time has increased, instead of decreased.

Unknowns	Time CPU (s)	Time GPU (s)	CPU → GPU (s)	GPU → CPU (s)	Total GPU (s)	Accel-eration	Max Accel-eration
16,384	0.0005	0.0004	0.0025	0.0006	-	-	-
	0.0005	0.0003	0.0113	0.0006	-	-	-
	0.0006	0.0003	0.0025	0.0006	-	-	-
	0.0006	0.0003	0.0025	0.0006	-	-	-
	0.0006	0.0003	0.0114	0.0006	-	-	-
Mean	0.0006	0.0003	0.00604	0.0006	0.00694	0.086	2
65,536	0.0020	0.0004	0.0026	0.0007	-	-	-
	0.0017	0.0004	0.0113	0.0006	-	-	-
	0.0019	0.0004	0.0111	0.0007	-	-	-
	0.0018	0.0004	0.0110	0.0007	-	-	-
	0.0022	0.0003	0.0111	0.0007	-	-	-
Mean	0.00192	0.0004	0.00942	0.0007	0.01052	0.18	4.8
262.144	0.0087	0.0006	0.0135	0.0016	-	-	-
	0.0075	0.0006	0.0048	0.0016	-	-	-
	0.0072	0.0006	0.0046	0.0016	-	-	-
	0.0074	0.0006	0.0136	0.0017	-	-	-
	0.0069	0.0006	0.0138	0.0017	-	-	-
Mean	0.00754	0.0006	0.01006	0.0016	0.01226	0.62	12.6
1,048,576	0.0297	0.0020	0.0131	0.0036	-	- -	
	0.0311	0.0020	0.0199	0.0039	-	-	-
	0.0290	0.0019	0.0126	0.0035	-	-	-
	0.0307	0.0020	0.0196	0.0038	-	-	-
	0.0311	0.0021	0.0131	0.0036	-	-	-
Mean	0.03032	0.0020	0.01566	0.00356	0.02112	1.45	15.1
4,194,304	0.0868	0.0067	0.0356	0.0115	-	-	-
	0.0817	0.0067	0.0355	0.0111	-	-	-
	0.0870	0.0067	0.0378	0.0126	-	-	-
	0.0870	0.0067	0.0379	0.0126	-	-	-
	0.0903	0.0057	0.0379	0.0126	-	-	-
Mean	0.08656	0.0065	0.03694	0.01208	0.05552	1.56	13.3
16,777,216	0.3576	0.0204	0.0920	0.0386	-	-	-
	0.3674	0.0260	0.0887	0.0428	-	-	-
	0.3648	0.0204	0.0887	0.0429	-	-	-
	0.3664	0.0259	0.0886	0.0429	-	-	-
	0.3656	0.0259	0.0888	0.0435	-	-	-
Mean	0.36436	0.02372	0.08936	0.04214	0.15522	2.35	15.4

Table C.2: Single precision results in seconds on CPU and GPU for a varying number of unknowns.

Unknowns	Time GPU (s) (work-group size: 128)	Time GPU (s) (work-group size: 256)	Time GPU (s) (work-group size: $NULL$)
16,777,221	0.0202	0.0205	0.0201
	0.0260	0.0258	0.0259
	0.0204	0.0204	0.0204
	0.0260	0.0203	0.0259
	0.0203	0.0257	0.0259
Mean	0.2264	0.02244	0.02364

Table C.3: Single precision results in seconds on CPU and GPU for a varying number of unknowns.

Unknowns	Time GPU (s)	CPU → GPU (s)	GPU → CPU (s)	Total GPU (s)
16.777.216	0.0820	0.1103	0.0594	0.2517

Table C.4: Single precision results in seconds on a GPU for two work dimensions.

Bibliography

[1] S. C. Medeiros and S. C. Hagen, *Review of wetting and drying algorithms for numerical tidal flow models,* International journal for numerical methods in fluids **71**, 473 (2013).

[2] M. Baptist, V. Babovic, J. R. Uthurburu, M. Keijzer, R. Uittenbogaard, A. Mynett, and A. Verwey, *On inducing equations for vegetation resistance,* Journal of Hydraulic Research **45**, 435 (2007).

[3] P. Wesseling, *Principles of computational fluid dynamics*, Vol. 29 (Springer Science & Business Media, 2009).

[4] C. Vuik and C. Lemmens, *Programming on the GPU with CUDA* (Springer Science & Business Media, 2015).

[5] S. Temmerman, T. Bouma, J. Van de Koppel, D. Van der Wal, M. De Vries, and P. Herman, *Vegetation causes channel erosion in a tidal landscape,* Geology **35**, 631 (2007).

[6] Salt marsh, (2017), https://en.wikipedia.org/wiki/Salt_marsh [Accessed: 24-10-2017].

[7] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, Vol. 31 (Cambridge university press, 2002).

[8] L. C. Van Rijn, *Principles of fluid flow and surface waves in rivers, estuaries, seas and oceans*, Vol. 11 (Aqua Publications Amsterdam, The Netherlands, 1990).

[9] Underdetermined system, (2017), https://en.wikipedia.org/wiki/Underdetermined_system [Accessed: 24-10-2017].

[10] J. H. Ferziger and M. Peric, *Computational methods for fluid dynamics* (Springer Science & Business Media, 2012).

[11] W.-Y. Tan, *Shallow water hydrodynamics: Mathematical theory and numerical solution for a two-dimensional system of shallow-water equations*, Vol. 55 (Elsevier, 1992).

[12] C. B. Vreugdenhil, *Numerical methods for shallow-water flow*, Vol. 13 (Springer Science & Business Media, 2013).

[13] Deltares, *Delft3d-flow, simulation of multi-dimensional hydrodynamic flows and transport phenomena, including sediments. user manual.* Delft, the Netherlands (2028).

[14] O. Gourgue, R. Comblen, J. Lambrechts, T. Kärnä, V. Legat, and E. Deleersnijder, *A flux-limiting wetting–drying method for finite-element shallow-water models, with application to the scheldt estuary,* Advances in Water Resources **32**, 1726 (2009).

[15] Deltares, *Delft3d-tide, analysis and prediction of tides. user manual.* Delft, the Netherlands (2018).

[16] Range variation: springs and neaps, (2017), https://en.wikipedia.org/wiki/Tide [Accessed: 6-11-2017].

[17] Sea level: Dutch coast and worldwide, 1890-2014, (2016), http://www.clo.nl/en/indicators/en0229-sea-level-dutch-coast-and-worldwide [Accessed: 8-11-2017].

[18] Sea level: Dutch coast and worldwide, 1890-2014, (2017), https://nl.wikipedia.org/wiki/Getijde_(waterbeweging) [Accessed: 8-11-2017].

[19] S. Fagherazzi, M. Hannion, and P. D'Odorico, *Geomorphic structure of tidal hydrodynamics in salt marsh creeks,* Water resources research **44** (2008).

[20] L. C. Van Rijn *et al.*, *Principles of sediment transport in rivers, estuaries and coastal seas*, Vol. 1006 (Aqua publications Amsterdam, 1993).

[21] A. D'Alpaos, S. Lanzoni, M. Marani, and A. Rinaldo, *Landscape evolution in tidal embayments: modeling the interplay of erosion, sedimentation, and vegetation dynamics,* Journal of Geophysical Research: Earth Surface **112** (2007).

[22] S. Fagherazzi, M. L. Kirwan, S. M. Mudd, G. R. Guntenspergen, S. Temmerman, A. D'Alpaos, J. Koppel, J. M. Rybczyk, E. Reyes, C. Craft, *et al.*, *Numerical models of salt marsh evolution: Ecological, geomorphic, and climatic factors,* Reviews of Geophysics **50** (2012).

[23] A. Giardino, E. Ibrahim, S. Adam, E. A. Toorman, and J. Monbaliu, *Hydrodynamics and cohesive sediment transport in the ijzer estuary, belgium: Case study,* Journal of waterway, port, coastal, and ocean engineering **135**, 176 (2009).

[24] A. Crosato, I. Tanczos, M. De Vries, and Z. Wang, *Quantification of biogeomorphological variables for Dutch tidal systems*, Tech. Rep. (Deltares (WL), 2002).

[25] M. W. Straatsma and M. Baptist, *Floodplain roughness parameterization using airborne laser scanning and spectral remote sensing,* Remote Sensing of Environment **112**, 1062 (2008).

[26] J. van Kan, A. Segal, and F. Vermolen, *Numerical methods in scientific computing* (VSSD, 2005).

[27] D. Mike, *21&MIKE 3 FLOW MODEL FM Hydrodynamic and Transport Module Scientific Documentation*, Tech. Rep. (Copenhagen: DHI, 2011).

[28] Performance and Correctness of GPGPU Applications, (2017), http://fmt.ewi.utwente.nl/Workshops/NIRICT_GPGPU/ [Accessed: 24-10-2017].

[29] J. van de Koppel, R. Gupta, and C. Vuik, *Scaling-up spatially-explicit ecological models using graphics processors,* Ecological modelling **222**, 3011 (2011).

[30] D. R. Kaeli, P. Mistry, D. Schaa, and D. P. Zhang, *Heterogeneous Computing with OpenCL 2.0* (Morgan Kaufmann, 2015).

[31] Thread and block heuristics in cuda programming , (), http://cuda-programming.blogspot.nl/2013/01/thread-and-block-heuristics-in-cuda.html [Accessed: 24-10-2017].

[32] R. Gupta, *Implementation of the deflated preconditioned conjugate gradient method for bubbly flow on the graphical processing unit (gpu),* (2010).

[33] Erros, (), https://www.khronos.org/registry/OpenCL/sdk/1.0/docs/man/xhtml/errors.html [Accessed: 2-4-2018].

[34] S. Temmerman, T. Bouma, G. Govers, Z. Wang, M. De Vries, and P. Herman, *Impact of vegetation on flow routing and sedimentation patterns: Three-dimensional modeling for a tidal marsh,* Journal of Geophysical Research: Earth Surface **110** (2005).

[35] Precision-of-numeric-calculation, (), https://nl.mathworks.com/help/symbolic/increase-precision-of-numeric-calculations.html [Accessed: 2-4-2018].

[36] C. S. S. T. Olivier Gourgue, Jim van Belzen and J. van de Koppel, *Hedwige and prosper polders modeling project,* .

[37] M. Aissa, T. Verstraete, and C. Vuik, *Toward a gpu-aware comparison of explicit and implicit cfd simulations on structured meshes,* Computers & Mathematics with Applications **74**, 201 (2017).