2011

An Euler solver for nonlinear water waves using a modified staggered grid and Gaussian quadrature approach

Qi Fan

Louisiana State University and Agricultural and Mechanical College

AN EULER SOLVER FOR NONLINEAR WATER WAVES USING A MODIFIED STAGGERED GRID AND GAUSSIAN QUADRATURE APPROACH

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Engineering Science
in
The Interdisciplinary Program in Engineering Science

By
Qi Fan
B.S., Wuhan University, 2003
December 2011

ACKNOWLEDGEMENTS

The deepest appreciation goes to my advisor Dr. Qin Jim Chen who has always been supportive and an inspiration. His academic rigour helps me develop a good attitude about research, and his guidance and influence not only in research but also in other aspects are an estate for life. I also want to thank my committee members Dr. Haosheng Huang and Dr. Xiaoliang Wan for the valuable guidances and suggestions I received from them when I got into difficulties in research.

I also want to thank all of my collaborators. I really appreciate Dr. Haihong Zhao and Dr. Kelin Hu's enormous help in every aspects which made my work and life here much easier. I 'd also like to thank my fellow students Ranjit Jadhav, Qian Zhang, Ling Zhu, Ke Liu for the wonderful working environment they create and the kind help they offered.

I am also very grateful for the help I get from my collaborators in LSU's Center of Computational Technology (CCT). I want to thank Dr. Jian Tao and Lei Jiang's technical support for my work. Also I 'd like to thank Dr. Soon-Heum Ko who spent a lot of his own time helping me solve specific problems.

Finally, I appreciate my parents for their love and continuous support and my wife, Shanshan Cai, who has always been supporting me and influencing me with her great personality.

Table of Contents

List of Figures

ABSTRACT

A structured, finite-volume Euler solver for non-hydrostatic, free surface flows is developed to simulate coastal nonlinear dispersive water waves. A semi-implicit projection method which splits the pressure term into hydrostatic and non-hydrostatic parts is employed. A vertically shifted, staggered grid is designed to accommodate a new Gaussian discharge calculator and to facilitate the enforcement of a non-hydrostatic free-surface boundary condition. The Gaussian discharge calculator on the shifted grid increases the dispersion accuracy compared to traditional approaches that calculate discharges on a regular staggered grid. Numerical results are presented to demonstrate the improvements of these methods.

Chapter 1

Introduction

1.1 An introduction to water waves

As one of the most powerful natural phenomena on Earth, water waves strongly affect coastal areas. In the vast majority of coastal engineering projects, from beach nourishment to harbor construction, we need to take into account the influence of water waves and their impact on these projects. Therefore, accurate and efficient wave modeling is very important in coastal engineering.

Even though waves can be generated by earthquakes or boat wakes, the main energy source for ocean waves is wind. The amplitude and period of waves are determined by the strength of the wind, fetch (the distance over which the wind blows), and the duration of the wind. The behavior of waves changes when they propagate from deep water to shallow water. The definition of deep water or shallow water is determined by the ratio of water depth (h) to wave length (L). A wave is categorized as a deep water wave when $h/L \geq 0.5$ and a shallow water wave when $h/L \leq 0.05$. Deep water waves are dispersive, which means wave speed is determined by wave period(T). The deep water wave speed approximates $\frac{g}{2\pi}T$, where g is the acceleration of gravity. This means the fastest waves are the ones with the longest wave period or the longest wavelength. Shallow water waves are non-dispersive waves. The expression of the shallow water wave speed is $\sqrt{gh}$. Thus the speed of shallow water waves is only determined by water depth. The shallow water equation that is the depth-integrated Euler equations of motion can be used to model shallow water waves. The shallow water equation is under the hydrostatic assumption which is only correct when vertical-to-horizontal scales are very small. Thus it fails in modeling deep water waves because the vertical velocity distribution needs to be considered for an accurate deep water simulation. Therefore non-hydrostatic pressure has to be included in a dispersive wave model.

1.2 Numerical modeling of water waves

Navier-Stokes (N-S) equation with free surface boundary conditions is the most accurate equation for wave modeling. However, solving the N-S equation in 3D with a free surface in a large domain is very expensive in terms of computational power. Therefore, many approximate models have been developed for practical applications. The main effort has been made to reduce the computational time and extend model from shallow water to deep water, and to the surf zone.

The first attempt is ray approximation for infinitesimal waves propagating over mildly-varied depth. In this approximation, wave rays are found by employing the geometrical optic theory, which defines the wave ray as a curve tangential to the wave number vector. Then the spatial variation of the wave envelope along the rays can be calculated by invoking the energy conservation law. However, the approximation does not take into account the energy transfer between wave rays. Hence, it fails when neighboring wave rays intersect, and diffraction and nonlinearity are important.

An improvement to ray approximation was later made by the mild slope equation which was first suggested by Eckart(1952) and then rederived by Berkhoff(1972, 1976). The two dimensional mild slope equation can deal with large regions of refraction and diffraction. The assumption made to derive the equation is that the evanescent modes are not important for waves propagating over a slowly varying bathymetry. The mild slope equation is essentially depth integrated using an analytical velocity profile, which enables it to model wave propagation from deep water to shallow water.

Boussinesq equations have been the dominant model for coastal wave simulation in the past two decades. The classic Boussinesq equations for variable depth under the assumption of weak nonlinearity and weak frequency dispersion were derived by Peregrine (1967). A depth averaged horizontal velocity is used in the classical model. The assumption of this set of equations makes it impossible to use the model in deep water. Later the effort of extending Boussinesq equations to deeper water was made by Madsen et al. (1991) and Nwogu (1993). The method proposed by Nwogu uses a reference velocity instead of the depth averaged velocity, which extends Boussinesq equations to deep water, where the water depth (h) to wave length (L) ratio is 0.5.

Even though the modified Boussinesq equations have good dispersion properties in water of $0 < \frac{h}{L} \leq 0.5$, they are still restricted to weakly nonlinear waves. The defect can be remedied by removing the weak nonlinearity assumption as in Liu (1994), Wei et al. (1995), Wu (2001) and Madsen et al. (2003).

The continuous increase in computing power and the development of High Performance Computing (HPC) technologies make it possible to use the full N-S equation to model waves. A fractional step method for solving free surface N-S equation was first proposed by Casulli and Stelling (1998). One problem with this method early on is that a large number of vertical layers, like 10-40, were need to achieve a good dispersion accuracy in deep water. A lot of effort has been devoted to reducing the number of vertical layers for wave modeling. Stelling and Zijlema (2003) employed a Keller-box scheme that enables the use of only two vertical layers to achieve acceptable dispersion accuracy for deep water modeling ($h/L \leq 0.5$). Another impressive work was done by Yuan and Wu (2004). In order to better represent the non-hydrostatic pressure at the top layer, they integrated the vertical velocity from the top layer center to free surface to get the accurate top layer non-hydrostatic pressure.

1.3 Objective and overview of thesis

The objective of the present study is to develop a numerical model for nonlinear, dispersive waves in coastal regions, which not only has good dispersion properties but also accurately predicts the vertical distribution of velocities.

In this study, a numerical model using the pressure correction method is developed based on the Euler equations. A modified staggered grid is introduced to improve the accuracy of the model. This new grid shifts all the variables on a regular staggered grid by half a layer and determines the layer interface according to the zeros of Gauss-Jacobi polynomial. The new modified grid makes it possible to achieve a good wave dispersion accuracy using only two layers and an accurate velocity profile as well. The outline of the dissertation is as follows. In Chapter one, a brief introduction to water waves and wave modeling is given. A literature review of N-S wave models are presented in Chapter two. The details of the new model on a modified staggered grid are presented in Chapter three. Numerical results generated by the model to verify the model is shown in Chapter four. Conclusions and discussion are given in Chapter five.

Chapter 2

Literature review

There are many N-S equation based free surface flow models that are capable of simulating non-linear dispersive waves. To capture the moving free surface, several methods, such as the arbitrary Lagrangian-Eulerian method (e.g. Hodeges and Street 1999, and Zhou and Stansby 1999), marker and cell method (e.g. Park et al. 1999), volume of fluid method (e.g. Ng and Kot 1992, Shen et al. 2004, and Nielsen and Mayer 2004), and level-set method (e.g. Iafrati and Campana 2003, and Yue et al. 2004) have been successfully incorporated in the N-S model. However, these methods are yet to be used for large scale, near-shore wave modeling due to high computational costs and strict stability requirements.

More than a decade ago, a fractional step method was used by Casulli and Stelling (1998), Casulli (1999) and Stelling and Busnelli (2001) to model free surface flows at a reasonable computational cost. This method added a non-hydrostatic step to the hydrostatic model.

The incorporation of non-hydrostatic pressure enables the modeling of short waves. Since the fractional step method can be implemented on an existing hydrostatic model, the extra effort for model development has been minimized. The pressure in the fractional step method is separated into two parts, the hydrostatic and non-hydrostatic parts. The non-hydrostatic pressure is not involved in the momentum equation solved in the hydrostatic step and only calculated in the non-hydrostatic step by solving a Poisson equation. This will cause a cut-off error that leads to severe wave damping (Zijlema and Stelling 2005). This problem is solved by the pressure correction method (Stansby and Zhou 1998, Zhou and Stansby 1999, and Kocyigit et al. 2002). In this pressure correction method, the non-hydrostatic pressure is included in the momentum equation in the hydrostatic step, and only the increment of the non-hydrostatic pressure is calculated in non-hydrostatic step. The computational cost of these pressure correction models in the early publications was still high because a large number of vertical layers, normally 10-40, are need to achieve acceptable dispersion accuracy.

Stelling and Zijlema (2003) was the first to use the Keller-box scheme to address the issue of the classic staggered grid in which the top layer non-hydrostatic pressure was not correctly determined. In the classic staggered grid, non-hydrostatic pressure is located at the layer center, which makes it difficult to apply the boundary condition for the non-hydrostatic pressure (q) at the free surface. The Keller-box scheme replaces the derivative of q at the layer center by an average of the derivative of q at the upper and lower layer interfaces. This treatment makes it convenient to impose the free surface dynamic boundary condition for

q. The result shows that Stelling and Zijlema's model can accurately simulate deep water waves ($h/L \leq 0.5$) using only two layers. An alternative was presented by Yuan and Wu (2004). In order to obtain an accurate non-hydrostatic pressure on the top layer, they integrated the vertical velocity from the center of top layer to the free surface to get the expression of non-hydrostatic pressure. This method has been further enhanced by using high order interpolation schemes for the top layer (Badiei et al. 2008 and Young et al. 2009).

The present study is focused on developing a model that is able to simulate dispersive waves with a good accuracy of velocity profile in the water column using a minimal number of vertical layers.

Chapter 3

Method

3.1 Mathematical Formulation

3.1.1 Computational domain

A two dimensional model is developed to demonstrate the method introduced in the present study. The discretization and solution procedure of a three dimensional model is similar to that of the two-dimensional model. The computational domain is shown in Fig. (3.1). It is bounded by a free surface on top and a rigid boundary at bottom, which can be represented by $-d(x) \leq z \leq \eta(x,t)$, where, z is the vertical coordinate with an origin sitting on the still water and pointing upward, $d(x)$ is the water depth from the still water level and $\eta(x,t)$ is the surface elevation.

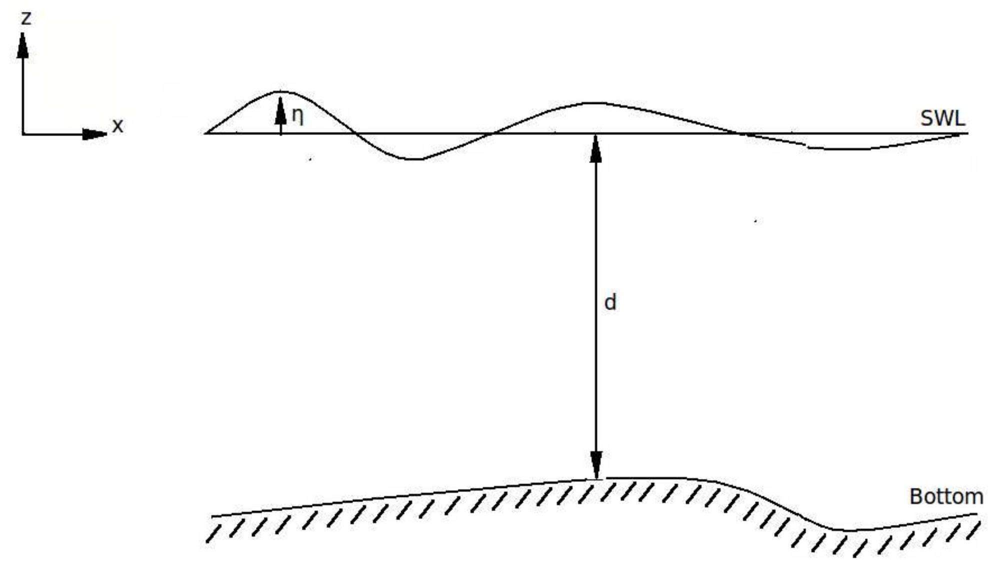

Figure 3.1: Computational domain

3.1.2 Governing equations

The governing equations used here are the incompressible Euler equations, which read:

$$\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z} = 0 \tag{3.1}$$

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial wu}{\partial z} + g\frac{\partial \eta}{\partial x} + \frac{\partial q}{\partial x} = 0 \tag{3.2}$$

$$\frac{\partial w}{\partial t} + \frac{\partial uw}{\partial x} + \frac{\partial w^2}{\partial z} + \frac{\partial q}{\partial z} = 0 \tag{3.3}$$

where u and w are velocity components in the horizontal x direction and the vertical z direction, respectively; g is gravitational acceleration; η is free surface elevation; q is non-hydrostatic pressure. Here the pressure is split into two parts. The total pressure p is represented by:

$$p = \rho g(\eta - z) + q \tag{3.4}$$

In order to get the free surface equation, we need to integrate the mass equation (3.1) from the bottom to the free surface and use the corresponding kinematic boundary conditions, which read:

$$u\frac{\partial d}{\partial x} + w = 0 \tag{3.5}$$

$$\frac{\partial \eta}{\partial t} + u\frac{\partial \eta}{\partial x} - w = 0 \tag{3.6}$$

That gives the free surface equation:

$$\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}\int_{-d}^{\eta} u dz = 0 \tag{3.7}$$

3.1.3 Boundary conditions

Boundary conditions are needed at all the boundaries of the two dimensional domain, including free surface, bottom, inflow and outflow.

1) Free surface boundary: The kinematic boundary condition (3.6) is needed here. Actually,

as we will see later, the kinematic boundary condition is not directly implemented at the free surface, because we do not have the grid point of velocity located at the free surface. However, we do use the kinematic boundary condition implicitly as we set the momentum flux at the free surface zero. The free surface boundary condition for non-hydrostatic pressure q is also needed to be implemented in the model. This condition is very important because the achievement of high dispersion accuracy relies on the accurate calculation of the non-hydrostatic pressure. Here we just set $q = 0$ at the free surface.

2) Bottom condition: Similar to the free surface, we also need the kinematic boundary condition at the bottom. The bottom is treated as a free-slip bottom without including bottom friction. As for the non-hydrostatic boundary condition, we set $\frac{\partial q}{\partial z} = 0$ in the case of flat or mild slope bottom where the vertical velocity near bottom is negligible. And a very thin bottom layer, which is determined by our unique grid, also reduces the error for this assumption.

3) Inflow boundary: At the inflow boundary, although only the grid points of u are located right at the inflow boundary, the boundary conditions for other variables are still needed when doing interpolation or solving the Poisson equation. Horizontal velocity u is always given at the inflow boundary. Vertical velocity w and non-hydrostatic pressure q are set to be zero. The horizontal derivative of surface elevation η are assumed to be zero as well. We may also apply wall boundary condition to the inflow boundary. Then it will be treated as an free-slip impermeable boundary with the variables' values set as $u = 0, \frac{\partial w}{\partial x} = 0$ and $\frac{\partial q}{\partial x} = 0$.

4) Outflow boundary: In our model, the outflow boundary is treated as the wall boundary. A spongy layer can be added in front of the outflow boundary for absorbing the wave energy. In the spongy layer, a damping coefficient function presented in Chen et al. (1999) with the form listed below is used to gradually decrease the amplitude of surface elevation and the velocities.

$$C_s = \alpha_s^{\gamma_s^{i-1}}, i = 1, 2, ...n \tag{3.8}$$

where α_s and γ_s are two free parameters. Numerical experiments suggest that $\alpha_s = 2$,$\gamma_s = 0.88 - 0.92$, and $n = 50 - 100$ lead to an efficient absorption of shortwaves.

Thus, the variables in the spongy layer are calculated by:

$$\eta_j = \eta_j / C_s, u_j = u_j / C_s, w_j = w_j / C_s \tag{3.9}$$

3.2 Numerical Discretization

A structured finite volume method is used in this model. The entire domain is divided into $N * K$ quadrilaterals, where K is the number of layers in the vertical direction and N is the number of cells in the horizontal direction. The width in the x direction is constant for every cell. So the x coordinate in this Cartesian grid at the cell centers is:

$$x_{i+1/2} = (i + 1/2)\Delta x \tag{3.10}$$

where Δx is the width of the cell.

The height of the cell varies in space and time. However, it is proportional to the total water depth at a given time and location. The sum of the height of cells in the water column is equal to the total water depth:

$$\sum_{k=1}^{K} h_{i,k} = \eta(x_i, t) + d(x_i) \tag{3.11}$$

So the z coordinate of the layer interface can be expressed by:

$$z_{i,k+1/2} = z_{i,k-1/2} + h_{i,k} \tag{3.12}$$

with:

$$z_{i,1/2} = -d(x_i), \quad z_{i,K+1/2} = \eta(x_i, t) \tag{3.13}$$

If the ratio between the layer thickness and total water depth is fixed to be f_k, then we can rewrite (3.12) to be:

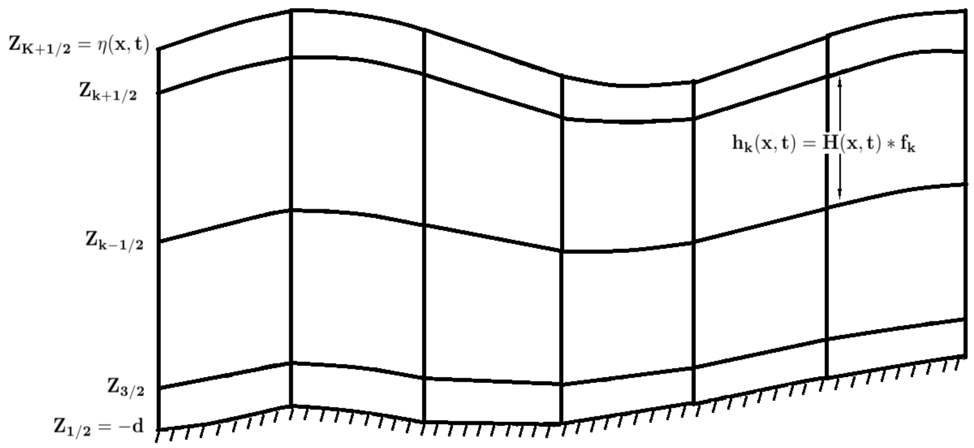

Figure 3.2: Modified staggered grid

$$z_{i,k+1/2} = z_{i,k-1/2} + f_k(\eta(x_i,t) + d(x_i)) \tag{3.14}$$

The choice of f_k can significantly influence the performance of the model. f_k may change in the x direction or be constant for every layer, but it can not be negative. In our model, f_k is determined according to the zeros of Gauss-Jacobi quadrature for the purpose of implementing Gauss-Jacobi quadrature. This will be illustrated in details later. The computational mesh is shown in Fig. (3.2).

Since the layer interfaces move with time, we need a relative vertical velocity to accurately calculate the flux passing the cells interface in the vertical direction. The formula for this relative vertical velocity $\omega_{k+1/2}$ is expressed as:

$$\omega_{k+1/2} = w(z_{k+1/2}) - \frac{\partial z_{k+1/2}}{\partial t} - u(z_{k+1/2})\frac{\partial z_{k+1/2}}{\partial x} \tag{3.15}$$

By applying the kinematic boundary condition to the expression of $\omega_{k+1/2}$ at the bottom and free surface we have:

$$\omega_{1/2} = 0, \quad \textit{and} \quad \omega_{K+1/2} = 0 \tag{3.16}$$

3.2.1 Computational grid

A modified staggered grid is proposed in this study. Before introducing the new grid, we review the classic staggered grid first. In the classic staggered grid, velocities u and w are located at the cell surface while non-hydrostatic pressure q is located at the cell center. The indices for u, w, q are $(i + 1/2, k)$, $(i, k + 1/2)$ and (i, k), respectively. η is a one dimensional variable, thus the index of η is i. In our model, all the variables on the grid are shifted in the vertical direction by half a cell. The consequence of this shift is that all the variables, which are originally located at the cell center, are relocated to the cell surface and vice versa. Now, the new indices for u, w, q are $(i + 1/2, k + 1/2)$, (i, k) and $(i, k + 1/2)$, respectively. The index for η remains the same because the variable on the grid only shifts in the vertical direction. There are two benefits of the new modified grid. One is that we now have non-hydrostatic pressure on the layer interface, therefore we can apply the boundary condition for q at the free surface directly. Another benefit is that this grid will make it more convenient to apply a Gauss-Jacobi quadrature to calculate the integration in (3.7). Both of the benefits will considerably improve the dispersion accuracy.

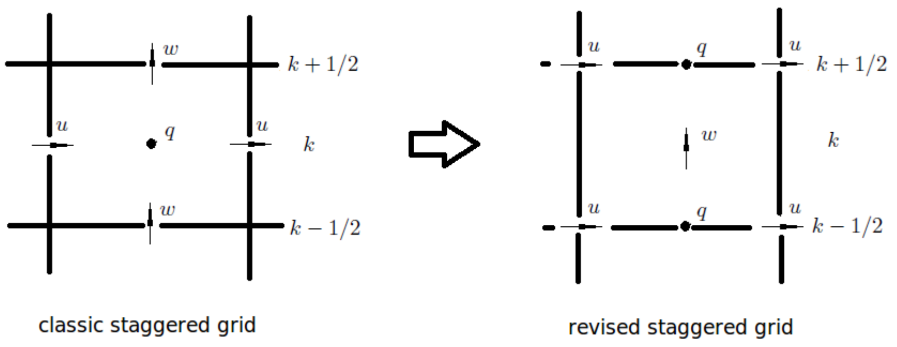

Figure 3.3: Locations of variables on grid

The detail of applying Gauss-Jacobi quadrature will be discussed next. The modified grid is illustrated in Fig. (3.3).

3.2.2 Application of Gauss-Jacobi Quadrature

Base on (3.7), it can be seen that the dispersion accuracy is determined by the integration accuracy of horizontal velocity from bottom to free surface. In a classic staggered grid, the horizontal velocity is located at the layer center. This integration is done by:

$$\int_{-d}^{\eta} u dz = \sum_{k=1}^{K} u_k h_k \tag{3.17}$$

If the velocity is located at the zeros of Gauss-Jacobi polynomial, then we can use a more accurate Gaussian-Jacobi Quadrature for the integration. The numerical result shows that this can improve the model accuracy in terms of dispersion.

The formula for Gauss-Jacobi quadrature is:

$$\int_{-1}^{1} f(t)dt = \sum_{i=1}^{n} w_i f(t_i) \tag{3.18}$$

where t_i is the zeros of Gauss-Jacobi polynomial and w_i is the weight for t_i. n is the number of the zeros and it is determined by the order of the polynomial. The detailed information about Gauss-Jacobi polynomial and the evaluation of the zeros and weights of the polynomial can be found in the Appendix B in Karniadakis and Sherwin (2005). A C++ code for generating zeros and weights of Gauss-Jacobi polynomial can be downloaded from www.nektar.info/code/page_polylib.html.

A transformation is needed to change the integration domain from [-1,1] to any given interval [a,b].

$$\int_{a}^{b} f(x)dx = m\int_{-1}^{+1} f(c+mt)dt = m\sum_{i=1}^{n} w_i f(c+mt_i) \tag{3.19}$$

where $m = \frac{1}{2}(b-a), \quad c = \frac{1}{2}(b+a)$ and $x = c+mt$.

In order to apply Gauss-Jacobi quadrature in our model, we need to place the layer interface at the zeros.This can be achieved by setting:

$$f_1 = \frac{t_1+1}{2} - 0; \quad f_k = \frac{t_k - t_{k-1}}{2}, k = 2 : K-1; \quad f_K = 1 - \frac{t_K+1}{2}; \tag{3.20}$$

Based on (3.18), we can get:

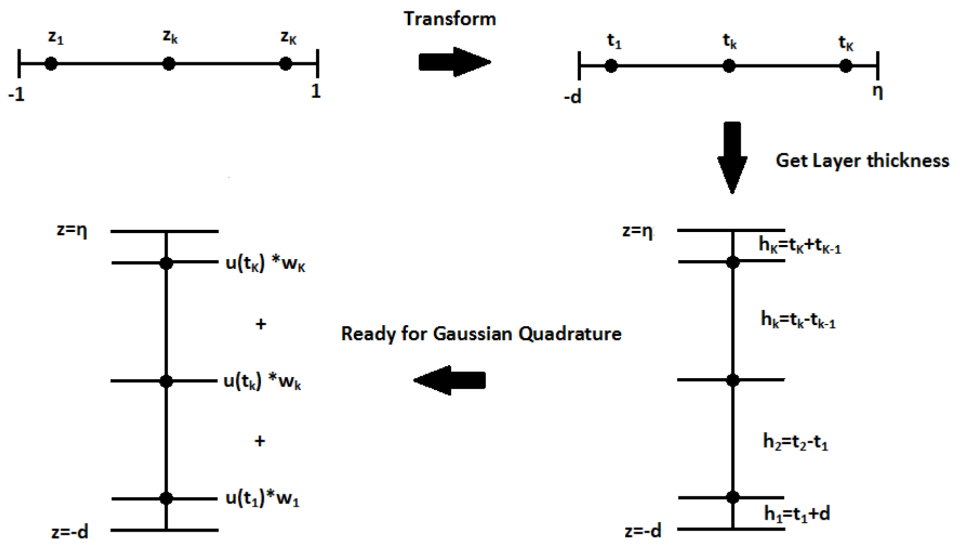

Figure 3.4: Determination of the layer thickness

$$\int_{-d}^{\eta} u dz = (\eta + d) \sum_{k=1}^{K-1} w_{k+1/2} u_{k+1/2} \tag{3.21}$$

The whole process is illustrated by Fig. (3.4).

3.2.3 Control volume

The control volume for the variables on the new grid also needs to be redefined, as shown in Fig. (3.5). The shift of the variables creates two half control volumes for u near the free surface and the bottom. The two half cells are actually very thin because of the distribution of the zeros of Gauss-Jacobi polynomial. And we don't have to solve horizontal velocity at the free surface and the bottom. Therefore, we merge these two half cells to the cells next to them.

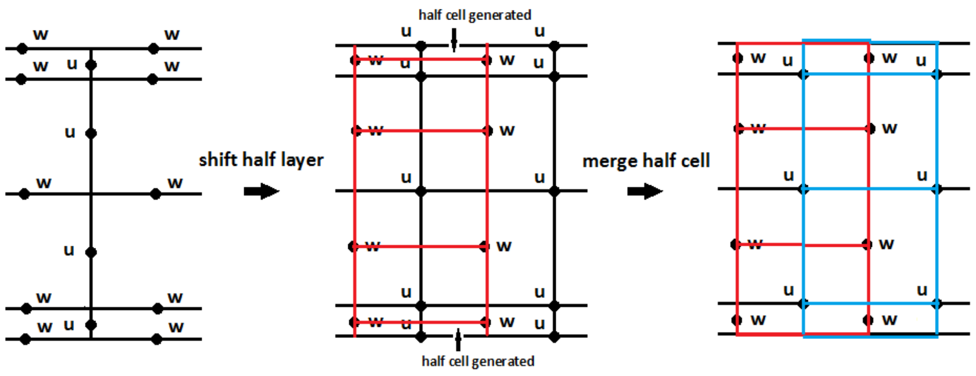

Figure 3.5: Control volume for the modified staggered grid

According to the finite volume method, velocities at the faces of a control volume are needed to calculate the flux. At the left and right boundaries of the control volume, the velocities are calculated by taking the average of the velocities at the cell centers.

$$u_{i,k+1/2} = 0.5 * (u_{i-1/2,k+1/2} + u_{i+1/2,k+1/2}), \quad w_{i+1/2,k} = 0.5 * (w_{i,k} + w_{i+1,k}) \tag{3.22}$$

At the upper and lower face of the control volume, we have:

$$u_{i+1/2,k} = 0.5 * (u_{i+1/2,k-1/2} + u_{i+1/2,k+1/2}),$$

$$w_{i,k+1/2} = (h_{i,k} * w_{i,k+1} + h_{i,k+1} * w_{i,k})/(h_{i,k} + h_{i,k+1}) \tag{3.23}$$

A weighted averaged is needed for w at cell surface because the layer thickness changes from layer to layer.

3.2.4 Spatial discretization of continuity equation and free surface equation

To discretize the continuity equation we first integrate (3.1) in the vertical direction between the center of two adjacent layers k and $k+1$.

$$\int_{z_k}^{z_{k+1}} (\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}) dz = \frac{\partial h_{k+1/2} u_{k+1/2}}{\partial x} - u \frac{\partial z}{\partial x} |_{z_k}^{z_{k+1}} + w_{k+1} - w_k = 0 \tag{3.24}$$

Here, $h_{k+1/2}$ is obtained by:

$$h_{k+1/2} = 0.5 * (h_k + h_{k+1}) \tag{3.25}$$

Then we integrate (3.24) in the horizontal direction from $x_{i-1/2}$ to $x_{i+1/2}$, which gives

$$\begin{aligned} h_{i+1/2,k+1/2} u_{i+1/2,k+1/2} - h_{i-1/2,k+1/2} u_{i-1/2,k+1/2} + u_{i,k}(z_{i+1/2,k} - z_{i-1/2,k}) \\ - u_{i,k+1}(z_{i+1/2,k+1} - z_{i-1/2,k+1}) + (w_{i,k+1} - w_{i,k}) \Delta x = 0 \end{aligned} \tag{3.26}$$

The discretized free surface equation is:

$$\frac{\partial \eta}{\partial t} + Q_{i+1/2} - Q_{i-1/2} = 0 \tag{3.27}$$

where Q is the integration determined by (3.21).

3.2.5 Spatial discretization of horizontal momentum equation

The control volume for horizontal velocity is from x_i to x_{i+1} in the x direction and from z_k to z_{k+1} in the z direction. Here we treat terms in the horizontal momentum equation over control volume separately. These terms include time derivative term, convective term, surface elevation gradient term and non-hydrostatic pressure gradient term.

Integrating the time derivative term in the vertical direction from z_k to z_{k+1} gives:

$$\int_{z_k}^{z_{k+1}} \frac{\partial u}{\partial t} dz = \frac{\partial h_{k+1/2} u_{k+1/2}}{\partial t} - u \frac{\partial z}{\partial t}|_{z_k}^{z_{k+1}} \tag{3.28}$$

Then integrating the convective term over the same vertical interval gives:

$$\int_{z_k}^{z_{k+1}} (\frac{\partial u^2}{\partial x} + \frac{\partial wu}{\partial z}) dz = \frac{\partial}{\partial x} \int_{z_k}^{z_{k+1}} u^2 dz + u_{k+1}(\omega_{k+1} + \frac{\partial z_{k+1}}{\partial t}) - u_k(\omega_k + \frac{\partial z_k}{\partial t}) \tag{3.29}$$

In the formula above, we have:

$$\int_{z_k}^{z_{k+1}} u^2 dz = h_{k+1/2} u_{k+1/2}^2 + \int_{z_k}^{z_{k+1}} (u - u_k)^2 dz \tag{3.30}$$

The integral term in the right-hand side of the equation above is due to the non-uniformities of the horizontal velocity. Here we assume it is small and negligible.

For q we have:

$$\int_{z_k}^{z_{k+1}} \frac{\partial q}{\partial x} dz = \frac{\partial}{\partial x} \int_{z_k}^{z_{k+1}} q dz - q_{k+1} \frac{\partial z_{k+1}}{\partial x} + q_k \frac{\partial z_k}{\partial x} \tag{3.31}$$

The integral on the right-hand size is approximated by:

$$\int_{z_k}^{z_{k+1}} q dz \approx \frac{1}{2} h_{k+1/2} (q_{k+1} + q_k) = h_{k+1/2} q_{k+1/2} \tag{3.32}$$

Putting all the terms together, the vertical integration of the horizontal momentum equation reads

$$\begin{aligned} \frac{\partial h_{k+1/2} u_{k+1/2}}{\partial t} + \frac{\partial h_{k+1/2} u_{k+1/2}^2}{\partial x} + u_{k+1}\omega_{k+1} - u_k \omega_k \\ + g h_{k+1/2} \frac{\partial \eta}{\partial x} + \frac{\partial h_{k+1/2} q_{k+1/2}}{\partial x} - q_{k+1} \frac{\partial z_{k+1}}{\partial x} + q_k \frac{\partial z_k}{\partial x} \end{aligned} \tag{3.33}$$

Then integrating (3.33) over interval in the x direction from x_i to x_{i+1} gives

$$\begin{aligned}\frac{\partial h_{i+1/2,k+1/2}u_{i+1/2,k+1/2}}{\partial t}\Delta x + u_{i+1,k+1/2}\phi_{i+1,k+1/2} - u_{i,k+1/2}\phi_{i,k+1/2}\\ + (u_{i+1/2,k+1}\omega_{i+1/2,k+1} - u_{i+1/2,k}\omega_{i+1/2,k})\Delta x\\ + g(\eta_{i+1} - \eta_i)h_{i+1/2,k+1/2} + h_{i+1,k+1/2}q_{i+1,k+1/2} - h_{i,k+1/2}q_{i,k+1/2}\\ - q_{i+1/2,k+1}(z_{i+1,k+1} - z_{i,k+1}) + q_{i+1/2,k}(z_{i+1,k} - z_{i,k}) = 0 \quad (3.34)\end{aligned}$$

where $\phi = hu$ is the value at the side faces of a control volume.

3.2.6 Spatial discretization of vertical momentum equation

We follow the same procedure as that for the horizontal momentum equation. The control volume for the vertical momentum equation is from $x_{i-1/2}$ to $x_{i+1/2}$ in the x direction and from $z_{k-1/2}$ to $z_{k+1/2}$ in the z direction.

After we integrate the vertical momentum equation in the vertical direction in the interval $[z_{k-1/2}, z_{k+1/2}]$, we get

$$\frac{\partial h_k w_k}{\partial t} + \frac{\partial h_k u_k w_k}{\partial x} + w_{k+1/2}\omega_{k+1/2} - w_{k-1/2}\omega_{k-1/2} + q_{k+1/2} - q_{k-1/2} = 0 \quad (3.35)$$

Then we do the integration in the x direction and obtain

$$\frac{\partial h_{i,k}w_{i,k}}{\partial t}\Delta x + h_{i+1/2,k}w_{i+1/2,k}u_{i+1/2,k} - h_{i-1/2,k}w_{i-1/2,k}u_{i-1/2,k}$$
$$+ [w_{i,k+1/2}\omega_{i,k+1/2} - w_{i,k-1/2}\omega_{i,k-1/2}]\Delta x + (q_{i,k+1/2} - q_{i,k-1/2})\Delta x = 0 \quad (3.36)$$

Notice that the time derivative terms in the discretized vertical and horizontal equation have the layer thickness in the differentiation. To get a time derivative only for the velocities, we apply the chain rule as follows:

$$\frac{\partial u}{\partial t} = \frac{1}{h}(\frac{\partial hu}{\partial t} - u\frac{\partial h}{\partial t}) \quad (3.37)$$

Based on (3.15) and (3.24), we can verify that:

$$\omega_{k+1} = \omega_k - \frac{\partial h_{k+1/2}}{\partial t} - \frac{\partial h_{k+1/2}u_{k+1/2}}{\partial x} \quad (3.38)$$

Applying (3.37) and (3.38) to (3.34), we can get the discretized equation with time derivative only for u, as follows:

$$
\begin{aligned}
h_{i+1/2,k+1/2}\frac{\partial u_{i+1/2,k+1/2}}{\partial t}\Delta x &+ (u_{i+1,k+1/2}-u_{i+1/2,k+1/2})\phi_{i+1,k+1/2}-(u_{i,k+1/2}-u_{i+1/2,k+1/2})\phi_{i,k+1/2} \\
&+ [(u_{i+1/2,k+1}-u_{i+1/2,k+1/2})\omega_{i+1/2,k+1}-(u_{i+1/2,k}-u_{i+1/2,k+1/2})\omega_{i+1/2,k}]\Delta x \\
&+ g(\eta_{i+1}-\eta_i)h_{i+1/2,k+1/2}+h_{i+1,k+1/2}q_{i+1,k+1/2}-h_{i,k+1/2}q_{i,k+1/2} \\
&- q_{i+1/2,k+1}(z_{i+1,k+1}-z_{i,k+1})+q_{i+1/2,k}(z_{i+1,k}-z_{i,k})=0 \quad (3.39)
\end{aligned}
$$

The same procedure is applied to the vertical momentum equation and the discretized equation is obtained:

$$
\begin{aligned}
h_{i,k}\frac{\partial w_{i,k}}{\partial t}\Delta x &+ h_{i+1/2,k}(w_{i+1/2,k}-w_{i,k})u_{i+1/2,k}-h_{i-1/2,k}(w_{i-1/2,k}-w_{i,k})u_{i-1/2,k} \\
&+ [(w_{i,k+1/2}-w_{i,k})\omega_{i,k+1/2}-(w_{i,k-1/2}-w_{i,k})\omega_{i,k-1/2}]\Delta x+(q_{i,k+1/2}-q_{i,k-1/2})\Delta x=0
\end{aligned}
\quad (3.40)
$$

3.3 Solution Procedure

How the solution at the time level $n+1$ is evolved from the time level n using the discretized equations derived above is discussed in this section. This is achieved by doing the time integration. A pressure correction method is applied in our model. The time integration in this method is completed in two steps, namely the hydrostatic step and the non-hydrostatic step. The spatial discretization versions of (3.1), (3.2), (3.3) and (3.7) listed below are employed to demonstrate the time integration procedure.

$$\frac{\delta u_{i,k}}{\delta x} + \frac{\delta w_{i,k}}{\delta z} = 0 \tag{3.41}$$

$$\frac{du_{i,k}}{dt} + g\frac{\delta \eta_i}{\delta x} + \frac{q_{i,k}}{\delta x} = F_u \tag{3.42}$$

$$\frac{dw_{i,k}}{dt} + \frac{\delta q_{i,k}}{\delta z} = F_w \tag{3.43}$$

$$\frac{d\eta_i}{dt} + \frac{\delta Q_i}{\delta x} = 0, \quad Q_i = \int_{-d}^{\eta} u_i dz \tag{3.44}$$

F_u and F_w represent the spacial discretization of convective terms in the horizontal momentum and vertical momentum equations, respectively. $\delta/\delta x$ and $\delta/\delta z$ are linear algebraic operators of gradients in the $x-$ and $z-$direction, respectively. Different time integration schemes are applied to different terms. An explicit time stepping scheme is employed for convective terms and a semi-implicit θ-scheme is used for both hydrostatic and non-hydrostatic pressure terms, as follows:

$$\frac{\delta u_{i,k}^{n+1}}{\delta x} + \frac{\delta w_{i,k}^{n+1}}{\delta z} = 0 \tag{3.45}$$

$$\frac{u_{i,k}^{n+1} - u_{i,k}^{n}}{\Delta t} + g(\theta\frac{\delta \eta_i^{n+1}}{\delta x} + (1-\theta)\frac{\delta \eta_i^{n}}{\delta x}) + \theta\frac{q_{i,k}^{n+1}}{\delta x} + (1-\theta)\frac{\delta q_{i,k}^{n}}{\delta x} = F_u^n \tag{3.46}$$

$$\frac{w_{i,k}^{n+1} - w_{i,k}^{n}}{\Delta t} + \theta\frac{\delta q_{i,k}^{n+1}}{\delta z} + (1-\theta)\frac{\delta q_{i,k}^{n}}{\delta z} = F_w^n \tag{3.47}$$

$$\frac{\eta_i^{n+1}-\eta_i^n}{\Delta t}+\theta\frac{\delta Q_i^{n+1}}{\delta x}+(1-\theta)\frac{\delta Q_i^n}{\delta x}=0, \quad Q_i=\int_{-d}^{\eta} u_i dz \tag{3.48}$$

The value of θ should be in interval [0.5,1] to give a stable model. The explicit treatment of convective terms makes this scheme conditionally stable. A pressure correction scheme is employed to solve (3.45)-(3.48). In the hydrostatic step of the scheme, intermediate variables u^*, w^*, η^* are calculated based on the variables at the time level n. Then in the non-hydrostatic step, a Poisson solver is constructed based on the intermediate variables and the incremental Δq is calculated by the Poisson solver. Finally, $u^{n+1}, w^{n+1}, q^{n+1}$ are all updated based on Δq. The details are discussed below.

3.3.1 Hydrostatic step

First, we find out the intermediate variables u^* and η^* using the variables at the time level n.

$$\frac{u_{i,k}^*-u_{i,k}^n}{\Delta t}+g(\theta\frac{\delta\eta^*}{\delta x}+(1-\theta)\frac{\delta\eta_i^n}{\delta x})+\frac{\delta q_{i,k}^n}{\delta x}=F_u^n \tag{3.49}$$

$$\frac{\eta_i^*-\eta_i^n}{\Delta t}+\theta\frac{\delta Q_i^*}{\delta x}+(1-\theta)\frac{\delta Q_i^n}{\delta x}=0, \quad Q_i^*=GLQ(u_{i,k}^*) \tag{3.50}$$

$GLQ(u_{i,k}^*)$ means the Gauss-Jacobi quadrature of $u_{i,k}^*$ defined before. It is easy to see that the two equations above are coupled. To solve it we have to define another variable $u_{i,k}^{**}$ which is calculated based on η_i^n, as follows:

$$\frac{u_{i,k}^{**} - u_{i,k}^{n}}{\Delta t} + g\frac{\delta \eta_i^n}{\delta x} + \frac{\delta q_{i,k}^n}{\delta x} = F_u^n \tag{3.51}$$

Subtracting (3.49) by (3.51) and introducing $\Delta\eta_i = \eta_i^* - \eta_i^n$, we can calculate u_i^* by:

$$\frac{u_{i,k}^{*} - u_{i,k}^{**}}{\Delta t} + g\theta\frac{\delta \Delta\eta_i}{\delta x} = 0 \tag{3.52}$$

Integrating (3.52) from the bottom to the free surface gives

$$Q_i^* = Q_i^{**} - g\theta\Delta t H_i^* \frac{\delta\Delta\eta_i}{\delta x}, \quad H_i^* = \eta_i^* + d_i \tag{3.53}$$

Based on (3.50) and (3.53), an equation for $\Delta\eta_i$ is constructed as follows:

$$\frac{\Delta\eta_i}{\Delta t} - g\theta^2\Delta t\frac{\delta}{\delta x}(H_i^*\frac{\delta\Delta\eta_i}{\delta x}) = -\theta\frac{\delta Q_i^{**}}{\delta x} - (1-\theta)\frac{\delta Q_i^n}{\delta x} \tag{3.54}$$

We update u_i^* and η_i^* after (3.54) is solved.

w_i^* can be solved by:

$$\frac{w_{i,k}^{*} - w_{i,k}^{n}}{\Delta t} + \frac{\delta q_{i,k}^n}{\delta z} = F_w^n \tag{3.55}$$

3.3.2 Non-hydrostatic step

In this step, the intermediate variables u^*, w^*, η^* will be updated to the time level $n+1$.

By subtracting (3.46) and (3.47) by (3.49) and (3.55), we get:

$$\frac{u_{i,k}^{n+1} - u_{i,k}^{*}}{\Delta t} + g\theta \frac{\eta_i^{n+1} - \eta_i^{*}}{\delta x} + \theta \frac{\delta \Delta q_{i,k}}{\delta x} = 0 \tag{3.56}$$

$$\frac{w_{i,k}^{n+1} - w_{i,k}^{n}}{\Delta t} + \theta \frac{\delta \Delta q_{i,k}}{\delta z} = 0 \tag{3.57}$$

where $\Delta q_{i,k} = q_{i,k}^{n+1} - q_{i,k}^{n}$. We are not going to update surface elevation in the non-hydrostatic step, which means $\eta_i^{n+1} - \eta_i^{n} = 0$. Thus (3.56) can be rewritten as:

$$\frac{u_{i,k}^{n+1} - u_{i,k}^{*}}{\Delta t} + \theta \frac{\delta \Delta q_{i,k}}{\delta x} = 0 \tag{3.58}$$

A Poisson equation can be constructed by substituting (3.57), (3.58) to (3.45).

In order to better demonstrate the construction of Poisson equation, a set of simplified indices are used here and is shown in Fig. (3.6).

The black dots are grid locations of the variables. The number and letter in the brackets are the simplified index. Their real index in the original grid is indicated by i and k. The

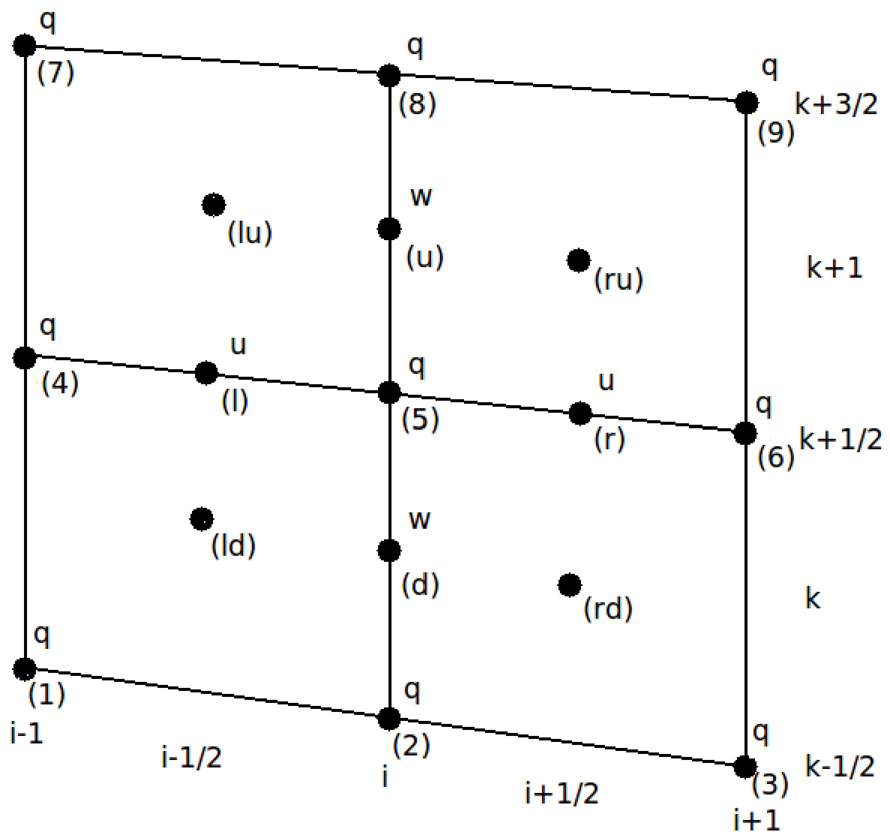

Figure 3.6: Simplified index for constructing Poisson equation

black dots with an index from 1 to 9 is where the non-hydrostatic pressure is located. Take q_6 for example, its index in the original grid is $q_{i+3/2,k+1/2}$. Index u,d,l and r means up, down, left and right, respectively. lu,ld,ru and rd are the combinations of u,d,l and r. u_l and $u_{i-1/2,k+1/2}$ represent the same grid variable. So are w_u and $w_{i,k+1}$.

With the simplified indices, (3.22) can be rewritten as:

$$h_r u_r - h_l u_l - u_u \Delta z_u + u_d \Delta z_d + (w_u - w_d)\Delta x = 0 \tag{3.59}$$

where $\Delta z_u = z_{ru} - z_{lu}, \Delta z_d = z_{rd} - z_{ld}$

The discretized form of (3.57) and (3.58) are part of (3.34) and (3.40). Applying them to u_l, u_r, w_u, w_d yields

$$\frac{u_r h_r - u_r^* h_r}{\Delta t}\Delta x + \theta(h_6 \Delta q_6 - h_5 \Delta q_5 - \Delta q_{ru} \Delta z_{ru} + \Delta q_{rd} \Delta z_{rd}) = 0 \tag{3.60}$$

$$\frac{u_l h_l - u_l^* h_l}{\Delta t}\Delta x + \theta(h_5 \Delta q_5 - h_4 \Delta q_4 - \Delta q_{lu} \Delta z_{lu} + \Delta q_{ld} \Delta z_{ld}) = 0 \tag{3.61}$$

$$\frac{w_u \Delta x - w_u^* \Delta x}{\Delta t} h_u + \theta(\Delta q_8 \Delta x - \Delta q_5 \Delta x) = 0 \tag{3.62}$$

$$\frac{w_d \Delta x - w_d^* \Delta x}{\Delta t} h_u + \theta(\Delta q_5 \Delta x - \Delta q_2 \Delta x) = 0 \tag{3.63}$$

We still need to get $u_u \Delta z_u$ and $u_d \Delta z_d$ to complete the continuity equation. They are represented by:

$$\frac{u_u h_u - u_u^* h_u}{\Delta t}\Delta x + \theta(h_{ru}\Delta q_{ru} - h_{lu}\Delta q_{lu} - \Delta q_8 \Delta z_8 + \Delta q_5 \Delta z_5) = 0 \tag{3.64}$$

$$\frac{u_d h_d - u_d^* h_d}{\Delta t}\Delta x + \theta(h_{rd}\Delta q_{rd} - h_{ld}\Delta q_{ld} - \Delta q_5 \Delta z_5 + \Delta q_2 \Delta z_2) = 0 \tag{3.65}$$

By replacing every term in (3.59) using (3.60)-(3.65), we can get an equation with only known intermediate variables and unknown non-hydrostatic pressure. Put all the intermediate variables to the right hand side:

$$RHS = u_r^* h_r - u_l^* h_l + w_u^* \Delta x + w_d^* \Delta x - u_u^* \Delta z_u + u_d^* \Delta z_d \tag{3.66}$$

$\Delta q_{ru}, \Delta q_{rd}, \Delta q_{lu}, \Delta q_{ld}$ can be determined by linear interpolation listed below:

$$\Delta q_{ru} = 0.25 * (\Delta q_5 + \Delta q_6 + \Delta q_8 + \Delta q_9) \tag{3.67}$$

$$\Delta q_{rd} = 0.25 * (\Delta q_2 + \Delta q_3 + \Delta q_5 + \Delta q_6) \tag{3.68}$$

$$\Delta q_{lu} = 0.25 * (\Delta q_4 + \Delta q_5 + \Delta q_7 + \Delta q_8) \tag{3.69}$$

$$\Delta q_{ld} = 0.25 * (\Delta q_1 + \Delta q_2 + \Delta q_4 + \Delta q_5) \tag{3.70}$$

Finally, we can get the coefficients $c1, c2, ...c9$ of $\Delta q1, \Delta q2, ...\Delta q9$, as follows:

$$c1 = -0.25\theta\Delta t(\Delta z_{ld} + h_{ld}\frac{\Delta z_d}{h_d})/\Delta x \tag{3.71}$$

$$c2 = [0.25\theta\Delta t(\Delta z_{rd} - \Delta z_{ld} + h_{rd}\frac{\Delta z_d}{h_d} - h_{ld}\frac{\Delta z_d}{h_d}) + \Delta z_2\frac{\Delta z_d}{h_d}]/\Delta x + \frac{\Delta x}{h_d} \tag{3.72}$$

$$c3 = 0.25\theta\Delta t(\Delta_{rd} + h_{rd}\frac{\Delta z_d}{h_d})/\Delta x \tag{3.73}$$

$$c4 = [0.25\theta\Delta t(\Delta z_{lu} - \Delta z_{ld} + h_{lu}\frac{\Delta z_u}{h_u} - h_{ld}\frac{\Delta z_d}{h_d} + h_4]\Delta x \tag{3.74}$$

$$\begin{aligned} c5 = [0.25\theta\Delta t(\Delta z_{rd} - \Delta z_{ru} - \Delta z_{ld} + \Delta z_{lu} + \frac{\Delta z_u}{h_u}h_{lu} - \frac{\Delta z_u}{h_u}h_{ru} + \frac{\Delta z_d}{h_d}h_{rd} - \frac{\Delta z_d}{h_d}h_{ld}) \\ - 2h_5 - \frac{\Delta z_u}{h_u}\Delta z_5 - \frac{\Delta z_d}{h_d}\Delta z_5]/\Delta x - \frac{\Delta x}{h_u} - \frac{\Delta x}{h_d} \end{aligned} \tag{3.75}$$

$$c6 = [0.25\theta\Delta t(\Delta z_{rd} - \Delta z_{ru} - h_{ru}\frac{\Delta z_u}{h_u} + h_{rd}\frac{\Delta z_d}{h_d}) + h_6]/\Delta x \tag{3.76}$$

$$c7 = 0.25\theta\Delta t(\Delta z_{lu} + \frac{\Delta z_u}{h_u}h_{lu})/\Delta x \tag{3.77}$$

$$c8 = [0.25\theta\Delta t(\Delta z_{lu} - \Delta z_{ru} - \frac{\Delta z_u}{h_u}h_{ru} + \frac{\Delta z_u}{h_u}h_{lu}) + \frac{\Delta z_u}{h_u}\Delta z_8]/\Delta x + \frac{\Delta x}{h_u} \tag{3.78}$$

$$c9 = -0.25\theta\Delta t(\Delta z_{ru} + \frac{\Delta z_u}{h_u}h_{ru})/\Delta x \tag{3.79}$$

The constructed Poisson equation is solved by a software library called HYPRE, which is a high performance preconditioners and solvers for the solution of large, sparse linear systems of equations on massively parallel computers.The details of implementing HYPRE in our model is discussed next.

3.3.3 Implementation of HYPRE

HYPRE is a library of high performance preconditioners and solvers for the solution of large, sparse linear systems of equations on massively parallel computers. More information about HYPRE can be found in its website(http://acts.nersc.gov/hypre/). HYPRE offers different conceptual interface for different applications. They are listed below:

- Structured-Grid System Interface (Struct): This interface is for structured grid with fixed stencil pattern of non-zeros at each grid point. This interface supports only a single unknown per grid point.

- Semi-Structured-Grid System Interface (SStruct): This interface is for applications whose grids are mostly structured, but with some unstructured features. Examples include block-structured grids, composite grids in structured adaptive mesh refinement (AMR) applications, and overset grids. This interface supports multiple unknowns per cell.

- Finite Element Interface (FEI): This interface is for finite element application. The interface mirrors typical finite element data structures, including element stiffness matrices.

- Linear-Algebraic System Interface (IJ): This is the traditional linear-algebraic interface which can be used where other interfaces are not appropriate. It requires more work on the user's part.

In our case, the Struct interface is used. There are five basic steps involved in setting up the linear system and solving it:

1) set up the grid

2) set up the stencil

3) set up the matrix

4) set up the right-hand-side and solution vector

5) solve the linear system

Each step is explained in detail with a piece of code as follows.

1) Set up the grid

```
HYPRE_StructGridCreate(MPI_COMM_WORLD, 2, &grid);

int ilower[2]=0,1, iupper[2]=Nc-1,Nr-2;

HYPRE_StructGridSetExtents(grid, ilower, iupper);

HYPRE_StructGridAssemble(grid);
```

The grid is described via a global index space, i.e., via integer singles in 1D, tuples in 2D, or triples in 3D. The basic component of the grid is a box: a collection of abstract cell-centered indices in index space, described by its "lower" and "upper" corner indices. The Create() routine creates an empty 2D grid object. The SetExtents() routine adds a new box defined by ilower and iupper. The Assemble() routine assembles the grid and makes the grid ready to use.

2) Set up the Struct Stencil

```
HYPRE_StructStencilCreate(2, 9, &stencil);

int entry;

int offsets[9][2] = {{0,0}, {-1,0}, {1,0}, {0,-1}, {0,1},{-1,-1},{1,-1},{-
1,1},{1,1}};

for (entry = 0; entry ≤ 8; entry++)

HYPRE_StructStencilSetElement(stencil, entry, offsets[entry]);
```

The geometry of discretization stencil is described by an array of indices, each representing a relative offset from any given point on the grid. The offsets array represent a 9-point stencil. The (0,0) entry represents the "center" coefficient, and is the 0th stencil entry. The (0,-1) entry represents the "south" coefficient, and is the 3rd stencil entry, and so on. The Create() routine creates an empty 2D, 9-point stencil object. The SetElement() routine defines the geometry of the stencil and assigns the stencil numbers for each of the stencil entries.

3) Set up the Struct Matrix

```
HYPRE_StructMatrixCreate(MPI_COMM_WORLD, grid, stencil, &A);

HYPRE_StructMatrixInitialize(A);

int ilower[2]=0,1, iupper[2]=Nc-1,Nr-2;

int stencil_indices[9] = {0,1,2,3,4,5,6,7,8};

int nentries = 9;

int nvalues = N*9;

double values[N*9];

HYPRE_StructMatrixSetBoxValues(A,     ilower,     iupper,     nen-

tries,stencil_indices, values);

HYPRE_StructMatrixAssemble(A);
```

The matrix is set up in terms of the grid and stencil objects described above. The coefficients associated with each stencil entry will typically vary from grid point to grid point, and their calculations are based on the derivation given in the last section. The Create() routine creates an empty matrix object. The initialize() routine indicates that the matrix coefficients are ready to be set. The SetBoxValues() routine sets the matrix coefficients for some set of stencil entries over the grid points in some box. The Assemble() routine assembles the matrix and makes it ready to use.

4) Set up the struct right-hand-side and solution vector

```
HYPRE_StructVectorCreate(MPI_COMM_WORLD, grid, &b);

HYPRE_StructVectorCreate(MPI_COMM_WORLD, grid, &x);

HYPRE_StructVectorInitialize(b);

HYPRE_StructVectorInitialize(x);

int ilower[2]=0,1, iupper[2]=Nc-1,Nr-2;

double values[N];

HYPRE_StructVectorSetBoxValues(b, ilower, iupper, values);

HYPRE_StructVectorSetBoxValues(x, ilower, iupper, values);

HYPRE_StructVectorAssemble(b);

HYPRE_StructVectorAssemble(x);
```

The right-hand-side and solution vector are set up similar to the matrix set up described above. The main difference is that there is no stencil. The right-hand-side vector is assigned based on the derivation in the previous section, and the solution vector is assigned zeros. The Initialize() routine indicates that the vector coefficients are ready to be set. This routine follows the same rules as its corresponding Matrix routine. The SetBoxValues() routine sets the vector coefficients over the grid points in some box, and again, follows the same rules as its corresponding Matrix routine. The Assemble() routine assembles the vector and makes them ready to use.

5) Solve the linear system

```
HYPRE_StructPCGCreate(MPI_COMM_WORLD, &solver);
HYPRE_StructPCGSetTol(solver, 1.0e-15);
HYPRE_StructPCGSetMaxIter(solver, 1000);
HYPRE_StructPCGSetPrintLevel(solver, 100);
HYPRE_StructPCGSetup(solver, A, b, x);
HYPRE_StructPCGSolve(solver, A, b, x);
```

The Create() routine creates a solver object. The SetTol() routine sets the tolerance of the solver. SetMaxIter() routine sets the maximum iteration times. SetPrintLevel() routine sets the frequency to print out the results. PCGSetup() routine prepares to solve the system using the PCG solver. The PCGSolve() routine solves the linear system.

3.3.4 Overall solution procedure

The overall solution procedure is listed below:

1. Assign value to variables at the time level n including η^n, u^n, w^n, q^n.
2. Calculate u^{**} by (3.51).
3. Solve (3.54) to obtain $\Delta\eta$.
4. Get intermediate variables u^* by (3.52) and η^* by $\eta^* = \eta^n + \Delta\eta$.
5. Set $\eta^{n+1} = \eta^*$.
6. Calculate w^* by (3.55).

7. Solve the system of linear equations constructed by (3.66) and (3.71)-(3.79) to get Δq.

8. Update the non-hydrostatic pressure by $q_{n+1} = q_n + \Delta q$.

9. Calculate u_{n+1} and w_{n+1} by (3.58) and (3.57).

10. Update ω by (3.15).

Chapter 4

Results

Several test cases have been carried out to verify the code and demonstrate the performance. These are standing waves in a closed channel, linear progressive waves on a flat bottom and laboratory experiments of waves over an uneven bottom.

4.1 Standing waves

In this test case, a sinusoidal wave is released in a closed channel at time zero, as shown in Fig. (4.1). Without considering diffusion, there should be no energy loss and the time series of the water level at a fixed location should be a sinusoidal function with constant amplitude. In our test case, the length of the basin is 20m and the depth is 10m. The initial surface profile is given by

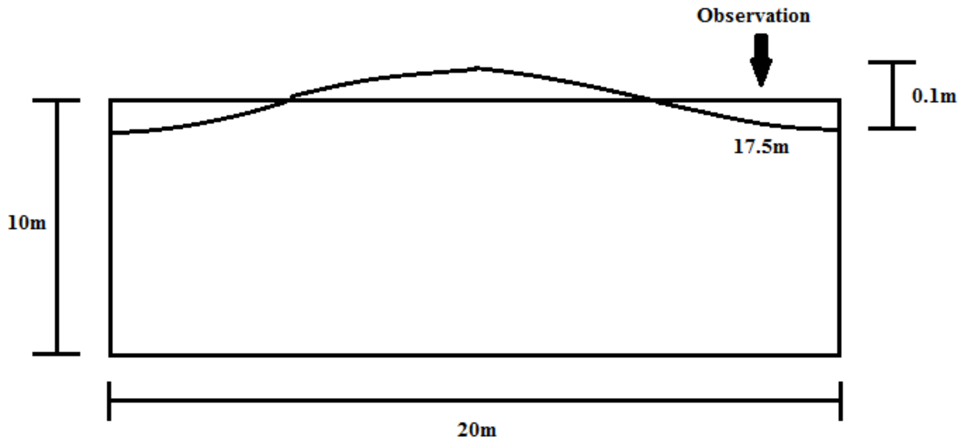

Figure 4.1: Basin for standing wave test

$$\eta = a\, cos(kx) \tag{4.1}$$

where $k = 2\pi/L$ is the wave number. $L = 20m$ is the wavelength. The wave amplitude is $a = 0.1m$. Based on the linear dispersion relationship, we have the wave period as:

$$T = \sqrt{\frac{2\pi L}{g tanh(kd)}} \tag{4.2}$$

In this case, when $L = 20m, d = 10.0m, g = 9.81m/s^2$, the period is 3.5858s.

The time series of water level at $x = 17.5m$ is used for the comparison between analytical solution and the result from our two-layer model. Grid spacing and time-step is $\Delta x = 1.0m$ and $\Delta t = 0.025$, respectively. The comparison is shown in Fig. (4.2). Good agreement is found.

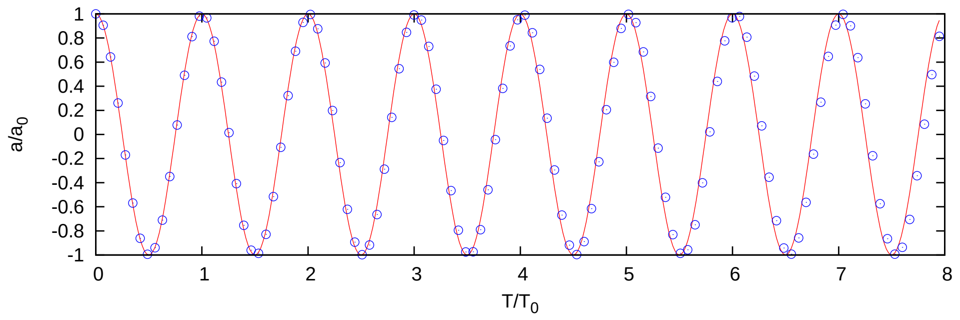

Figure 4.2: Comparison between analytical solution (solid line) and numerical result (circle)

4.2 Velocity profile and dispersion accuracy

A progressive wave test case is used to demonstrate the improvement in dispersion accuracy achieved by applying the modified staggered grid and Gauss-Jacobi quadrature. A progressive wave train of 0.01m with a period of 3.5858s is generated at the left end of a channel with water depth d. The water depth will vary while the wave period remains unchanged. When $d = 10m$, the velocity and non-hydrostatic pressure profile comparisons between analytical solution and numerical results at $x = 33.3m$ are shown in Fig. (4.3). The grid spacing and time spacing are $\Delta x = 1.0m, \Delta t = 0.025s$. Nine layers are used here. Excellent agreement with the analytical solution of velocities and pressure is found.

In order to demonstrate the dispersion accuracy improvement of the modified grid along with the Gauss-Jacobi quadrature, we compare our result to a modified grid with constant layer thickness along with the integration represented by (3.17). Six layers are used for both models and d is changed to vary the wave dispersion. The comparison is shown below in Fig. (4.4). The figure shows an obvious improvement.

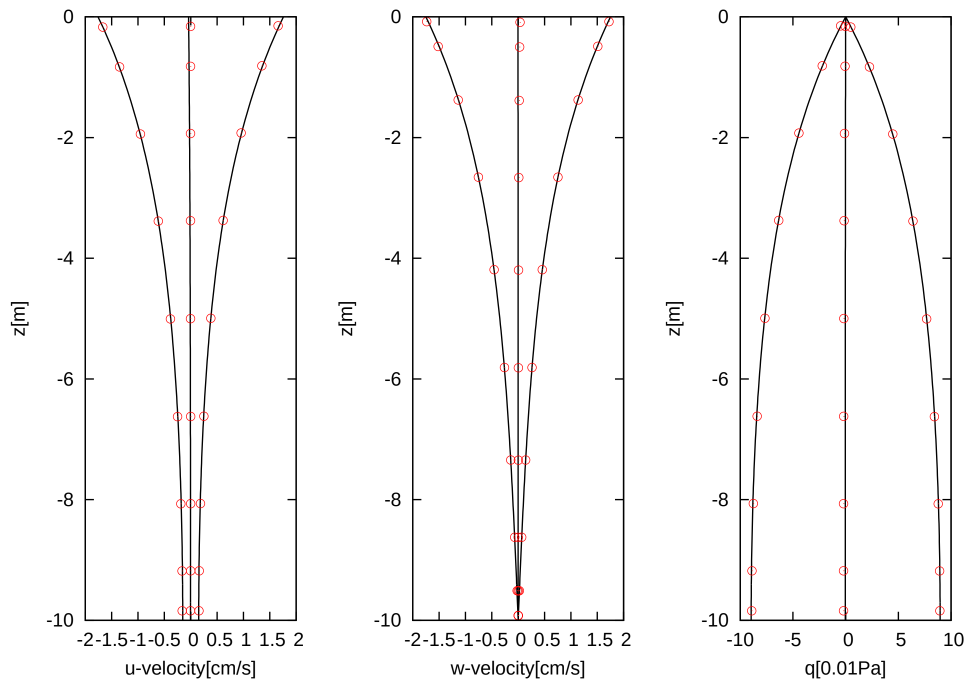

Figure 4.3: Comparison of velocity and non-hydrostatic pressure profiles between analytical solution (solid line) and numerical result (circle)

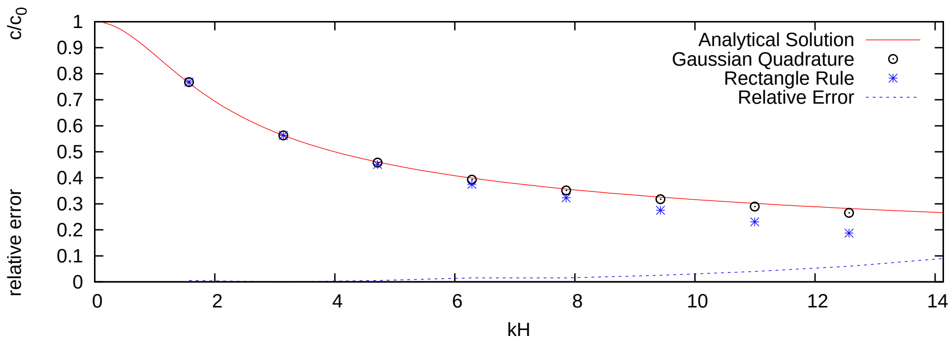

Figure 4.4: Comparison of dispersion accuracy between rectangular integration and Gauss-Jacobi quadrature

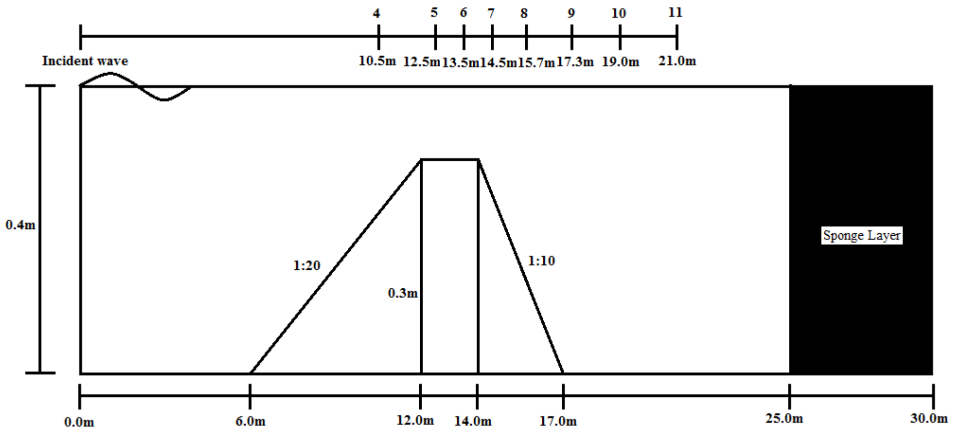

Figure 4.5: Geometry of submerged bar and location of wave gauges

4.3 Submerged bar test case

This test case is a laboratory experiment of a non-linear wave train propagating over a submerged bar done by Beji and Battjes (1993). The experiment was performed in a flume with a length of 30m and a water depth of 0.4m. As shown in Fig. (4.5). A sinusoidal wave train with an amplitude of 0.01m and a period of 2.02s is generated from the left end. The time series of water level at different stations is recorded. The comparison between the numerical result and experimental data at Station 4 to Station 11 is shown in the Fig. (4.6) and Fig. (4.7). Generally good agreement is found.

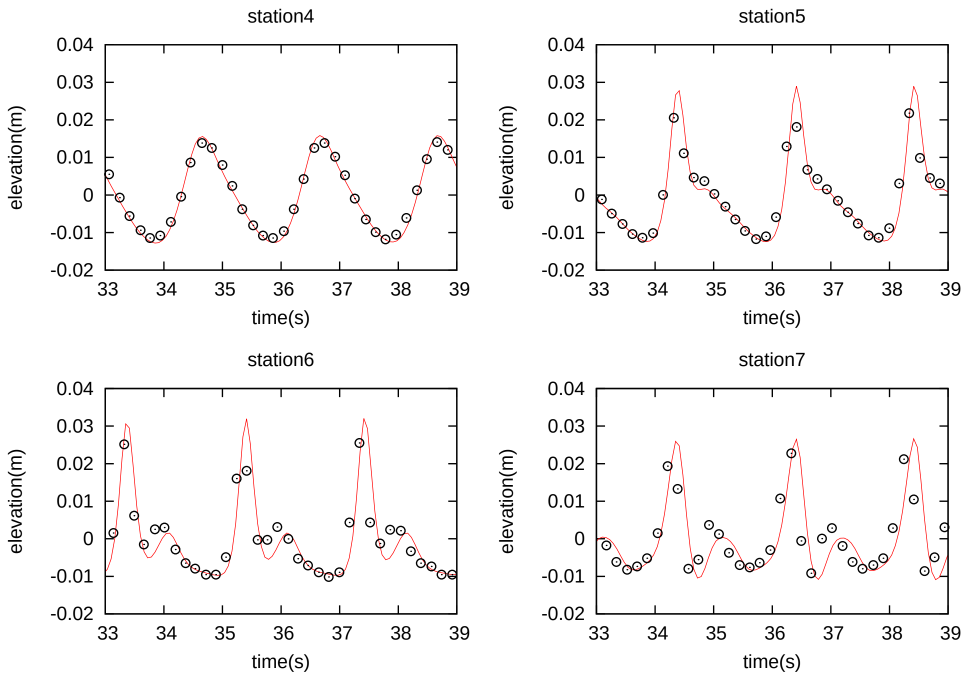

Figure 4.6: Comparison between analytical solution (circle) and numerical result (solid line) from Station 4 to Station 7

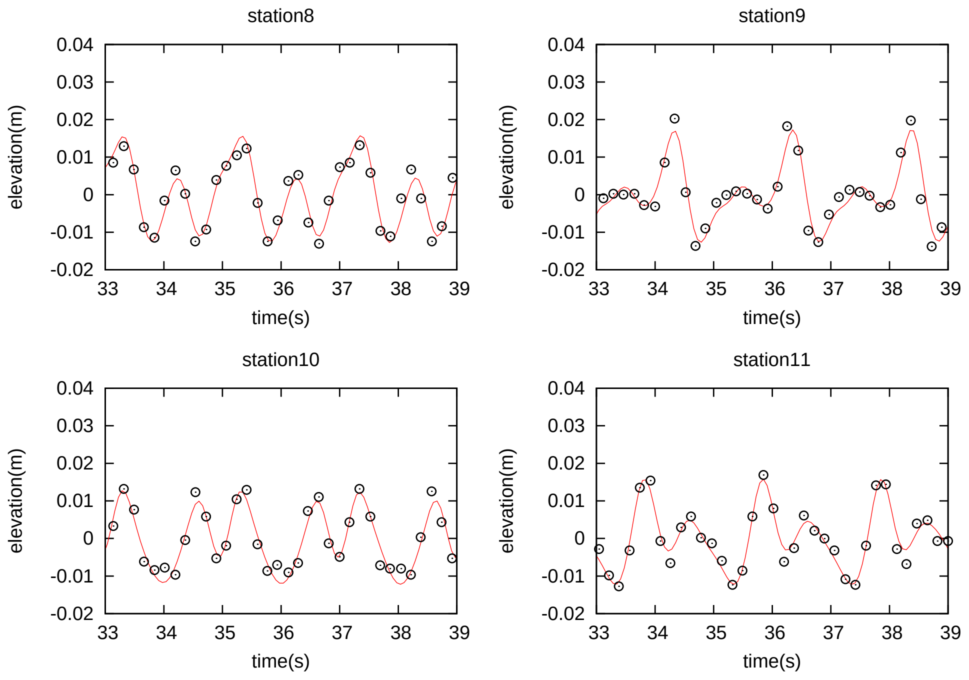

Figure 4.7: Comparison between analytical solution (circle) and numerical result (solid line) from Station 8 to Station 11

Chapter 5

Conclusion

5.1 Summary and conclusion

A two-dimensional finite-volume Euler solver for free-surface flow on a modified staggered grid using a Gaussian Quadrature method has been developed and verified. The pressure correction method which splits the pressure into the hydrostatic part and the non-hydrostatic part is used to enhance the dispersion accuracy of the model. In this method, a hydrostatic step is followed by a non-hydrostatic step with a Poisson solver that calculates the increment of the non-hydrostatic pressure. The time integrations of both the hydrostatic and non-hydrostatic pressure are treated in a semi-implicit way. This solver employs a vertically boundary-conforming coordinate system. A modified staggered grid with the non-hydrostatic pressure on the cell surface enables the direct use of the dynamic free-surface boundary condition. The layer thickness of the computational mesh is determined by the zeros of Gauss-Jacobi polynomial. This facilitates the use of the Gauss-Jacobi quadrature to calculate the discharge rate. The solution of Poisson equation relies on HYPRE, a library for solving large, sparse linear systems of equations on massively parallel computers. The numerical results show that the modified grid with Gauss-Jacobi quadrature enables the modeling of deep water waves ($h/L \leq 0.5$) using only two vertical layers with good accuracy. Furthermore, the model is able to generate accurate vertical profiles of velocities and non-hydrostatic pressure using a limited number of vertical layers.

5.2 Future work

1) Extending the two-dimensional code to three dimensions. Although a two dimensional model is good to demonstrate the method and performance, a three-dimensional model is needed for more complicated cases and practical applications. There should be no major difficulties when extending the two-dimensional code into a three-dimensional one. However, how to solve a three-dimensional Poisson equation efficiently is a question that needs to be thought of.

2) Parallelizing the code using openMP and MPI. Some two-dimensional applications have already been computationally intensive, like highly dispersive progressive waves in a long channel, not to mention a three dimensional code with a massive Poisson equation to solve. With openMP, it is easy to obtain several times speedup on a multiple-core computer by

adding a few extra lines into the code. An MPI version of the three dimensional code would be a better option. For both openMP and MPI, the bottleneck will be the Poisson equation. Therefore, most of the effort should be devoted to finding an efficient parallel algorithm to solve the Poisson equation.

REFERENCES

Badiei, P., Namin, M.M., Ahmadi, A., 2008. A three-dimensional non-hydrostatic vertical boundary fitted model for free-surface flows. International Journal for Numerical Methods in Fluids 56, 607-627.

Beji, S., Battjes, J.A., 1993. Experimental investigations of wave propagation over a bar. Coastal Engineering 19(1,2), 151-162.

Berkhoff, J.C.W., 1972. Computation of combined refraction and diffraction. Proceedings of the 13th International Coastal Engineering Conference(ASCE), 471-490.

Berkhoff, J.C.W., 1976. Mathematical models for simple harmonic linear water waves; wave refraction and diffraction. PhD thesis, Delft Technical University of Technology.

Casulli, V., Stelling. G.S., 1998. Numerical simulation of 3D quasi-hydrostatic, free-surface flows. Journal of Hydraulic Engineering (ASCE) 124, 678-686.

Casulli V., 1999. A semi-implicit finite difference method fro non-hydrostatic, free-surface flows. International Journal for Numerical Methods in Fluids 30, 425-440.

Chen, Q., Madsen, P.A., Basco, D.R., 1999. Current effects on nonlinear interactions of shallow water waves. Journal of Waterway, Port, Coastal, and Ocean Engineering 125, 176-186.

Eckart, C., 1952. The propagation of gravity waves from deep to shallow water. Circular 20, National Bureau of Standards, 165-173.

Hodges, B.R., Street, R.L., 1999. On simulation of turbulent nonlinear free-surface flows. Journal of Computational Physics 151 (2), 425-457.

Iafrati, A., Campana, E.F., 2003. A domain decomposition approach to compute wave breaking (wave-breaking flows). International Journal for Numerical Methods in Fluids 41 (4), 419-445.

Karniadakis, G.E., Sherwin, S., 2005. Spectral/hp Element Methods for Computational

Fluid Dynamics. New York: Oxford University Press.

Kocyigit, M.B., Falconer, R.A., Lin, B., 2002. Three-dimensional numerical modeling of free surface flows with non-hydrostatic pressure. International Journal for Numerical Methods in Fluids 40, 1145-1162.

Liu, P.L.F. 1994 Model equations for wave propagation from deep to shallow water. Advances in Coastal and Ocean Engineering 1, 125-158.

Madsen, P.A., Murray, R., Sorensen, O.R., 1991. A new form of the Boussinesq equations with improved linear dispersion characteristics. Coastal Engineering 15, 371-388.

Madsen, P.A., Bingham, H.B., Schaffer, H.A., 2003. Boussinesq-type formulations for fully nonlinear and extremely dispersive water waves: derivation and analysis. Proceedings of Royal Society of London Series A-Mathematical Physical and Engineering Sciences 495 (2033), 1075-1104.

Ng, C.O., Kot, S.C., 1992. Computations of water impact on a 2-dimensional flat-bottom body with a volume-of-fluid method. Ocean Engineering 19 (4), 337-383.

Nielsen, K.B., Mayer, S., 2004. Numerical prediction of green water incidents. Ocean Engineering 31 (3-4), 363-399.

Nwogu, O., 1993. Alternative form of Boussinesq equations for near-shore wave propagation. Journal of Waterway, Port, Coastal and Ocean Engineering 119, 618-638.

Park, J.C., Kim, M.H., Miyata, H., 1999. Fully nonlinear free-surface simulations by a 3D viscous numerical wave tank. International Journal for Numerical Methods in Fluids 29 (6), 685-703.

Peregrine, D.H., 1967. Long Waves on a beach. Journal of Fluid Mechanics 27, 815-827.

Shen, Y.M., Ng, C.O., Zheng, Y.H., 2004. Simulation of wave propagation over a submerged bar using the VOF method with a two-equation K-=epsilon turbulence modeling. Ocean Engineering 31 (1), 87-95.

Stelling, G.S., Busnelli, M.M., 2001. Numerical simulation of the vertical structure of discontinuous flows. International Journal for Numerical Methods in Fluids 30, 425-440.

Stelling, G.S., Zijlema, M., 2003. An accurate and efficient finite-difference algorithm for non-hydrostatic free-surface flow with application to wave propagation. International Journal for Numerical Methods in Fluids 43, 1-23.

Wei, G., Kirby, J.T., Grilli, S.T. and Subramanya, R., 1995. A fully nonlinear Boussinesq model for surface waves. Part I Highly nonlinear unsteady waves. Journal of Fluid

Mechanics 294, 71-92.

Wu, T.Y., 2001. A unified theory for modeling water waves. Advances in Applied Mechanics 37, 1-88.

Young, C.C., Wu, C.H., Liu, W.C., Kuo, J.T., 2009. A higher-order non-hydrostatic σ model for simulating non-linear refraction-diffraction of water waves. Coastal Engineering 56, 919-930.

Yuan, H., Wu, C.H., 2004. A two-dimensional vertical non-hydrostatic σ model with an implicit method for free-surface flows. International Journal for Numerical Methods in Fluids 44(8), 811-835.

Yue, W.S., Lin, C.L., Patel, V.C., 2004. Numerical simulation of unsteady multidimensional free surface motions by level set method. International Journal for Numerical Methods in Fluids 42 (8), 853-884.

Stansby, P.k., Zhou, J.G., 1998. Shallow-water flow solver with non-hydrostatic pressure: 2D vertical plane problems. International Journal for Numerical Methods in Fluids 28, 514-563.

Zhou, J.G., Stansby, P.K., 1999. An arbitrary Lagrangian-Eulerian σ (ALES) model with non-hydrostatic pressure for shallow water flows. Computer Methods in Applied Mechanics and Engineering 178, 199-214.

Zijlema, M., Stelling, G.S., 2005. Further experiences with computing non-hydrostatic free-surface flows involving water waves. International Journal for Numerical Methods in Fluids 48, 169-197

Zijlema, M., Stelling, G.S., 2008. Efficient computation of surf zone waves using the non-linear shallow water equations with non-hydrostatic pressure. Coastal. Engineering 55, 780-790.

APPENDIX A: USERS' MANUAL FOR THE CODE

A.1 Program flow chart

The code was written using C++. The libraries used include Boost multi_array and HYPRE. The flow chart is shown in Fig. (A.1).

A.2 Class and function descriptions

1) Classes and class functions

GF1D: one-dimensional grid function. It is used to represent one-dimensional grid functions such as the surface elevation and bottom elevation. The private variables include "row" and "gf". "row" is the size of the grid function. "gf" is an array to store the grid function.

GF1D::plotM: output grid function to a file that can be plotted by matlab.

GF1D::plotG: output grid function to a file that can be plotted by Gnuplot.

GF1D::showpoint: output an element of grid function with the index i.

GF1D::showgrid: output grid function to screen.

GF1D::input: input an element of grid function with the index i.

GF1D::getpoint: return an element of grid function with the index i.

GF1D::getrow: return the size of the grid function.

GF1D::getgf: return the grid function.

GF2D: two-dimensional gird function. It is used to represent two-dimensional grid functions such as velocities and the non-hydrostatic pressure. The private variables include "row", "column" and "gf". "row" is the number of rows of the grid function. "column" is the number of columns of the gird function. ”gf” is a two-dimensional array to store the grid function.

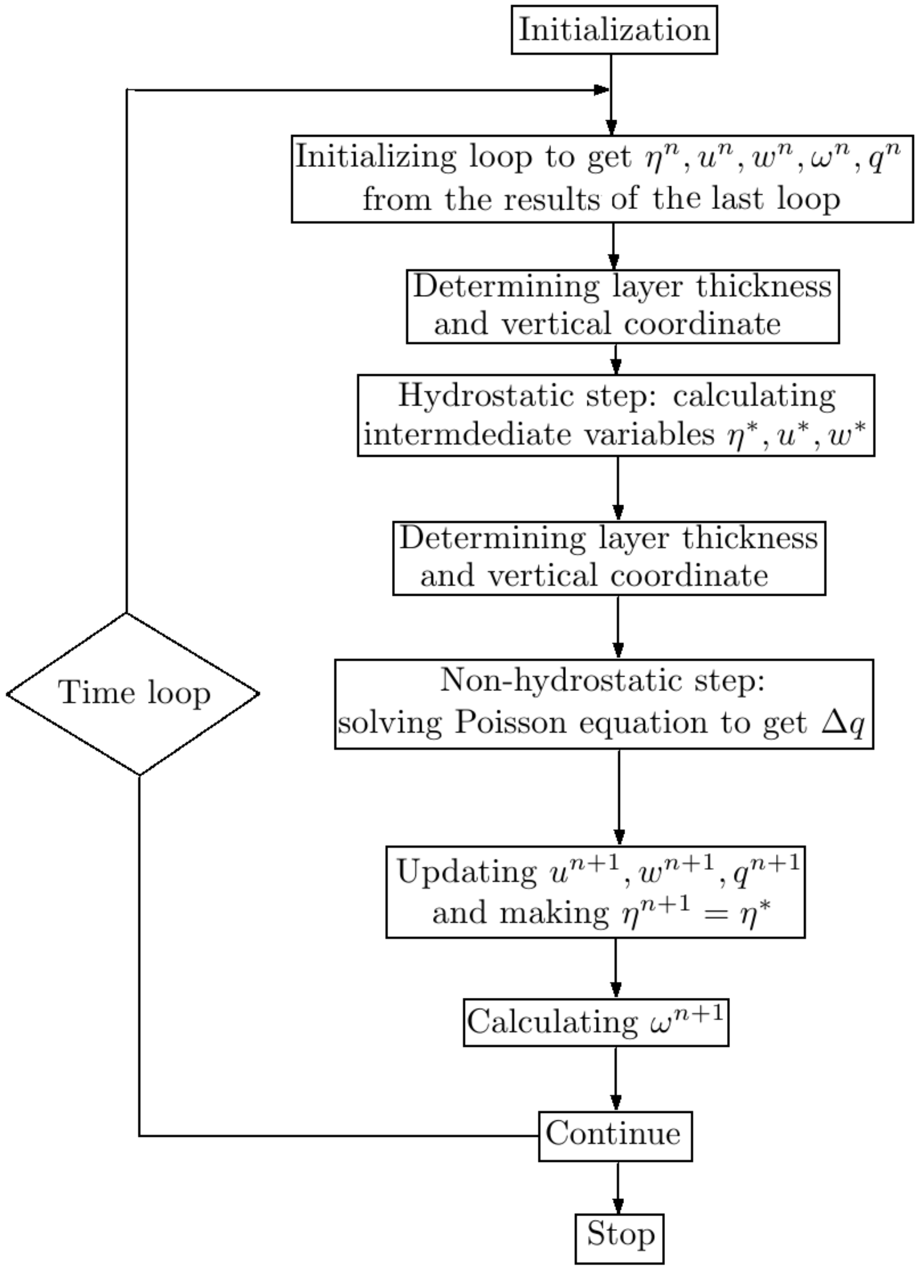
Initialization
Initializing loop to get $\eta^n, u^n, w^n, \omega^n, q^n$ from the results of the last loop
Determining layer thickness and vertical coordinate
Hydrostatic step: calculating intermdediate variables η^*, u^*, w^*
Determining layer thickness and vertical coordinate
Non-hydrostatic step: solving Poisson equation to get Δq
Updating $u^{n+1}, w^{n+1}, q^{n+1}$ and making $\eta^{n+1} = \eta^*$
Calculating ω^{n+1}
Continue
Stop
Time loop

Figure A.1: Flow chart

GF2D::plotM: output grid function to a file that can be plotted by matlab.

GF2D::plotG: output grid function to a file that can be plotted by Gnuplot.

GF2D::showpoint: output an element of grid function with the index i, j.

GF2D::showgrid: output grid function to screen.

GF2D::input: input an element to grid function with the index i, j.

GF2D::getpoint: return an element of grid function with the index i, j.

GF2D::getrow: return the number of rows of the grid function.

GF2D::getcolumn: return the number of columns of the grid function.

GF2D::getgf: return the grid function.

2) Functions

zwgj: calculate the zeros and weights of Gauss-Jacobi polynomial. It calls functions "jacobz", "jacobd" and "gammaF" during its calculation.

Initialization: assign the initial conditions to surface elevation, bottom elevation, velocities and the non-hydrostatic pressure.

Gethandz: calculate the layer thickness and z coordinates.

GetQ: calculate the discharge.

GetH: calculate the total water depth.

TriSolver: solve the tridiagonal matrix equation.

GetMVar: calculate the intermediate variables including surface elevation and velocities. This function implements the hydrostatic step.

PoissonSolver: calculate the non-hydrostatic pressure increment and update the intermediate variables.This function implements the non-hydrostatic step.

Getomega: calculate the relative vertical velocity.

ProgressiveUniform: generate progressive waves with a uniform horizontal velocity distribution at inflow boundary. This function is used in the submerged bar test case.

ProgressiveAnalytical: generate linear progressive wave using the analytical horizontal velocity distribution at inflow boundary.

AddSpongy: add a spongy layer in front of outflow boundary.

A.3 Permanent variables

blineco: bathymetry at $x_{j\pm1/2}$.

eta: surface elevation plus still water level at x_j. Time level is $n+1$.

eta_p: surface elevation plus still water level at x_j. Time level is n.

eta_m: intermediate surface elevation plus still water level at x_j.

elevation: surface elevation at x_j. Time level is $n+1$.

q: non-hydrostatic pressure. It locates at $(i \pm 1/2, j)$. Time level is $n+1$.

q_p: non-hydrostatic pressure. It locates at $(i \pm 1/2, j)$. Time level is n.

q_m: intermediate non-hydrostatic pressure. It locates at $(i \pm 1/2, j)$.

checkdq: the increment of non-hydrostatic pressure calculated in non-hydrostatic step.

uco: horizontal velocity. It locates at $(i \pm 1/2, j \pm 1/2)$. Time level is $n+1$.

uco_p: horizontal velocity. It locates at $(i \pm 1/2, j \pm 1/2)$. Time level is n.

uco_m: intermediate horizontal velocity. It locates at $(i \pm 1/2, j \pm 1/2)$.

wcc: vertical velocity. It locates at (i, j). Time level is $n+1$.

wcc_p: vertical velocity. It locates at (i, j). Time level is n.

wcc_m: intermediate vertical velocity. It locates at (i, j).

omega: relative vertical velocity. It locates at (i, j). Time level is $n+1$.

omega_p: relative vertical velocity. It locates at (i, j). Time level is n.

omega_m: intermediate relative vertical velocity. It locates at (i, j).

zco: vertical coordinate at $(i \pm 1/2, j \pm 1/2)$. Time level is $n+1$.

zco_p: vertical coordinate at $(i \pm 1/2, j \pm 1/2)$. Time level is n.

zco_m: intermediate vertical coordinate at $(i \pm 1/2, j \pm 1/2)$.

zcc: vertical coordinate at (i, j). Time level is $n+1$.

zcc_p: vertical coordinate at (i, j). Time level is n.

zcc_m: intermediate vertical coordinate at (i, j). Time level is n.

hco: layer thickness at $x_{j\pm1/2}$. Time level is $n+1$.

hco_p: layer thickness at $x_{j\pm1/2}$. Time level is n.

hco_m: intermediate layer thickness at $x_{j\pm1/2}$.

hs: layer thickness at x_j. Time level is $n+1$.

hs_p: layer thickness at x_j. Time level is n.

hs_m: intermediate layer thickness at x_j.

hcointer: the average of the layer thickness of two adjacent layers. It locates at $x_{j\pm1/2}$. Time level is $n+1$.

hcointer_p: the average of the layer thickness of two adjacent layers. It locates at $x_{j\pm1/2}$. Time level is n.

hcointer_m: intermediate the average of the layer thickness of two adjacent layers. It locates at $x_{j\pm1/2}$.

hsinter: the average of the layer thickness of two adjacent layers. It locates at x_j. Time level is $n+1$.

hsinter_p: the average of the layer thickness of two adjacent layers. It locates at x_j. Time level is n.

hsinter_m: intermediate the average of the layer thickness of two adjacent layers. It locates at x_j.

zeros: the zeros of Gauss-Jacobi polynomial.

weights: the accordingly weights of Gauss-Jacobi polynomial

blinterpoc: the extrapolation coefficient to get the horizontal velocity in the center of the top and the bottom layer.

A.4 Installation and compilation

Our code relies on HYPRE to solve the Poisson equation. So first we need to install and compile HYPRE. Before installing HYPRE, we have to make sure that mpi and blas are installed on the computer. Then we can install HYPRE by executing "./configure" and "make" in command line under "src" folder in HYPRE. After that, we can copy our source code to "examples" folder and use the make file provided by HYPRE in that folder to compile our code with HYPRE. We need to install multi_array in Boost before we compile our code. Finally, the executable file can be executed using command "mpirun -np $(np)

./$(executable_file)". Where np is the number of processors. np is 1 for single-processor application.

A.5 Input and output

Input includes initial condition and inflow boundary condition. They can be set in Initialization, ProgressiveUniform and ProgressiveAnalytical. The methods in classes GF1D and GF2D, like plotG, plotM, showgrid and getpoint, can be used to output the grid function into files or to screen.

VITA

Qi Fan was born in 1982, in Ezhou, Hubei, China. He received his Bachelor of Electrical Engineering from Wuhan University, China in June 2003. He entered the Department of Civil and Environmental Engineering (Engineering Science) at Louisiana State University in spring 2008 and has been working on numerical modeling of water waves and coastal hydrodynamics. He expects to earn a Master of Science in Engineering Science in December 2011 and then pursue a doctorate.