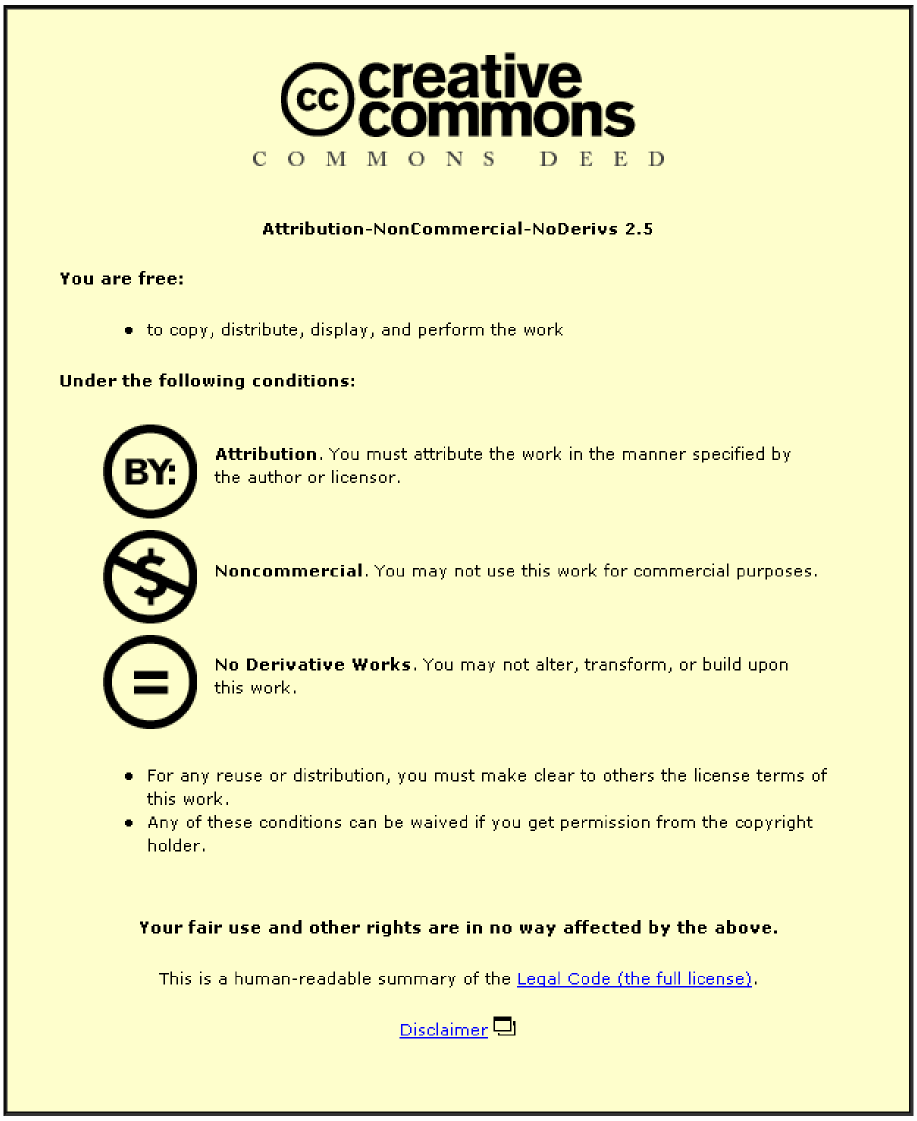


Object-Oriented Hyperbolic Solver on 2D-Unstructured Meshes Applied to the Shallow Water Equations

Atmanand Jha

Department of Civil and Building Engineering

Loughborough University

A Thesis Submitted for the Degree of

Doctor of Philosophy [DRAFT COPY]

April 2006

I would like to dedicate this thesis to ......

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Motivation

Fluid dynamics, like other physical sciences, is divided into theoretical and experimental branches. However, computational fluid dynamics (CFD) is third branch of Fluid dynamics, which has aspects of both the previous two branches. CFD is a *supplement* rather than a *replacement* to the experiment or theory. It turns a computer into a virtual laboratory, providing insight, foresight, return on investment and cost savings[1]. This work is a step toward an approach that realise a new and effective way of developing these CFD models.

A common practice in CFD, seen over the last four decades, is to take existing modelling codes, written in the procedural languages, and create *ad hoc* assemblies to add extra features to it. This may be valuable in the short term, but it leads to great duplication of effort and does not represent a long term solution [Harvey *et al.* (2002)]. In this work a framework and a model is presented to overcome this problems is by using object-oriented technology[2]. This model is named *Riemann2D*, which is:

- simple, flexible and robust compared to the traditional models;

[1]Computers have, over the half century, developed to unprecedented levels of performance at an ever decreasing cost.

[2]Object-oriented technology (OOT) comprises area of object-oriented approach (OOA), object-oriented design (OOD) and object-oriented programming (OOP). [Booch (1991), Booch (1993), Gamma *et al.* (1995), Dubois (1996), Larman (2004), Horstmann (2005) and Bennett *et al.* (2005)]

- reusable and extensible, so that the existing model can be extended without duplicating the previous work; and
- self-explanatory to help others for an easy adoption.

1.2 The Vision

The vision which the *Riemann2D* development represents is a step towards developing a generic framework to solve hyperbolic problems, which can be implemented by a new piece of code (extended model) to solve different hyperbolic problems. The object-oriented design of the *Riemann2D* is influenced by the desire to develop a model with following features:

- Any number of developers/users should be able to implement the *Riemann2D* for solving various hyperbolic problems.
- Developers should be able to overwrite, if required, the functionalities of the *Riemann2D* for their individual needs, without making any change to the generic part of the model.
- New developers/users with less knowledge of object-oriented technology or the background of the generic model should also be able to to adopt it for their own purpose.
- The system should facilitate multiple developers/users to simultaneously work and contribute to the development of one model.

1.3 Problem Selection

1.3.1 Hyperbolic Problems

Hyperbolic problems have key interest in this work due to their applications in a broad spectrum of disciplines where wave motion is important: shallow water problems, gas dynamics, acoustics, elastodynamics, optics, geophysics, and biomechanics, to name a few [see Toro (1999) and LeVeque (2002)].

Many single-purpose one-, two- or three-dimensional models have been developed in the past (see Chapter 2) to solve the problems arising in the above mentioned disciplines. However, this work uses object-oriented technology to unify the common features required by these models and develops a generic (multi-purpose) hyperbolic solver[1].

Objective of studying and developing a generic solver is to:

1. identify the common features of various problems (as unified by the *hyperbolic theory*), which will lead to better understanding of its theory and greater applicability of these techniques later to new problems; and

2. make it possible to implement the generic solver for solving a wide variety of hyperbolic problems simply by providing the case-specific extension through appropriate piece of code. These extensions will require less effort than developing a single-purpose individual model[2].

1.3.2 Shallow Water Equations

Historically, many of the fundamental ideas of hyperbolic partial differential equations were first developed for the special case of compressible gas dynamics using the *Euler's equations*, for applications in aerodynamics, astrophysics and detonation waves. Over the last two decades the numerical methods developed for solving Euler's equations have been used to study the simpler equations such as advection equation, Burger's equation, and the shallow water equations.

In this work shallow water equations is selected for the extension of *Riemann2D* , because:

1. it provide many challenging problems [Toro (2001)], which give an opportunity for further development of the numerical techniques.

2. it demonstrates the implementation procedure of the generic model, which would help in developing extensions for other hyperbolic systems.

1.4 Thesis Presentation

This work has two main components:

[1]Only two-dimensional models have been studied in this work

[2]Implementation of generic solver by shallow water model is shown in this work.

1. **Theory of the Problem:** It comprises of the theory of hyperbolic problems, high-resolution methods and shallow water equations. Mathematical expressions are an important component to these theories, which are fed to the model.

2. **Computer Technology:** As stated in the previous sections, the object-oriented technology is used for the *Riemann2D* development. The design of the *Riemann2D* is very much influenced by the theory of the problem.

Both of these components are inter-dependent and equally important. On one hand studying the theory of the problem is a science, whereas, on the other hand model design is more of an art than science. Therefore, due to the multi-disciplinary nature of this work, organisation of thesis has been a challenging task.

The chapters in this thesis is based on the above two components. Wherein, chapters 4, 5 & 6 present the theory, and chapters 2 & 7 presents the design of the model. To relate the theory with the object-oriented design, gray comment boxes (see sample box in Figure 1.1) are used in the chapters 4, 5 and 6 to serves two purposes:

1. It helps to relate the theory with the object-oriented design.

2. It separate the comments from the theory, keeping the flow of the chapter.

Figure 1.1: A sample object-oriented design comments box used in the chapters 3,4 and 5.

The details of these chapters and appendices are as following:

- **Chapter 1** *i.e.,* this chapter.

- **Chapter 2** presents **literature review** on the relevant methods of modelling hyperbolic and shallow water problems. Initially, this chapter describes the fundamental of PDEs and then selection of numerical method. The sections of this chapter provides the brief background on the chapters 4, 5, 6 and 7 *i.e.,* the hyperbolic problem, high-resolution methods, shallow water equations and object-oriented modelling.

- **Chapter 3** provides description of the **object-oriented approach** of the *Riemann2D* development. It also present the basics to the model design, which helps to further understand relationship of the theory and the model design. The detailed model design is then presented in chapter 7.
- **Chapter 4** provides discussion of the **hyperbolic systems**. Initially it presents the formulation of the one- and two-dimensional hyperbolic problem, followed by discussion on Riemann problem and its solution. After this, the chapter discusses the selection of mesh, control volume, solution for these control volumes and stability criteria of the solution. At the end it presents a generic and efficient approach to solve the boundary conditions.
- **Chapter 5** provides discussion on the **high resolution methods** with theory behind them. Continued from the pervious chapter on hyperbolic system, it present the traditional numerical methods followed by the high-resolution methods used in *Riemann2D*.
- **Chapter 6** presents the theory of the **shallow water equations**. It shows the direction for extending the generic solver. Apart from discussing the basics of the shallow water equations, it also presents the hyperbolic characteristics of the shallow water problems.
- **Chapter 7** presents the **object-oriented design and development** of *Riemann2d*. This chapter can be seen as continuation of chapter 2, where the object-oriented approach to the model development is already given.
- **Chapter 8** presents the results and discussion, when ***Riemann2D* is applied to the shallow water equations**. A series of test cases are presented to performs the model verification and validation. The tests results presented here are compared with the numerical, analytical and experimental results.
- **Chapter 9** presents **summary** of this work. Finally, future research possibilities are sketched.

CHAPTER 2

Literature Review

2.1 Introduction

This chapter presents a review of the work done in the field of hyperbolic problems, high resolution methods and shallow water equations followed by a brief review of various work on object-oriented modelling.

The classification of sections in this chapter reflects the division of the chapters of this thesis (as discussed in section 1.4). The importance of this chapter is that:

- it helps to outline the work done in the related fields; and
- separate these contents from the relevant chapters, so that the immediate problems can be addressed.

2.2 Hyperbolic Problems

This section reviews the literature to present the partial differential equations (PDEs), hyperbolic PDEs (or problems) and their importance, numerical methods to solve hyperbolic problems, and how the finite volume methods have been used for solving hyperbolic problems.

2.2.1 Partial Differential Equations

The study of partial differential equations (PDEs) started in the 18th century with the work of Euler, d'Alembert, Lagrange and Laplace. PDEs are central tool in the description of mechanics of continua and more generally, as the principal mode of analytical study of models in the physical science [Brezis (1988)]. The analysis of physical models has remained to the present day one of the fundamental concerns of the development of PDEs. Beginning in the middle of the 19th century, particularly with the work of Riemann, PDEs also became an essential tool in other branches of mathematics. Ames (1992) identified three basic types of physical problems that lead to PDEs, *i.e.*, equilibrium, eigenvalue and propagation problems, which corresponds to elliptic, parabolic and hyperbolic type PDEs respectively(see Figure 2.1 and table 2.1).

A second-order PDEs can be written as:

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F = 0. \tag{2.1}$$

Based on the matrix of coefficients of the second order terms in the above equation 2.1 *i.e.*, $Z \equiv \begin{bmatrix} A & B \\ B & C \end{bmatrix}$, PDEs are classified as shown in the table 2.1.

Table 2.1: Classification of PDEs based on the coefficients of the second order terms in equation 2.1.

Type	Condition	Example
Elliptic	$det(Z) > 0$	Laplace's equation
Parabolic	$det(Z) = 0$	Diffusion/Schrdinger equation
Hyperbolic	$det(Z) < 0$	Wave equation

In the Figure 2.1 three type of PDEs based on the propagation of information are shown, in which the domain is divided into:

- **Domain of Dependence**: For a point **P**, it is defined as the region of the solution domain, upon which the solution at a point **P**, $f(x_p, y_p) = f_p$ depends. In other words, f_p depends on everything that has happened in domain of dependence.
- **Domain of Influence**: For a point **P**, it is defined as the region of the solution domain in which the solution at point $f(x, y)$ depends on the solution at **P**, f_p In other words, f_p influences the solution at all points in the range of influence.

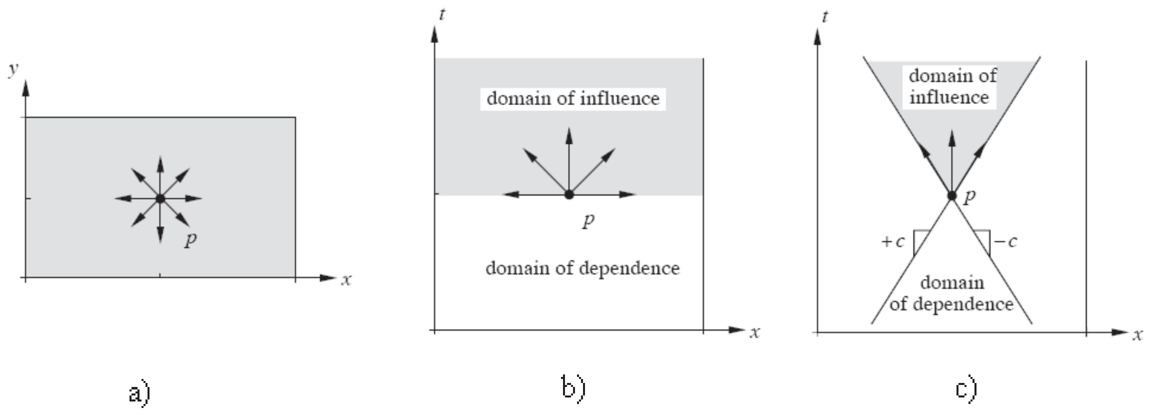


Figure 2.1: Information propagation along characteristics of the wave equation(a) elliptic (b) parabolic (c) hyperbolic PDE [Recktenwald (2004)].

Generally, elliptic equations have boundary conditions which are specified around a closed boundary (Figure 2.1(a)). This has domain of influence in all the directions and the domain of dependence is all of the the boundary. Usually all the derivatives are with respect to spatial variables, such as in Laplace's or Poisson's Equation. Parabolic (Figure 2.1(b)) and hyperbolic equations (Figure 2.1(c)), by contrast, depend on time and have at least one open boundary. The boundary conditions for at least one variable, usually time, are specified initially and the system is integrated indefinitely. Thus, the diffusion equation and the wave equation contain a time variable and there is a set of initial conditions at a particular time. These properties are, of course, related to the fact that an ellipse is a closed object, whereas hyperbola and parabola are open.

2.2.2 Solutions of Partial Differential Equations

Generally the number of solutions satisfying a PDE are very large and the functions may be completely different from one another. However, when applying PDEs to model a physical system, other conditions are imposed, enabling a unique solution to be found. For example, the solution $\mathbb{U}$ may assume specific values on the boundary of the domain, and if, time dependent, have a condition specified at $t = 0$; these are the so called *boundary conditions* and *initial conditions*. This system of equations consisting of PDEs together with some initial and boundary values is **properly posed**[Harpham (1997)] if it has the following properties:

1. **Existence:** For any sufficiently smooth data functions (say from the initial conditions) there is a solution $\mathbb{U}$ of the system of equations.

2. **Uniqueness:** There is at most one function $\mathbb{U}$ satisfying the system of equations, for given data.

3. **Continuity:** The solutions $\mathbb{U}$ corresponding to data which differ by small amounts in the appropriate norms, also differ by small amounts. (Continuity will normally imply uniqueness.)

2.2.3 Hyperbolic PDEs

Hyperbolic PDEs, as discussed in the above section, arise in a broad spectrum of disciplines where wave motion or advective transport is important: These areas include gas dynamics, acoustics, elastodyanmics, optics, geophysics and biomechanics. Historically, many of the fundamental ideas were first developed for the special case of compressible gas dynamics (the Euler equations), for applications in aerodynamics, astrophysics, detonation waves, and related fields, where shock wave arise. The study of simpler equations such as the advective equations, Burgers' equations and the shallow water equations have played an important role in the development of these methods. Often the model problem for testing methods has been the application to the Euler equation. This is also reflected in many of the texts on these methods. Nordstro and Gong (2006) states that these hyperbolic equations involved in modeling still remain a computational challenge both for academia and industry.

For a one-dimensional (in space) wave equation (hyperbolic PDE)

$$u_{tt} = cu_{xx} \qquad 0 \leq x \leq L, \quad t \geq 0, \tag{2.2}$$

with boundary and initial conditions

$$\begin{array}{ll} u(t,0) = u_0 & u(t,L) = u_L \\ u(0,x) = f(x) & \frac{\partial u}{\partial t}\Big|_{t=0} = g(x), \end{array} \tag{2.3}$$

where $u = u(t,x)$ and $\sqrt{c}$ is the wave speed, the solution can be represented as a field in the semi-infinite strip, as shown in Figure 2.1. The state of the solution at time t and position x influences the solution at all other points in the forward wedge of space time labelled the domain of influence. This wedge is defined by two characteristics with slope $\pm c$.

2.3 Hyperbolic Systems

In the hyperbolic systems of PDEs, the problems we consider are generally time-dependent (see section 2.2.1), so that the solution depends on time as well as one or more spatial variables. In one space dimension, a homogeneous first order system of PDEs in x and t has the form

$$u_t(x,t) + Au_x(x,t) = 0 \tag{2.4}$$

in the simplest constant-coefficient linear case. Here $u : \mathbb{R} \times \mathbb{R} = \mathbb{R}^m$ is a vector with m components representing the unknown functions (pressure, velocity, *etc.*) that are to be determined, and A is the constant $m \times m$ real matrix. In order for this type of problem to be *hyperbolic*, the matrix A must satisfy certain properties.

System 2.4 is said to be hyperbolic at a point (x,t), if matrix A has m real eigenvalues $\lambda_1, \lambda_2, ..., \lambda_m$ and a corresponding set of m linearly independent right eigenvectors $K^{(1)}, K^{(2)}, ..., K^{(m)}$. The system is said to be strictly hyperbolic if the eigenvalues λ_i are all distinct [Toro (1999)].

The eigenvalues λ_i of a matrix A are the solution of the characteristic polynomial

$$|A - \lambda_i I| = \det(A - \lambda_i I) = 0, \tag{2.5}$$

where I is the identity matrix. The eigenvalues of the coefficient matrix A of a system of this form, equation 2.5, are also called as *eigenvalues of the system*. Physically these eigenvalues represent speeds of propagation of information. Whereas, the eigenvector of a matrix A corresponding to an eigenvalue λ_i of A is a vector, $K^{(i)} = \left[K_1^{(i)}, K_2^{(i)}, ..., K_m^{(i)}\right]^T$ satisfying $AK^{(i)} = \lambda_i K^{(i)}$.

The simplest case is the constant-coefficient *scalar* problem, in which $m = 1$ and the matrix A reduces to a scalar value. This problem is hyperbolic provided the scalar A is real. This simple equation can model either advective transport or wave motion, depending on the context. For more detail refer LeVeque (2002).

Since the PDEs continue to hold away from discontinuities, one possible approach is to combine a standard finite volume method in smooth regions with some explicit procedure for tracking the location of discontinuities. This is the numerical analogue of the mathematical approach in which the PDEs are supplemented by jump conditions across discontinuities. This approach is often called as *shock tracking* or *front tracking*. In two-dimensions the discontinuities typically lie along curves or the surface of the cell/element/grid and this makes it more complicated. Moreover in realistic problems there may be many such surfaces that interact in complicated ways as time evolves. This approach can be found in the work by Glimm *et al.* (1981), Davis (1992), Grove (1994), LeVeque and Shyue (1995), and Mao (2000).

However, in this work a different approach is adopted, known as *shock capturing*, where the goal is to capture discontinuities in the solution automatically, without explicitly tracking them. The success of this method lies on implicitly incorporating the correct jump condition, reducing the smearing to a minimum, and no introduction of the non-physical oscillation near the discontinuities. The development of high-resolution shock-capturing schemes has a long history (see, the classical references [Godunov (1959), Harten (1983), vanLeer (1979) and Yee (1987)] or the recent textbooks [Godlewski and Raviart (1996), Kroner (1997), LeVeque (1992), and LeVeque (2002)]. The main reason for using shock-capturing methods is that the high-resolution finite volume methods based on Riemann solutions often perform well and are much simpler to implement than shock-tracking methods.

2.3.1 Conservation Laws

This work is concerned with an important class of homogeneous hyperbolic equations called *conservation laws*. In physics, a conservation law states that a particular measurable property of an isolated physical system does not change as the system evolves. The simplest form of a conservation law is a one-dimension PDE

$$u_t(x,t) + f(u(x,t))_x = 0, \tag{2.6}$$

where $f(u)$ is the *flux function*. This form (equation 2.6) is also know as the *differential form* of the conservation laws. Rewriting 2.6 in the quasi-linear form

$$u_t + f'(u)u_x = 0 \tag{2.7}$$

suggests that the equation is hyperbolic if the flux Jacobian matrix $f'(u)$ satisfies the conditions previously given for the matrix A (see equation 2.4). In fact the linear problem in equation 2.4 is a conservation law with the linear flux problem $f(u) = Au$.

but many physical problems give rise to *nonlinear conservation laws* in which $f(u)$ is a non-linear function of u.

The principal feature of nonlinear hyperbolic conservation laws, demonstrated in the physical phenomenon of the breaking of waves, is the breakdown of classical solutions and the development of discontinuities that propagate as shock waves (*e.g.*, propagating phase boundaries, fluid interfaces, gravitational waves, *etc.*), which play a dominant role in multiple areas of physics: astrophysics, cosmology, dynamics of (solid-solid) material interfaces, multiphase (liquid-vapour) flows, combustion theory, *etc.* In recent years, major progress has been made in both the theoretical and numerical aspects of the field, while the number of applications has highly increased. Dafermos *et al.* (2003) states that the challenge for mathematicians is to comprehend the properties of these non-linear waves and their relationships with the dynamics of many physical phenomena.

2.3.2 Numerical Methods for Hyperbolic PDEs

There are many methods available to solve PDEs, of which the four commonly used methods, applied to hyperbolic PDEs, are method of characteristics (MOC), finite element method (FEM), finite difference method (FDM) and finite volume method (FVM).

Method Of Characteristics (MOC) is an analysis-based method, which involves the transformation of the PDEs to ordinary differential equations (ODEs) along the characteristics. These transformed ODEs are then solved numerically using explicit or implicit schemes. Classical work using these methods can be seen in the literature by Abbott (1966), Simpson (1967), Abbott and Verwey (1970) and Edenhofer and Schmitz (1981).

The application of MOC is limited to general cases, since spatial variability, slope, surface roughness, and infiltration pattern cannot be adequately characterized. Most practical applications do not have analytical solutions, therefore, analysis-based methods, like the MOC are not very useful for such problems.

Consequently, the governing equations, which are in continuous forms, are transformed into discrete forms, which then result in series of algebraic equations that can be solved with the computer. The solution of these discrete algebraic equations represents an approximation of the continuous problem, and several methods have been developed to find the most appropriate discrete representation of the actual continuous equation, like FEM, FDM and FVM.

The **Finite Element Method** (FEM) involves the discretization of the system into a series of sub-domains (usually triangular or quadrilateral), called finite elements, connected at a discrete number of nodal points. Thereafter, each of the dependent state variables is approximated in terms of the unknown values and the known *shape function* at the nodal points. Some of the work using these methods can be seen in the literature/books by Lee *et al.* (1987), Cockburn *et al.* (1989), Schwab (1998), Berzins (2001) and Sheu *et al.* (2003). The most attractive feature of the FEM is its ability to handle complex geometries (and boundaries) with relative ease.

A common limitation of the method is that every improvement in the predictive ability implies a corresponding increase in number of nodal points to be handled, which consequently increases the number of points for solution. This always results in increased time for efficient computations, particularly when the variable surface properties and time-varying rainfall intended in this study are used.

One of the most frequently used methods is the **finite difference method** (FDM). Some work using this method can be seen in the literature by Courant *et al.* (1952), Roe (1981), Smith (1985), Carpenter *et al.* (1993) and Rasulov *et al.* (2004). There are three basic steps in the application of this method to differential equations [Singh (1996)]:

1. The continuous solution domain is discretized and replaced by a grid point called the finite-difference mesh.

2. Then the continuous derivatives of the differential equation are replaced by finite differences on the grid points, thus the solution equations, their variables and coefficients are established at all grid points (nodes).

3. In the final step, for each node, all the equations are solved using the values of the dependent variable given by the initial and boundary conditions. This set of solutions is then used as initial and boundary conditions when the solution at the next time step is desired.

The most attractive feature of FDM is that it can be very easy to implement. However, the main problem with the FDM is that it is not flexible enough to deal with the irregular boundaries.

Finite volume method (FVM) is more flexible than finite difference schemes to treat complicated geometry and adaptivity [Xing and Shu (2006)]. It involves the discretization of the continuous equation into a number of finite volumes. In each of these finite volumes, the integral equations are applied to obtain the exact conservation within each element (also known as cell or grid). It is particularly useful in cases like hydrodynamic modelling, where the equation is solved based on the principle of conservation of momentum. By the discretization of the integral form of the conservation equation, the mass and momentum remain conserved. The resulting expression in the FVM solution appears similar to FDM depending on the techniques applied. As such, it is often considered as a FDM applied to the differential conservative form of the conservation law in arbitrary coordinates [Ajayi (2004)]. The method can be applied also using an unstructured grid system as FEM, but will generally require less computational effort than FEM.

The main advantage of FVM is that it combines the simplicity of FDM with the geometric flexibility of FEM [Mingham and Causon (1998)]. Lomax *et al.* (1999) stated that FVM have become popular in CFD as a result, primarily, of two advantages:

1. They ensure that the discretization is conservative, *i.e.*, mass, momentum, and energy are conserved in a discrete sense. While this property can usually be obtained using a finite difference formulation, it is obtained naturally from a finite volume formulation as well; and

2. Finite volume methods do not require a coordinate transformation in order to be applied on irregular meshes. As a result, they can be applied on unstructured meshes consisting of arbitrary polyhedra in three dimensions or arbitrary polygons in two dimensions. This increased capability can be used to great advantage in generating grids about arbitrary geometries.

2.3.3 Selection of Method

One of the characteristics of hyperbolic equations is that, unlike elliptic and parabolic, they admit discontinuous solutions (See chapter 5) *i.e.*, the dependent variables may be discontinuous. Even if the initial and boundary conditions are smooth, a discontinuity may develop in the solution [Liggett (1994)]. These discontinuities, often called *shock* *waves*, lead to computational difficulties and one of the subjects in this work is to develop a good understanding of such approximate solutions.

The basic characteristics that makes finite volume method (FVM) suitable for the applications in the hyperbolic problems is that at any mesh density level transported quantities are fully conserved. Hirsch (1990) stated that FVM are based on the integral form of the conservation equations, thus a scheme in conservation form can easily be constructed to capture shock waves (shock-capturing property). Classical finite difference methods, which are very similar to the finite volume methods in 1D cases and in which derivatives are approximated by finite differences, can be expected to break down near discontinuities in the solution where the differential equation does not hold [LeVeque (2002)].

2.3.4 Finite Volume Method for Hyperbolic Problems

The key problem in FVM is to estimate the normal flux through each cell interface. There are several algorithms to estimate these fluxes. Valiani *et al.* (1999) stated that hyperbolic problems have an inherent directional property of signal propagation, therefore algorithms to estimate these fluxes should handle this property appropriately. In FVM the mass and momentum are conserved in each cell, even in the presence of flow discontinuities. The fluxes can be evaluated at these cell faces by solving a Riemann problem, which accurately captures wave propagation and shocks generated in the system. Also, numerical oscillations due to sudden jumps (*shocks*) in the state parameter may be suppressed with the use of a slope limiter [Bradford and Sanders (2002)]. Two principal ingredients in a FVM are *reconstruction* and *flux evaluation.*

- Reconstruction is the process of computing the spatial gradients of the flow variables.

- In structured methods, this is done using finite difference formulas and the (i, j, k) grid indices. Whereas, in unstructured methods, other techniques have been devised to compute the gradient of the flow variables.

One successful technique is the least-square reconstruction algorithm proposed by Barth (1993), where the flow gradients in a cell are computed through a least-square fitting procedure. Although this technique has been used quite successfully for the Euler and Reynolds-averaged Navier-Stokes (RANS) simulations, it has not yet been applied for Large Eddy Simulation (LES) or Detached Eddy Simulation (DNS) of turbulent flows [Trong (2000)].

Generally, in FVM a flux formula is needed to compute a single flux at a cell boundary given the two different flow states on the left and right sides of the cell boundary. If the physical propagation of information of the inviscid flow is taken into account in computing the inviscid flux, then this results in a family of numerical methods known as upwinding [Trong (2000)]. In recent years, finite volume methods have attracted wide attention and achieved a series of successes in the numerical simulation of hyperbolic problems. Using Roe's Riemann solver, Alcrudo and Garcia-Navarro (1993) developed a high-resolution Godunov-type MUSCL[1]-FVM and reported impressive results for rapidly varying inviscid flow. Zhao *et al.* (1994) designed a FVM on unstructured meshes based on Osher's scheme. Anastasiou and Chan (1997) introduced a Roe-type second-order accurate upwind FVM on unstructured triangular meshes. Following Harten-Lax-van Leer (HLL) Riemann solver, Hu *et al.* (1998) developed a HLL-type MUSCL-FVM. Tseng (1999) proposed an explicit FVM, which takes Roe, *total variation diminishing* (TVD) and *essentially non-oscillatory* (ENO) method as the special case. A composite FVM on unstructured triangular meshes was advanced by Ji-Wen and Ru-Xun (2000).

It is a very difficult to review the whole work done related to finite volume methods and the hyperbolic problems. Therefore, a few pieces of literature given below to show the variety of application and advances in the last decade.

are selected to show the variety of application and recent advances in this field.

Sonar (1997) developed ENO finite volume methods on conforming triangulations for the numerical solution of hyperbolic conservation laws. Besides theoretical results concerning the recovery of data from cell averages, they gave a description of practicable algorithms for the stencil selection to recover polynomials of arbitrary degree. Through extensive numerical tests, the accuracy of the methods was confirmed.

Zhang *et al.* (2000) proposed an a posteriori error estimation technique for hyperbolic conservation laws. The error distributions were obtained by solving a system of equations for the errors which was derived from the linearized hyperbolic conservation laws. The error source term was estimated using the modified equation analysis.

[1]MUSCL stands for Monotone Upstream-centred Scheme for Conservation Laws

Numerical tests for one-dimensional linear and non-linear scalar equations and systems of equations were also presented. The results demonstrated that the error estimation technique can correctly predict the location and magnitude of the errors.

Ghidaglia *et al.* (2001) proposed a method for the discretization of non-linear systems of PDEs occurring in the numerical simulation of two phase flows. This method was based on a cell centered finite volume discretization on unstructured meshes. They were able to consider conservative and non-conservative systems of equations and the method belonged to the class of shock-capturing upwind ones. Authors put the emphasis on the treatment of terms involving first-order derivatives since we deal with the change of type (hyperbolic to non-hyperbolic).

Souadnia *et al.* (2005) used the general multimaterial formulation of Kashiwa and Rauenzahn (1994) to derive an hydrodynamic model for a trickle-bed reactor operating under trickling flow conditions. This kind of trickle-bed reactor has applications in many fields *e.g.*, bio-industry, electrochemical industry and remediation of underground water resources other than the traditional chemical, petrochemical and petroleum industries. For the model they used the FVM and the second order Godunovs method combined with the solution of Riemann problem.

Kroger and Lukacova-Medvid'ova (2005) proposed a finite volume evolution Galerkin (FVEG) scheme for the shallow water magnetohydrodynamic equations. This was one of the first attempts to apply multidimensional Evolution Galerkin (EG) techniques to a magnetohydrodynamic model.

Calle *et al.* (2005) proposed a stabilization technique and studied for the discontinuous Galerkin method applied to hyperbolic equations. In order to avoid the use of slope limiters, a streamline diffusion-like term was added to control oscillations for arbitrary element orders. The scheme combines ideas from both the RungeKutta discontinuous Galerkin method [Cockburn and Shu (2001)] and the streamline diffusion method [Brooks and Hughes (1982)].

Rossmanith (2006) presented an explicit FVM for solving general hyperbolic systems on the surface of a sphere. Applications where such systems arise include passive tracer advection in the atmosphere, shallow water models of the ocean and atmosphere, and shallow water magnetohydrodynamic models of the solar tachocline. This approach used TVD wave limiters, which allowed the method to be accurate for both smooth solutions and solutions in which large gradients or discontinuities can occur in the form of material interfaces or shock waves.

Nordstro and Gong (2006) proposed a stable hybrid method for hyperbolic problems that combines the unstructured finite volume method with high-order finite difference methods. The coupling procedure was based on energy estimates and stability was guaranteed.

2.4 High Resolution Methods

In fluid dynamics problems, discontinuities usually do not develop from smooth initial conditions; except in cases of breaking waves, such as the formation of hydraulic jumps that evolve in the shallow-water flows from smooth initial data. For instance in mid-latitudes, fronts can be formed in low-pressure systems, yet these fronts are not entirely discontinuities. Atmospheric fronts (also substances such as chemical pollutants) are transported from one location to another, described very well by a tracer advection model. Due to the deformation (stretching and shearing) of the velocity field that advects the front, discontinuous can be formed on the resolution scale of the (computational) model [Durran (1999)]. Therefore, from a purely computational stand point, there is a need to apply numerical schemes devised for numerical solutions of conservation laws which support discontinuous solutions.

In recent years, a tremendous amount of research has been done in developing and implementing modern high-resolution methods for approximating solutions of hyperbolic systems of conservation laws. A review of such numerical methods can be found in the books by Godlewski and Raviart (1996) and LeVeque (2002).

Rood (1987) provided a detailed analysis and comparison of various advection schemes on a simple linear atmospheric transport model. Lin *et al.* (1994) have analysed the effect of varying the slope limiters using an atmospheric general circulation model. Lin and Rood (1996) compared the first order upwind, central difference, PPM (modified monotonic and positive definite) and monotonic vanLeer schemes, and concluded that their monotonic version [vanLeer (1977b)] of PPM yields the most accurate results. Towards the development of a fully operational atmospheric general circulation model based on FV discretization, Lin and Rood (1997) implemented slope limited vanLeer schemes and the PPM scheme on a shallow water equation model using a semi-Lagrangian semi-implicit time integration scheme. For a discussion and applications of other popular schemes such as MPDATA of Smolarkiewicz (1984) and QUICK of Leonard [Leonard (1979) and Leonard (1991)], see Vukicevic *et al.* (2001).

Some of the most popular methods in the finite volume context are Lax-Wendroff, Lax-Friedrichs, Roes, flux corrected transport methods of Boris- Book and Zalesak, slope limited methods of van Leer, piecewise parabolic method (PPM) of Colella and Woodward essentially non-oscillatory schemes of Harten-Shu-Osher to name a few [see LeVeque (2002), Laney (1998), Durran (1999)].

There are two ways of implementing the high resolution methods *i.e.*, using structured and unstructured meshes. In the case of irregular boundaries the structured and unstructured mesh use different techniques:

- **Structured mesh** algorithms employed generally involve complex iterative smoothing techniques that attempt to align elements with irregular boundaries or physical domains. Where non-trivial boundaries are required, *block-structured* techniques can be employed which allow the user to break the domain up into topological blocks.

- **Unstructured mesh** generation, on the other hand, relaxes the node valence requirement, allowing any number of elements to meet at a single node. Triangle meshes are most commonly thought of when referring to unstructured meshing, although quadrilateral and hexahedral meshes can also be unstructured.

2.4.1 High Resolution Methods for Structured Meshes

The last two decades have witnessed a sustained effort by the CFD community to develop robust high resolution methods for the simulation of advection-dominated flows [Darwish and Moukalled (2003)]. Many of these methods have been implemented on structured mesh/grid within the framework of finite volume methods. The main ingredients common to all these methods are a high order profile for the reconstruction of cell face values from cell averages, combined with a monotonicity criterion.

The high order reconstruction is usually based on an upwind biased, sometimes symmetric, high order interpolation profile [Leonard (1979), Leonard (1991) and Holnicki (1996)]. To satisfy monotonicity, a number of concepts have been proposed over the years [LeVeque (2002)], most of them are within a structured mesh/grid framework.

In the Flux Corrected Transport (FCT) approach of Book *et al.* (1981), a first order accurate monotone scheme was converted to a high resolution scheme by adding limited amounts of anti-diffusive flux. In the monotonic upstream-centered scheme for conservation laws (MUSCL) of vanLeer (1979), monotonicity is enforced through a limiter function applied to a piecewise polynomial flux reconstruction procedure. Harten (1983) expressed monotonicity as a measure of discrete variation in the solution fields, hence the name Total Variational Diminishing (TVD). This criterion was then expressed as a flux limiter by Sweby (1984) using the $r - \psi$ diagram. Leonard (1991) presented his monotonicity criterion using a relation between a normalized face value and a normalized upwind value. While on the conceptual level the above-mentioned monotonicity criteria can be shown to be related and sometimes equivalent, but implementation-wise they are very different. However within the framework of structured grids these differences have not translated into increased difficulties in implementation.

While there is certainly some overlap between structured and unstructured mesh generation technologies, the main feature which distinguish the unstructured grids is the advantage of generality in that they can be made to conform to nearly any desired geometry.

2.4.2 High Resolution Methods for Unstructured Meshes

For unstructured meshes the situation is more complicated and high resolution methods are not as advanced as for structured meshes [Venkatakrishnan (1996) and Fursenko *et al.* (1993)]. This is specifically due to the difficulty in implementing and enforcing a monotonicity criterion that relies on logical or directional next-neighbor information, which is readily available in structured meshes but missing in unstructured meshes. To overcome this difficulty a number of approaches have evolved, with varying degrees of success, based on different monotonicity criteria, such as the FCT [LeVeque (2002), Zalesak (1979), Boris and Book (1973) and Boris and Book (1976)], the flux difference splitting concepts [Lohner *et al.* (1987) and Ferzoui and Stoufflet (1989)], or the MUSCL approach [Venkatakrishnan and Barth (1989) and Frink (1992)].

The MUSCL-based technique developed by Barth and Jespersen (BJ) [Venkatakrishnan (1994)], by modifying the Spekreijse (1987) definition of monotonicity to bound the cell face values rather than the cell nodal value, was one of the most popular and successful approaches for the implementation of high resolution methods in unstructured meshes [Venkatakrishnan (1993), Anderson (1994) and Swanson *et al.* (1998)], partly because of its simplicity. Unfortunately, most of the limiters developed for structured meshes cannot be implemented using the BJ technique as it is restricted to schemes where the base high order profile uses a cell based gradient, which is basically equivalent to the FROMM scheme [vanLeer (1979)], whose bounded version is equivalent to the MUSCL scheme. In one-dimension the BJ method can be shown to be equivalent to the TVD-MUSCL scheme [Bruner (1996)]. Bruner (1996) suggested a more general approach to bound convective schemes. In this approach he used the Sweby $r - \psi$ diagram with a modified r factor defined for unstructured grids. Unfortunately his modification did not recover the exact r factor on structured grids.

Many limiters have been developed and used for the hyperbolic problems with different level of success. A few of the well known limiters that are used for two-dimensional unstructured meshes are:

- **Minmod Limiter:** The minmod flux limiter applies the maximum possible limiting allowed within the second order TVD region. Anastasiou and Chan (1997) stated the minmod limiter produces satisfactory results, but numerical diffusion was evident, as is always the case with any dissipative algorithm with a limiter. It is also used in several pieces of literature, few of them are by Sweby (1984), Hirsch (1990), Roe and Sidilkover (1992), Wang *et al.* (2000), Caleffi *et al.* (2003) and Lipnikov and Shashkov (2006).

- **Superbee Limiter:** The superbee limiter applies the minimum limiting and maximum steepening possible to remain TVD. It is known to suffer from excessive sharpening of slopes as a result. It is also used in several literature, few of them are by Roe and Sidilkover (1992), Arora and Roe (1997) and Szpilka and Kolar (2003).

- **LCD Limiter:** The LCD (Limited Central Difference) limiter is one of the earliest limiters in the context of MUSCL-schemes. This limiter's advantages lie in its simplicity and speed. Batten *et al.* (1996) and Vollmer (2003) have used the LCD limiter.

- **MLG Limiter:** It stands for Maximum Limited Gradient, developed by Batten *et al.* (1996). Author has shown that it reduces to Roe's superbee limiter in one dimension. Vollmer (2003) found that it is less diffusive than the LCD limiter, but Wang and Liu (2002) states that this advantage of minimum numerical dissipation is expensive to compute.

2.4.3 Numerical Methods for High Resolution Shock Capturing

Modern numerical ideas of shock capturing for computational fluid dynamics can date back as early as 1944 when von Neumann first proposed a new numerical method, a centered difference scheme, to treat the hydrodynamical shock problems, for which numerical calculations showed oscillations on mesh scale [Lax (1986)]. von Neumann's dream of capturing shocks was first realized when von Neumann and Richtmyer (1950) had the ingenious idea of adding to the hydrodynamic equations a numerical viscous term of the same size as the truncation error. Their numerical viscosity guarantees that the scheme is consistent with the Clausius inequality, the entropy inequality. The shock jump conditions *i.e.*, the Rankine-Hugoniot jump conditions, are satisfied provided that the equations of fluid dynamics are discretized in conservation form. Then oscillations were eliminated by the judicious use of the artificial viscosity; solutions constructed by this method converge uniformly except in a neighborhood of shocks, where they remain bounded and are spread out over a few mesh intervals [Chen (2000)].

The analytical idea of shock capturing *i.e.*, vanishing viscosity methods, is quite old. Chen (2000) states that good numerical schemes should be numerically simple, robust, fast, and low cost, and have sharp oscillation-free resolutions and high accuracy in domains where the solution is smooth. It is also desirable that the schemes capture contact discontinuities, and are coordinate invariant, among other things. For the one-dimensional case, examples of success include the Lax-Friedrichs scheme [Lax (1954)], the Glimm scheme [Glimm (1965)], the Godunov scheme [Godunov (1959)] and related high order schemes *e.g.*, vanLeer's MUSCL [vanLeer (1979)], Colella-Wooward's PPM [Colella and Woodward (1984)], Harten-Engquist-Osher-Chakravarthy's ENO [Harten *et al.* (1987)], and the Lax-Wendroff scheme [Lax and Wendorff (1960)] and its two-step version, the Richtmyer scheme [Richtmyer and Morton (1967)] and the MacCormick scheme [MacCormack (1969)].

The most common approach to shock capturing is to first develop a one-dimensional, total-variation-diminishing (TVD) upwind scheme for a scalar conservation law and then apply it to systems using one-dimensional characteristic decompositions or approximate Riemann solvers. Upwind schemes have been used very successfully for gas dynamical calculations, where the Riemann problem can be solved exactly and many approximate Riemann solvers are available [Tai *et al.* (2002)]. Tracking shocks, especially when and where new shocks arise and interact in the motion of fluids, is scientifically extremely important but numerically burdensome. The main motivation in developing numerical shock capturing algorithms is to treat the shock problem in fluids [Chen (2000)].

Some of the classical references on development of high-resolution shock-capturing schemes are Glaz and Liu (1984), Glimm *et al.* (1985), Chen *et al.* (1993) and Canuto *et al.* (1998), or the textbooks Glass (1974), Chainais-Hillairet (1999), Toro (1999), and LeVeque (2002).

2.5 Shallow Water Equations

In recent years free-surface flow models have been increasingly developed using explicit schemes, *vis-á-vis* implicit schemes [see Fennema and Chaudhry (1990), Nujic (1995) and Chan and Anastasiou (1999)]. One of the main reasons for using the explicit scheme is that the resulting numerical models behave better when used to simulate flows with sharp gradient free surfaces, such as dam break flow. Several techniques have been published in the literature concerning the use of the finite volume method to solve the two-dimensional shallow water equations to model free surface flows. Zhao *et al.* (1994) and Zhao *et al.* (1996) used three-types of Riemann solvers, including the flux vector splitting, the flux difference splitting and the Riemann solver of Osher, to simulate shock wave problems. The model was based on the finite volume method and used 4-sided grids. Numerical tests showed that all of these three schemes were capable of predicting both gradually and rapidly varying flows, including those with sharp surface gradients. However, since these schemes were first order accurate, the effect of numerical damping was observed when studying model predicted surface profiles. Anastasiou and Chan (1997) and Chan and Anastasiou (1999) developed a finite volume scheme based on a Godunov-type second-order upwind formulation to solve incompressible flows, both with and without a free surface and using an unstructured triangular mesh. Mingham and Causon (1998) developed a high resolution finite volume scheme using a MUSCL reconstruction and with a slope limiter to capture surface discontinuities. Their model was applied to simulate bore wave diffraction in both internal and external hydraulic flows.

2.5.1 Riemann Problem

Prediction of flows with discontinuities, such as hydraulic jumps, has been a great challenge to numerical schemes for hydrodynamics [Benedict *et al.* (2003)]. Extensive research has been performed in the area of shallow-water equations, and several upwind methods originally designed to solve the Euler equations have been extended to the shallow-water system [Mohamadian *et al.* (2005)]. These schemes are introduced for the solution of the hyperbolic sets of equations in order to take into account the information about the direction of signal propagation, enclosed in this class of equations.

One method is based on the solution of the local Riemann problem at cell interfaces. This approach was proposed by Godunov (1959), so the derived schemes are called **Godunov-type**. In the well known work of Godunov, the exact solution of the Riemann problem was used. Today the exact solution of the Riemann problem is replaced with an approximate solution in order to reduce computational time; this type of scheme is called Flux Difference Splitting (FDS) [Caleffi *et al.* (2003)]. In the last twenty years, many efforts have been made by several researchers in the field of approximate Riemann solvers. The more remarkable results are obtained in aerodynamics, but are also easily applicable to the shallow water equations. In fact, a strong analogy between compressible flow and free surface shallow water flow exists [Liggett (1994)].

Many approximate Riemann solvers now exist to evaluate numerically convective fluxes. Those are for example, vanLeer (1977b), Roe (1981), Osher and Solomon (1982), Harten (1983), and extended later on to free surface hydraulics in several papers, including those by Glaister (1988), Alcrudo *et al.* (1992), Alcrudo and Garcia-Navarro (1993), essentially non oscillatory (ENO) schemes [Nujic (1995)], the Harten, Lax and van Leer (HLL) solver [Mingham and Causon (1998)] and Valiani *et al.* (2002). Most of these methods have the capability of shock capturing with a high level of accuracy in few computational cells, and the flux vector is determined based on the wave propagation structure.

Some of these methods perform well for particular types of flow like discontinuous or transcritical flows over flat topographies, but in the case of flows over variable topography there is room for considerable improvement. The advantages of using Riemann solvers to describe rapidly varying shallow water flows became apparent in the early 1990s. The shallow-water equations describe the conservation of mass and momentum in shallow water bodies, and are particularly amenable to solution by finite volume Godunov-type approaches where Roe's approximate Riemann solver can be used to evaluate inviscid fluxes [Alcrudo and Garcia-Navarro (1993), Anastasiou and Chan (1997) and Fujihara and Borthwick (2000)].

Although Roe's approximate method is robust, difficulties arise in solving the Riemann problem when source terms are included in the analysis [Gascon and Coberan (2001)]. Essentially, a numerical imbalance is created by the artificial splitting of physically meaningful terms in the governing equations between flux gradients and source terms in order to generate a mathematically hyperbolic formulation. These terms are then evaluated by different methods at different locations within the computational grid creating the numerical imbalance.

2.5.2 Source Terms

The imbalance problem is particularly acute for the shallow water equations where the surface gradient term within the momentum equations is conventionally split into an artificial flux gradient and a source term that includes the effect of the bed slope. Thus, many numerical solvers of the shallow water equations based on the conventional formulation give unphysical results for flows over physically realistic variable bathymetries, solely because of this mathematically convenient splitting.

Bermudez and Vazquez (1994) extended the vanLeer's Q-scheme for variable topographies by using an upwind discretization of the source terms and they introduced the C property, which states that the scheme should preserve the stagnant conditions.

Nujic (1995) used the water level variable instead of the depth and he extracted the gravitational terms from flux functions. Nugic proposed a revised mathematical formulation of the shallow water equations, by reallocating all bed-slope related flux gradients to the source terms. Then, using the Shu and Osher (SO) scheme [Shu and Osher (1988)], computed the flux vector and obtained good results for dam break problems over variable topographies.

Ambrosi (1995) noted that the effectiveness of using Roe's approximate Riemann solver was lost when the bottom slope varied, giving a quantitative estimation of the error of the scheme as first-order, but accepted that the quiescent still water solution was not computed, in favour of preserving the formal accuracy of the scheme.

LeVeque (1998) introduced a Riemann problem inside a cell for balancing the source terms and the flux gradients, and proposed a wave propagation algorithm by artificially introducing another discontinuity within each computational cell to account for the propagation of source terms. The resulting method was found to preserve both stagnant and quasi-steady state conditions. Although suitable for quasi-steady conditions, LeVeque's method is reportedly less robust when predicting steady transcritical flows that contain shocks [Benedict *et al.* (2003)]. Also, LeVeque's scheme is not directly transportable to unstructured grids.

Vazquez-Cendon (1999) used numerical upwinding of the source terms to achieve equilibrium between flux gradient and source terms in the shallow water equations. Hubbard and Garcia-Navarro (2000) and Garcia-Navarro and Vazquez-Cendon (2000) have since extended this numerical treatment to higher order total variation diminishing (TVD) schemes.

Zhou *et al.* (2001) introduced the surface gradient method and showed that interpolating the depth without considering the bed variations may lead to erroneous results. They used the interpolated water surface elevation to calculate the depth at the interface and showed that this approach combined with the HLL flux function [Harten (1983)] satisfies the C property. Their scheme performs very well for variable topographies without any extra efforts for balancing the source terms and the flux gradients. However, the C property does not hold for unstructured grids and moreover, the HLL flux induces a high level of numerical viscosity in recirculating flows.

Burguete and Garcia-Navarro (2001) investigated different explicit schemes and presented conservative schemes in a non-conservative formulation of the equations with flux-adjusted source terms discretised using either a semi-implicit or upwinding technique. Gascon and Coberan (2001) present another approach to deal with the balancing difficulty by transforming non-homogeneous conservation laws into homogeneous ones by introducing a new flux generated by the addition to the physical flux of the primitive of the source term.

Alcrudo and Benkhaldoun (**2001**) defined the topography such that a sudden change in the topography occurs at the interface of two cells. They also developed a Riemann solver at the interface with a sudden change in the bed elevation. However, their approach leads to several cases of Riemann fan and it is numerically too expensive.

Vukovic and Sopta (**2002**) extended the ENO and WENO schemes to shallow-water equations with the source terms in 1D channels.

Xu (**2002**) proposed a second order gas kinetic scheme for shallow-water flows over variable topographies. The gas kinetic schemes are basically different from characteristics based schemes.

Mohamadian *et al.* (**2005**) has presented an efficient numerical method for recirculating flows over variable topographies on unstructured grids. In order to fulfill this goal, Nujic's method [Nujic (1995)] is combined with the Roe's scheme, and the surface gradient method [Zhou *et al.* (2001)] is used for calculating the water depth at the interface.

2.6 Object-Oriented Modelling

The idea behind object-oriented modelling is that a computer program may be seen as composed of a collection of individual units, or objects, that act on each other, as opposed to a traditional view in which a program may be seen as a collection of functions or procedures, or simply as a list of instructions to the computer. Each object is capable of receiving messages, processing data, and sending messages to other objects. It is very broad subject and each model is different. To develop an object-oriented model, knowledge of object-oriented technologies and self-visualization of the problem is necessary. Therefore, this section is just an overview of some of the work done related to the use of object-oriented modelling.

Crutchfield and Welcome (**1993**) have built an adaptive mesh refinement algorithm model using a mixture of C++ classes for high organizational levels and Fortran for low level numerical routines. Effectively, the C++ classes encapsulate Fortran routines and provide a hybrid solution that allows better memory management than pure Fortran coding.

Larsen and Gavranovic (**1994**) reviewed the state of object-orientation within hydroinformatics and concluded that the *Dinosaur effect* was taking place where even the average word processor application was getting so large that not only users failed to understand the complexity but that development may well have also stalled due to the ever-increasing code sizes. Deckers (1994) produced a geohydrological information based system using an object-oriented programming approach and both spatial and non-spatial information. The system used a proprietary Geographic Information Systems (GIS) with its own object-oriented language. This system allowed the coupling of the user interface, GIS, tools and databases into a seamless application. Ruland and Rouve (1994) described the integration of hydraulic models into object-oriented GIS using relational databases. GIS data classes were defined for points, chains, area, polygons and topographical objects, to which 1D backwater and water quality models and 2D finite element models were coupled, also written in objects. They found that this enabled easy operation of simulation models at reduced cost and allowed better integration of technologies.

Perla and Ponnambalam (1994) discuss the use of object-oriented programming within a multi-reservoir system simulation framework using Microsoft Foundation Classes. The Microsoft Foundation Classes provide the front end and Windows implementation and C++ classes provide the numerical algorithms. Solomatine (1994) addresses the need for object-orientation within the hydroinformatics community and shows an experimental hydraulic modelling system, HIS (water distribution modelling system). He argues that object-orientation will eventually replace procedural code as code *toolboxes* arrive, much like programming environments. These toolboxes will not limit object-oriented techniques to just interfaces, GIS and databases, but will spread into the numerical engines and artificial intelligence components and code reuse will then be possible. Finally, the author concludes that the large amount of Fortran code is holding back the use of object-oriented programming techniques.

Solomatine (1996) takes the original Hydraulic Modeling System (HIS) model and adapts and broadens it. In doing so, he has built on his previous work and used the same base classes, extending the hydraulic features and user interfaces within an event framework. The entire system has been implemented using Pascal with objects within a Windows, icons, menus and pointer interface.

Shane *et al.* (1996) have produced an object-oriented water resources management system model (PRSYM), written in C++. The software is capable of modelling reservoir storage, different types of routing, power generation, control rules, water quality and linear programming optimization to optimise policy and economic constraints. Kutija (1998) describes the development of an object-oriented model for the solution of free surface flows in dendritic channel networks. The model uses the de Saint Venant equations, solved by the Abbott-Ionescu [Abbott and Basco (1989)] implicit finite difference method. This gives alternating H (elevation) and Q (flow) calculating points with the nodes always being H points. The model was designed to handle rectangular crosssections and flow and elevation boundary conditions, solved by a classical dendritic solver [Cunge *et al.* (1980)]. Using this, the object hierarchy shown in Figure 2.2 [Kutija (1998)] was derived.

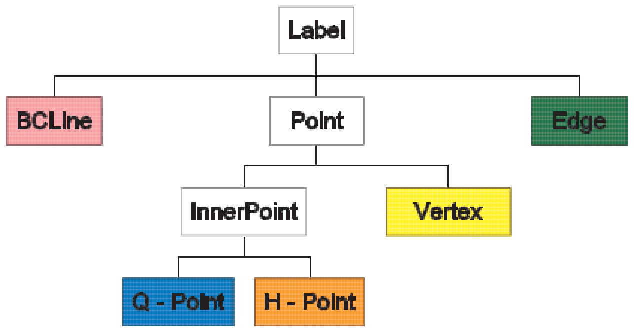


Figure 2.2: Family tree of Label type objects [Kutija (1998)]

Tomicic and Yde (1998) demonstrated a typical example of legacy procedural software integration across a wide range of models. Using six models: STORMPAC (Rainfall generation); MOUSE (Urban Sewer Modelling); STOAT (Waste water treatment); MIKE 11 (River modelling); MIKE 21 (Coastal Modelling); and SIMPOL (Simplified urban drainage), they progressively integrated the system of models, starting with input/output files, to produce integrated procedures and a common user interface on a multi-processor machine. This was carried out using procedural techniques to couple the various codes together and allow information to be passed between the various models that became fully integrated into the whole. Thus, the modelling system became fixed into a monolith of code highly suited for its task but incapable of being used in other configurations. Alternatively, Cate *et al.* (1998) took a more flexible approach to model coupling within a framework so that any collection of models reaching a common standard could be coupled in a multitude of common ways. CORBA technology was tried but was found to be not ready for the connection of these models. Whilst they found shared memory and common files adequate for loosely coupled systems, it was not found to be suitable for more closely coupled systems where data and information needed to flow in many directions.

Harvey *et al.* (2002) promoted the use of models being broken down into smaller components that could be arranged by a model developer into new models without the need for constant rewriting, much like user interfaces are now designed by dragging buttons onto a form. This would create a component framework rather than a model framework. Alfredsen (2000) shows a good example of code wrapping. Components were divided into river reaches, lakes, reservoirs, *etc.* Each component was fitted with a standard interface that allowed flow to be taken from the upstream component and then routed to the next component. Using these basic types of component, further sub types were derived for different types of routing and storage. The system does not offer any real interaction as information flow is one way and each component is fired off one after another. Unlike other frameworks, the components can still be used individually.

Tachikawa *et al.* (2000) and **Ichikawa *et al.* (2000)** describe the development of the OhyMoS object-oriented hydrological modelling system which has been applied to *Chao Phraya* River basin in Thailand. The model was built around a grid system in which channels and links (ports) to neighbouring grid boxes could be made to form a channel network. Flow could therefore be routed through the grid boxes using a routing equation and rainfall added using a rainfall runoff (Xinanjiang) model. The entire model is built around the grid structure that allows different model representations to be selected for different grid boxes, such as HEC-HMS (Hydrologic Engineering Center - Hydrologic Modelling System, U.S. Army Corps of Engineers) where each model has a standardised form, inputs and outputs and can be wrapped in a class.

CHAPTER 3

Object-Oriented Approach

3.1 Introduction

This chapter aims to provide reader with background knowledge of object-oriented approach used for the *Riemann2D* development. This chapter first gives a brief outline of various approaches to the model development followed by discussion on the basic design of the *Riemann2D* model.

3.2 Approaches to Software Design

A software model is an abstract representation of a system that enables us to answer questions about the system. These models are useful when dealing with systems that are too large, too small, too complicated, or too expensive to experience firsthand. Usually the model gets complex as it grows or develops in size. To remain useful the model needs to evolve with the end users' requirement. Traditional approaches to the design of software have been either,

- **Data-oriented methodologies**: It emphasise the representation of information and the relationships between the parts of the whole. The actions (processes) which operate on the data are of less significance. (*e.g.*, dBase, a relational database access language; M ,an ANSI standard general purpose language with specializations for database work; and SQL); or

- **Process-oriented methodologies**: It emphasise the actions performed by a software artifact; the data are of lesser importance.

The *data-oriented* perspective focuses on complete and thorough data analysis and its relationships; the *process-oriented* perspective focuses on analysis of functions/processes of real world systems; whereas for the real-world engineering problems require focus on both *i.e.,* data and process.

Hence, to maintain the usefulness of the *Riemann2D* as it grows, a different approach is adopted, called *object-oriented*, in which an equal emphasis is given to both, *data* and *process*, of the system. Preiss (1998) also justifies the use of object-oriented design as:

> *"**Object-oriented** methodologies are more effective for managing the complexity which arises in the design of large and complex software artefact than either data-oriented or process-oriented methodologies. This is because data and processes are given equal importance. Objects are used to combine data with the procedures that operate on that data."*

3.3 Object-Oriented Technique

Objects are key to the object-oriented technique, which can be defined as *a software bundle of variables and related methods*[1]. In essence, an **object** is only an object if it is described in fundamental components or is built using other objects. It shares two characteristics of the real world: They all have *state* and *behaviour*.

Some programming languages do have objects but cannot properly be called *object-oriented*. The original "Ada" is one such. Ada-83 has what it terms *packets*, which satisfy the requirements for objects but lack two very important features that must be present to earn the label object-oriented. These are:

1. *Inheritance:* Each subclass shares common characteristics with the class from which it is derived. Inheritance enables the derivation of a new class using all the existing properties of its base classes, and derived classes can add new properties

[1]See `http://java.sun.com/docs`

of their own. Inheritance shortens an object-oriented program and clarifies the relationship among model elements; and

2. *Polymorphism:* This technique allows functions or operators to act in different ways, depending on the element on which they are operating. Polymorphism simplifies code design and makes programming more efficient.

These two features give object-oriented technology much of its power compared with conventional languages [Brown (1997)]. Inheritance and polymorphism make possible vast potential for reuse of program code, as well as analysis of the results. Although, in a few cases most of the advantages of object-oriented programming carry over to object-based programming in Fortran 90, but that is at the expense of extra work and discipline [Gray and Roberts (1997)].

3.3.1 Object-Oriented Models

Generally, the object-orientated models for the scientific purpose are developed in three ways:

1. One way is that the models are the fundamental building blocks of objects. These *objects* are normally formed from legacy code wrapped in COM, DCOM, CORBA or ActiveX, *etc.* containers. These these methods offer a good way for distributed computing across a wide range of different systems and platforms. By wrapping the legacy codes up in the latest standards, the entire model can be labelled *object-oriented*.

2. The second way is to consider that objects are for data user interfaces, graphics and sundry items. An example of this is the EPANET model, developed by United States Environmental Protection Agency [1]. The code, which is freely available, is split into two parts: numerical code and user interface. The numerical part is written in C in a procedural coding style with data stored in record structures and compiled as a Dynamic Link Library (DLL). The user interface is written in Delphi using standard object-oriented design and visual components. The user interface calls the numerical DLL and then during a model run uses object-oriented techniques to animate the model results.

[1]`http://www.epa.gov/ORD/NRMRL/wswrd/epanet.html`

3. The above two ways are not fully object-oriented. Although they are good to wrap up the legacy codes, they often lack the full exploitation of the object-oriented technologies to make the model simpler and more robust. An example of a truly object-oriented model is HYPRESS [Vanecek *et al.* (1994)], which is built around a MS Window based object-oriented database. The model examined water hammer and water distribution in pipe networks. The database was designed so that an entire windows framework was stored within the database.

In this work the last approach is adopted to develop *Riemann2D* as truly object-oriented model. It can be seen in the following sections and chapters that this object-oriented approach produces robust and more reliable code, which can easily perform various task, and can be shared by a number of researchers and developers.

3.4 Design of the *Riemann2D* Objects

In Object-Oriented design (OOD) of *Riemann2D* , main focus is on creating *objects* (a.k.a *classes* in programming languages). Each of these object contains data as well as the set of functions that manipulate those data. The *data components* of a class are called *data members* or *attributes* or *variables* and the function components of the class are called *member functions* or *behaviours* or *methods*. *Riemann2D's* OOD advocates claim that Object-Oriented techniques provide a natural and intuitive view of the programming process, by modelling the attributes and behaviours of real-world objects.

Problem structuring is the initial activity in any *object-oriented* design, the success of which is largely dependent on the correct identification and definition of the chosen components of the problem. Problem structuring is a mixture of art[1] and science which seeks to build a formal representation, integrating the objective components of the model and the problem. An important aspect of object-oriented problem structuring is the degree of disaggregation, which depends on the problem to be solved [Murray (2003)].

[1]Problem structuring is considered an art as there is no attempt to uncover the scientific principles underlying the intuition and craftsmanship of the individual developer [Winterfeldt (1980)].

3.4.1 Object-Oriented Problem Structuring of *Riemann2D*

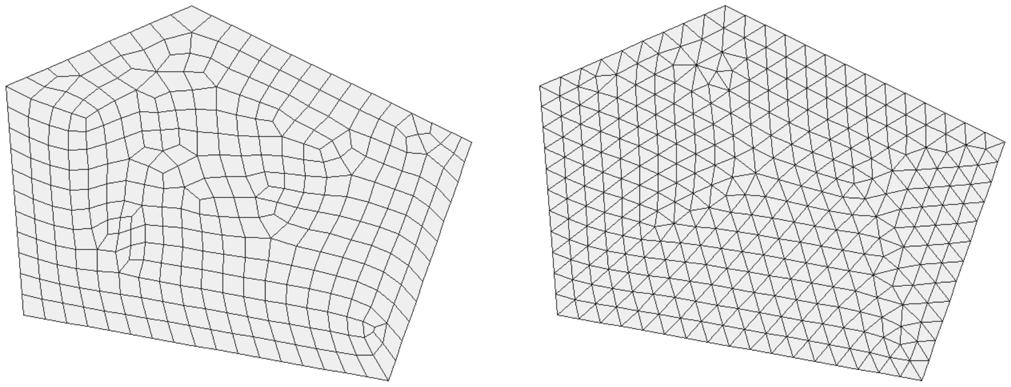


Figure 3.1: (a) Quadrilateral Mesh (b) Triangular Mesh

In order to numerically solve PDEs on a surface (2D problem), it is necessary to create a mesh (Figure 3.1) on the surface. This mesh specifies the nodes/areas where the equation is solved as well as the relation between these nodes/areas. The derivation of a numerical method is based on these meshes and control volumes.

These mesh are made up of elements of different shape (triangle, quadrilateral etc.), where each element is made up of sides, which in turn are made up of two nodes. Therefore for the *Riemann2D*, it is logical to build *objects* (or classes) using this premise: Node, side, element, mesh and a solver.

3.4.2 Physical Description of *Riemann2D*

As described above, the *Riemann2D* has primary building blocks namely, nodes, sides, elements, a mesh and a solver to create a model representation of the physical hyperbolic system. It is easy to understand (see Figure 3.2) that nodes are the building blocks of sides, which in turn are building blocks for elements and similarly elements are building blocks for meshes.

The code in *Riemann2D* is organised into packages[1] based on the objects' functionalities. The primary benefit of packages is the ability to organise many *classes* into a single unit. Classes in the model often need to communicate with other classes to access their *data components i.e.,* variables and methods directly. However, it is important

[1]Packages are collections of classes and interfaces that are related to each other in some useful way.

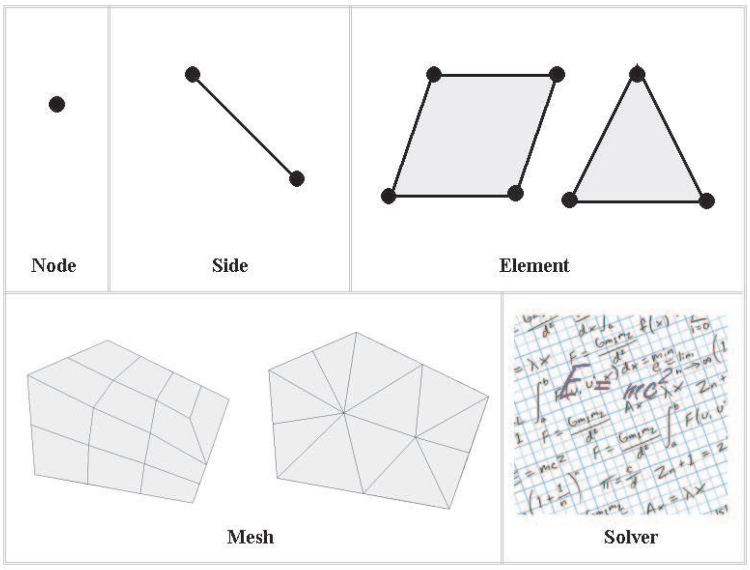


Figure 3.2: Objects of *Riemann2D*: Node, Side, Element, Mesh and Solver

to keep the *data components* open or closed for the access from other classes, which is based on the requirements of the model. *Riemann2D* follow the open-close principle of object-oriented design.

Open-Close Principle *states that software entities like classes, modules and functions should be open for extension but closed for modifications. To extend the behavior of the system, we add new code, but we do not modify old code.* [Martin (2002)]

This is achieved through abstraction, polymorphism, inheritance and interfaces. The Open-Close principle is the heart of OOD and conformance to this principle yields the high level of *Riemann2D* reusability and maintainability.

As discussed in section 1.3.1, the *Riemann2D* model is an attempt towards developing a generic hyperbolic solver, which is empowered by its object-oriented design. In this work focus is on pulling all the common attributes of the various hyperbolic models and place it in a package of classes. The distribution of the data components would follow the object design as discussed before. This approach of pulling all the common component is known as hierarchy system of object-oriented design, which follows the Liskov-substitutability principle of object oriented design.

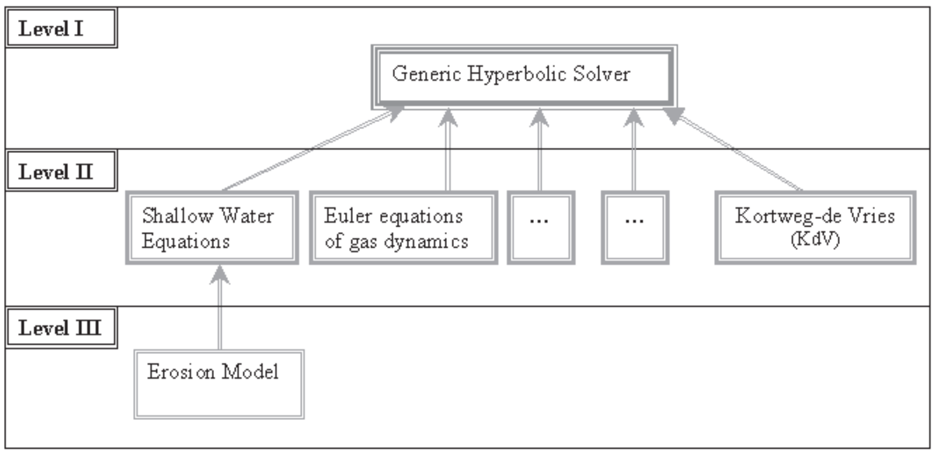


Figure 3.3: Level of packages in the *Riemann2D* model.

Liskov-Substitutability Principle *defines the use of hierarchy as an important component in OOD. Hierarchy allows the use of type families, in which higher level super-types capture the behaviour that all of their subclasses have in common. This also explains the method of reuse in which new functionality is obtained by extending the implementation of an existing object.* [Liskov and Wing (1994)]

In the present form *Riemann2D* is divided into three level of packages, as shown in Figure 3.3, which can be described as:

- **Level I**: It is the highest level of packages in *Riemann2D*. This package contains all the common *data components* required[1] by any model to solve a hyperbolic problem. It is important to note that the model at this level is not a stand-alone model *i.e.*, it doesn't solve any problem.

- **Level II**: This level contains the hyperbolic systems to be solved. The Figure 3.3 shows[2] the packages including the shallow water equations, Euler equations

[1]The decision of the common *data components* should be based on the knowledge of other systems. For *Riemann2D* the common *data components* are based on the knowledge from Toro (1999) and LeVeque (2002). Therefore, this model can evolve as it will be used for different systems.

[2]It should be noted that only shallow water equations are solved in this work. However, other system are shown in level II for the illustration purpose.

of gas dynamics and other hyperbolic system. At this level a hyperbolic system is solved and it uses the *data components* of level I.

- **Level III**: This is the lowest level of packages in the *Riemann2D* . However, more level can be derived depending on the system properties and requirements. This system shows the erosion problem[1] of shallow water equations.

Figure 3.3 and above description of levels show the flexibility of the *Riemann2D* model. Following the Liskov-substitutability principle this relationship can be further explained as:

- Level I is the super-type (also called superclass) for level II, which is sub-type (also called subclass) for level I.
- Level II is superclass for level III, which is subclass for level II.
- Level I is superclass of level II, which is superclass of level III, therefore, level I is also superclass of level III.

This relationship between packages and their level is an important factor in deciding the *data components* for a new package. In this work the *generic hyperbolic solver* of level I and *shallow water equations* of level II is studied and developed (see Figure 3.3). The Figure 3.4 shows the relationship between objects of the two packages, which is an enhanced view of Figure 3.3 covering only *generic hyperbolic solver* and *shallow water equations* packages.

It can be seen in Figure 3.4 that the objects (or classes) of ShallowWater package *i.e.*, *Node, Side, Element, Mesh* and *Solver*, are subclass of the objects of Generic package[2] *i.e.*, *GenericNode, GenericSide, GenericElement, GenericMesh* and *GenericSolver* respectively.

[1]Erosion problem is not solved in this work. However, it is included in the present object-oriented design of the *Riemann2D* . Detailed UML diagram and description is given in chapter 7.

[2]The name of the objects or classes in the generic package has “Generic” as suffix. This is only for the ease of representation, otherwise same object names can be given to superclass and subclass packages.

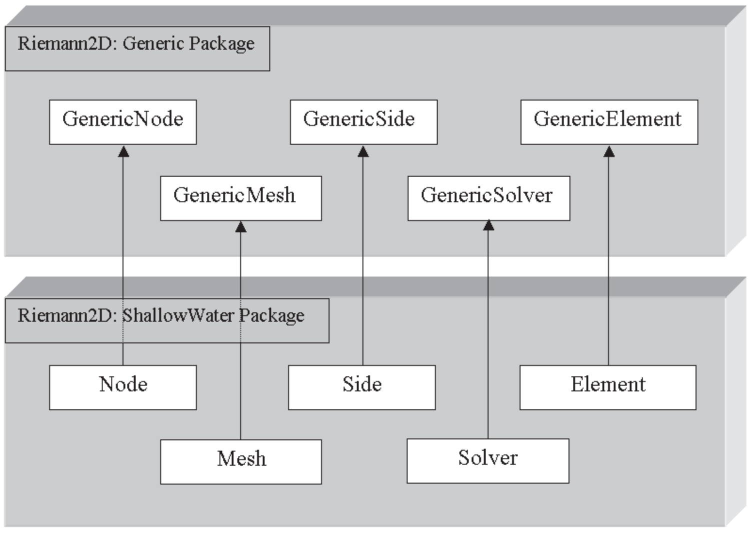


Figure 3.4: Relationship between the various classes in the Generic package and the shallow water package.

3.4.3 Controlling Access to Members of a Class

One of the important properties of any model is to define the access level to the *data components*. An access level determines whether other classes can use a particular member *variable* or call a particular *method*. Different programming language has different access specifiers. For this work Java programming language is used, which supports four access specifiers for member variables and methods: private, protected, public, and, if left unspecified, package private. The table 3.1 shows the access permitted by each specifier.

Table 3.1: Access levels used in *Riemann2D* development.

Specifier	**Class**	**Package**	**Subclass**	**Model**
Private	Y	N	N	N
No specifier	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

The first column in table 3.1 indicates whether the class itself has access to the member defined by the access level. It can be seen that a class always has access to its own members. The second column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member. A package groups related classes and provides access protection and name space management. The third column indicates whether subclasses of the class declared outside this package have access to the member. The fourth column indicates whether all classes have access to the member.

Access levels affect the model development in two ways:

- Firstly, when we use classes that come from another source, such as the classes in the superclass, access levels determine which members of those classes new classes can be used.
- Secondly, when we write a class, we need to decide what access level every member variable and every method in the class should have.

Much effort has been giving into deciding the access level for a member. Most important factor that helps in deciding the access level is that if other developers will use the class, we need to ensure that errors from misuse cannot happen. The most restrictive access level that makes sense for a particular member *i.e.*, "private" is used unless there is a good reason not to use it.

3.5 Conclusions

This chapter has presented an overview of the object-oriented approach to the *Riemann2D* design. However, it should be noted that the design is based on the basics of CFD modelling, which can easily be adopted for any other CFD model. This design has following benefits:

- It helps to classify the code of the model into various classes. These classes (node, sides, elements, mesh and solver) can be formed for any CFD code.
- Classification of code make the development process faster and easier to debug.
- It helps to reuse the previous work, thus reducing the effort in creating new models.

- The simple design helps the new developer to quickly understand the model design.

The knowledge of approach to the design, presented in this chapter, has been used in the chapters 4, 5 and 6 to link *Riemann2D* design with the theory. Detailed design of the classes is presented in chapter 7.

CHAPTER 4

Finite Volume Methods for Hyperbolic Problems

4.1 Introduction

One of the tasks in developing *Riemann2D* is to separate the commonly used functionalities of the hyperbolic models into a generic package (see section 3.4.2). This chapter presents these common features, based on which the object-oriented *data components* are created for generic package objects (or classes) (see Figure 3.4).

The objective of this chapter is to present:

- the numerical solution for the hyperbolic problems using finite volume methods (one-dimensional formulation followed by the two-dimensional formulation);
- the problems due to discontinuity and its solution (the Riemann problem and solvers for its solution);
- the procedure for mesh selection, formation of control volume, approach to solve problems on these control volume and the CFL condition for the stable solution;
- generic boundary condition; and
- the object-oriented implementation of the theory in the *Riemann2D* design.

Few important things to note before reading this chapter:

- The gray object-oriented comment boxes, mentioned in the section 1.4, are only for the discussion on object-oriented implementation.
- The high-resolution methods is a part of generic package, however, it is presented separately in Chapter 5 for a detailed discussion.

4.2 Finite Volume Formulation

The finite volume method (FVM) is a discretization method which is well-suited for the numerical simulation of various types of hyperbolic problems (see section 2.3.4). Some of the important features of the FVM that gives *Riemann2D* flexibility to solve variety of real-world problems are as follows:

- It can be used on arbitrary geometries, using high resolution methods on unstructured meshes, and leads to robust schemes.
- An additional feature is the local conservation of the numerical fluxes, that is the numerical flux is conserved from one discretization cell to its neighbour.
- Conservation of these numerical fluxes makes it quite attractive when modelling problems for which the flux is of importance, such as in fluid mechanics.

This conservation property of FVM is because the method is based on a *balance* approach[1] by the divergence formula, an integral formulation of the fluxes over the boundary of the control volume is then obtained. The fluxes on the boundary are discretized with respect to the discrete unknowns.

Finite volume methods are closely related to finite difference methods, and a finite volume method can often be interpreted directly as a finite difference approximation to the differential equations. In the next section a general formulation, adapted from LeVeque (2002), for the one-dimensional problem is derived for completeness of the two-dimensional formulation, presented in the following section 4.2.2.

[1]A local balance is written on each discretized cell which is often called *control volume* (discussed in section 4.4.4)

4.2.1 The One-Dimensional Problem

In this section the finite volume methods for the basic solution of conservation laws and hyperbolic systems are derived. The methods are based on subdividing the spatial domain into intervals (the *finite volumes*) and keeping track of an approximation to the integral of u over each of these volumes. In each time step these values are updated using an approximation to the flux through the endpoints of the intervals.

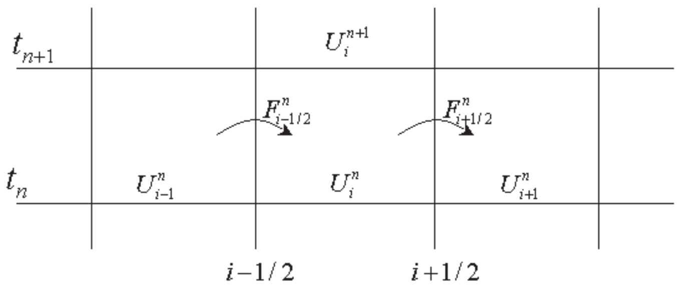


Figure 4.1: Illustration of a finite volume method for updating the cell average U_i^n by fluxes at the cell edges. Shown in $x-t$ space

For simplicity the grid is assumed to be uniform. Now, we denote the i^{th} grid cell by $\Omega_i = (x_{i-1/2}, x_{i+1/2})$ (Figure 4.1). The value in the cell, U_i^n (called state variable in this work), approximates the average value of $u(x,t)$ over the i^{th} interval at time t_n:

$$U_i^n \approx \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} u\left(x, t_n\right) \mathrm{d}x \equiv \frac{1}{\Delta x}\int_{\Omega_i} u(x, t_n)\,\mathrm{d}x, \tag{4.1}$$

where $\Delta x = x_{i+1/2} - x_{i-1/2}$, *i.e.*, the length of the cell.

If $u(x,t)$ in equation 4.1 is a smooth function, then this integral equation agrees with the value of u at the midpoint of the interval to second order of cell length ($O(\delta x^2)$). By working with cell averages, however, it is easier to use important properties of the conservation law in deriving basic numerical methods. Later high-resolution methods are used to determine the values at each side of a cell/element (discussed in Chapter 4). In particular, it can be assumed that the numerical method is conservative in a way that mimics the true solution, and this is extremely important in accurately calculating *shock waves*. This is because $\sum_{i=1}^{N} U_i^n \Delta x$ approximates the integral of u over the entire interval, say $[a,b]$, and if we use a method that is in conservation form, then this discrete sum will change only due to fluxes at the boundaries *i.e.*, at $x = a$ and at $x = b$. Therefore, the total mass within the computational domain will be preserved, or will vary correctly with the given boundary conditions.

The integral form of the conservation law (equation 2.6) gives:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega_i} u(x,t)\mathrm{d}x = f\left(u\left(x_{i-1/2,t}\right)\right) - f\left(u\left(x_{i+1/2,t}\right)\right). \tag{4.2}$$

This expression can be used to develop an explicit time-marching algorithm. Given U_i^n, the cell averages at time t_n, we want to approximate U_i^{n+1}, the cell averages at the next time t_{n+1} after a time step of length $\Delta t = t_{n+1} - t_n$. Integrating equation 4.2 in time from t_n to t_{n+1} yields

$$\int_{\Omega_i} u\left(x,t_{n+1}\right)\mathrm{d}x - \int_{\Omega_i} u\left(x,t_n\right)\mathrm{d}x = \int_{t_n}^{t_{n+1}} f\left(u\left(x_{i-1/2},t\right)\right)\mathrm{d}t - \int_{t_n}^{t_{n+1}} f\left(u\left(x_{i+1/2},t\right)\right)\mathrm{d}t. \tag{4.3}$$

Rearranging this and dividing by Δx gives

$$\begin{aligned}\frac{1}{\Delta x}\int_{\Omega_i} u\left(x,t_{n+1}\right)\mathrm{d}x &= \frac{1}{\Delta x}\int_{\Omega_i} u\left(x,t_n\right)\mathrm{d}x \\ &\quad - \frac{1}{\Delta x}\left[\int_{t_n}^{t_{n+1}} f\left(u\left(x_{i+1/2},t\right)\right)\mathrm{d}t - \int_{t_n}^{t_{n+1}} f\left(u\left(x_{i-1/2},t\right)\right)\mathrm{d}t\right].\end{aligned} \tag{4.4}$$

In general, however, the time integrals on the right-hand side of equation 4.4 cannot be evaluated exactly, since $u(x_{i\pm1/2},t)$ varies with time along each edge of the cell, and there is no exact solution to work with. Therefore, numerical methods of the form

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - F_{i-1/2}^n\right), \tag{4.5}$$

where $F_{i-1/2}^n$ is some approximation to the average flux along $x = x_{i-1/2}$, can be studied:

$$F_{i-1/2}^n \approx \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} f\left(u\left(x_{i-1/2},t\right)\right)\mathrm{d}t. \tag{4.6}$$

If this average flux can be approximated based on the values U^n, then a fully discrete method will be obtained.

In a hyperbolic system information propagates with finite speed, so it is reasonable to first suppose that we can obtain $F_{i-1/2}^n$ based only on the values U_{i-1}^n and U_i^n, the cell averages on either side of this interface. Then we might use a formula of the form

$$F_{i-1/2}^n = \mathfrak{f}(U_{i-1}^n, U_i^n), \tag{4.7}$$

where $\mathfrak{f}$ is some numerical *flux function*. The equation 4.5 then becomes

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x}\left[\mathfrak{f}(U_i^n, U_{i+1}^n) - \mathfrak{f}(U_{i-1}^n, U_i^n)\right]. \tag{4.8}$$

The specific method obtained depends on how we choose the formula $\mathfrak{f}$, but in general any method of this type is an explicit method with a three-point stencil, meaning that the value U_i^{n+1} will depend on the three values U_{i-1}^n, U_i^n, and U_{i+1}^n at the previous time level. Moreover, it is said to be in *conservation form*, since it mimics the property 4.4 of the exact solution. Note that if we sum $\Delta x U_i^{n+1}$ from equation 4.5 over any set of cells, we obtain

$$\Delta x \sum_{i=I}^{J} U_i^{n+1} = \Delta x \sum_{i=I}^{J} U_i^n - \frac{\Delta t}{\Delta x}\left(F_{J+1/2}^n - F_{I-1/2}^n\right). \tag{4.9}$$

The sum of flux differences cancels out except for the fluxes at the extreme edges. Over the full domain we have exact conservation except for fluxes at the boundaries. The method 4.8 can be viewed as a direct difference approximation to the conservation law $u_t + f(u)_x = 0$, since rearranging it gives

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + \frac{F_{i+1/2}^n - F_{i-1/2}^n}{\Delta x} = 0. \tag{4.10}$$

In the above equation flux F is a function of the state variable U. Therefore, flux can also be written as $F(U)$ and the equation 4.10 as:

$$U_t + F\left(U\right)_x = 0, \tag{4.11}$$

where

$$U = \begin{bmatrix} u_1 \\ u_2 \\ : \\ : \\ u_m \end{bmatrix}, F\left(U\right) = \begin{bmatrix} f_1 \\ f_2 \\ : \\ : \\ f_m \end{bmatrix}. \tag{4.12}$$

In the above equation 4.11, U is known as the vector of *conserved* or *state variables*, $F = F(U)$ is the vector of *fluxes* and each of its components f_i is a function of the components u_j of U [Toro (1999)]. The Jacobian of the flux function $F(U)$ in the equation 4.11 is the matrix

$$A(U) = \frac{\partial F}{\partial U} = \begin{bmatrix} \partial f_1/\partial u_1 & .. & .. & \partial f_1/\partial u_m \\ \partial f_2/\partial u_1 & .. & .. & \partial f_2/\partial u_m \\ : & : & : & : \\ : & : & : & : \\ \partial f_m/\partial u_1 & .. & .. & \partial f_m/\partial u_m \end{bmatrix} . \tag{4.13}$$

Object-Oriented Design

The vectors of state variables, U and the fluxes, $F(U)$ is equations 4.12 are required by every hyperbolic system. However, the components u_j and f_i of these vectors are different for every system, which depends on the system of governing equations. Therefore, the vectors U and $F(U)$ are declared in the GenericElement class and the components u_j and f_i are initialised and calculated in the Element class (subclass of GenericElement).

4.2.2 The Two-Dimensional Problem

Based on the understanding of the one-dimensional formulation it is easy to extend it for the two-dimensional case. The equation 4.11 has flux in the x-direction, which can be extended for two-dimensional problems by adding an addition flux term in y-direction. Now, the differential conservation law form in two-dimensional can be written as:

$$U_t + F\left(U\right)_x + G\left(U\right)_y = S(U) \tag{4.14}$$

Here $S = S(U)$ is a *source* or *forcing term*. Terms such as body forces (*e.g.*, gravity) and injection of mass, momentum or energy may be represented in S. Usually, S(U) is a prescribed algebraic function of the flow variables and does not involve derivatives of these, but there are exceptions[Toro (1999)]. In the present case the equation 4.14 is *inhomogeneous*, but when $S(U) \equiv 0$ the equation 4.14 is *homogeneous*. From now, we would consider the homogeneous form of the equation because the source term depends on the system[1] for which the equations are applied. Therefore, rewriting equation 4.14 in the homogeneous form:

[1]The source term for 2D shallow water hyperbolic equations are discussed in the Chapter 6.

$$U_t + F\left(U\right)_x + G\left(U\right)_y = 0. \tag{4.15}$$

The corresponding integral form of the above equation (4.15) can be written as:

$$\frac{\mathrm{d}}{\mathrm{dt}} \int\int_V U \mathrm{d}V = -\int_\Omega H.n\mathrm{d}\Omega, \tag{4.16}$$

where V is a control volume, Ω is the boundary of V, $H = (F, G)$ is the tensor of the flux, $n = [n_1, n_2]$ is the outward unit vector normal to the surface Ω, $\mathrm{d}\Omega$ is an area element and $H.n\mathrm{d}\Omega$ is the flux component normal to the boundary Ω. The cell average for this control volume can be defined as:

$$\hat{U} = \frac{1}{|V|} \int\int_V U \mathrm{d}V, \tag{4.17}$$

where $|V|$ denotes the volume of V. Substituting the values from equation 4.17 into the equation 4.16 and rewriting the right hand flux in terms of summation, gives

$$\frac{d}{dt}\hat{U} = -\frac{1}{|V|}\sum_{s=1}^{N} F_s. \tag{4.18}$$

with fluxes

$$F_s = \int_{A_s}^{A_{s+1}} \left[n_1 F\left(U\right) + n_2 G\left(U\right)\right] d\Omega. \tag{4.19}$$

where N is the number of sides of control volume[1].

Expression 4.18 forms the basis of *semi-discrete* numerical methods, in that the right-hand side is assumed to be discretised in *space*, leaving the left-hand side continuous in *time*. Now replacing the time derivative by a forward in time approximation, we obtain *fully discrete scheme*

[1]The expression in equation 4.19 is valid for any number of sides of a control volume. However, in the present case the control volume is triangular in shape. Section on control volume is discussed later in section 4.4.4

$$U^{n+1} = U^n - \frac{\Delta t}{A} \sum_{s=1}^{N=3} [n_1 F\,(U) + n_2 G\,(U)]. \tag{4.20}$$

Object-Oriented Design

The equation 4.20 is used to update the vector of state variables (defined in equation 4.12) at every time step of the calculation. This operation is common for every hyperbolic system, which makes it a method of the GenericElement class.

It should be noted that the fluxes $F(U)$ and $G(U)$ of equation 4.20 are the side fluxes and different than the fluxes of equation 4.15, which are the property of the element. These fluxes of equation 4.20 are properties of the sides of a control volume, therefore, they are declared in the GenericSide class. Similar to the fluxes of the elements, these fluxes of sides are calculated in the subclass, *i.e.*,Side class.

Various methods exist to define the numerical flux function of equation 4.20. For an easy understanding we start with the equation 4.7. There are several considerations that go into judging how good a particular flux function is for numerical computation. One essential requirement is that the resulting method should be convergent, *i.e.*, the numerical solution should converge to the exact solution of the differential equation as the grid is refined (as δx , $\Delta t \to 0$). This generally requires two conditions:

- The method must be consistent with the differential equation, meaning that it approximates it well locally;
- The method must be stable in some appropriate sense, meaning that the small errors made in each time step do not grow too fast in later time steps.

For a general finite volume method (see Figure 4.1) for a hyperbolic system there are a number of ways which might be defined. The average flux at $x_{i-1/2}$ is calculated using the state variables at the left(U_{i-1}^n) and right(U_i^n) of this point. A first attempt might be a simple arithmetic average

$$F_{i-1/2}^n = \mathcal{F}(U_{i-1}^n, U_i^n) = \frac{1}{2}\left[f(U_{i-1}^n) + f(U_i^n)\right], \tag{4.21}$$

using this in equation 4.5 would give,

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{2\Delta x}\left[f(U_{i+1}^n) - f(U_{i-1}^n)\right]. \tag{4.22}$$

Unfortunately, this method is unstable for hyperbolic problems and cannot be used even if the time step is small enough that the CFL condition is satisfied [LeVeque (2002)]. This supports the need for the *high-resolution* methods to produce stable solutions.

The application of high-resolution methods has caused a revolution in solving numerical hyperbolic problems. These methods satisfy in a quite natural way the basic properties required for any acceptable numerical method [Marti and Muller (2003)]:

- high order of accuracy,
- stable and sharp description of discontinuities, and
- convergence to the physically correct solution.

Moreover, these high resolution methods are conservative, and because of their shock capturing property discontinuous solutions are treated both consistently and automatically whenever and wherever they appear in the flow. High resolution is usually achieved by using monotonic polynomials in order to interpolate the approximate solutions within numerical cells. A detailed discussion on high resolution methods is presented in Chapter 4 of this thesis.

As these methods are written in conservation form, the time evolution of element averaged state vectors is governed by some functions (the numerical fluxes) evaluated at element interfaces separated by constant states at the two sides, also known as the *Riemann problem* (discussed in next section).

4.3 The Riemann Problem

As discussed in the previous sections on finite volume formulation (section 4.2) and on the control volume (section 4.4.4), the numerical flux is calculated at each side of the control volume with two states separated by a discontinuity. A simplified case of one spatial dimension is show below in which the Riemann problem is the simplest possible initial value problem of discontinuity for hyperbolic systems. Apart from being an important test bench, the Riemann problem is a basic building block for a large class of modern numerical methods, called *upwind* or *Godunov schemes* [Kurganov and Tadmor (2002)].

The Riemann problem consists of computing the breakup of a discontinuity, which initially separates two arbitrary constant states **L**(left) and **R** (right). The solution to this problem depends on the two constant states defining the discontinuity.

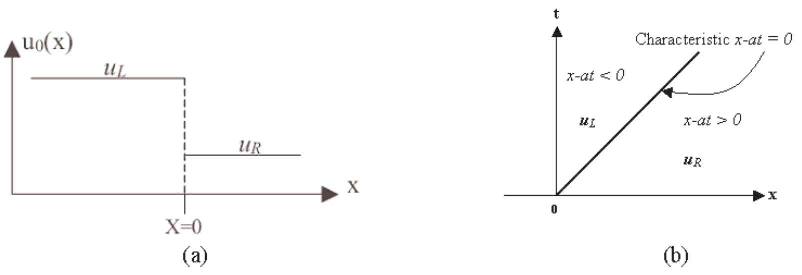


Figure 4.2: Illustration of the Riemann problem (a) Initial data at $t = 0$, consists of two constant states separated by a discontinuity at $x = 0$. (b) Solution in the $x - t$ plane for the linear advection equation with positive characteristic speed a.

Mathematically, a special initial value problem (IVP) of the form:

$$PDE \qquad u_t + au_x = 0 \tag{4.23}$$

$$IC \qquad u(x,0) = u_0(x) = \begin{cases} u_L & \text{if} \quad x < 0, \\ u_R & \text{if} \quad x > 0, \end{cases} \tag{4.24}$$

is called a Riemann problem [Toro (1999)], where u_L and u_R are two constant values, as shown in Figure 4.2. The initial data has a discontinuity at $x = 0$. The trivial case would result when $u_L = u_R$. The solution to this problem helps to obtain information that is used to compute a numerical flux and update the cell averages over a time step. For hyperbolic problems the solution to the Riemann problem is typically a similarity solution, a function of x/t alone (see Figure 4.2(b)), and consists of a finite set of waves that propagate away from the origin with a constant wave speed. For linear hyperbolic systems the Riemann problem is easily solved in terms of the eigenvalues and eigenvectors of the matrix A, (see section 6.5 and 6.6.1 for matrix A, eigenvalues and eigenvectors of 2D shallow water equations). In a numerical method we solve these Riemann problems using the Riemann solvers, as discussed in the next section.

Object-Oriented Design

Riemann problem is solved at every side within the mesh domain and in some cases also solved at the boundary sides (see section 4.5). To solve a Riemann problem of a side, left and right states are required. These left and right states are set by a method called *setStateAtSides()* in GenericMesh class, which cycles through all the elements of the mesh to set the state at the element's sides.

4.3.1 Riemann Solvers: Exact Vs Approximate

Numerical methods for solving systems of hyperbolic conservation laws *via* finite volume methods require, as the building block, a numerical flux. The choice of the building block has a profound influence on the properties of the resulting schemes. One approach utilizes wave propagation information contained in the differential equations [Toro and Titarev (2006)] and the solution of such problems is governed by the hyperbolic properties of the system. It is built from simple waves, either discontinuities, called *shock waves* or *rarefaction fans*, which have to be assembled appropriately. This is done through the *exact* or *approximate solution* of the Riemann problem, giving rise to upwind methods [classical references: Courant *et al.* (1952), Godunov (1959), Roe (1981) and vanLeer (1974)]. A Few recent references on these methods are Godlewski and Raviart (1996), Toro (1999), Kulikovskii *et al.* (2002) and LeVeque (2002).

The exact solution of the one-dimensional Riemann problem is well-known [see chapters 5, 6 and 7 of Toro (2001)], which consists of a combination of wave types: shocks (S) and rarefaction waves (R). Toro (2001) states that making a choice between exact and approximate Riemann solver is motivated by:

1. computational cost;
2. simplicity; and
3. correctness

where correctness should be the overriding criterion. Arguments of computational cost is one of the important factors, but depends on the problem to be solved. For instance, in the shallow water equations the arguments of computational cost is not as strong as for the Euler equations or some other complex systems of equations. In the study of real compressible gases, the adoption of complicated equations of states that make use of exact Riemann solvers is prohibitively expensive. LeVeque (2002) states that computationally, the exact Riemann solution is often too expensive to compute for non-linear problems and approximate Riemann solvers are used in implementing numerical methods.

Therefore, as a general solution and choice, approximate Riemann solvers are used in this work.

4.3.2 The Approximate Riemann Solvers

To apply Godunov's method on a system of equations we need flux F_s (equation 4.19), based on the Riemann data u_l and u_r (Figure 4.2). One of the benefits of these problems is that we do not need the entire structure of the Riemann problem. Usually, we compute F_s across each side with the help of full wave structure and wave speeds. LeVeque (2002) states that the process of solving the Riemann problem is often quite expensive, even though in the end we use very little information from this solution in defining the flux. Also, it is often true that it is not necessary to compute the exact solution to the Riemann problem in order to obtain good results.

A wide variety of approximate Riemann solvers have been proposed that can be applied much more cheaply than the exact Riemann solver and yet give results that in many cases are equally good when used in the Godunov or high-resolution methods. In the present study we have carefully selected three Riemann solvers to understand the behaviour of the Riemann solvers for two-dimensional triangular control volumes. A variety of other approximate Riemann solver can be found in Godlewski and Raviart (1996), Kurganov and Tadmor (2000) and Kroner *et al.* (1995).

Object-Oriented Design

In this work three approximate Riemann solvers are used *i.e.*, Roes', HLL and Shu & Oshers' solver. All these solvers calculate the side fluxes and supply it to the the *update* method in the GenericElement class.

4.3.3 Roe's Solver

Perhaps the most well-known of all approximate Riemann solvers today is one due to Roe, which was first presented in 1981 [Roe (1981)]. Since then the method has not only been refined, but it has been applied to a variety of physical problems. Appropriate adoption and modification to the Roe approach was introduced by Roe and Pike (1984), Harten and Hyman (1983), Roe (1992), Dubois and Mehlman (1993) and Einfeldt *et al.* (1991). Many application of the Roe scheme have been presented over the last two decades, a few of them were by; Brio and Wu (1988) for Magneto-Hydrodynamic equations (MDH), Clarke *et al.* (1993) for detonation wave in solid materials, Giraud and Manzini (1996) for two-dimensional gas dynamics, Marx (1994) for incompressible Navier-Strokes equations, and Sainsaulieu (1995) for multiphase flow problems.

Due to the wide application of Roe's solver and it capability of good resolution of the waves, it has been considered in the present study.

In this section we describe the Roe approach for a general system of m hyperbolic conservation laws (adapted from Toro (1999)).

Roe (1981) solved the Riemann problem (equations 4.23 and 4.24) approximately. By introducing the Jacobian matrix (see equation 4.13)

$$A(U) = \frac{\partial F}{\partial U}, \tag{4.25}$$

and using the chain rule the conservation laws

$$U_t + FU_x = 0, \tag{4.26}$$

in equation 4.23 may be written as

$$U_t + A(U)U_x = 0. \tag{4.27}$$

Roe's approach replaces the Jacobian matrix $A(U)$ in equation 4.27 by a constant Jacobian matrix

$$\tilde{A} = \tilde{A}(U_L, U_R), \tag{4.28}$$

which is a function of the data states U_L, U_R. In this way the original PDEs in equation 4.23 are replaced by

$$U_t + \tilde{A}U_x = 0. \tag{4.29}$$

This is a linear system with constant coefficients. The original Riemann problem 4.23 and 4.24 is then replaced by the *approximate Riemann problem.*

$$\left.\begin{array}{l} U_t + \tilde{A}U_x = 0 \\ U_x = \begin{cases} U_L, & x < 0 \\ U_R, & x > 0 \end{cases} \end{array}\right\}, \tag{4.30}$$

which is then solved exactly. The approximate problem results from replacing the original non-linear conservation laws by a linearised system with constant coefficients but the initial data of the exact problem is retained.

For a general hyperbolic system of m conservation laws, the Roe Jacobian matrix is required to satisfy the following properties:

<u>PROPERTY (A)</u>: Hyperbolicity of the system. $\tilde{A}$ is required to have real eigenvalues $\tilde{\lambda}_i = \tilde{\lambda}_i(U_L, U_R)$, which are normally ordered as

$$\bar{\lambda}_1 \leq \bar{\lambda}_2 \leq .... \leq \bar{\lambda}_m, \tag{4.31}$$

and a complete set of linearly independent right eigenvectors

$$\tilde{K}^{(1)}, \tilde{K}^{(2)}, ...\tilde{K}^{(m)}. \tag{4.32}$$

<u>PROPERTY (B)</u>: Consistency with the exact Jacobian

$$\tilde{A}\left(U, U\right) = A\left(U\right). \tag{4.33}$$

<u>PROPERTY (C)</u>: Conservation across discontinuities

$$F\left(U_R\right) - F\left(U_L\right) = \tilde{A}\left(U_R - U_L\right). \tag{4.34}$$

Property (A) on hyperbolicity is an obvious requirement; the approximate problem should at the very least preserve the mathematical character of the original non-linear system. Property (B) ensures consistency with the conservation laws. Property (C) ensures conservation. It also ensures exact recognition of isolated discontinuities; that is, if the data U_L, U_R are connected by single isolated discontinuities, then the approximate Riemann solver recognises this wave exactly. However, this does not mean that the corresponding approximate Godunov method with the Roe approximate numerical flux will in general give exact solutions for isolated discontinuities.

The construction of matrices satisfying properties(A)-(C) for general hyperbolic systems can be very complicated and thus computationally unattractive. For a specific case of the Euler equations of gas dynamics Roe (1981) proposed a relatively simple way of constructing a matrix $\tilde{A}$.

4.3.3.1 The Intercell Flux

Once the matrix $\tilde{A}\left(U_L, U_R\right)$, its eigenvalues $\tilde{\lambda}\left(U_L, U_R\right)$ and the right eigenvector $\tilde{K}^{(i)}\left(U_L, U_R\right)$ are available, one solves Riemann problem (equation 4.30) by direct application of various methods like CIR scheme, Godunov's method *etc.* By projecting the data difference

$$\Delta U = U_R - U_L \tag{4.35}$$

onto the right eigenvectors we can write

$$\Delta U = U_R - U_L = \sum_{i=1}^{m} \tilde{\alpha}_i \tilde{K}^{(i)}, \tag{4.36}$$

from which one finds the wave strengths $\tilde{\alpha}_i = \tilde{\alpha}_i\left(U_L, U_R\right)$. The solution $U_{I+1/2}(x/t)$ evaluated along the $t-$axis, $x/t = 0$, is given by

$$U_{i+1/2}(0) = U_L + \sum_{\tilde{\lambda}_i \leq 0} \tilde{\alpha}_i \tilde{K}^{(i)}, \tag{4.37}$$

or

$$U_{i+1/2}(0) = U_R - \sum_{\tilde{\lambda}_i \leq 0} \tilde{\alpha}_i \tilde{K}^{(i)}. \tag{4.38}$$

We can now find the corresponding numerical flux. As we have replaced the original set of conservation laws in equation 4.23 by the constant coefficient linear system (equation 4.29); this can be viewed as a modified system of conservation laws

$$\overline{U} + \overline{F}\left(\overline{U}\right)_x = 0, \tag{4.39}$$

with the flux function

$$\overline{F}\left(\overline{U}\right) = \tilde{A}\overline{U}. \tag{4.40}$$

The corresponding numerical flux is not the obvious choice,

$$F_{i+1/2} = \tilde{A}\overline{U}_{i+1/2}(0) \tag{4.41}$$

where $\overline{U}_{i+1/2}(0)$ is given by either of equations 4.37-4.38. That this would be incorrect becomes obvious when, for instance, assuming right supersonic flow in 4.37 one would compute an intercell flux. Instead, the correct expression for the corresponding numerical flux is obtained from any of integral relations

$$F_{0L} = F_L - S_L U_L - \frac{1}{T}\int_{TS_L}^{0} U\left(x, T\right) \mathrm{d}x \tag{4.42}$$

$$F_{0R} = F_R - S_R U_R + \frac{1}{T}\int_0^{TS_R} U\left(x,T\right)\mathrm{d}x. \tag{4.43}$$

Here S_L, S_R are the smallest and largest signal speeds in the exact solution of the Riemann problem with data U_L,U_R and T is a positive time. If the integrand $U(x,t)$ in equation 4.42 or equation 4.43 is replaced by some approximate solution, then equality of the fluxes F_{0L} and F_{0R} requires the approximate solution to satisfy a consistency condition.

If $\overline{U}_{i+1/2}(x/t)$ is the solution of the Riemann problem for the modified conservation laws 4.39 with data U_L, U_R, then the integrals in 4.42 and 4.43 respectively, are

$$\int_{TS_L}^{0} \overline{U}_{i+1/2}\left(x,T\right)\mathrm{d}x = T\left[\overline{F}\left(U_L\right) - \overline{F}\left(\overline{U}_{i+1/2}(0)\right)\right] - TS_L U_L \tag{4.44}$$

and

$$\int_0^{TS_R} \overline{U}_{i+1/2}\left(x,T\right)\mathrm{d}x = T\left[\overline{F}\left(\overline{U}_{i+1/2}(0)\right) - \overline{F}\left(U_R\right)\right] - TS_R U_R. \tag{4.45}$$

Substitution of equations 4.44 and 4.45 into equations 4.42 and 4.43 gives

$$F_{0L} = \overline{F}\left(\overline{U}_{i+1/2}(0)\right) + F\left(U_L\right) - \overline{F}\left(U_L\right) \tag{4.46}$$

and

$$F_{0R} = \overline{F}\left(\overline{U}_{i+1/2}(0)\right) + F\left(U_R\right) - \overline{F}\left(U_R\right). \tag{4.47}$$

Finally by using $\overline{U}_{i+1/2}(0)$ as given by equation 4.42 or 4.43 and the definition of the flux $\overline{F} = \tilde{A}\overline{U}$ we obtain the numerical flux F_{Roe} at the boundary of the element as:

$$F_{Roe} = F_L + \sum_{\tilde{\lambda}_i \le 0} \tilde{\alpha}_i \tilde{\lambda}_i \tilde{K}^{(i)}, \tag{4.48}$$

or

$$F_{Roe} = F_R - \sum_{\tilde{\lambda}_i \ge 0} \tilde{\alpha}_i \tilde{\lambda}_i \tilde{K}^{(i)}. \tag{4.49}$$

Alternatively, we may also write

$$\boxed{F_{Roe} = \frac{1}{2}\left(F_L + F_R\right) - \frac{1}{2}\sum_{i=1}^{m} \tilde{\alpha}_i \tilde{\lambda}_i \tilde{K}^{(i)}.} \tag{4.50}$$

The relations 4.44-4.50 are valid for any hyperbolic system and any linearisation of it. In order to compute Roe's numerical flux for a particular system of hyperbolic conservation laws, one requires expressions for the wave strengths $\tilde{\alpha}_i$, the eigenvalues, and the right eigenvectors in any flux expressions 4.48-4.50. It is important to note that the Jacobian matrix is not explicitly required by the numerical flux. This method is further derived for the shallow water equations and used in the present study (see section 6.6.1).

4.3.4 HLL (Harten-Lax-van Leer) Solver

The HLL Riemann solver is another class of linear approximation methods, originally developed for the Euler equations of gas dynamics by Harten *et al.* (1983), that are extended for many other equations (see section 6.6.2 on HLL sovers for shallow water equations). These solvers are based on estimating the speeds at which information or waves propagate away from a Riemann problem. A linear solution with two discontinuities is then constructed[1], using estimates for the speeds of the propagating discontinuities[2]. The estimates for the speeds are based on the initial data, and general properties of exact Riemann solutions [George (2004)]. The approach produced practical schemes after the contributions of Davis (1988) and Einfeldt (1988), who independently proposed various ways of computing the wave speeds required to completely determine the intercell flux. The resulting Riemann solver forms the basis of a very efficient and robust approximation to Godunov-type methods. One difficulty with these schemes is the assumption of a two-wave configuration. This is true only for a hyperbolic systems of two equations, such as the one-dimensional shallow water equations. For larger systems, such as the Euler equations or the split two-dimensional shallow water equations for example, the two-wave assumption is incorrect [Toro (1999)]. Therefore, HLL solvers gave rise to family of solvers for example,

- **HLLE** (Harten-Lax-vanLeer-Einfeldt): Keeps only largest and smallest characteristics, averages intermediate states in-between [Einfeldt (1988)].

[1]Two speeds are used in the original HLL method even for equations with more than two characteristic families.

[2]Contrast this to a method such as the Roe solver, where a constant estimate to the Jacobian matrix is constructed first, and the eigenvalues of this estimate subsequently affect the approximate Riemann solution.

- **HLLC** (Harten-Lax-vanLeer and "C" denotes Contact): Adds entropy wave back into HLLE method, giving two intermediate states [Toro *et al.* (1994)].
- **HLLD** (Harten-Lax-vanLeer and "D" stands for Discontinuities): The HLLC-type Riemann solvers for magnetohydrodynamic (MHD) [Miyoshi and Kusano (2005)].

Due to the simple, efficient and robust of the HLL solver, it has been one of the Riemann solvers considered in the present study. One other advantage is that these solver can be easily extended to its family of solvers.

Harten *et al.* (1983) suggested a way of solving the Riemann problems approximately by finding directly an approximation to the numerical intercell flux as shown below (adapted from Miyoshi and Kusano (2005)).

Let us consider general hyperbolic conservation laws (equations 2.4), where U and F are not specified here. The integral form of the conservation laws for a rectangle in figure 4.3 $(x_1, x_2) \times (t_1, t_2)$ is given by

$$\int\limits_{x_1}^{x_2} U\left(x, t_2\right) - \int\limits_{x_1}^{x_2} U\left(x, t_1\right) + \int\limits_{t_2}^{t_2} F\left(U\left(x, t_2\right)\right) dt - \int\limits_{t_1}^{t_2} F\left(U\left(x, t_2\right)\right) dt = 0. \tag{4.51}$$

Harten *et al.* (1983) showed that the Godunov-type scheme for equation 2.4 can be written in conservative form:

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x}\left[F\left(R\left(0; U_i^n, U_{i+1}^n\right)\right) - F\left(R\left(0; U_{i-1}^n, U_i^n\right)\right)\right], \tag{4.52}$$

where n and i indicate a time step and a element/cell number, respectively, and $R\left(x/t; U_{i-1}^n, U_i^n\right)$ is the approximate solution of the Riemann problem around the interface $x_{i+1/2}$. In this form, the appropriate numerical fluxes are obtained by applying the integral conservation laws 4.51 over the rectangle $(x_i, x_{i+1/2}) \times (t^n, t^{n+1})$ (Figure 4.1) as

$$F_{i+1/2} = F_i - \frac{1}{\Delta t} \int\limits_{x_i}^{x_{i+1/2}} R\left(\frac{x - x_{i+1/2}}{\Delta t}; U_i^n, U_{i+1}^n\right) \mathrm{d}x + \frac{x_{i+1/2} - x_i}{\Delta t} U_i^n, \tag{4.53}$$

where $F_{i+1/2} = F\left(R\left(0; U_i^n, U_{i+1}^n\right)\right)$, $F_i = U_i^n$ and $\Delta t = t^{n+1} - t^n$. We note that the exact solution of the Riemann problem R_{exact} produces the fluxes of the original Godunov scheme. The numerical fluxes $F_{i+1/2}$ obtained by the other integral conservation laws

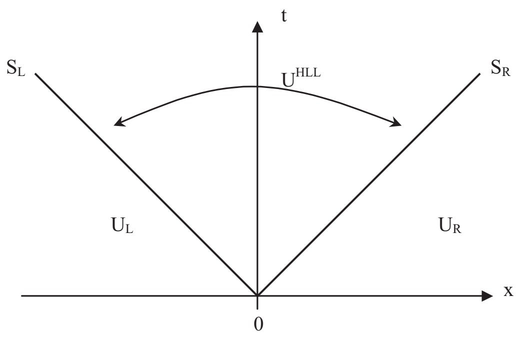


Figure 4.3: Approximate HLL Riemann solver. Solution in the star region consists of a single state U^{HLL} separated from data states by two waves of speeds S_L and S_R

over $(x_{i+1/2}, x_{i+1}) \times (t_n, t_{n+1})$ must coincide with equation 4.52 due to the consistency with the integral form of conservation laws over $(x_i, x_{i+1}) \times (t_n, t_n + 1)$.

The HLL Riemann solver is constructed by assuming an average intermediate state between the fastest and slowest waves. Consider a *subsonic* solution of the single-state approximate Riemann problem at the interface between the left and right states, U_L and U_R, where the minimum signal speed S_L and the maximum signal speed S_R are negative and positive, respectively (Figure 6.5). By applying the integral conservation laws (equation 4.51) over the Riemann fan, $(\Delta t.S_L, \Delta t.S_R) \times (0, \Delta t)$, the intermediate state is given by

$$U^* = \frac{S_R U_R - S_L U_L - F_R + F_L}{S_R - S_L}. \tag{4.54}$$

After that, as denoted by equation 4.52, the integral over $(\Delta t.S_L, 0) \times (0, \Delta t)$ gives the HLL fluxes,

$$F^* = \frac{S_R U_L - S_L U_R + S_R S_L\left(S_R + S_L\right)}{S_R - S_L}. \tag{4.55}$$

If both signal speeds are of the same sign, the fluxes must be evaluated only from the upstream side. Therefore, in general, the HLL fluxes become

$$F_{HLL} = \begin{cases} F_L & if \quad S_L > 0 \\ F^* & if \quad S_L \leq 0 \leq S_L \\ F_R & if \quad S_R < 0. \end{cases} \tag{4.56}$$

4.3.5 Shu and Osher Solver

In the computation of discontinuous solutions of hyperbolic conservation laws, TVD (total-variation- diminishing), TVB (total-variation-bounded), and the EN0 (essentially non-oscillatory) schemes have proven to be very useful. Shu and Osher (1988) discussed two improvements:

1. a simple TVD Runge-Kutta type time discretization, and
2. an EN0 construction procedure based on fluxes rather than on cell averages.

These improvements simplify the schemes considerably specially for multi-dimensional problems with forcing terms. Shu and Osher (1988) finds the solution in one- and multi-space dimension hyperbolic problem. Conservative schemes of the form:

$$u_i^{n+1} = u_i^n - \lambda \left(f_{i+1/2} - f_{i-1/2} \right), \qquad \lambda = \Delta t / \Delta x, \tag{4.57}$$

with a consistent numerical flux

$$f_{i+1/2} = f\left(u_{j-1}, ..., u_{j+K} \right), \tag{4.58}$$

was considered. In order to guarantee that any convergent bounded *i.e.*, subsequence has as its limit a weak solution of equation 4.11. Shu and Osher (1988) constructed the shock capturing methods. The $\lambda = \Delta t / \Delta x$ term in equation 4.57 is the CFL condition. Therefore in a one-dimensional case it should be less than one. Whereas, the two-dimensional schemes are stable under CFL numbers, one-half of those used for one dimension. In two-dimensional problems, stable solutions were observed for $\lambda = 0.4$ and $\lambda = 0.3$.

This simplified method proposed by Shu and Osher (1988) (SO) is adopted in the present study to compare its performance with others, like Roes' and HLL Riemann solvers. This scheme simplifies the equation 4.50 and can be written as

$$F_{SO} = 0.5 \left\{ F_L + F_R - |a| \left(U_R - U_L \right) \right\} \tag{4.59}$$

where $a = \lambda \left| a_{\max} \right|$, wherein $a_{\max}$ = maximum value of the eigenvalues of the average Jacobean matrix (equation 4.13) and λ = a positive coefficient, with $\lambda < 1$ (usually, set between 0.3 - 0.4 for stable solutions).

Object-Oriented Design

The Roes', HLL and Shu & Oshers' solvers are generic in nature, but the left and right element fluxes required to solve the Riemann problem is dependent on the governing equation of a particular hyperbolic system. Therefore, the properties of Riemann solvers (for *e.g.*, wave speed, eigen vlaues, eigen vectors, etc.) are declared in the GenericSide class, and the body of these solver methods are present in the Side Class (subclass of the GenericSide).

4.4 Selection of Mesh and Control Volume for *Riemann2D*

In order to numerically solve PDEs on a surface (for 2D problems), it is necessary to create a mesh on the problem domain. This mesh specifies the points where the equation will be solved as well as the connections between these points. The derivation of a finite volume numerical solution is based on these meshes and control volumes.

4.4.1 Mesh Selection

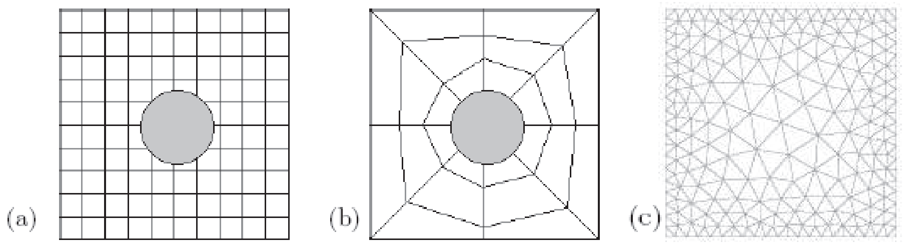


Figure 4.4: (a) Cartesian grid, (b) body-fitted structured grid, (c) unstructured grid with varying element size

One category of meshes are the so-called *structured meshes.* The Cartesian grid is the simplest such example (Figure 4.4a). The solution points are located at the intersection of the grid lines, while the elements are the rectangles determined by the intersecting lines. This type of mesh is extremely simple and quick to generate. However, if the problem to be solved has curved internal and/or external boundaries, solving on a structured Cartesian grid requires one to modify the numerical scheme near these boundaries. This is usually quite difficult. See, for example, Helzel *et al.* (2005).

Another type of structured mesh is the *body-fitted* structured mesh. Here, the grid is still essentially Cartesian, but the quadrilateral elements are shaped to fit the boundary (Figure 4.4b). Numerical methods that were originally developed for Cartesian grids can be modified to body-fitted grids through grid mappings that do not change the essence of the numerical scheme [see LeVeque (2002)]. The difficulty in implementing this method lies in generating the mesh, particularly if there are multiple objects in the interior.

The alternative that we have considered and adapted in this work is *unstructured* (triangular) meshes. Unlike a structured mesh, elements and solution points are not determined by the intersection of sets of parallel lines. One advantage of an unstructured mesh is that points can be set on the boundaries of the surface, allowing for greater accuracy later when applying a numerical method. Further-more, unstructured meshes allow for elements of varying sizes, permitting accurate representation of boundaries without requiring an excessive number of points and elements (Figure 4.4c).

Object-Oriented Design

GenericMesh class reads the mesh from a user supplied mesh file. It then creates the list of sides and elements in the domain and their neighbouring elements. Also creates ghost nodes and ghost elements in the process to facilitate the application of boundary conditions.

4.4.2 Control Volume

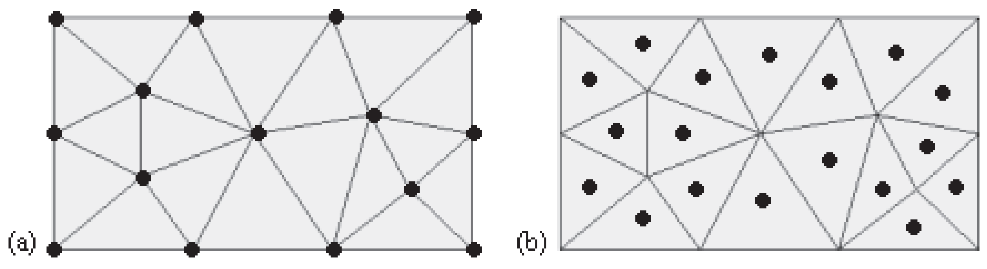


Figure 4.5: Types of two-dimension control volume (a) Node-Centred (b) Element-Centred

Two types of control volumes (CV) can be made in the two-dimensional (2D) triangular mesh : *node-centred* and *element-centred* (Figure 4.5).

In a **node-centred** CV the unknown variables are positioned at the nodes of a elements and the CV is an irregular shape surrounding the corresponding vertex. In Figure 4.6 three possible definitions of the node-centred CV are shown. The *centroid dual* is created by connecting the mid-points of the elements which are joined to the concerned vertex. The *Dirichlet tesselation* is formed by connecting the centres of the circum-circles of the same elements. Finally the *median dual* is obtained by linking the mid-points of the elements and edges around the vertex.

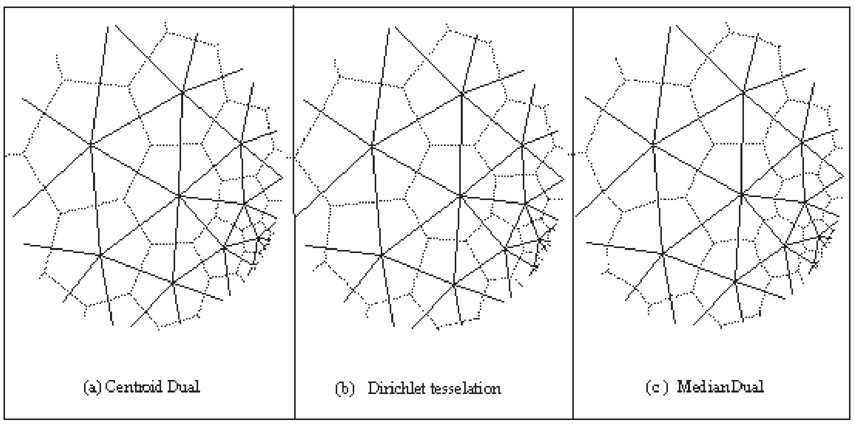


Figure 4.6: Different possible choices of the control volumes for node centered control volume. Dashed lines are the control volume's faces and the solid lines are the elements (triangles)[Namin *et al.* (2004)]

In a **element centered** CV, each triangular element is considered as a control volume and the state variables are located at its mid-point, so that the number of unknown variables is the same as the number of elements or triangles.

In an element-centred CV the Riemann problem is solved for three sides, which is shared with another triangle. Whereas in the node-centred CV the Riemann problem is solved as many times as the number of the mesh elements connected to that particular node. If we assume that the number of boundary sides are negligible compared to the interior sides of the domain, then we can say that the number of Riemann problem solved from each triangle is 1.5 times that for the element-centred CV. Conversely, for node-centred number of Riemann problem for each mesh element is 1. This information is used in table 4.1 to show that total number of Riemann problems solved for a particular domain is the same for element-centred and node-centred CV.

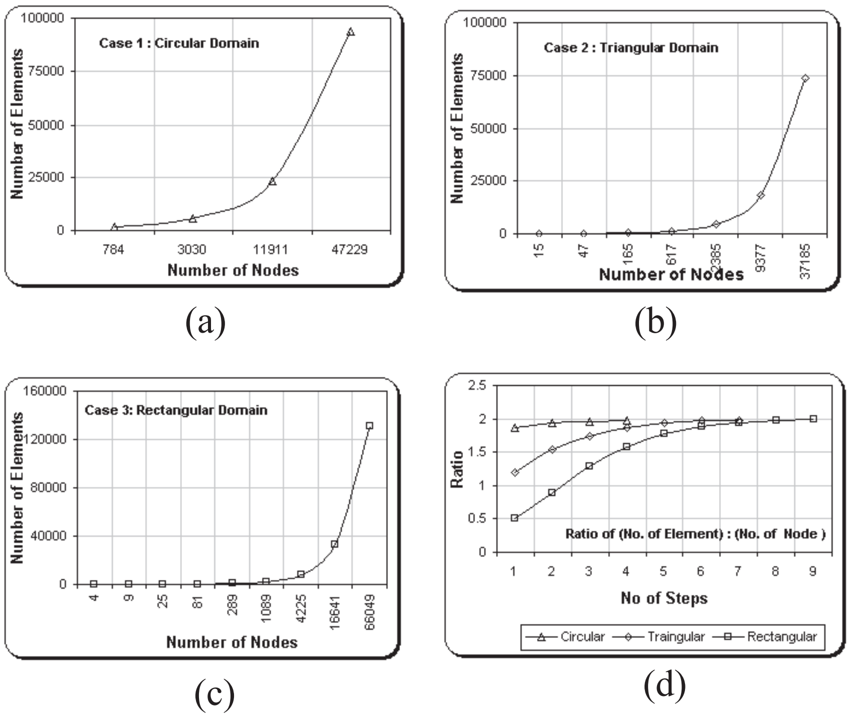


Figure 4.7: Three domains of different shapes and with different level of mesh refinement. (a) Circular Domain (b) Triangular Domain (c) Rectangular Domain (d) Ratio of (No. of Elements) to (No. of Nodes) for all the three cases. It can be seen in Figure (d) that the ratio tends to 2 as the mesh is refined. Note: The "No of Steps" in (d) is the number of times mesh is refined for each domain.

Table 4.1: Comparison of the properties of the *element-centred* and *node-centred* control volumes.

Approximate Ratio of:	Element Centred Control Volume	Node Centred Control Volume
No. of Control Points	2	1
Average no. of Riemann problems solved per CV	1	2
Total No. of Riemann problems solved	1	1

Namin *et al.* (2004) have used both types of grid layout and showed that both types produce reliable predictions. In this work *element-centred* CV are adopted for the following reasons:

1. For a particular mesh, element-centred CV has approximately twice as many computational points as the node-centred CV (see figure 4.7), but the computational cost is approximately same for both the CVs (see table 4.1). Therefore, it can be easily said that the element-centred CV is expected to give more information than the node-centred CV.

2. In a element-centred CV, an element of mesh itself is considered to be a CV, this save computation time compared to the node-centred CV, where an extra computation effort is required to create CV around each node.

3. Applying boundary conditions in element-centred CV is easier than applying them to node-centred CV. Mirror image of the boundary element are created and called Ghost Elements (see section 4.5), which easily implements the boundary condition to the system. Also in cases where boundary fluxes have to be specified it can be easily done, as the boundary of the domain and the centre of the element coincide with each other.

Object-Oriented Design

The control volumes coincide with the elements of the mesh. Therefore, the generic properties of a control volume (*i.e.*, state variables, fluxes, area etc.) are defined and stored in the GenericElement class. Whereas, the system specific properties (*e.g.*, bed depth for a shallow water problem) are defined and stored in the Element class (subclass of GenericElement).

4.4.3 Approach to Triangular Control Volume

The traditional approach for the development of schemes to solve numerical systems of m hyperbolic conservation laws in several space dimensions was as follows; First one develops a scheme for the one dimensional problem, a good introduction to a large class of one dimensional schemes can be seen in LeVeque (1992). All these schemes use in one way or another the fact that all the propagation speeds are finite. This is due to the hyperbolicity of the differential equations. That is, the Jacobian of the flux function can be diagonalized. There are m real eigenvalues and a full set of eigenvectors. In the large class of Godunov-type schemes, propagation speeds are computed implicitly by solving a Riemann problem exactly or approximately across a element boundary.

This approach is then extended to several space dimensions using the one-dimensional Godunov-type methods at each element interval. Clawpack[1] is a good example of this approach.

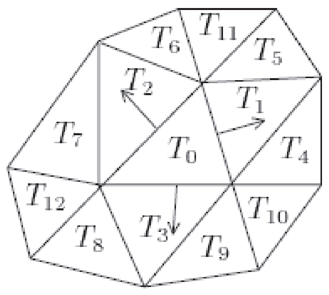


Figure 4.8: Discretization of space with triangular control volume. The information propagates normal to the element faces in three directions.

In this work triangular control volume are used. To use this theory of computing fluxes across element sides, an approach is adopted where the axis of the problem domain is rotated along the sides and the normal fluxes are calculated. In this approach, the numerical approximation on each side of the element is considered to be constant and the 1D scheme is applied. Hence, information from one element of the mesh travels normal to the element face in the three space directions given by the normal vectors of the sides of the triangles (see Figure 4.8). The triangle T_0 has only three neighbouring triangles T_1, T_2 and T_3 into which information travels in the next time step.

4.4.4 Control Volume and Numerical Fluxes

In this work, the control volumes coincide with the mesh element, resulting in a element-centred scheme. Thus, $\vec{f}_k^*$ is effectively an approximation to the flux through edge k of $\partial\Omega$ and depends on the state (*i.e.*, the value of the solution variable(s)) on both sides of the edge and (possibly) on its position as well as time. This numerical flux is usually computed as the solution to a standard *Riemann problem* with the two states separated by a discontinuity (discussed in section, 4.3).

But for the resulting finite volume method to be conservative, they still have to satisfy a number of properties:

[1]`http://www.amath.washington.edu/~claw/`

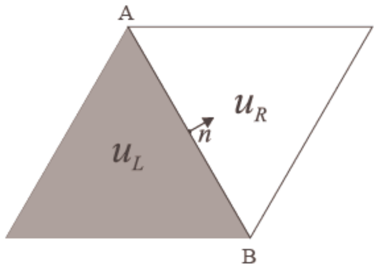


Figure 4.9: Conventions for the flux computation across edge AB (as seen from the shaded element)

1. **Property 1:** The union of all control volumes covers the whole domain *i.e.*, $\sum_{j=0}^{N} \Omega_j = \Omega$.
2. **Property 2:** Adjacent volumes should not overlap the internal boundary so that the corresponding internal fluxes cancel out.
3. **Property 3:** Fluxes along a volume boundary have to be computed independently of the control volume in which they are considered.

For a numerical scheme to be conservative, the discretisation of the flux integrals has to satisfy the above properties. Properties (1) and (2) are trivially satisfied by a element-centred scheme and property (3) can be reduced to the requirement that for each interior edge AB in the discretisation of Ω

$$f_{AB}.\hat{n}_{AB} = -f_{BA}.\hat{n}_{BA}, \tag{4.60}$$

as $\hat{n}_{AB} = -\hat{n}_{BA}$,

$$f_{AB} = f_{BA}. \tag{4.61}$$

Object-Oriented Design

The fluxes for each side of the element is calculated using the Riemann solvers present in the Side class (subclass of GenericSide). Using these fluxes each element is updated at every time step. The *update* method is defined in the GenericElement class.

4.4.5 The CFL Condition

The CFL condition is a necessary condition for the stability of a difference approximation to a hyperbolic problem. The condition is named after Courant, Friedrichs, and Lewy who published the result in 1928 [Courant *et al.* (1928)]. It is sometimes referred to as the Courant condition. It is one of the (historically and practically) most important results in numerical solutions of PDEs.

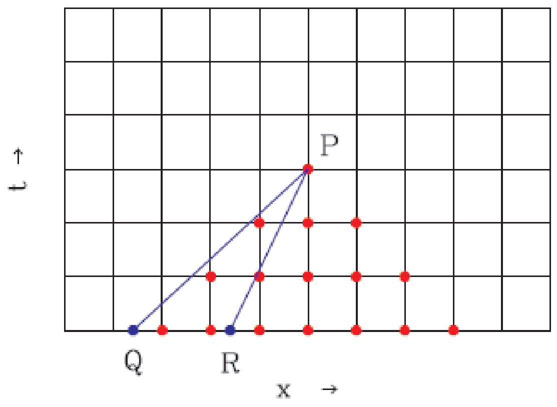


Figure 4.10: Numerical domain of dependence shown in circular points at the mesh node. Two possible characteristics are shown by lines. QP corresponds larger c and the CFL conditions is violated. RP corresponds smaller c and the CFL condition is satisfied.

The CFL condition is based on the concept of **domain of dependence**. For the advection equation with $c =$ *constant wave speed*, the domain of dependence is just the ray $x = ct + x_0$ which passes through **P** (see Figure 4.10). More generally the domain of dependence for the advection equation is the characteristic passing through point **P**. For the wave equation, there will be two characteristics passing through **P**.

For hyperbolic equations, consider a point **P** on the space-time grid. Let U_P be the numerical value of U at **P**, u_P be the actual value of the solution to the PDE at **P**. Let **Q** be some point in the domain of dependence of **P**. Now,

The CFL conditions states that a necessary condition of a numerical scheme to be stable is that the numerical domain of dependence must contain the actual domain of dependence for the partial differential equation.

It is easy to see why this must hold. Consider the case in which the numerical domain of dependence does not contain the actual domain of dependence for the PDE. Think of $\mathbf{Q}$ as a point at time $t = 0$. u_P will hence depend on initial condition at $\mathbf{Q}$. However, since $\mathbf{Q}$ is not in the numerical domain of dependence, the numerical value U_P cannot depend on the initial condition at $\mathbf{Q}$. Consider changing the initial condition at $\mathbf{Q}$ (or in some small neighborhood of $\mathbf{Q}$). u_P will change but U_P will not. Hence there is no possibility, in general, of U_P converging to u_P as the grid is refined (as $\Delta t, h \to 0$).

For the advection equation with $c > 0$ and any three point stencil, the CFL condition is:

$$\frac{\Delta t}{h} \leq \frac{1}{c}, \tag{4.62}$$

Similarly for $c < 0$ the CFL condition is:

$$\frac{\Delta t}{h} \leq \frac{-1}{c}, \tag{4.63}$$

Combining equation 4.62 and 4.63, we get:

$$|c| \frac{\Delta t}{\Delta x} \leq 1. \tag{4.64}$$

The same condition holds for the wave equation with 2 sets of characteristics. It should be noted that the CFL condition is necessary but not sufficient *i.e.*,

- Satisfying the CFL condition does not guarantee a scheme converges.
- Not satisfying the CFL condition guarantees failure.

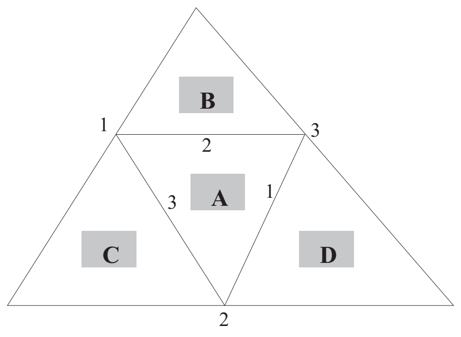


Figure 4.11: Element and its neighboring elements

Equation 4.64 is sufficient for a one dimensional or a two-dimensional rectangular grid. But in the case of a 2D unstructured triangular mesh the calculation of CFL is complicated. The problem of triangular element is that all its sides have different lengths, also the orientation of the element poses a problem in determining the side through which information is propagating. In finite volume explicit schemes the flux is calculated through each side. Generally the CFL condition in such methods are calculated through these sides based on the velocities of the waves passing through them. To generalise the CFL condition based on Figure 4.11 an equation is developed as shown in equation 4.65:

$$Max\left(\frac{1}{2}\frac{\Delta t.\underset{side=1,2,3}{l}.s}{Min\left(A_{area},\underset{N=D,B,C}{N_{area}}\right)}\right)\leq 1, \tag{4.65}$$

where Δt is the time step, l is the length of the sides $(1,2,3)$, N_{area} denotes the corresponding area of the triangle (D,B,C), and s is the speed of the wave, which is calculated based on the Riemann solvers. The CFL number is a dimensionless quantity.

Object-Oriented Design

The generalised formula of equation 4.65 calculates the CFL number based on the speed of the waves across each side. This generalisation of CFL number makes it a part of the Generic package.

CFL criteria is used to check the the convergence of the solution for the mesh. Whereas, the formula 4.65 is operated on each side of a triangle. Therefore, the CFL number is declared in the GenericMesh class but updated in the GenericSide class.

4.5 Boundary Conditions

For the numerical methods discussed above we need to devise a method for applying the boundary condition. One approach is to develop special formulas for use near the boundaries, which will depend both on what type of boundary conditions are specified and on what sort of method we are trying to match. However, in general it is much easier to think of expanding the computational domain to include a few additional elements on the boundary, called *ghost elements*. The values of the ghost elements are set at the beginning to each time step in some manner that depends on the boundary conditions and perhaps the interior solution.

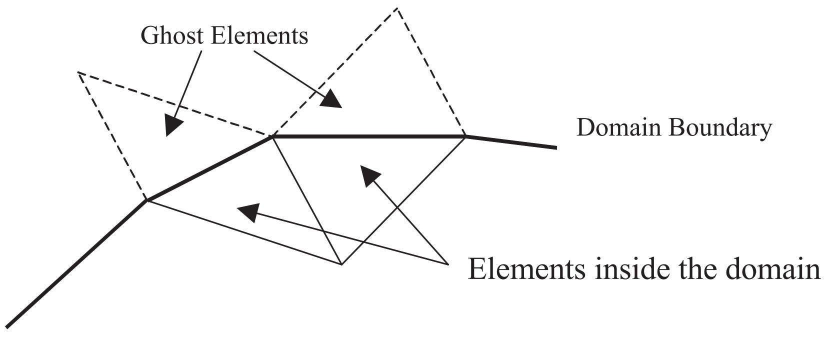


Figure 4.12: Ghost Elements which is made as the mirror image of the element inside the domain

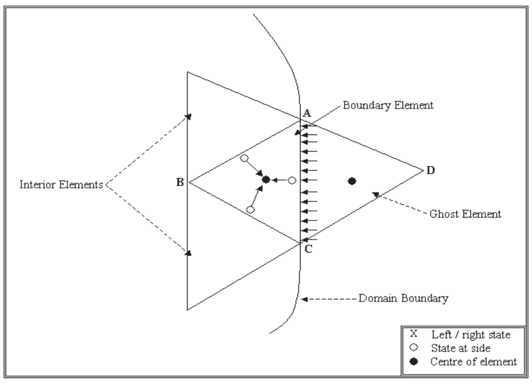
The ghost elements are made in such a way that it creates a parallelogram with the boundary elemnet, shown in Figure 4.12. Ghost elements make it easier to implement the boundary conditions. At the start of each time step we have the values of parameter of the interior element obtained from the previous step (or from the initial step), and we apply a boundary condition procedure to fill the ghost elements with values, before applying the method to solve the problem for the next time step. These values provide the neighbouring element values needed in updating the element near the physical domain. The boundary conditions are usually dealt with using one or more ghost elements, but in this work only one ghost element is used.

In many simulations there is a need to have fluid flow specified on the face of the boundary, generally, in terms of flux. It is difficult to correctly model this type of boundary condition by specifying the values in the ghost element. It is important to understand that when boundary condition specified at the face of the side are extrapolated to the centre of the element, it may incur unwanted error. This could result from the application of the limiters. Therefore, to model this type of boundary condition flux type boundary condition is used.

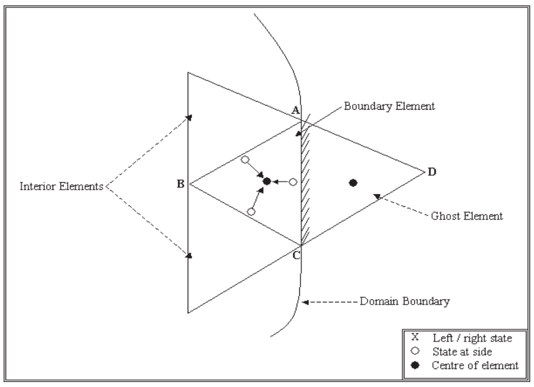
Before the integration step the value of the boundary condition for the side is set in a pre-determined way, depending on the problem. Using both options of applying boundary condition there are four possible choices, shown in table 4.2. These boundary condition are discussed in the following subsections.

Table 4.2: Classification of boundary conditions

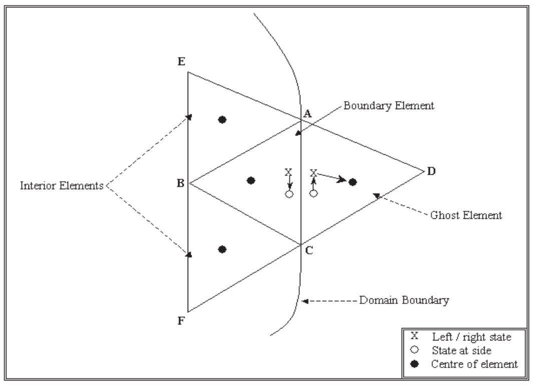
	Boundary condition	Riemann problem solved	Limiters used
1	Flux-type	No	No
2	Reflective-type	No	No
3	State-type	Yes	Yes
4	Transmissive-type	Yes	Yes



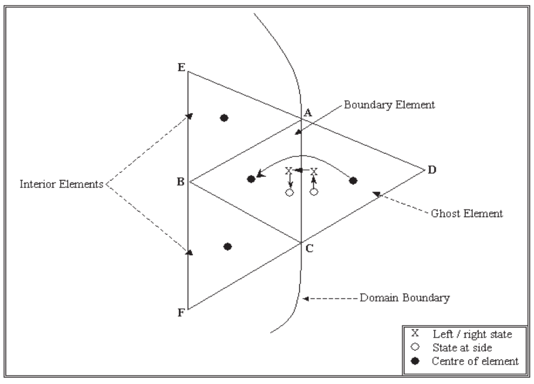
(a)



(b)



(c)



(d)

Figure 4.13: Object-oriented implementation of boundary conditions: (a) Flux-type (b) Reflective-type (c) State-type (d) Transmissive-type zero order extrapolation

Object-Oriented Design

For efficient handling of the boundary condition, object-oriented method pointer is used. This allows a method to be initialised indirectly by way of a reference to the variable, which is a once-only transfer of the appropriate reference to the left and right states for a side. It is part of the Mesh initialisation process and done only once. In other words, references are used to hook up the states at sides (calculated from limiters)and the left and right state of the respective side.

Application of the limiters are discussed in the next chapter. For the time being it can be assumed that limiters help to specify the states at three side of a triangular control volume. These states at sides are used to set left and right states when solving a Riemann problem at a side.

Figure 4.13 shows four type of situations where different boundary conditions (Table 4.2) are applied using appropriate reference pointers. The detail of these boundary conditions are described in the following subsections.

4.5.1 Flux-Type

When flux (mass and momentum) values are assigned to faces of the elements, the boundary condition are classified as flux-type boundary conditions (Figure 4.13 a). The corresponding boundary conditions are a numerical attempt of imposing fluxes into the system. In some cases this could be in the reserve direction *i.e.*, the fluid leaves the system at a defined flux rate.

There are two possible condition in which this type of boundary condition are used, that is:

1. when the fluxes are fixed at the boundary. In this case the fluxes imposed at the boundary will be same for every time step, and
2. when the fluxes vary with the time. In this case the state variable can be passed through an input file and defined at every time step of the calculation.

It should be noted that in this type of boundary condition, Riemann problems are not solved at the corresponding boundary sides. Also, the limiters are not applied to the boundary element.

Object-Oriented Design

Figure 4.13 (a) shows the situation for the implementation of the flux-type boundary condition. It can be seen that the flux values are specified at the boundary side of the boundary element. Therefore, no Riemann problem is solved and the three states at the sides of the element point to the centre of the element. This means that when state at the sides are called for the values, they return the same values stored at the centre of the element *i.e.*, the vector of state variables (see equation 4.12).

State variables of elements within the domain are updated at every time step and for this fluxes at three sides are required (see equation 4.20). In this type of boundary condition the code directly assign the values of the fluxes for the boundary side.

4.5.2 Reflective-Type

The domain boundary, which physically consists of a fixed or reflective impermeable wall are classified as the reflective-type boundary condition (Figure 4.13 b). The corresponding boundary conditions are a numerical attempt to produce boundaries that do not allow the passage of waves and fully reflect them. There are two ways of implementing this type of boundary condition:

1. One way of modelling this situation is by creating a fictitious state $U^n_{ghost\_element}$ in the ghost element and defining the boundary Riemann problem. The fictitious state $U^n_{ghost\_element}$ (see equation 4.12 for the vector of conserved variables) is defined from the known state $U^n_{boundary\_element}$ inside the computational domain, namely

$$u^n_{ghost\_element} = u^n_{boundary\_element} \tag{4.66}$$

for all the non-momentum conserved variables. For the momentum conserved variables the $U^n_{ghost\_element}$ is defined as

$$u^n_{ghost\_element} = -u^n_{boundary\_element}. \tag{4.67}$$

Toro (1999) states that the exact solution of this boundary Riemann problem consists of either (i) two shock waves if momentum conserved variable, $u^n_{boundary\_element} > 0$, or (ii) two rarefaction waves momentum conserved variable, $u^n_{boundary\_element} \leq 0$. In both cases $u_{boundary} = 0$ along the boundary; this is the desired condition at the solid, fixed impermeable boundary.

2. Alternatively, zero flux can be assigned at the boundary side of the boundary elements. The benefit of using this method is:

   - Firstly, it doesn't not require to solve the Riemann problem at the boundary side. Therefore, it save the computational cost of solving riemann problem and then updating the parameters.
   - Secondly, in this process the limiters are not applied to the boundary element. This ensures that that depth at the sides in the boundary element are same as the states of the element and there is no extra reflection created due to zero-order extrapolation.

In the present study second approach is adopted. But, for those who wish to apply the reflective type boundary condition using first approach can achieve this with little modification in the code. The implementation of the boundary condition using first approach would follow the principles of the state type boundary condition, explained in the next subseciton. The only difference would be that the state variables need to be updated at every time step, as described above.

Object-Oriented Design

Figure 4.13 (b) shows the situation for the implementation of the reflective-type boundary condition. It can be seen that the situation in this case is same as the flux-type boundary condition except that the flux is zero at the boundary side.

Therefore, Riemann problem is not solved at the boundary and limiters are not applied to the boundary element. For the update method using equation 4.20, the code will return zero flux.

4.5.3 State-Type

When the vector of state variables(equation 4.12) are defined at the boundary then it is classified as the state-type boundary condition (Figure 4.13 c). The assumption made to this type of boundary condition is that the state variables are defined in the ghost element. There are two possible condition in which this type of boundary condition are used:

1. when the state variables are fixed at the boundary. In this case the state variables will be same for every time step, and

2. when the state variables vary with the time. In this case the state variable can be passed through an input file and defined at every time step of the calculation.

Object-Oriented Design

Figure 4.13 (c) shows the situation for the implementation of the state-type boundary condition. In this type of boundary condition the vector of state variables (see equations 4.12) is specified at the centre of the ghost element.

While creating sides in the GenericElement the code cycles (anticlockwise) through each element to make sides and specify the left and right elements to this sides. To solve the Riemann problem it also specify the left and right state (according to the left and right element) at the centre of the side. During this process the ghost elements always comes on the right of the boundary element.

In the ghost element DAC, the state at side points to the right state of the boundary side, which in turn points to the centre of the ghost element. In other words, the state at boundary side and right state will always return the values from the vector of state variable. Whereas, for the boundary element ABC, left state at the boundary side points to the state at the boundary side. This state at the boundary side is calculated using the limiters. The limiter set the value at the sides of the boundary element ABC by using the values in ABC and its neighbouring elements *i.e.*,DAC,AEB and BFC (see section 5.9.1).

4.5.4 Transmissive-Type

Transmissive-type (Figure 4.13 c), boundary conditions are a numerical attempt to produce boundaries that allow the passage of waves without any reflection. This is achieved by

$$U^n_{ghost\_element} = U^n_{boundary\_element}, \tag{4.68}$$

and equation produces a trivial Riemann problem. Toro (1999) states that no wave of finite strength is produced at the boundary that might affect the flow inside the domain. This last statement might be true for some cases, but in the present study it has been found that this approach (also known as *zero-order* extrapolation) produces reflection at the transmissive boundaries.

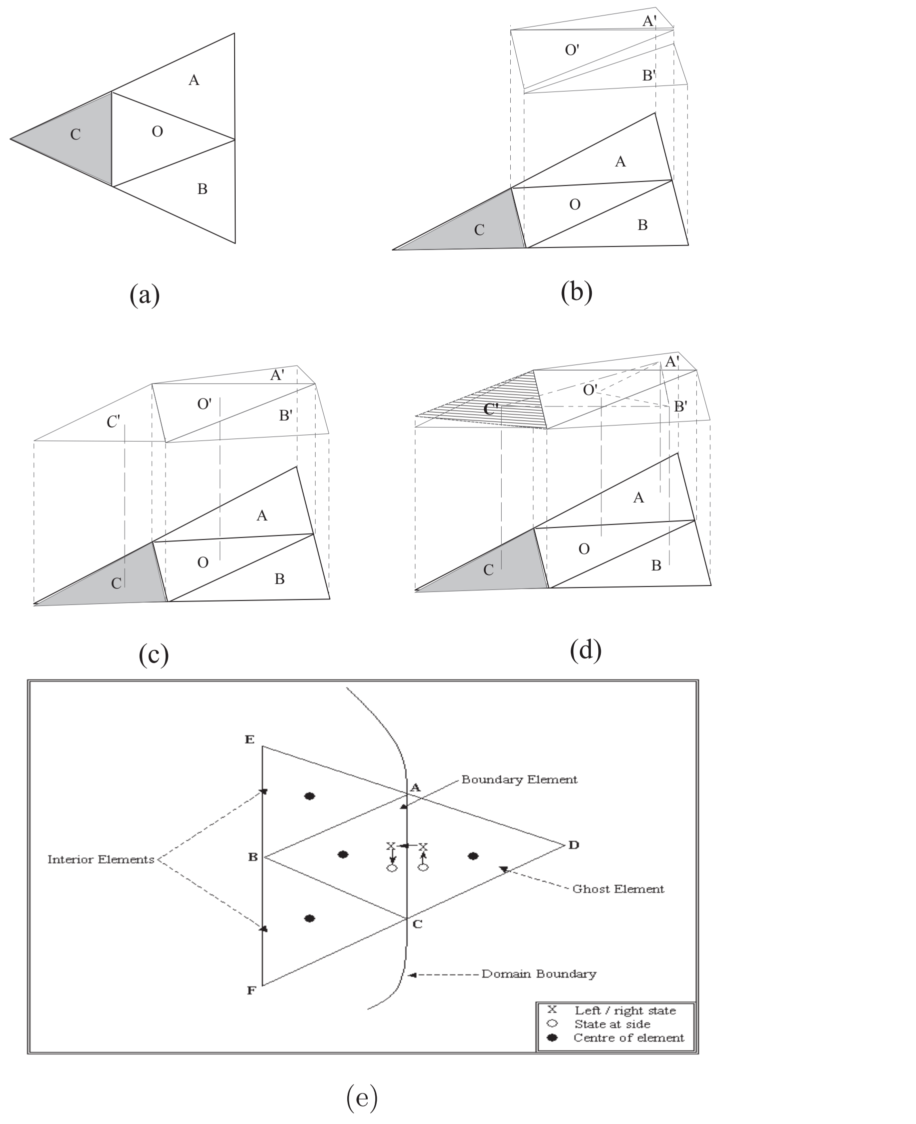


Figure 4.14: **First-order extrapolation for 2D triangular elements**: (a) Elements in plan view. Ghost Element C is in grey shade. (b) Elements in plan and elevation view. Ghost Element C elevation view is excluded. (c) Elements shown after the limiters are applied. Ghost Element C is given same value (+ or -) as the boundary element O *i.e.*, $\|CC'\| = \|OO'\|$. This is often know as *zero-order* extrapolation. (d) Ghost Element O is given value (+ or -) based on the gradient of the triangle formed by values in the OO', AA' and BB'. This is often know as *first-order* extrapolation. Shaded triangle C shows that the value in the ghost element could be different from the zero-order value, depending on the gradient of O'A'B'. (e) Object-oriented implementation of first-order transmissive-type boundary condition.

Object-Oriented Design

Figure 4.13 (d) shows the situation for the implementation of the transmissive-type zero order boundary condition. It can be seen in Figure 4.13 that the centre of the ghost element points to the centre of the boundary element. Therefore, when limiters are set for boundary element ABC, the ghost element will always return the values from the centre of boundary element. However, on the boundary side the state at boundary side in the ghost element points to the right state. The right state points to the left state, which in turn point to the state at the boundary side of the boundary element. The value of state at the boundary side of boundary element ABC is assigned by setting up the limiters on ABC.

To overcome the problem of reflection, a first order approach is developed. This is based on fitting a *first-order* plane through the interior solutions and extrapolating the boundary elements. The process of *first-order* extrapolation is described in Figure 4.14, which shows the steps followed for the implementation of this process.

Object-Oriented Design

Figure 4.14 (a)-(d) shows the process of setting up first order boundary condition. This is done using the extrapolation of the values in the boundary element and its neighbouring interior elemnets. The extrapolated values set the vector of state variable in the centre of the ghost element.

Figure 4.14 (e) shows the situation for the implementation of the transmissive-type first order boundary condition. It can be seen that the Figure 4.14 (e) is similar to the Figure 4.13 (d). The only difference in the two process (*i.e.*, zero-order and first-order) is that the values in the ghost elements are the extrapolated values. Otherwise, the process reference pointing is same.

4.6 Conclusions

This chapter has presented the common theory of hyperbolic systems and discussed the object-oriented implementation. The theory presented in this chapter do not develop a stand-alone model, or solve any particular hyperbolic problem. However, the model developed using this theory can easily extended using the object-oriented model of

Riemanns2D. The benefits of developing *Riemann2D* as a generic hyperbolic model are as follows:

- It helps to identify common features of various model and thus leads to better understanding of its theory.
- It has greater ability to apply techniques, used for one model, to solve new problems.
- It is easier to develop new model with less effort.
- It is possible to keep the model open for the future development.

Other important developments in this chapter are as follows:

- discussion on element-centered and node-centered control voulme (see section 4.4.2);
- generalised CFL criteria for 2D triangular mesh for hyperbolic problems (equation 4.65); and
- introduction of new and effictive approach to the solution of boundary conditions, (see section 4.5).

CHAPTER 5

High-Resolution Methods

5.1 Introduction

In Chapter 4, the finite volume methods developed for the hyperbolic systems are only first order accurate in time and space. In this chapter these methods are further derived to find second order accuracy in space and time. Focus has been on providing a solution for discontinuities and eliminating the spurious/numerical oscillations.
The objective of this chapter is to present:

- the traditional numerical methods to emphasise the need for the used high-resolution methods;
- various methods to solve the problem due to oscillations in the vicinity of high gradients;
- the slope limiters used in *Riemann2D* to limit the oscillations.
- an extended version of one-dimensional vanLeer limiter [vanLeer (1979)] for triangular elements.
- the object-oriented implementation of the theory in the *Riemann2D* design.

Few important things to note before reading this chapter:

- The gray object-oriented comment boxes, mentioned in the section 1.4, are only for the discussion of object-oriented implementation.
- The high-resolution methods is a part of generic package.

5.2 Shock Capturing

The problem of spurious oscillations in the vicinity of high gradients (also known as *shocks*[1]) is depicted in Figure 5.1, where the solid line denotes the exact solution and the dotted line denotes the numerical solution obtained by some linear method of second order or higher order of accuracy. According to Godunov's theorem [see Toro (1999)], spurious oscillation in the vicinity of high gradients are expected [Toro (2001)], which is highly undesirable in any numerical scheme.

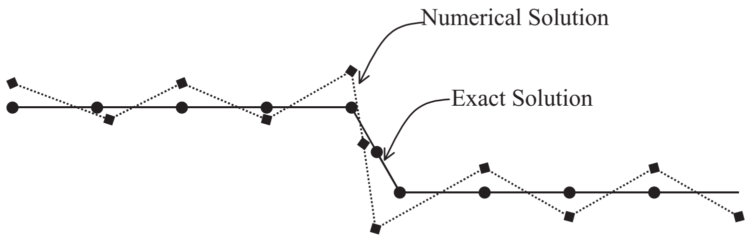


Figure 5.1: Illustration of the numerical phenomenon of spurious oscillations near high gradient

The main difficulty in calculating fluid flows with *shocks* is that it is very hard to predict, even in the process of a flow calculation, when and where new *shocks* will arise and interact; tracking the *shocks*, especially their interactions, is numerically burdensome.

Many methods used to control (or limit) spurious oscillations and diffusion around discontinuities by either,

- *Artificial viscosity*: The artificial viscosity, introducing by fictitious term, suppresses numerical oscillations near discontinuities, which otherwise would have been generated; or

- *Non-linear limiters*: The use of non-linear limiters suppresses oscillations by controlling the numerical solution in a non-linear way, so that the appearance of any new local extremes is prohibited during the process of reconstruction of the cell variables.

[1]Shock waves are discontinuous solutions of hyperbolic conservation laws obeying some precise mathematical conditions. See Toro (2001) (page 3) for a brief introduction to shocks from the viewpoint of numerical methods.

In present work, non-linear limiters are studied and used to achieve:

1. correct speed of propagation and correct width of the shock layer;
2. second or higher order of accuracy in time and space;
3. schemes producing numerical solutions free from spurious oscillations; and
4. schemes producing high-resolution of discontinuities, *i.e.*, the number of mesh points in the transition zone containing the numerical wave is less in comparison to that of first-order monotone methods.

For a two-dimensional case, one direct approach is to generalize directly the one-dimensional methods to solve multi-dimensional problems; such an approach has led to several useful numerical methods including *semi-discrete* methods and *Strang's dimension-dimension splitting* methods [Chen (2000)]. Multi-dimensional effects play a significant role in the behaviour of the solution locally, and any approach that only solves the one-dimensional Riemann problem in the coordinate directions is clearly not using all the multi-dimensional information.

5.3 Traditional Numerical Methods

In this section some traditional numerical schemes are presented that are used in different literatures to solve hyperbolic systems of equations. The benefit of studying the traditional methods is that it helps better understanding and usage of modern methods.

5.3.1 The Lax-Friedrichs Method

The Lax-Friedrichs (LxF) method [Strikwerda (1989), Thomas (1995) and LeVeque (2002)] is a basic method for the solution of hyperbolic partial differential equations. Its use is limited because its order is only one, but it is easy to program, applicable to general PDEs, and has good qualitative properties because it is monotone. The LxF method is often used to show the effects of dissipation, but it is not actually a dissipative method [Strikwerda (1989)].

The classical Lax-Friedrichs (LxF) method has the form:

$$U_i^{n+1} = \frac{1}{2}\left[U_{i-1}^n + U_{i+1}^n\right] - \frac{\Delta t}{2\Delta x}\left[f(U_{i+1}^n) - f(U_{i-1}^n)\right], \tag{5.1}$$

which is very similar to the unstable method (equation 4.22), but the value of U_i^{n+1} is replaced by the average $\frac{1}{2}\left[U_{i-1}^n + U_{i+1}^n\right]$. For the linear hyperbolic equation this method is stable provided the Courant number (section 4.4.5) is less then 1. The numerical equation 5.1, alternatively, can be written in the terms of numerical flux as:

$$\mathcal{F}(U_{i-1}^n, U_i^n) = \frac{1}{2}\left[f(U_{i-1}^n) + f(U_i^n)\right] - \frac{\Delta x}{2\Delta t}(U_i^n - U_{i-1}^n). \tag{5.2}$$

this flux look like the unstable flux (equation 4.21; also see Figure 4.1) with the addition of another term similar to the flux of the diffusion equations. By using this flux we appear to be modelling the advection-diffusion equation $q_t + f(q)_x = \beta q_{xx}$ with $\beta = \frac{1}{2}(\Delta x)^2/\Delta t$. But if we fix the ratio, $\Delta t/\Delta x$, then this coefficient vanishes as the grid is refined, so in the limit, the method is still consistent with the original hyperbolic equation. This additional term can be interpreted as the *numerical diffusion* that damps the instability arising in the equation 4.22 and gives a method that can be shown to be more stable for Courant number (see section 4.4.5) up to 1. However the Lax-Friedrichs method introduces much more diffusion than is actually required, and gives numerical results that are typically badly smeared unless a very fine grid is used [LeVeque (2002)].

5.3.2 The Richtmyer Two-Step Lax-Wendroff Method

The Lax-Friedruchs method is only first-order accurate and can be achieved by using a better approximation to the integral in equation 4.5. One approach is to first approximate q at the midpoint in time, and evaluates the flux at this point. The Richtmeyer Two-Step Lax-Wendroff method [Richtmyer and Morton (1967)] is of the form:

$$F_{i-1/2}^n = f(U_{i-1/2}^{n+1/2}), \tag{5.3}$$

where

$$U_{i-1/2}^{n+1/2} = \frac{1}{2}\left(U_{i-1}^n + U_i^n\right) - \frac{\Delta t}{2\Delta x}\left[f(U_i^n) - f(U_{i-1}^n)\right]. \tag{5.4}$$

$U_{i-1/2}^{n+1/2}$ is obtained by applying the Lax-Friedrichs method at the cell interface with Δx and Δt replaced by $\frac{1}{2}\Delta x$ and $\frac{1}{2}\Delta t$ respectively.

For a linear system of equations, $f(q) = Aq$, the Richtmyer method reduces to the standard Lax-Wendroff method. These methods often lead to spurious oscillations in solutions, particularly when solving problem with discontinuous solutions. Additional numerical diffusion (or artificial viscosity) can be added to eliminate these oscillations, as proposed by von Neumann and Richtmyer (1950).

5.4 The Lax-Wendroff Method

A large number of second-order methods can be developed to solve the linear system $q_t + Aq_x = 0$ by using different finite difference approximations. Most of them are directly based on finite difference approximations of the model equations with the exception of the Lax-Wendroff method. This method is based on the Taylor series expansion

$$q(x, t_{n+1}) = q(x, t_n) + \Delta t q_t(x, t_n) + \frac{1}{2}(\Delta t)^2 q_{tt}(x, t_n) + \dots \tag{5.5}$$

Rewriting $q_t + Aq_x = 0$ as $q_t = -Aq_x$ and differentiating this gives

$$q_{tt} = -Aq_{xt} = -A^2 q_{xx} \tag{5.6}$$

where we have used $q_{xt} = q_{tx} = (-Aq_x)_x$. Using these expressions for q_t and q_{tt} in equation 5.5 gives

$$q(x, t_{n+1}) = q(x, t_n) - \Delta t A q_x(x, t_n) + \frac{1}{2}(\Delta t)^2 A^2 q_{xx}(x, t_n) + \dots . \tag{5.7}$$

Keeping only the first three term on the right hand side and replacing the spatial derivatives by central finite difference approximation gives the *Lax-Wendroff method.*

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{2\Delta x} A(Q_{i+1}^n - Q_{i-1}^n) + \frac{1}{2}\left(\frac{\Delta t}{\Delta x}\right)^2 A^2(Q_{i+1}^n - 2Q_i^n + Q_{i-1}^n). \tag{5.8}$$

By matching three terms in the Taylor series and using centered approximations, we obtain a second-order accurate method.

This derivation of the method is based on a finite difference interpolation, with Q_i^n approximation point wise value $q(x, t_n)$. However it can be interpolated as a finite volume method of the form of equation 4.5 with the flux function

$$F_{i-1/2}^n = \frac{1}{2}A\left(Q_{i-1}^n - Q_i^n\right) - \frac{1}{2}\frac{\Delta t}{\Delta x}A^2(Q_i^n - Q_{i-1}^n). \tag{5.9}$$

5.5 The Beam-Warming Method

The Lax-Wendroff method is a *centred* three point method. If three is a system for which all the eigenvalues are positive (*e.g.*, the scalar advection equation with $\overline{u} > 0$), then it might be preferable to use a one-sided formula. In place of a centred formula for q_x and q_{xx} , we can use

$$q_x\left(x_i, t_n\right) = \frac{1}{2\Delta x}\left[3q\left(x_i, t_n\right) - 4q\left(x_{i-1}, t_n\right) + q\left(x_{i-2}, t_n\right)\right] + \mathrm{O}(\Delta x^2) \tag{5.10}$$

$$q_{xx}\left(x_i, t_n\right) = \frac{1}{(\Delta x)^2}\left[q\left(x_i, t_n\right) - 2q\left(x_{i-1}, t_n\right) + q\left(x_{i-2}, t_n\right)\right] + \mathrm{O}(\Delta x). \tag{5.11}$$

Using these in equation 5.7 gives a method that is again second order accurate

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{2\Delta x}A(3Q_i^n - 4Q_{i-1}^n + Q_{i-2}^n) + \frac{1}{2}\left(\frac{\Delta t}{\Delta x}\right)^2 A^2(Q_i^n - 2Q_{i-1}^n + Q_{i-2}^n). \tag{5.12}$$

This is know as *Beam-Warming method*, which can be written in the flux differencing finite volume method with

$$F_{i-1/2}^n = AQ_{i-1}^n + \frac{1}{2}A(1 - \frac{\Delta t}{\Delta x}A)(Q_{i-1}^n - Q_{i-2}^n). \tag{5.13}$$

5.5.1 Problems with Traditional Schemes

The methods discussed in the above sections (5.3.1 and 5.3.2) are *centered* methods *i.e.*, symmetric about the point where solution is updated. Pudasaini (2003) states that the traditional central difference methods introduce dispersive effects that lead to unphysical oscillations in the numerical solution for physical problems with large gradients of variables. Second order accutate methods such as Lax-Wendroff or Beam-Warming give much better accuracy on a smooth solution than the upwind methods, as seen in Figure 5.2[1], but fail near discontinuities, where oscillations are generated. Even for the smooth solutions, oscillations may appear due to the dispersive nature of these methods.

Literature on second order central schemes in one space dimension can be found in Nessyahu and Tadmor (1990), Jiang *et al.* (1998), Kurganov and Petrova (2000), Lie and Noelle (2003) and Breuss (2005). Some third and higher order central schemes can

[1]Animation of these tests are available `http://www.amath.washington.edu/~claw/book/chap6/compareadv/www/`

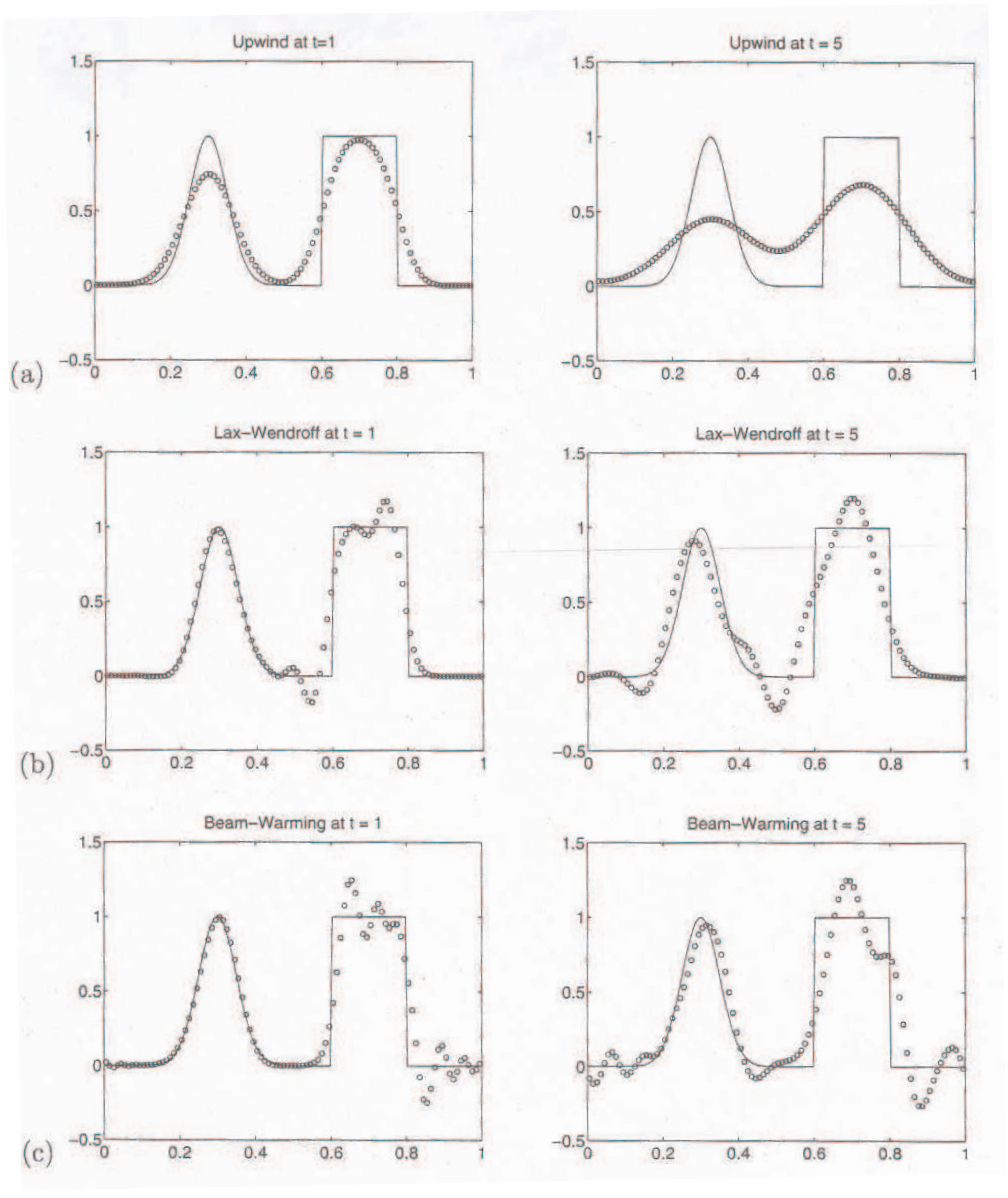


Figure 5.2: Test on the advection equation with different linear methods. Result at time t = 1 and t = 5 , corresponding to 1 and 10 revolutions through the domain in which the equation $q_t + q_x = 0$ are shown with periodic boundary conditions: (a) upwind, (b) Lax-Wendroff, (c) Beam-Warming. [LeVeque (2002)]

be found in Liu and Tadmor (1998), Kurganov and Levy (2000), Qiu and Shu (2002) and Tadmor and Tanner (2003).

For hyperbolic equations it is often the case that the numerical oscillations are so large that a stable simulation may not be reached. For such cases, in order to avoid possible (emerging) instabilities or to limit numerical oscillations to an acceptable level, certain limiting operator must be incorporated. In order to avoid the above problem non-centered upstream difference schemes may be used. However, this introduces alternative difficulties in the implicit numerical diffusion.

5.6 Upwind Methods

For hyperbolic problems, information propagates with the waves along its characteristics. In a system of equation, several waves propagate at different speeds and in different directions. *Upwind methods* [Bermudez and Vazquez (1994), Navarro *et al.* (1995), LeVeque (1998), Vukovic and Sopta (2002)] provide a method to find the better numerical flux functions, by using the knowledge of the structure of the solution in which the information for each characteristic wave is obtained by looking in the direction from which this information should be coming. Therefore, using upwind methods provides better results.

The upwind schemes have to solve Riemann problems on the boundaries of each cell, which is interpreted as an *upwinding procedure*. However, a general scheme for the (exact or approximate) solution of the Riemann problems is not known, and the upwind approach may be rather complicated and costly, especially in multidimensional cases. But with the ever increasing advances in the computer hardware the computational cost should not be an issue over the accuracy and robustness of a scheme.

For the constant-coefficient equation

$$u_t + cu_x = 0 \tag{5.14}$$

where c is a real constant or the wave speed or the velocity of propagation.

Figure 5.3(b) shows the flux through the left edge is entirely determined by the value, U_{i-1}^n, of the left cell. This suggests defining the numerical flux as

$$F_{i-1/2}^n = cU_{i-1}^n. \tag{5.15}$$

This leads to the standard first-order upwind method for the advection equation,

$$U_i^{n+1} = U_i^n - \frac{c\Delta t}{\Delta x}(U_i^n - U_{i-1}^n). \tag{5.16}$$

Now this can be written as

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + c\left(\frac{U_i^n - U_{i-1}^n}{\Delta x}\right) = 0, \tag{5.17}$$

whereas the unstable method (see equation 4.22) applied to the advection equation is

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + c\left(\frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x}\right) = 0. \tag{5.18}$$

The upwind method used one side approximation to the derivative u_z in place of the centered approximation.

Another interpretation of the upwind method is suggested by Figure 5.3(a). If U_i^{n+1} is supposed to be the value at the center of the grid points, as is standard in the finite difference method, then $u(x,t)$ is constant along characteristics. Therefore, we can write

$$U_i^{n+1} \approx u(x_i, t_{n+1}) = u(x_i - c\Delta t, t_n). \tag{5.19}$$

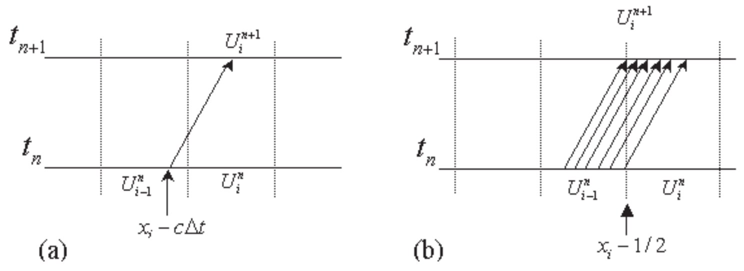


Figure 5.3: Two representations of the upwind method for advection. (a) If U_i^n represents the value at a point where information is available, then we can trace the characteristic back and interpolate. (b) If U_i^n represents the cell average, then the flux at the interface is determined by the cell value on the upwind side.

If the value on the right side is approximated by a linear interpolation between the grid values U_{i-1}^n and U_i^n , we obtain the method

$$U_i^{n+1} = \frac{c\Delta t}{\Delta x}U_{i-1}^n + \left(1 - \frac{c\Delta t}{\Delta x}\right)U_i^n. \tag{5.20}$$

This is the simple upwind method, it is a rearrangement of equation 5.16, but the condition applied is

$$0 \leq \frac{c\Delta t}{\Delta x} \leq 1, \tag{5.21}$$

in order for the characteristics to fall between the neighbouring points so that this interpolation is sensible. In fact equation 5.21 must be satisfied in order for the upwind method to be stable and also follows from the CFL condition [see section 4.4.5]. Also if equation 5.21 is satisfied then equation 5.20 expresses U_i^{n+1} as a convex combination of U_i^n and U_{i-1}^n (*i.e.*, the weights are both non-negative and sum to 1).

In the above discussion we have assumed that $c > 0$. On the other hand if $c < 0$ then the upwind direction is to the right and so the numerical flux at $x_{1-1/2}$ is

$$F_{i-1/2}^n = cU_i^n. \tag{5.22}$$

The two formulas in equation 5.15 and 5.22 can be combined into a single upwind formula that is valid for c of either sign, then we can write

$$F_{i-1/2}^n = c^- U_i^n + c^+ U_{i-1}^n, \tag{5.23}$$

where

$$c^+ = max(c, 0) \qquad and \qquad c^- = min(c, 0). \tag{5.24}$$

This information[1] is useful in extending the methods to more general hyperbolic problems. Not all hyperbolic equations are in conservation form; for example the equations of acoustics in a heterogeneous medium (where the density and bulk modulus vary with x). Such equations do not have a flux function, and so numerical methods of the form 4.5, where $F_{i-1/2}^n = \mathcal{F}(U_{i-1}^n, U_i^n)$ cannot be applied. However, these hyperbolic problems can still be solved using the finite volume approach of the model that results from a simple generalization of the high-resolution methods developed for hyperbolic conservation laws. The unifying feature of all hyperbolic equations is that they model waves that travels at finite speed. In particular the Riemann problem with piecewise constant initial data (as discussed in Chapter 3, section 4.3) consists of waves traveling at constant speeds away from the location of the jump discontinuities in the initial data.

[1]See sections 6.6 on Riemann solvers for shallow water equations where this information is used.

5.7 Godunov's Method

Godunov-type schemes have proved to be powerful tools for simulating discontinuous flows when they are described by systems of non-linear, hyperbolic conservation laws [Woodward and Colella (1984), and Toro (1999)]. Guinot (2003) describes Godunov-type schemes as a good candidates for the next generation of commercial modelling software packages, the capability of which to handle discontinuous solutions will be a basic requirement. Although they have gained popularity in the research area, such schemes are rarely found in simulation packages available commercially. Such schemes use many *ad hoc* techniques, such as slope limiters or wave splitting for multidimensional problems. Also, because the stencil of a scheme should include the domain of dependence of the solution for stability, a CFL [see section 4.4.5] stability constraint is necessarily attached to fixed-stencil schemes. Therefore, in such schemes, the computational time step has to be limited so the Courant number associated with each of the eigenvalues of the hyperbolic system should often be smaller than a fixed value. Consequently, these schemes should have the cpabilities to provide efficient solutions when:

- large contrasts exist between the various eigenvalues, and
- the computational grid is highly irregular.

In these cases, the schemes performance is limited by the largest of the eigenvalues of the hyperbolic system and by the size of the smallest cell, leading to a strong degradation of the numerical solution in regions where the cell is larger.

Guinot (2002) identified various ways to overcome the stability problems, one of them leads to a first class of numerical methods where the governing partial differential equations (PDEs) are approximated with ordinary differential equations (ODEs), thus breaking the dependence of the time step on the cell size. Although this allows the computational time step to be increased, such an approach has the drawback that it only works for smooth solutions to the PDEs.

Another possible approach consists of adapting the stencil of the scheme to the size of the domain of dependence of the solution. The characteristics can be traced forward or backward across several computational cells. This has been the subject of various works in the domain of Lagrangian or semi-Lagrangian techniques [Goldberg and Wylie (1983)] as well as flux-based (or conservative) Eulerian schemes. Originally, the approach consisted of seeking the departure point of the characteristic lines across several cells when needed. It was applied to problems with (almost) constant wave speeds, such as water hammer problems. The accurate determination of the feet of the characteristic lines is not straightforward and has been studied by a number of authors [Savic and Holly (1993)]. Moreover, when strong non-linearity is present in the equations, the irreversible character of the solution makes it difficult, if not impossible, to determine accurately the extension of the domain of dependence using backward-tracking algorithms.

A third approach consists of making the scheme implicit by solving a set of linear or non-linear equations using the unknown variables at the next time level to be computed. These techniques are time consuming, inasmuch as the size of the system to be solved is equal to the number of computational cells, or to twice this number in some methods.

The fourth approach well suited to the solution of conservation laws in the presence of shock waves, consists of using front-tracking-based methods, except that the domain of influence and the potential merging of *shocks* is explored forward in time rather than backward.

Instead we have used the shock-capturing methods, where the goal is to capture discontinuities in the solutions automatically, without explicitly tracking them.

5.7.1 Godunov's Method For Linear Systems:

The upwind method for the advection equation can be derived as a special case of the following approach, which can also be applied to systems of equations. LeVeque (2002) referred it as the REA for *reconstruct-evolve-average*:

1. Reconstruct a piecewise polynomial function $\tilde{u}^n(x, t_n)$ defined for all x, from the cell averages U_i^n . In the simple case this is a piecewise constant function that takes the value U_i^n in the grid cell, *i.e.*,

$$\tilde{u}^n(x, t_n) = U_i^n \qquad \text{for all} \quad x \in \Omega_i. \tag{5.25}$$

2. Evolve the hyperbolic equation exactly (or approximately) with this initial data to obtain $\tilde{u}^n(x, t_{n+1})$ at time Δt later.

3. Average this function over each grid to obtain new cell averages

$$U_i^{n+1} = \frac{1}{\Delta x} \int_{\Omega_i} \tilde{u}^n(x, t_{n+1}) \mathrm{d}x. \tag{5.26}$$

This whole process is repeated in the next time step. To introduce this procedure, it must be able to solve the hyperbolic equation in step 2. Because it is starting with piecewise constant data, this can be done using the theory of Riemann problems. When applied to the advection this leads to upwind algorithm.

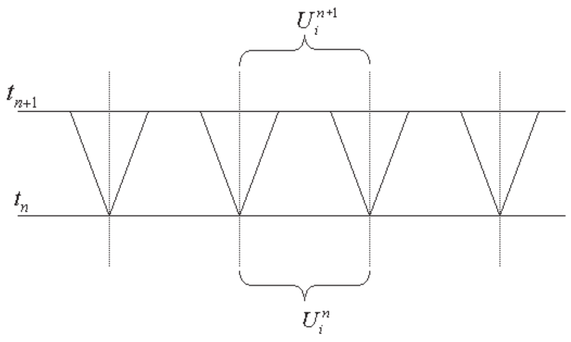


Figure 5.4: An illustration of reconstruct-evolve-average algorithm for the case of linear acoustics (adapted from LeVeque (2002)). The Riemann problem is solved at each cell interface, and the wave structure is used to determine the exact solution time Δt later. This solution is averaged over the grid cell to determine U_i^{n+1}.

In step 1 we reconstruct a function $\tilde{u}^n(x, t_n)$ from the discrete cell average. In Godunov's original approach this is the piecewise constant function. It leads most naturally to Riemann problems, but gives only a first order accurate method. To obtain better accuracy, a better reconstruction can be used, for example a piecewise linear function that is allowed to have a non-zero slope σ_i^n in the i^{th} grid cell. This idea forms the basics of the high-resolution methods.

The exact solution at time t_{n+1} can be constructed by piecing together the Riemann solution, provided that the time step is short enough that the waves from two adjacent sides in this process have not yet started to interact. Figure 5.4 shows a systematic diagram of this process for the equation of linear acoustics with constant sound speed c, in which case this requires that

$$|c|\, \Delta t \leq \frac{1}{2} \Delta x, \tag{5.27}$$

so that each wave goes at most halfway through though the grid cell. Rearranging gives

$$|c|\,\frac{\Delta t}{\Delta x} \leq \frac{1}{2}. \tag{5.28}$$

The quantity $|c|\,\frac{\Delta t}{\Delta x}$ is simply the Courant number [see section 4.4.5], so it is apparent that the condition is limited only within 1/2.

5.8 Total Variation Diminishing Method

Although first-order finite difference methods are monotonic and stable, they are also strongly numerically diffusive, causing the solution to become smeared out. Second-order or higher-order techniques are less dissipative, but susceptible to non-linear, numerical instabilities that cause non-physical oscillations. The high-resolution methods are a compromise between the traditional first-order and higher-order difference schemes. Their central idea is, on the one hand, to avoid the introduction of under- and over-shoots (numerical oscillation), and on the other hand, to maintain the numerical diffusion as small as possible, that is often achieved by different cell reconstruction techniques.

It is well known that in computing discontinuous solutions, the first order method (upwind) gives very smeared solutions while the second order method (Lax-Wendroff or Beam-Warming) gives spurious oscillations [see figure 5.2]. In order to develop a method that is of higher order and at the same time non-oscillatory and capable of capturing *shocks*, we need to define a powerful concept called the *Total Variation Diminishing* (a.k.a. TVD) method.

For a grid function it can be defined as:

$$\mathrm{TV}(u) = \sum_{i=-\infty}^{\infty} \left|u_i^n - u_{i-1}^n\right|. \tag{5.29}$$

Any oscillation in the computed result increases the *total variation* (TV). The Total Variation Diminishing condition

$$TV(u^{n+1}) \leq TV(u^n) \tag{5.30}$$

provides a method that gives a solution without spurious oscillations near the discontinuities. Any numerical scheme which fulfils the TVD condition (equation 5.30) for all grid functions U^n is called a *Total Variation Diminishing* (TVD) method. Therefore, any TVD method is automatically monotonicity preserving. This means, in particular, that oscillations of the physical quantities like velocity jumps and other sharp gradients cannot arise near an isolated propagating discontinuity. As we will see later, another beautiful feature of the TVD requirement is that it is possible to derive higher-order accurate methods that also satisfy equation 5.30. It can also be shown that the true (*i.e.*, physically relevant weak) solution to a scalar conservation law possesses this TVD property [LeVeque (1992)].

5.9 Higher-Order Methods for *Riemann2D*

Much effort has been invested to achieve better than first-order accuracy with finite volume methods. The first hurdle is Godunov's Theorem, which states that non-oscillatory constant coefficient schemes can be at most first-order accurate [Vollmer (2003)]. This can be overcome by the introduction of non-linear schemes such as Weighted Average Flux (WAF) [Billett and Toro (1997)], MUSCL [vanLeer (1979)], ENO [Toro (1995)], Flux Corrected Transport (FCT) [Boris and Book (1973)] and Piecewise Linear (PLM) [Colella and Woodward (1984)]. One of the popular ones (also used in the present study) is vanLeers MUSCL[1] [vanLeer (1979)] approach, which belongs to the class of Godunov-type methods, a class of non-oscillatory finite volume schemes that incorporate the exact or approximate solution to Riemann's initial-value problem.

VanLeer [vanLeer (1977b), vanLeer (1977a), and vanLeer (1979)] introduced the idea of modifying the piecewise constant data in the first-order Godunov method, as a first step to achieving higher order accuracy. This approach has become known as the MUSCL or Variable Extrapolation approach. The piecewise constant states within each cell are modified to piecewise linear ones (figure 5.5), which are carefully constructed from neighbouring states both to maintain conservation and not increase *total variation* (section 5.8) (*i.e.*, do not create over- or under-shoots).

5.9.1 Data Reconstruction

In all second- and higher-order schemes the application of non-linear limiters involves the introduction of a parameter termed the *limiter* into the gradient terms that appear

[1]MUSCL stands for Monotone Upstream-centred Scheme for Conservation Laws

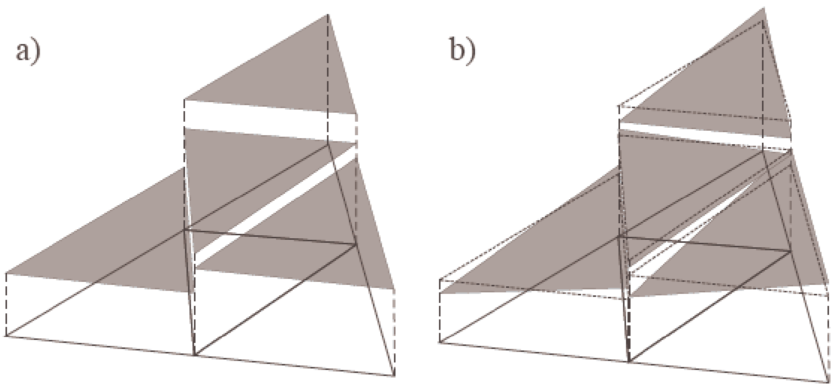


Figure 5.5: (a) Constant reconstruction vs (b) piecewise linear reconstruction

in the process of cell variable reconstruction. This step is necessary for higher-order schemes in order to maintain monotonicity.

Given an initial distribution of piecewise constant values in each control volume, a piecewise linear reconstruction of any scalar variable, u', over an element with centroid value as u may be expressed as

$$u' = u + \vec{r} \cdot \vec{L}. \tag{5.31}$$

Here $\vec{r}$ is a vector from the centroid of the control volume and $\vec{L}$ is a gradient operator. The linear reconstruction of u' still has to be conservative over the control volume Ω_j in the sense that

$$\frac{1}{V_{\Omega_j}} \int\int_{\Omega_j} u' \mathrm{d}x \mathrm{d}y = u, \tag{5.32}$$

Batten *et al.* (1996) recommended to construct a gradient plane through three nearby centroids A, B and C with normal vector

$$\vec{n} = (P_A - P_B) \times (P_C - P_B), \qquad \textit{with} \qquad P_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}, \tag{5.33}$$

and the subsequent gradient operator

$$\vec{\nabla}(\Delta ABC) = \begin{cases} \begin{bmatrix} -n_x/n_u \\ -n_y/n_u \end{bmatrix} & \text{if } n_u > \varepsilon \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \text{otherwise} \end{cases} \tag{5.34}$$

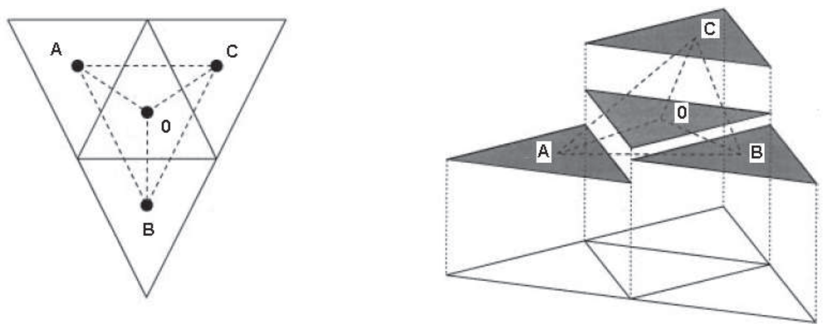


Figure 5.6: Naming convention for the Limiting Procedure defined for a triangular control volume (left) and a piecewise constant reconstruction of the solution (right)

The gradient operator defined by equation 5.34 is not yet limited and as such may exhibit non-physical over- or under-shoots at the points where the operator is evaluated usually at the midpoint of each edge. The limiting of the gradient operator therefore plays an important role as it directly influences the character and accuracy of the solution.

As a result we can write that the reconstruction of the state variables in the cell is limited in the form:

$$U(x, y) = U_0 + \Phi \nabla . r, \tag{5.35}$$

where Φ is a chosen limiter. When Φ is set to zero, equation 5.35 is a first-order-accurate reconstruction.

Object-Oriented Design

The reconstruction of elements (equation 5.35) are required to set the states at the sides of the elements. These states at the sides are used to solve Riemann problem. This operation is done on the elements, which makes the reconstruction as the part of the GenericElement class. In equation 5.35, Φ is supplied from the limiter package (discussed in the next section).

5.9.2 Limiters

We have introduced shock capturing schemes that reduce the numerical diffusion at discontinuities, sharpen the discontinuities in derivatives and avoid spurious oscillations, improving the behaviour of non-oscillatory schemes and piecewise hyperbolic methods.

In the present work we have introduced and analysed a number of limiter functions for the triangular meshes, based on the *rank* of the limiter. The term *rank*, which helps to understand the behaviour of limiters (discussed in chapter 8), is introduced in this work to identify and describe the number of limiting triangles (ΔABC, ΔAOC, ΔBOC and ΔCOB; see Figure 5.6) involved in the formation of a particular limiter.

Some literature on the selected limiters are given in section 2.4.2 and the mathematical expressions on these limiters is presented in the following sub-sections.

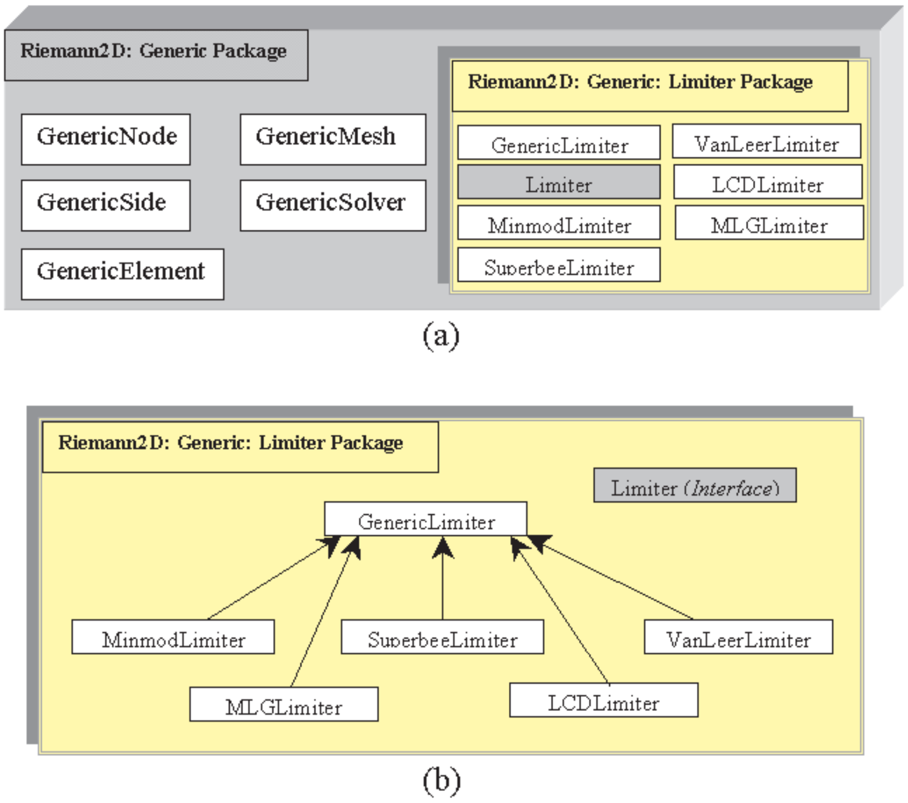


Figure 5.7: (a) Generic package with limiter package. (b) Relationship between various classes in the limiter package. GenericLimiter is superclass of other limiter classes.

Object-Oriented Design

Five different type of limiters are used in *Riemann2D*. These limiters are written in different classes and bundled into one package, called limiter (see Figure 5.7). The limiter package is part of the generic package, but due to its presence in a separate package it helps in better classification of the code. The GenericLimiter class in the limiter package (see Figure 5.7 (b)) is the superclass of other limiters and Limiter class is the interface for the limiter package. Detailed explanation is given in chapter 6.

5.9.2.1 No Limiters

"No limiter" is the case when $\Phi = 0$. It is a **rank 0** limiter. This means that the value at the centre of the element is same as the values at the sides of the elements.

Object-Oriented Design

The GenericLimiter class in the limiter package (see Figure 5.7 (b)) provides the "No Limiter" case. The GenericLimiter class is also superclass to other limiter classes in this package.

Note: Limiter class is the interface for the limiter package. Detailed explanation is given in chapter 6.

5.9.2.2 Superbee and Minmod Limiters

Superbee and Minmod limiters are **rank 1** limiters because they consider only one limiting triangle (Figure 5.6) *i.e.*, ΔABC.

$$\Phi_{Superbee/Minmod} = \min(\Phi_j), \qquad j = k(i), \tag{5.36}$$

where

$$\Phi_j = \max[\min(\beta r_j, 1), \min(r_j, \beta)], \tag{5.37}$$

with

$$r_j(u_j) = \begin{cases} (u_0^{\max} - u_0)/(u_j - u_0) & \text{if} \quad u_j - u_0 > 0 \\ (u_0^{\min} - u_0)/(u_j - u_0) & \text{if} \quad u_j - u_0 < 0 \\ 1 & \text{if} \quad u_j - u_0 = 0 \end{cases}, \tag{5.38}$$

and

$$u_0^{\min} = \min(u_0, u_{\text{neighbour}}), \qquad u_0^{\max} = \max(u_0, u_{\text{neighbour}}). \tag{5.39}$$

In the above equation (5.39) $u_{\text{neighbour}}$ is the value of conserved variables of elements A, B and C. The quantity β in equation 5.37 can take any value between one and two. In particular, $\beta = 1$ is the minmod limiter and $\beta = 2$ is Roe's Superbee limiter [Hirsch (1990)].

5.9.2.3 Limited Central Difference (LCD) Limiter

The LCD is also a **rank 1** limiter because it considers only one limiting triangle (Figure 5.6) *i.e.*, ΔABC. LCD is one of the earliest (and still most widely used) limiters in the context of MUSCL-schemes [Vollmer (2003)]. This *limiter's* advantages lie in its simplicity and speed.

1. Construct the unlimited gradient operator

$$\vec{L} = \vec{\nabla}(\Delta ABC). \tag{5.40}$$

2. For each edge k calculate the scalar

$$\alpha^k = \begin{cases} \frac{\max(u_k,u_0)-u_0}{\vec{r}_{0k}.\vec{L}} & \text{if}(u_0 + \vec{r}_{0k}.\vec{L}) > \max(u_k, u_0) \\ \frac{\max(u_k,u_0)-u_0}{\vec{r}_{0k}.\vec{L}} & \text{if}(u_0 + \vec{r}_{0k}.\vec{L}) < \min(u_k, u_0) \\ 1 & \text{otherwise} \end{cases}, \tag{5.41}$$

3. Set

$$\Phi_{LCD} = \vec{L}_{LCD} = \left(\min_{\text{all}\, k} \alpha^k\right) \vec{L}\ . \tag{5.42}$$

5.9.2.4 Extended vanLeer (EV) Limiter

The vanLeer limiter [vanLeer (1974)] for one-dimensional problem is extended in this work for two-dimensional triangular mesh. The Extended vanLeer is a **rank 3** limiter because it considers three limiting triangles (Figure 5.6) *i.e.*, ΔAOC, ΔBOC and ΔCOB. The gradient vectors in Figure 5.6 for ΔABC, ΔAOC, ΔBOC and ΔCOB can be denoted as S_1, S_2 and S_3.

$$\Phi_{EV} = S_0 = 0.8\frac{S_1.\left\|S_2\right\|.\left\|S_3\right\| + \left\|S_1\right\|.S_2.\left\|S_3\right\| + \left\|S_1\right\|.\left\|S_2\right\|.S_3}{\left\|S_2\right\|.\left\|S_3\right\| + \left\|S_1\right\|.\left\|S_3\right\| + \left\|S_1\right\|.\left\|S_2\right\|}. \tag{5.43}$$

5.9.2.5 Maximum Limited Gradient (MLG) Limiter

Batten *et al.* (1996) introduced the MLG operator in 1996 and they have shown that it reduces to Roe's Superbee limiter in one dimension, which is the most compressive limiter that still lies within Sweby's second order TVD region [Vollmer (2003)]. MLG is a rank 4 limiter because it considers four limiting triangles (Figure 5.6) *i.e.*, ΔABC, ΔAOC, ΔBOC and ΔCOB. The MLG limiter is based on computing various gradient operators in an LCD fashion and then retaining the steepest one of them as follows:

1. Compute

$$\vec{L}^0 = \vec{L}_{LCD}\left(\Delta ABC\right), \qquad \vec{L}^1 = \vec{L}_{LCD}\left(\Delta ABO\right), \\ \vec{L}^2 = \vec{L}_{LCD}\left(\Delta AOC\right) \text{ and } \qquad \vec{L}^3 = \vec{L}_{LCD}\left(\Delta OBC\right). \tag{5.44}$$

2. Set

$$\Phi_{MLG} = \vec{L}_{MLG} = \vec{L}^i \quad \text{such that} \quad \left|\vec{L}^i\right| = \max_{0<k<3}\left|\vec{L}^k\right| \quad . \tag{5.45}$$

5.9.3 The MUSCL-Hancock Scheme

vanLeer (1979) developed a method for higher-order in time and space and attributed this method to S. Hancock. Therefore, this approach is know as MUSCL-Hancock method. The MUSCL-Hancock scheme [Quirk (1994), Toro (1999)] is a second-order extension of the Godunov upwind method to provide second-order accuracy in in space and time. The MUSCL-Hancock scheme computes the intercell flux $f_{i+1/2}$ in the following three steps.

Step I: Data Reconstruction- Given a set of constants, average states within a cell represents the same value at any point within the corresponding cell. In the presence of discontinuities, as shown in the Figure 5.1, some numerical methods may generate oscillations in the results. Therefore, to tackle this problem of discontinuity over space, MUSCL-Hancock reconstructs the data using boundary interpolated values within each mesh cell. For the MUSCL interpolation we can use any of the limiters shown in section 5.9.2.

Step II: Evolution of State variables at half time- An intermediate solution is then found by advancing this reconstructed solution by half a time step. The reconstructed data is used to calculate flux for each element.

$$\mathbf{U}_k^{n+\frac{1}{2}} = \mathbf{U}_k^n - \frac{\Delta t}{2A_k}\left[\sum_{m=1}^{M}\mathbf{F}(\mathbf{U}_m)^n \cdot \mathbf{L}_m - \mathbf{S}_k^n\right]. \tag{5.46}$$

This step is entirely contained within each element as the side fluxes are evaluated at the boundary extrapolated values obtained from step I. At each side there are two fluxes, namely $F(U_R)$ and $F(U_L)$, which are in general distinct. This does not really affect the conservative character of the overall method, as this step is only an intermediate step; the side flux F_s to be used in 4.20 is yet to be evaluated.

Step III: Solution of the piecewise constant data Riemann problem- The intermediate solution (from step II) defines a set of state variables in each element, which then using a limiter sets the left- and right-hand states for a series of Riemann problems solved at the sides of the mesh. The solutions to these Riemann problems provide a set of upwinded interface fluxes which are used to integrate the original flow solution forward by one full time step.

$$\mathbf{U}_k^{n+1} = \mathbf{U}_k^n - \frac{\Delta t}{A_k}\left[\sum_{m=1}^{M} \mathbf{F}(\mathbf{U}_m^L, \mathbf{U}_m^R)^{n+\frac{1}{2}} \cdot \mathbf{L}_m - \mathbf{S}_k^{n+\frac{1}{2}}\right]. \tag{5.47}$$

Object-Oriented Design

MUSCL-Hancock scheme is a part of the GenericElement. The step I (*i.e.*, data reconstruction) is described in section 5.9.1. Steps II and III are also part of the GenericElement, where the state variables are updated for intermediate and final steps.

5.10 Conclusions

This chapter has presented the theory of high-resolution methods for hyperbolic problems and discussed its object-oriented implementation.

The slope limiters presented in this chapter is separated from other classes and bundled into one package. The benefits of modelling limiters in a separate package are as follows:

- All the limiters in this package subclass the GenericLimiter class and the limiters have to provide only gradients. Therefore, new limiters requires much less effort to introduce to be introduced.
- It helps to keep the code neat and allow more number of limiters to be introduces without disturbing the existing code.

Other important developments in this chapter are as follows:

- introduction of *rank* to the limiter, in chapter 8 it can be seen that this rank helps to predict the behaviour of the limiters.

- an extended version of one-dimensional VanLeer limiter for unstructured triangular mesh is developed (section 5.9.2.4). This limiter has shown stable and good results compared to other limiters (see chapter 8).

CHAPTER 6

Free-Surface Shallow Water Flows

6.1 Introduction

This chapter is intended to provide a brief background on the theory of the two-dimensional depth-averaged shallow water equations. As stated in the previous chapters, the shallow water equations are hyperbolic in nature and are extended in the present study using the generic *Riemann2D*.

The objective of this chapter is to present:

- the hyperbolic characteristic of shallow water equations;
- the theory related to the extended shallow water model;
- the object-oriented implementation of the theory in the *Riemann2D* design.

The practical value of this chapter is that it helps explain the significance of the input parameters and also highlights the reliability of the *Riemann2D* results as shown in chapter 8.

Note: The gray object-oriented comment boxes, mentioned in the section 1.4, are only for the discussion of object-oriented implementation.

6.2 Shallow Water Flows

The shallow water problem is also referred as *free surface flows*. The reason for the *free* designation arises from the large difference in the densities of the gas and liquid (*e.g.*, the ratio of water to air is 1000). A low gas density means that its inertia can generally be ignored compared to that of the liquid. In this sense the liquid moves independently, or freely, with respect to the gas. The only influence of the gas is the pressure it exerts on the liquid surface. In other words, the *gas-liquid* surface is not constrained, but free. The mathematical models of the so-called *free surface* type governs a wide variety of physical phenomena for scientific and practical problems of scientific interest, ranging from conventional water wave problems to sloshing in fuel tanks in rocket technology.

An important class of problems of practical interest involve water flows with a free surface under the influence of gravity. This class includes tides in the ocean, tsunami waves (see Figure 6.1) breaking of waves in rivers, surges, and dam-break wave modelling. A key assumption made in the derivation of the approximate shallow water theory concerns the pressure distribution; which is hydrostatics. This results from the assumption that the vertical acceleration of the water particles negligible compared to velocity of the water particles in horizontal plane.

Figure 6.1: On December 26, 2004, an earthquake off the Indonesian island of Sumatra triggered a tsunami that struck the coasts of Indonesia, Sri Lanka, India, Thailand, the Maldives, Malaysia, Burma, the Seychelles, and Somalia. Giant waves devastated the coastlines, leaving 181,516 people dead, 49,936 missing, and 1.8 million displaced [Renton and Palmer (2005)].

Depth-averaged modelling is based on the basic physical principles of conservation of mass and momentum and on a set of constitutive laws which relate the driving and resisting forces to fluid properties and motions. The differential equations of flow are derived by considering a differential volume element of fluid and describing mathematically:

1. The conservation of mass of fluid entering and leaving the control volume; the resulting mass balance is called the *equation of continuity.*

2. The conservation of momentum entering and leaving the control volume; called the *equation of motion.*

Applications of shallow water models can be found in many day-to-day events. For instance, modelling tidal fluctuations for those interested in capturing tidal energy for commercial purposes; predicting tidal ranges and surges which can then be used in the development planning of coastal areas; and, upon coupling to a transport model, considering flow and transport phenomena. The latter application makes it possible to study remediation options for polluted bays and estuaries, to predict the impact of commercial projects on fisheries, to model salinity intrusion effects, and to study the effects of wetting-induced mineral seepage into streams. Study of open channel problems like roll waves, flood waves in rivers, surges, and dam-break wave modelling helps the study of the inland water problems.

6.3 Water Flow with a Free Surface

Consider the flow of water with a free surface under gravity in a three-dimensional domain. Figure 6.2 depicts the convention for spatial coordinates; $x - y$ determining a horizontal plane whilst z defines the vertical direction, which is associated with the *free-surface* elevation.

The bottom boundary which can be called just as *bottom* or *bed*, is defined by a function

$$z = d(x, y), \tag{6.1}$$

and the free surface is defined by

$$z = s(x, y, z) \equiv d(x, y) + h(x, y, t), \tag{6.2}$$

where $h(x, y, t)$ is the depth of water, the vertical distance between the bottom and the free surface position. Figure 6.3 depicts the geometry for a simplified situation for a chosen value of y.

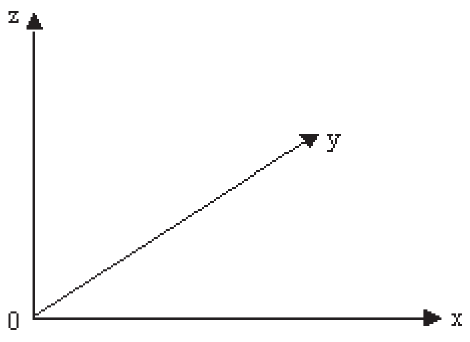


Figure 6.2: Coordinate convention for flow with a free surface under gravity, $x - y$ give the horizontal plane and z defines the vertical direction

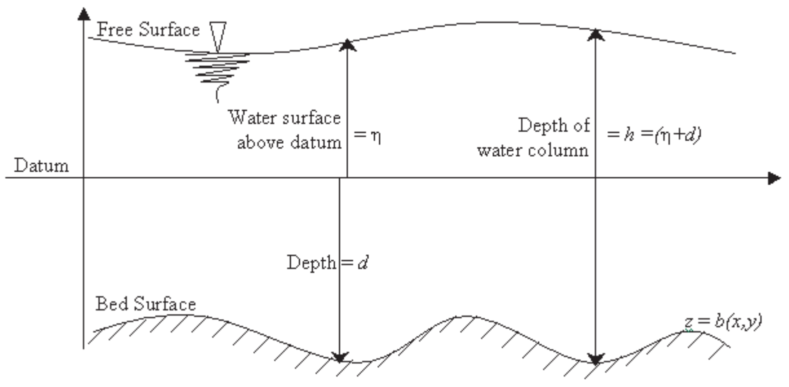


Figure 6.3: Flow with a free surface under gravity, for a fixed section y. The depth of water above datum, η is calculated in +ve direction above datum. Whereas, depth of bed surface is calculated in +ve direction below datum.

Assuming the density of the fluid is constant, the governing equations for the shallow water are given as:

$$u_x + v_y + w_z = 0 \tag{6.3}$$

$$u_t + uu_x + uv_y + uw_z = -\frac{1}{\rho}p_x \tag{6.4}$$

$$v_t + vu_x + vv_y + vw_z = -\frac{1}{\rho}p_y \tag{6.5}$$

$$w_t + wu_x + wv_y + ww_z = -\frac{1}{\rho}p_z - g \tag{6.6}$$

Here we have assumed that the body force vector is $\mathbf{g} = (0, 0, -g)$, where g is the acceleration due to gravity, taken as $9.8m/s^2$, a constant.

In principle, given initial conditions at time $t = 0$ and boundary conditions on the bottom and the free surface, the solution of the four equations 6.3 - 6.6 can be found for the four unknowns p, u, v, w. The main difficulty in solving the full problem is associated with the free surface, but the position of this boundary itself is unknown and therefore the domain on which the equations are to solved is not known. Approximate theories leading to simpler problem exist:

1. One such approximate theory assumes that the amplitude of the free-surface disturbance from the rest position is small with respect to a characteristic length, such as wave length. This assumption leads to linear boundary value problems and thus to a linear theory.

2. Another approximation, which is used for the shallow water extension of *Riemann2D* development, results from the assumption that the depth of water is small with respect to wavelength or free-surface curvature.

Before deriving the shallow water theory the boundary conditions for the full problem (equations 6.3 - 6.6) are discussed. Assuming that a boundary is given by the surface

$$\Omega(x, y, z) = 0, \tag{6.7}$$

then for the free surface we have

$$\Omega(x, y, z) \equiv z - s(x, y, t) = 0, \tag{6.8}$$

and for the bottom boundary we have

$$\Omega(x, y, z) \equiv z - d(x, y) = 0. \tag{6.9}$$

Two boundary conditions are imposed on the free surface $s(x, y, t)$, given by equation 6.8, namely the kinematic condition

$$\frac{\mathrm{d}}{\mathrm{d}t}\Omega(x, y, z) = \Omega_t + u\Omega_x + v\Omega_y + w\Omega_z = 0, \tag{6.10}$$

and the dynamic condition

$$p(x, y, z, t)|_{z=s(x,y)} = p_{atm} = 0, \tag{6.11}$$

where p_{atm} is the atmospheric pressure, which for convenience is taken as zero. For the bottom boundary (x, y), the condition in equation 6.10 also applies, with Ω given by 6.9.

6.4 Two-Dimensional Depth-Average Shallow Water Equations

The first assumption in the derivation of the shallow water equations is that the vertical component of acceleration, given by

$$\frac{\mathrm{d}w}{\mathrm{d}t} = w + uw_x + vw_y + ww_z, \tag{6.12}$$

is negligible. Insertion of this condition $\mathrm{d}w/\mathrm{d}t = 0$ into equation 6.6 gives

$$p_z = -\rho g. \tag{6.13}$$

Given the dynamic condition 6.11 that the atmospheric pressure is zero on the free surface, we obtain

$$p = \rho g(s - z). \tag{6.14}$$

Differentiation of equation 6.14 with respect to x and y gives

$$p_x = \rho g s_x \quad \text{and} \quad p_y = \rho g s_y. \tag{6.15}$$

Here, p_x and p_y are both independent of z and thus the x and y components of the acceleration of water particles $\mathrm{d}u/\mathrm{d}t$ and $\mathrm{d}v/\mathrm{d}t$ are independent of z. hence the x and y

velocity components u and v are also independent of z, that is $u_z = v_z = 0$. Therefore by the virtue of the above conditions and by making use of equation 6.15 in equations 6.4 and 6.5, we have

$$u_t + uu_x + vu_y = -gs_x, \tag{6.16}$$

and

$$v_t + uv_x + vv_y = -gs_y. \tag{6.17}$$

An important step in deriving the shallow water equations now follows. We integrate the continuity equation 6.3 with respect to z, the vertical coordinate, between $z = d(x, y)$ (bottom) and $z = s(s, y, t)$ (free surface). That is

$$\int_b^s (u_x + v_y + w_z)\mathrm{d}z = 0, \tag{6.18}$$

which leads to

$$w|_{z=s} - w|_{z=b} + \int_b^s u_x \mathrm{d}z + \int_b^s v_y \mathrm{d}z = 0. \tag{6.19}$$

We now apply the boundary condition in order to determine the first two terns in equation 6.19 above. Expanding equation 6.10 as applied to the free surface in equation 6.8 gives

$$(s_t + us_x + vs_y + ws_z - w)|_{z=s} = 0. \tag{6.20}$$

Expanding equation 6.10 as applied to the bottom boundary in equation 6.9 gives

$$(ub - vb - w)|_{z=d} = 0. \tag{6.21}$$

From equation 6.20 we obtain

$$w|_{z=s} = (s_t + us_x + vs_y)|_{z=s}, \tag{6.22}$$

and from equation 6.21 we obtain

$$w|_{z=b} = (ub_x + vb_y)|_{z=d}. \tag{6.23}$$

Substituting 6.22 and 6.23 into 6.19 gives

$$(s_t + us_x + vs_y)|_{z=s} - (ub_x + vb_y)|_{z=d} + \int_b^s u_x dz + \int_b^s v_y dz = 0. \tag{6.24}$$

To simplify equation 6.24 further, we use Leibniz's formula:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\int_{\xi_1(\alpha)}^{\xi_2(\alpha)} f(\xi,\alpha)\mathrm{d}\alpha = \int_{\xi_1(\alpha)}^{\xi_2(\alpha)} \frac{\partial f}{\partial \alpha}\mathrm{d}\xi + f(\xi_2,\alpha)\frac{\mathrm{d}\xi_2}{\mathrm{d}\alpha} - f(\xi_2,\alpha)\frac{\mathrm{d}\xi_1}{\mathrm{d}\alpha} \tag{6.25}$$

Applied to the last two integral terms in equation 6.24, we obtain

$$\int_b^s u_x dz = \frac{\partial}{\partial x}\int_b^s u dz - u|_{z=s}.s_x + u|_{z=d}.d_x, \tag{6.26}$$

and

$$\int_b^s v_y dz = \frac{\partial}{\partial y}\int_b^s v dz - v|_{z=s}.s_y + u|_{z=d}.d_y. \tag{6.27}$$

Substitution of equation 6.26 and 6.27 into 6.24 gives

$$s_t + \frac{\partial}{\partial x}\int_b^s u dz + \frac{\partial}{\partial y}\int_b^s v dz = 0. \tag{6.28}$$

Recall that both u and v are independent of z; also $s = d + h$ and $d_t = 0$. Equation 6.28 then simplifies to

$$h_t + (hu)_x + (hv)_y = 0. \tag{6.29}$$

This is the law of conservation of mass and is written in differential conservation law form.

The momentum equations 6.4 and 6.5 can also be expressed in differential conservation law form. To this end we add equation 6.29, pre-multiplied by u, to equation 6.16, pre-multiplied by h; we also make use of a relation that does assume differentiability of the water depth, namely

$$h\frac{\partial h}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{2}h^2\right), \tag{6.30}$$

to obtain

$$(hu)_t + (hu^2 + \frac{1}{2}gh^2)_x + (huv)_y = -ghd_x. \tag{6.31}$$

similarly, for the y momentum equation we obtain

$$(hv)_t + (huv)_x + (hv^2 + \frac{1}{2}gh^2)_y = -ghd_y. \tag{6.32}$$

All three partial differential equations 6.29, 6.31 and 6.32 can be written in differential conservation law form as the single vector equation

$$U_t + F(U)_x + G(U)_y = S(U), \tag{6.33}$$

with

$$U = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}, \qquad F(U) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ hv \end{bmatrix}, \qquad G(U) = \begin{bmatrix} hu \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix} \text{ and } \quad S(U) = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}. \tag{6.34}$$

Equation 6.33 may be written in the quasi-linear form as (4.25)

$$U_t + A(U)\,U_x + B(U)\,U_y = 0, \tag{6.35}$$

where $A(U)$ and $B(U)$ are the Jacobian matrices, as defined in equation 4.13, of the fluxes $F(U)$ and $G(U)$ respectively.

It should be noted that equation 6.33 has the same form as the generic equation for the hyperbolic equation, as shown in equation 4.14. In the system of equations 6.34, U is the vector of conserved variables, $F(U)$, $G(U)$ are flux vectors in the x and y directions respectively, and $S(U)$ is the source term vector. These equations can be compared to the generic equations for hyperbolic problems in equation 4.12. This comparison shows that hyperbolic equations follow the same behaviour and can be modelled using the generic *Riemann2D* model.

Due to the generic nature of the governing equations, the overall method of solving any hyperbolic equation would be the same. For the extension to superclass, it need to provide the conserved variable and thus the corresponding fluxes, such as equation 6.34 for shallow water equations.

Object-Oriented Design

Equation 6.34 defines the components of the vector of state variables and vector of fluxes in equations 4.12. These state variables and fluxes are specific to the shallow water equations, which makes them a part of the subclasses. These state variables are initialised in the Element class.

6.5 Hyperbolic Character of the Shallow Water Equations

It is important to note that the arguments of the flux functions, actually their components, are the components of the vector of the conserved variables. To make this clear, we express the flux functions in terms of the components u_1, u_2, u_3 of U,

$$q = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{6.36}$$

$$F(u) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix} = \begin{bmatrix} u_2 \\ (u_2)^2/u_1 + \frac{1}{2}g(u_1)^2 \\ u_2u_3/u_1 \end{bmatrix} \tag{6.37}$$

$$G(u) = \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix} = \begin{bmatrix} u_3 \\ u_2u_3/u_1 \\ (u_3)^2/u_1 + \frac{1}{2}g(u_1)^2 \end{bmatrix}. \tag{6.38}$$

The variables in the quasi-linear form (equation 6.70) are conserved variables, the formulation of the equation is non conservative. Next we calculate the Jacobian matrices (see equation 4.13) and express them in terms of the non-conservative variables u, v, c, where c is the speed of the wave defined as $c = \sqrt{gh}$.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ -(u_2/u_1)^2 + gu_1 & 2u_2/u_1 & 0 \\ -u_2u_3/(u_1)^2 & u_3/u_1 & u_2/u_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -u^2 + c^2 & 2u & 0 \\ -uv & v & u \end{bmatrix}. \tag{6.39}$$

Similarly,

$$B = \begin{bmatrix} 0 & 0 & 1 \\ -uv & v & u \\ -v^2 + c^2 & 0 & 2u \end{bmatrix}. \tag{6.40}$$

Now consider a matrix C that is a linear combination of the two Jacobian matrices A and B, namely

$$C = \omega_1 A + \omega_2 B, \tag{6.41}$$

where the coefficients ω_1 and ω_1 are two real parameters that define a non-zero vector $\omega = [\omega_1 + \omega_2]$

$$\omega = \sqrt{\omega_1^2 + \omega_2^2} > 0. \tag{6.42}$$

The matrix C is given by

$$C(U) = \begin{bmatrix} 0 & \omega_1 & \omega_2 \\ (-u^2 + c^2)\omega_1 - uv\omega_2 & 2u\omega_1 + v\omega_2 & u\omega_2 \\ -uv\omega_1 + (-v^2 + c^2)\omega_2 & v\omega_1 & u\omega_1 + 2v\omega_2 \end{bmatrix}. \tag{6.43}$$

The eigenvalues of C can be given as

$$\begin{aligned} \lambda_1 &= u\omega_1 + v\omega_2 - c\,|\omega| \\ \lambda_2 &= u\omega_1 + v\omega_2 \\ \lambda_3 &= u\omega_1 + v\omega_2 + c\,|\omega|\,. \end{aligned} \tag{6.44}$$

The time-dependent two-dimensional shallow water equations 6.33 - 6.34 are hyperbolic [Toro (2001)]. For a wet bed they are strictly hyperbolic. This is based on the fact that the above eigenvalues in equation 6.44 are all real and, for $h > 0$, distinct.

6.5.0.1 Eigen Structure for Shallow Water Equations with Solute

To model the rotation test[1] in the present work. Following derivation gives the eigenvalues and eigenvectors for the shallow water equations with solute.

Assume that there are I particles, whose concentration is given by ϕ. The vector of state variable can be given as,

$$\begin{aligned} q &= [h, hu, hv, h\phi_1,\ h\phi_2,\ .....\ ,\ h\phi_I]^T \\ &= [u_1, u_2, u_3, u_4,\ ...\ , u_{3+I}]^T \end{aligned} \quad , \tag{6.45}$$

and the vector of fluxes can be given as,

$$\begin{aligned} F(u) &= [uh,\ u^2h + 0.5gh^2,\ uvh\ ,\ uh\phi_1\ ,\ uh\phi_2,\ .....\ ,\ uh\phi_I]^T \\ &= [u_2,\ \tfrac{u_2^2}{u_1} + \tfrac{g}{2}u_1^2,\ \tfrac{u_2u_3}{u_1},\ \tfrac{u_2u_4}{u_1}\ ,\ \tfrac{u_2u_5}{u_1}\ ,\dots,\ \tfrac{u_2u_{3+I}}{u_1}]^T \end{aligned} \quad , \tag{6.46}$$

$$\begin{aligned} G(u) &= [vh\ ,\ uvh\ ,\ hv^2 + 0.5gh^2\ ,\ vh\phi_1\ ,\ vh\phi_2\ ,\ .....\ ,\ vh\phi_I]^T \\ &= [u_3\ ,\ \tfrac{u_2u_3}{u_1}\ ,\ \tfrac{u_3^2}{u_1} + \tfrac{g}{2}u_1^2\ ,\ \tfrac{u_3u_4}{u_1}\ ,\ \tfrac{u_3u_5}{u_1}\ ,\dots,\ \tfrac{u_3u_{3+I}}{u_1}]^T \end{aligned} \quad , \tag{6.47}$$

Next we calculate the Jacobian matrices (see equation 4.13) and express them in terms of the non-conservative variables u, v, c, where c is the speed of the wave defined

[1]Rotation test for shallow water equations has been performed and presented in chapter 8

as $c = \sqrt{gh}$.

$$A=\begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ -u_2^2/u_1^2 + c & 2u_2/u_1 & 0 & 0 & \cdots & 0 \\ -u_2u_3/u_1^2 & u_3/u_1 & u_2/u_1 & 0 & \cdots & 0 \\ -u_2u_4/u_1^2 & u_4/u_1 & 0 & u_2/u_1 & 0 & \vdots \\ & \vdots & \vdots & \vdots & \ddots & 0 \\ -u_2u_{3+I}/u_1^2 & u_{3+I}/u_1 & 0 & 0 & \cdots & u_2/u_1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ -u^2 + gh & 2u & 0 & 0 & \cdots & 0 \\ -uv & v & u & 0 & \cdots & 0 \\ -u\phi_1 & \phi_1 & 0 & u & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -u\phi_I & \phi_I & 0 & 0 & \cdots & u \end{bmatrix} \tag{6.48}$$

similarly,

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 & \cdots & 0 \\ -uv & v & u & 0 & \cdots & 0 \\ -v^2 + gh & 0 & 2v & 0 & \cdots & 0 \\ -v\phi_1 & 0 & \phi_1 & v & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -v\phi_I & 0 & \phi_I & 0 & \cdots & v \end{bmatrix} \tag{6.49}$$

The eigenvalues for Jacobian A can be given as

$[u - c, u, u + c, u, u, ..., u]$

and corresponding eigenvectors as

$[1\ ,\ u - c\ ,\ v\ ,\ \phi_1\ ,\ \phi_2\ ,\ ...\ ,\ \phi_I]^T$

$[0\ ,\ 0\ ,\ 1\ ,\ 0\ ,\ 0\ ,\ ....\ ,\ 0]^T$

$[1\ ,\ u + c\ ,\ v\ ,\ \phi_1\ ,\ \phi_2\ ,\ ...\ ,\ \phi_I]^T$

$[0\ ,\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ,\ ....\ ,\ 0]^T$

$[0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ,\ ....\ ,\ 0]^T$

continuing until

$[0\ ,\ 0\ ,\ 0\ ,\ ....\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ 1]^T$

Similarly, The eigenvalues for Jacobian B can be given as

$[v - c, v, v + c, v, v, ..., v]$

and corresponding eigenvectors as

$[1\ ,\ u\ ,\ v - c\ ,\ \phi_1\ ,\ \phi_2\ ,\ ...\ ,\ \phi_I]^T$

[0 , -1 , 0 , 0 , 0 , ... , 0]^T
[1 , u , $v+c$, ϕ_1 , ϕ_2 , ... , ϕ_I]^T
[0 , 0 , 0 , 1 , 0 , ... , 0]^T
[0 , 0 , 0 , 0 , 1 , 0 , ... , 0]^T
[0 , 0 , 0 , 0 , 0 , 1 , 0 , ... , 0]^T
[0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , ... , 0]^T
continuing until
[0 , 0 , 0 , ... , 0 , 0 , 0 , 1]^T

6.6 Approximate Riemann Solvers

6.6.1 The Roe Solver

Roe's approximate Riemann solver is discussed in section 4.3.3 for the generic approach. Due to dependence of the conserved variables on the type of the problem, the Riemann problem implementation is done here. The first approach of Roe's approximate Riemann solver to the shallow water equations can be found in the literature by Glaister (1988). Glaister followed the Roe-Pike approach [Roe and Pike (1984)] to derive the averages at the cell boundaries. Therefore, following Roe's approach from section 4.3.3, we can can write the Roe averages for shallow water equations using the parameter vectors $z = h^{-1/2}q$ *i.e.*,

$$\begin{bmatrix} z^1 \\ z^2 \\ z^2 \end{bmatrix} = \begin{bmatrix} \sqrt{h} \\ \sqrt{h}u \\ \sqrt{h}v \end{bmatrix}. \tag{6.50}$$

The Roe averages can be found as:

$$\tilde{u} = \frac{\overline{z}^2}{\overline{z}^1} = \frac{\sqrt{h_L}u_R + \sqrt{h_L}u_R}{\sqrt{h_L} + \sqrt{h_R}} \tag{6.51}$$

$$\tilde{v} = \frac{\overline{z}^3}{\overline{z}^1} = \frac{\sqrt{h_L}v_L + \sqrt{h_R}v_R}{\sqrt{h_L} + \sqrt{h_R}} \tag{6.52}$$

$$\tilde{h} = \frac{1}{2}(h_L + h_R) \tag{6.53}$$

$$\tilde{c} = \sqrt{g\tilde{h}}. \tag{6.54}$$

The average eigenvalues are:

$$\tilde{\lambda}_1 = \tilde{u} - \tilde{c} \qquad \tilde{\lambda}_2 = \tilde{u} \qquad \tilde{\lambda}_3 = \tilde{u} + \tilde{c}, \tag{6.55}$$

and the corresponding eigenvectors are:

$$\tilde{K}^{(1)} = \begin{bmatrix} 1 \\ \tilde{u} - \tilde{c} \\ \tilde{v} \end{bmatrix} \qquad \tilde{K}^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \tilde{K}^{(1)} = \begin{bmatrix} 1 \\ \tilde{u} + \tilde{c} \\ \tilde{v} \end{bmatrix}. \tag{6.56}$$

The wave strength $\tilde{\alpha}_j$, in terms of the averages, are

$$\begin{aligned} \tilde{\alpha}_1 &= \frac{\Delta h(\tilde{u} + c) - \Delta(hu)}{2c} \\ \tilde{\alpha}_2 &= -v\Delta(h) + \Delta(hv) \\ \tilde{\alpha}_3 &= \frac{-\Delta h(\tilde{u} - c) + \Delta(hu)}{2c}. \end{aligned} \tag{6.57}$$

Now the values from equations 6.55, 6.56 and 6.57 can be used to find the F_{Roe} in equation 4.50.

Object-Oriented Design

Equations 6.55, 6.56 and 6.57 defines the essential components (eigenvalues, eigenvectors and wave strength) of Roe's approximate Riemann solver. These components are derived according the generic rule of Roe's approximate Riemann solver. But their derivation require the state variables of the subclass, which make these components a part of Side class (subclass of GenericSide).

The Roe solver works well for shallow water flow problems but in certain situations it can produce negative depths. In addition to being physically incorrect, these negative depths usually pose computational difficulties introducing errors. Consider the solution to a linear Riemann problem (see section 4.3 for detailed description) with a Roe matrix $\tilde{A}$.

$$\begin{aligned} & u_t + Au_x = 0 \\ & u(x,0) = \begin{cases} u_l & x < 0 \\ u_r & x > 0 \end{cases} \end{aligned} \tag{6.58}$$

The solution's middle state, u^*, is connected to the state on the left, u_l, by a jump in the first eigenvector of $\tilde{A}$ and is connected to the state on the right, u_r, by a jump in the second eigenvector. The left and right states are therefore joined in phase space by the two eigenvectors, the middle state, u^*, being the intersection of the two vectors.

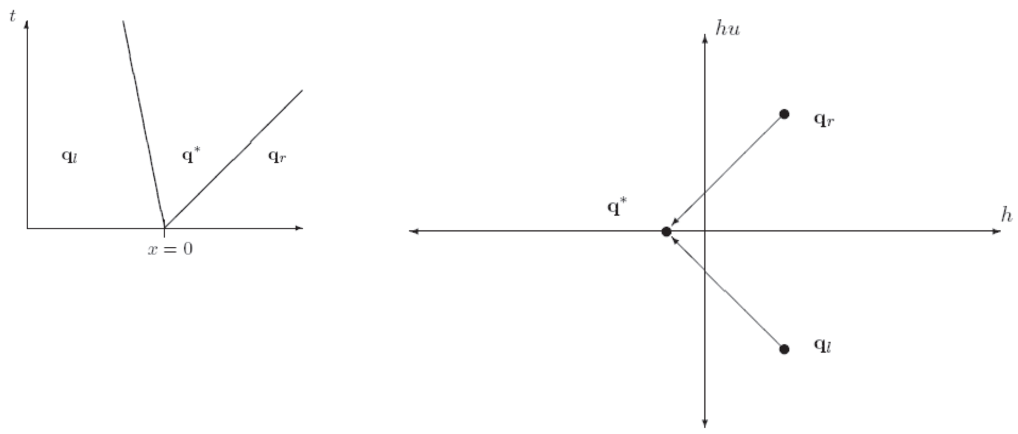


Figure 6.4: Failure of the Roe solver when the left and right states, u_l and u_r, lie in particular regions of phase space. In this example the left and right states are shallow, with opposite velocities. The arrows are in the direction of the Roe eigenvectors, the intersection of which is the middle state u^*, corresponding to a negative depth h [George (2004)]

This intersection corresponds to a negative depth when the left and right states lie in particular regions of phase space; see Figure 6.4. For instance, if the velocity on the right is much greater than the velocity on the left and the flow is already very shallow, the intersection often corresponds to a negative depth. For applications where the flow is very shallow in parts of the domain, or in fact becomes dry in regions, the Roe solver is almost certain to produce negative depths.

It is generally accepted that the numerical computation of dry/wet bed fronts is very difficult [Toro (2001)]. One way to resolve this difficulty is to compute the exact Riemann solution, which as mentioned is possible for the homogeneous shallow water equations. However, this is computationally expensive, and it is not clear how to extend this satisfactorily to a Riemann problem in a triangular element with a source term. Another approach often adopted is to assign the depth to a minimum level, which makes the water non-zero for all cases. The problem with this approach is that it has shock front which is generally missing in a dry bed situation.

6.6.2 The HLL (Harten-Lax-van Leer) Solver

Following the description and derivation of the HLL (Harten-Lax-van Leer) approximate Riemann solvers in section 4.3.4, these equations are extended for the homogeneous shallow water equations. These solvers are based on estimating the speeds that information or waves propagate away from a Riemann problem. A linear solution with two discontinuities is then constructed, using estimates for the speeds of the propagating discontinuities. Two speeds are used in the original HLL method even for equations with more than two characteristic families. This is in contrast to a method such as the Roe solver, where a constant estimate to the Jacobian matrix is constructed first, and the eigenvalues of this estimate subsequently affect the approximate Riemann solution. The estimates for the speeds are based on the initial data, and on general properties of exact Riemann solutions. A number of different estimates for the speeds have been used, and the particular choice of estimates gives the HLL method its particular properties; see Toro (1999).

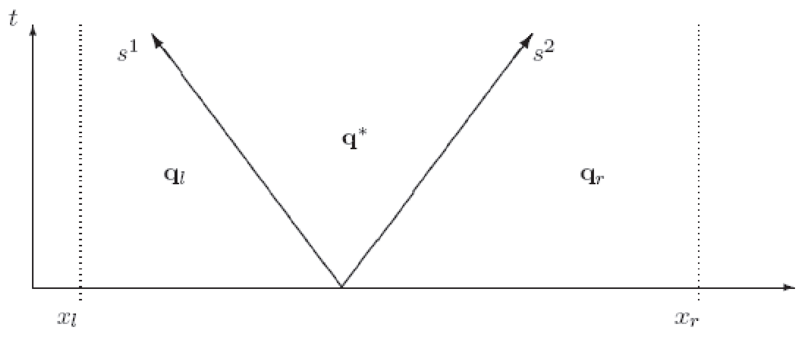


Figure 6.5: Constructing a solution using conservation and estimated speeds. If the speeds s^1 and s^2 are predetermined estimates, an approximate solution can be uniquely determined by applying the conservation law to the region $\omega = [x_l; x_r]$ surrounding the interface [George (2004)]

Only working on dry bed problems, the HLL approach highlights better behaviour, avoiding uni-dimensionalisation effects on the flow field [Caleffi *et al.* (2003)]. Such a reason leads to the choice of the HLL and its derived approximate Riemann solver (see section 4.3.4) for the development of the model where the dry bed problem occurs. Rewriting equation 4.56 in terms of the normal components, we have

$$F_{HLL} = \frac{S_R F_L.n - S_R F_L.n + S_L S_R \left(U_R - U_L\right)}{S_R - S_L}, \tag{6.59}$$

where n is the outward normal unit vector; $F_R = F(U_R)$ and $F_L = F(UL)$; subscripts R and L refer to the right and to the left side of the cell interface respectively. The

S_L and S_R symbols represent the wave speeds' propagation and they can be estimated through the *two expansion* approach [Caleffi *et al.* (2003)]:

$$\begin{array}{l} S_L = \min\left(q_L.n - \sqrt{gh_L}, u^* - \sqrt{gh^*}\right) \\ S_R = \min\left(q_R.n - \sqrt{gh_R}, u^* + \sqrt{gh^*}\right), \end{array} \tag{6.60}$$

where $q = (u, v)$ and:

$$\begin{array}{l} u^* = \frac{1}{2}\left(q_L + q_R\right).n + \sqrt{gh_L} - \sqrt{gh_R} \\ \sqrt{gh^*} = \frac{1}{2}\left(\sqrt{gh_L} + \sqrt{gh_R}\right) + \frac{1}{4}\left(q_L + q_R\right).n. \end{array} \tag{6.61}$$

Substituting the values from above equations 6.60 and 6.61 into equation 6.59 (or equation 4.56) we obtain the value of F_{HLL}.

Object-Oriented Design

Similar to the Roe's approximate Riemann solver, HLL is generic approximate Riemann solver (see section 4.3.4). Same as the Roe's, HLL's components are based on the governing equations of the hyperbolic system. Equations 6.60 and 6.61 defines these components, making it a part of Side class (subclass of GenericSide)

Consider a one dimensional Riemann problem where initial data is given, for *e.g.*, the left state, $[h_L, u_L]$ is specified, and the right state is dry so that $[h_R, u_R] = [0, 0]$. The exact solution to this problem can been seen in chapter 6 of Toro (2001). The solution consists of a single rarefaction associated with the left eigenvalue $\lambda_1 = u - a$. The expected right shock associated with the left eigenvalue $\lambda_2 = u + a$ is absent. The wet/dry front corresponds to the tail of the left rarefaction and has exact propagation speed $S_L = u_L + 2\sqrt{gh_L}$. It is interesting to note that the speed of this front is faster than one obtained from the usual evaluation of eigenvalues of the system.

A popular way of dealing with these kinds of problems is by artificially wetting the dry bed, that is by setting the water depth on the right-hand side in our problem to some small positive tolerance, namely $h_R = \epsilon > 0$ in the below illustration. Having done this the solution to the Riemann problem has a different structure to that of the exact problem for the dry bed conditions. The solution contains a relatively weak, right-propagation shock of speed S_R, which is meant to represent the wet/dry front speed. The speed S_R of this shock is considerably slower than that of the wet/dry front, S_{*L} (see Figure 6.4, S_{*L} is the speed of the wave in the star region). In the limit as ϵ tends to zero, the two speeds coincide. However, for practical values of the artificial bed-wetting parameter ϵ, the errors can be significant.

The above solution assumes that there exists a finite water depth everywhere. In the above example where the dry bed exists downstream $h_R = 0$, the two eigenvalues collapse into one and the system of equations is not strictly hyperbolic. Under these circumstances no shock exists and S_R represents the speed of the head of the rarefaction wave and S_L represents the speed of the toe of the rarefaction wave. Therefore, a general expression can be written as

$$S_L = \begin{cases} u_R - 2\sqrt{gh_R} & if \quad h_L = 0, \\ \text{usual estimate} & if \quad h_L > 0, \end{cases} \tag{6.62}$$

$$S_R = \begin{cases} u_L + 2\sqrt{gh_L} & if \quad h_R = 0, \\ \text{usual estimate} & if \quad h_R > 0. \end{cases} \tag{6.63}$$

Object-Oriented Design

HLL and its derived (see section 4.3.4) approximate solvers are well known for solving the dry bed problem. However, the equations 6.62 and 6.63 pose a problem when the state variables are η, hu and hv. This is because when we need to extract velocity components u and v from state variables, we have to divide the state variables hu and hv by h. For $h = 0$ or $h \rightarrow 0$, u and v will be ∞ or a large number. Therefore, in *Riemann2D* a non zero-depth is imposed if h is less than "mindepth". For the present case "mindepth" and non-zero depth is assumed to be 10^{-6}m, which is reasonably small for shallow water problems. This assumption also makes the Roe's solver to work for dry bed case in one-dimensional problems. Whereas, for two-dimensional problem this value of minimum depth needs to increased. In some test cases a value of 10^{-4} gives good results (see section 8.7)

6.6.3 Shu and Osher Solver

Similar to the Roes' and HLL approximate solvers, Shu and Osher (1988) method (SO) is generic in nature and have been known to work for hyperbolic systems. Nujic (1995) used the SO method with $\lambda = 0.4$ and found it attractive for solving the shallow water problems. Namin *et al.* (2004) has used SO method, when applying the model to mild gradient cases. Therefore, in this work $\lambda = 0.4$ and $a_{\max}$ = maximum value of the eigenvalues of the average Jacobean matrix. See equation 4.13 for Jacobian matrix and equation 6.55 for eigenvalues of shallow water equations.

Object-Oriented Design

Similar to the Roes' and HLL approximate Riemann solver, SO is generic approximate Riemann solver (see section 4.3.5). Same as the Roes' and HLL, SO's component (maximum eigenvalue) is based on the governing equations of the hyperbolic system, making it a part of Side class (subclass of GenericSide)

6.7 Source Terms

In recent years, the study of a hyperbolic system with source terms has attracted much attention in the CFD community [Xu (2002)]. One of the main reasons is that there exist wide engineering applications. For example, the Saint-Venant equations are widely used in ocean and hydraulic engineering to describe bore wave propagation, hydraulic jumps, and open-channel flow, among others. Many numerical schemes have been developed in the past decades for the above equations (see Bermudez and Vazquez (1994), Glaister (1988), Hubbard and Garcia-Navarro (2000), LeVeque (1998), Ghidaoui *et al.* (2001), Zhou *et al.* (2001) and references therein).

6.7.1 The Surface Gradient Method For Varying Bathymetry

In order to calculate the numerical fluxes in equations 4.50, 4.56 and 4.59, the source terms need to be calculated. Source terms due to varying bathymetry has been main concern in shallow water modelling on unstructured triangular meshes. The reason for this is shown in Figure 6.6, which can be explained as follows:

- Figure 6.6 (a): It shows the plan view of three triangles. In triangular mesh, the depth of the bed is assigned at the node, whereas, the free surface elevation is calculated at the centre of the triangle (*i.e.*, the element). The height of water column in each element is given be $h = \eta + d$, where $d = (d_1 + d_2 + d_3)/3$.

  For a stagnant ($u = 0, v = 0$) free surface and varying bathymetry, the left and right side of equations 6.31 and 6.32 should be equal. For a triangular element

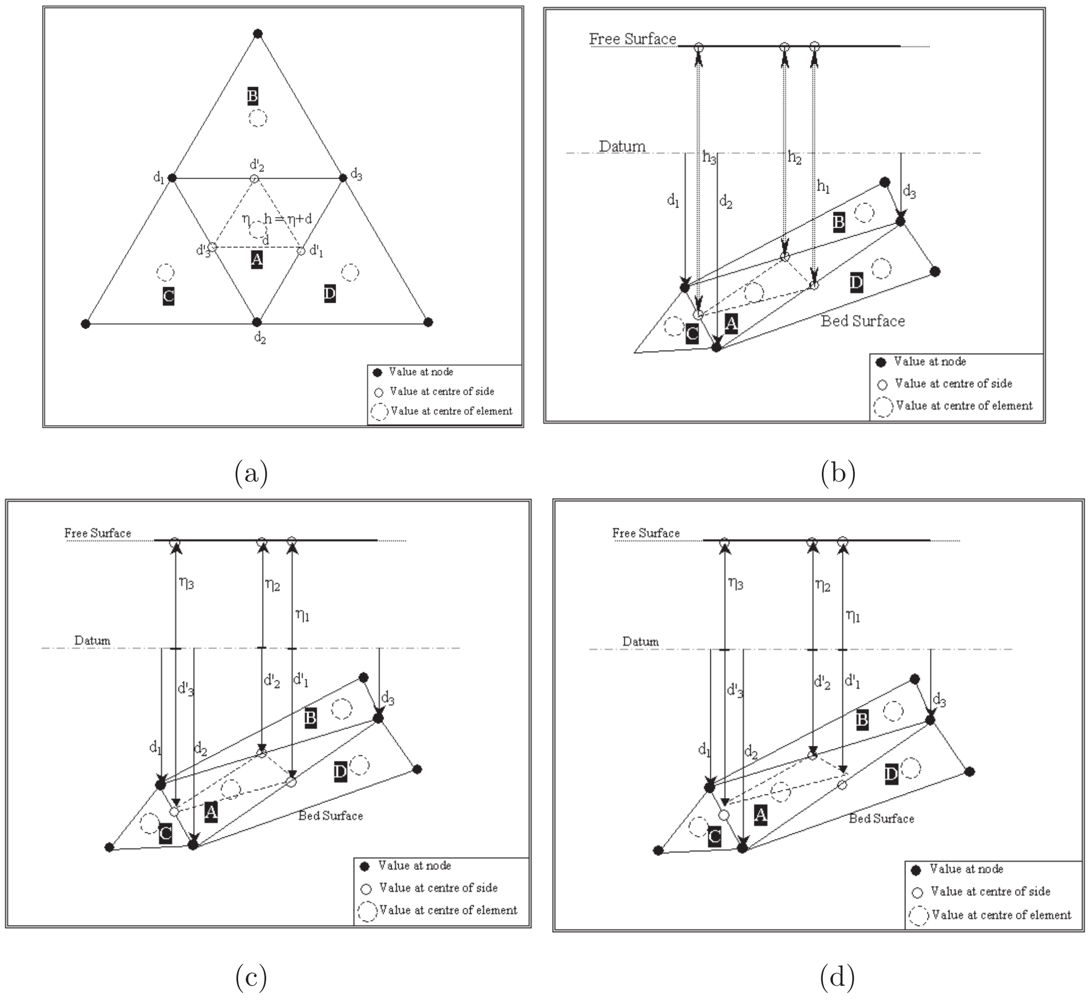


Figure 6.6: The Surface Gradient Method For Varying Bathymetry (a) Plan view of three triangles. (b) Elevation view: surface gradient for water column depth at side is equal to surface gradient of the bed depth. (c) Elevation view: surface gradient for water column depth (split into η, free surface height and d', depth at side) at side is equal to surface gradient of the bed depth. (d) Elevation view: surface gradient for water column depth (split into η, free surface height and d', depth at side) at side is not equal to surface gradient of the bed depth.

this relation can be written as:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & \sum_{side\,n=0}^{n=3} \left|0.5gh^2\right|_x & 0 \\ 0 & 0 & \sum_{side\,n=0}^{n=3} \left|0.5gh^2\right|_y \end{bmatrix} = \begin{bmatrix} 0 \\ -0.5ghd_x \\ -0.5ghd_y \end{bmatrix} \tag{6.64}$$

which can be written as:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & \sum_{siden=0}^{n=3} 0.5ghh_x & 0 \\ 0 & 0 & \sum_{siden=0}^{n=3} 0.5ghh_y \end{bmatrix} = \begin{bmatrix} 0 \\ -0.5ghd_x \\ -0.5ghd_y \end{bmatrix} \tag{6.65}$$

The left hand side of the above equations, surface gradient of water column depth, is calculated at the three sides of the triangle. Whereas, the right hand side of the above equation, surface gradient of bed depth, is calculated at the centre of the element. The correct solution can be obtained in the surface gradient (see Figure 6.6 a) of water column depth, $\Delta\, d_1 d_2 d_3$ is equal to surface gradient of bed surface, $\Delta\, d'_1 d'_2 d'_3$

- Figure 6.6 (b): It shows the situation when the surface gradient of the water column depth and the surface gradient of bed surface are equal. This situation is possible only when all the nodes lie on same plane.
- Figure 6.6 (c): It shows same situation as Figure 6.6 (b). The water column is split into η,free surface height and d', depth at side.
- Figure 6.6 (d): When the bottom surface is not doesnot lie on one plane, the surface gradient (see Figure 6.6 a) of water column depth, $\Delta\, d_1 d_2 d_3$ is not equal to surface gradient of bed surface, $\Delta\, d'_1 d'_2 d'_3$.

Many authors have presented ways to solve this problem, with reasonable success (see section 2.5.2). However, in this work, the surface gradient method proposed by Zhou *et al.* (2001) is adopted. In this approach the water surface elevation, η, is interpolated at the side of the element instead of the water depth, h. The water depth, h, at the side is then calculated using the water surface elevation and the bed topography, d, referenced from the datum. Mathematically, $h = \eta + d$. Therefore, in the present study the state variables are changed to η, hu, hv. Mohamadian *et al.* (2005) has shown that to satisfy the balance between the surface gradient of bed surface and water depth, the pressure term is moved from left side of the equation to right hand side[1] and deducted from the bed source term. This reduces to surface gradient of free surface height *i.e.*, η.

The surface gradient method leads to $\Delta h = 0$ at the cell faces in the case of stagnant water conditions. Zhou *et al.* (2001) used this method to solve the shallow water problem using HLL approximate Riemann solver. Mohamadian *et al.* (2005) used this approach to solve shallow water problems using Roe approximate solver.

In this work an same approach is adopted for Roes', HLL and Shu and Osher approximate solver. However, the depth of the element in the present work is defined at the sides of the element[2], same as the states at sides. The purpose of this is to allow the application of limiters to the bathymetry. This method can be useful for the moving bed problems. For the shallow water problems without moving bed, this method has given satisfactory results for the (test cases shown in chapter 7). In this work this is tested only for "no limiter" case.

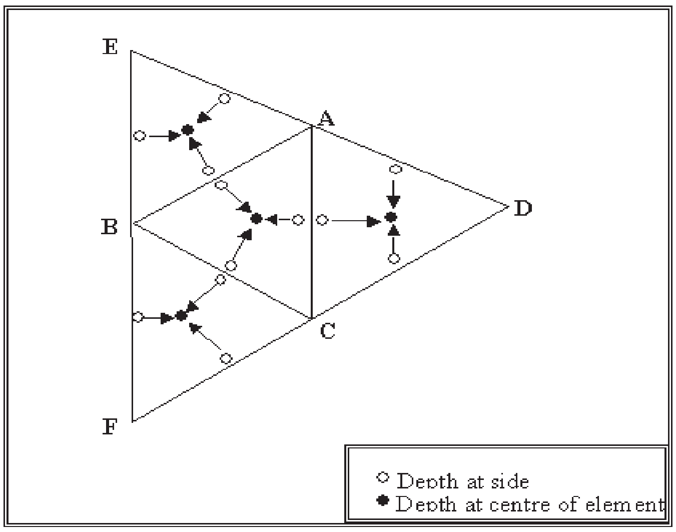


Figure 6.7: Reference pointers for implementing surface gradient within each element.

[1]Although pressure term is moved from left hand side of the equation to solve Riemann problem, the eigen structure of the shallow water equation remains same.

[2]Zhou *et al.* (2001) and Mohamadian *et al.* (2005) used the depth at the centre of the element.

Object-Oriented Design

Similar to the object-oriented method used for boundary condition, reference pointers are used to set the depth at the left and right side. This allows a method to be initialise indirectly by way of a reference to the variable, which is a once-only transfer of the appropriate reference to the left and right states for a side. It is part of the Element initialisation process and done only once. In other words, references are used to hook up the depth at sides to the depth at the centre of the element. Figure 6.7 shows the reference pointers, where depth at the sides of the triangle points to the centre depth of the element.

Other problem with the variable bathymetry is that the bed slope term in equations 6.31 and 6.32 is included in the bed slope, and it is discretised using a centered scheme.

6.7.2 Coriolis acceleration

Because the Earth rotates, a fluid that flows along the Earth's surface feels a coriolis acceleration perpendicular to its velocity. In the northern hemisphere, coriolis acceleration makes low pressure storm systems spin counterclockwise; however, in the southern hemisphere, they spin clockwise because the direction of the coriolis acceleration is reversed. The order of magnitude of the coriolis acceleration can be estimated from size of the *Rossby number*. The dimensionless ratio of inertia force to Coriolis force which gives an indication of the importance of rotation on flow in pipes. It is given by

$$R_0 \equiv \frac{v}{2\omega L \sin\theta}, \tag{6.66}$$

where v is the speed of fluid flow, ω is the angular velocity or rotation, and θ is the angle between the axis of rotation and the direction of fluid motion.

The *coriolis parameter*, f is defined as twice the vertical component of the Earth's angular velocity ω about the local vertical, and is given by

$$f = 2\Omega \sin\phi, \tag{6.67}$$

at latitude ϕ.

6.7.3 Bed Resistance

Resistance to flow is typically characterized by a roughness coefficient. The most commonly used equation for flow resistance is the Mannings equation. There are other resistance coefficients in use including the *Darcy-Weisbach* friction factor, f, and the Chezy C. These can all be converted easily to Mannings n.

Chezy Formula: This expression has the form:

$$V = C\,(RS)^{1/2} \tag{6.68}$$

Manning Formula: The following expression for the evaluation of the Chezy roughness coefficient was suggested by Flamant (1891):

$$C = \frac{R^{1/6}}{n} \tag{6.69}$$

where n is a roughness parameter which depends only upon the roughness characteristics of the boundary surface and which is known as the Manning roughness parameter.

In the present model the source term has forces: *coriolis force, bed stress, wind force, body force* and *bottom friction.* The source term is given as:

$$S\,(U) = \begin{pmatrix} q_s \\ -fhv + c_f u\sqrt{u^2+v^2} + gh\frac{\partial\eta}{\partial x} + c'_f w^2 \sin\alpha \\ fhu + c_f v\sqrt{u^2+v^2} + gh\frac{\partial\eta}{\partial y} + c'_f w^2 \sin\alpha \end{pmatrix}, \tag{6.70}$$

where u and$v$ are depth averaged velocities in the x and y directions respectively, h is the total water column depth, q is the source (or sink) discharge per cell area, g is the gravitational acceleration, f is the coriolis acceleration due to the Earth's rotation [wherein: $f = 2\Omega\sin\phi$ (where ω is Earth's angular velocity and ϕ latitude of the application area)], η is free surface elevation, υ_t is eddy viscosity, w is wind velocity, α is wind direction with regard to x axis, c'_f wind friction coefficient and c_f is bed resistant coefficient [wherein $c_f = \frac{g}{C^2} = \frac{gn^2}{h^{1/3}}$ (where C = Chezy coefficient and n = Manning coefficient)].

6.8 Conclusions

This chapter has presented the theory of shallow water equations and discussed its object-oriented implementation. The theory presented in this chapter provides extension to the hyperbolic model to solve shallow water problems. The benefits of developing extension models for the *Riemann2D* model are as follows:

- It can be seen that the knowledge required to provide extension as solve shallow water model is much less, as most of the modelling code is already present in the generic *Riemann2D* model.
- Similar to shallow water model extension other models for hyperbolic problems can be developed.

Other important developments in this chapter are as follows:

- eigen Structure for shallow water equations with Solute (see section 6.5.0.1);
- discussion on the problem of varying bathymetry (see section 6.7.1);
- introduction of reference pointers for the bed surface depth (see Figure 6.7). This method provides an efficient tool to introduce limiters to the bed surface. Which can lead to solve moving bed problems.

CHAPTER 7

Object-Oriented Design and Development

7.1 Introduction

A prerequisite to designing a model is a clear understanding of the fundamentals (theory) of the modelling system itself, which is already discussed in the previous chapters and the basic design of the *Riemann2D* is described in chapter 3. Therefore, this chapter focus only on the design of the *objects* of *Generic* and *ShallowWater* packages.

For better and clear representation of the model design standard Unified Modeling Language (UML) diagrams are thoroughly used in this chapter.
The objective of this chapter is to present:

- the UML diagram of the classes of *Riemann2D*.
- the development process of *Riemann2D*.

7.2 Object-Oriented Design Using UML

The Unified Modeling Language (UML[1]) is a standard language, which helps to specify, visualise, and document models of software systems, including their structure and design, in a way that meets all of its requirements. It simplifies the complex process

[1]See `http://www.omg.org/`

of model design, making a *blueprint* for construction. In other words, UML diagrams enable developers to communicate their ideas more quickly and more accurately than if only verbal descriptions or self drawn diagrams were used because the diagrams take a standard format. Thus, to explain the underlying architecture of object-oriented design of *Riemann2D*, UML diagrams are used in this chapter. The notation of the UML diagrams are shown in Figure 7.1.

The main objectives of using the UML diagrams in this thesis is to:

1. provide readers with a ready-to-use and expressive visual model of *Riemann2D* .
2. provide extensibility and specialisation mechanisms to extend the core concepts of *Riemann2D* .
3. be independent[1] of particular programming languages and development processes to develop another generic hyperbolic solver of similar nature.
4. support higher-level framework development concepts.

7.3 Riemann2D Design

The classes in the *Riemann2D* model are organised in packages based on their functionality. The superclass package (*riemann2d.generic*: see section 7.3.1) contains all the classes based on object-oriented problem structuring (section 3.4.1) to separate the common component (variables and methods) of hyperbolic models, based on the theory presented in chapter 4. The subclass package (*riemann2d.shallowwater*: see section 7.3.3) extends the superclass *i.e.*, the subclasses inherits the properties from the respective superclasses and provides the *variable* and *methods* that are necessary to solve a particular problem (the shallow water problem in this work). Because subclasses extend the subclasses they do not need to supply the variable or method that is inherited from its superclass. It is very noticeable in the *Riemann2D* that all the packages and class names refer to the functionality of its respective class or package (figure 7.2).

[1]The UML diagram in this chapter shows the components of Java, which can be neglected by readers who are interested in other programming language.

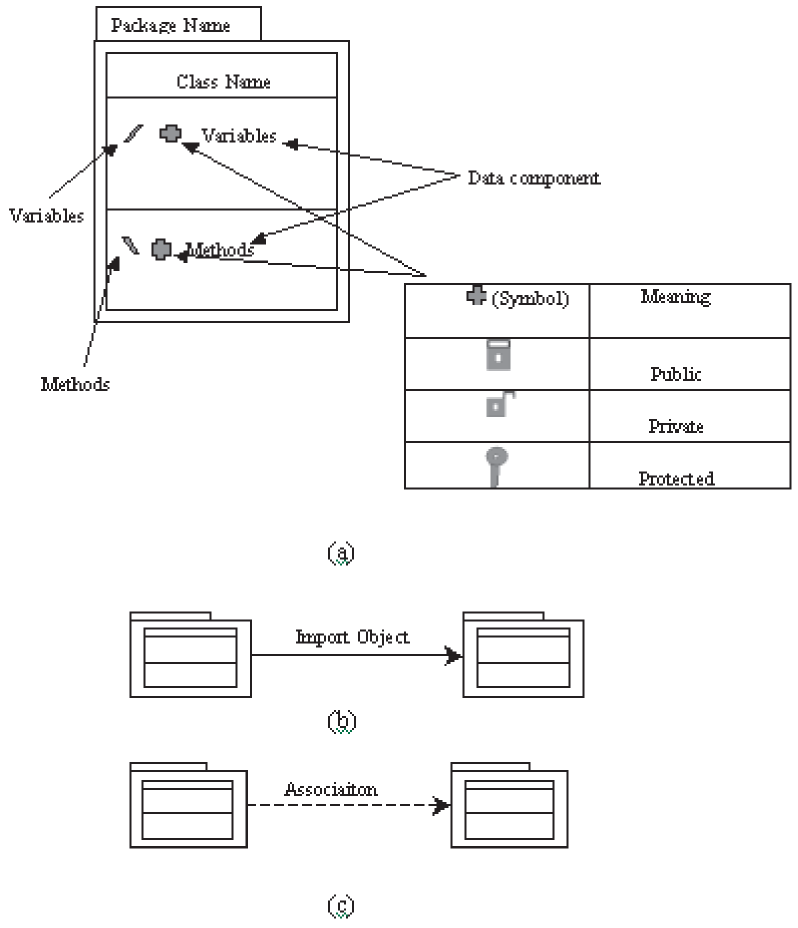


Figure 7.1: Notation used for the UML diagram. Note: The notations in this diagram are made using Microsoft Paint software, where the UML diagrams in this chapter are generated using JBuilder UML tool. Hence the shapes in this diagram are a little different than other UML diagrams.

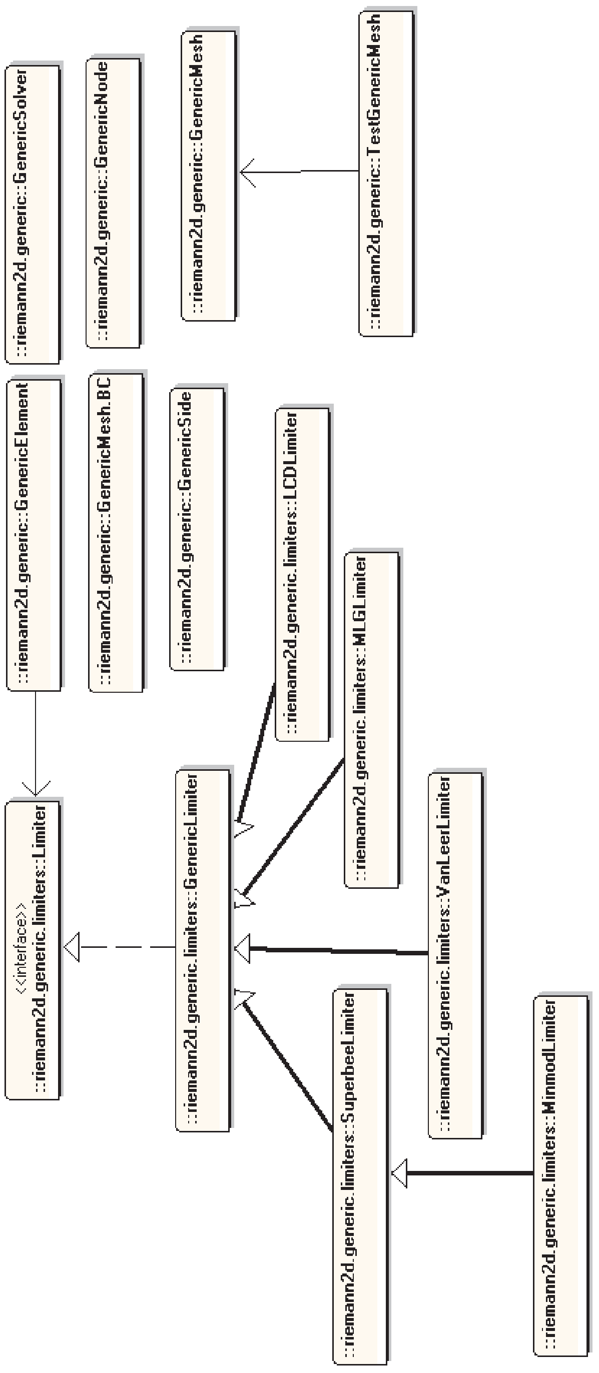

::riemann2d.generic::GenericSolver
::riemann2d.generic::GenericNode
::riemann2d.generic::GenericMesh
::riemann2d.generic::TestGenericMesh
::riemann2d.generic::GenericElement
::riemann2d.generic::GenericMesh.BC
::riemann2d.generic::GenericSide
::riemann2d.generic.limiters::LCDLimiter
::riemann2d.generic.limiters::MLGLimiter
<<interface>>
::riemann2d.generic.limiters::Limiter
::riemann2d.generic.limiters::GenericLimiter
::riemann2d.generic.limiters::VanLeerLimiter
::riemann2d.generic.limiters::SuperbeeLimiter
::riemann2d.generic.limiters::MinmodLimiter

Figure 7.2: UML diagram showing inheritance and associations of all the classes in the *Riemann2D* model. This gives a brief picture of the model. The detailed description and UML diagrams for each package is given in sections 7.3.1 - 7.29.

7.3.1 *riemann2d.generic* Package

The *riemann2d.generic* package contains the classes which represent the generic part of the model. The classes in this package are *GenericNode*, *GenericSide*, *GenericElement*, *GenericMesh* and *GenericSolver*. As the name suggests, these classes hold the generic properties (variable and methods) for node, side, element and solver, respectively. The naming convention for the variables and methods used in *Riemann2D* is self-descriptive.

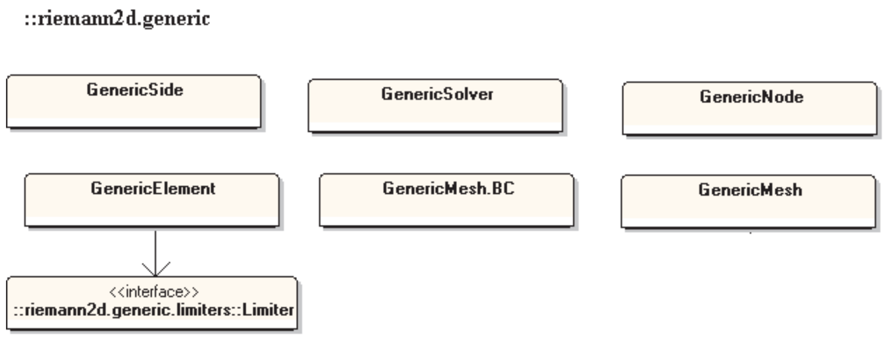


Figure 7.3: Inheritance and associations among the classes in the *riemann2d.generic* package. The detailed UML diagram for each of these classes can be found in Figure 7.5 - 7.9

A brief description of the classes and interface in the *riemann2d.generic* package are as follows:

GenericNode Class : The GenericNode class contains information on the node's label, it's (x,y)-coordinate location and a parameter array containing stored information read from the meshmaker input file associated with it. See Figure 7.5 for more details.

GenericSide Class : It contains the basic information related to a side of a triangle in a two-dimensional triangular mesh such as, the nodes making this element, label, length, slope, position within domain (*i.e.*, if it is on the boundary or not). The benefit of object-oriented design here is that for the information of its node, an object of node type can be created and its available properties (variable or methods) can be called. It also contains other properties that are generic and required by a side to solve any hyperbolic mesh. For example, it needs to know which solver (Roe or HLL etc.) is used, which elements are on its side, the

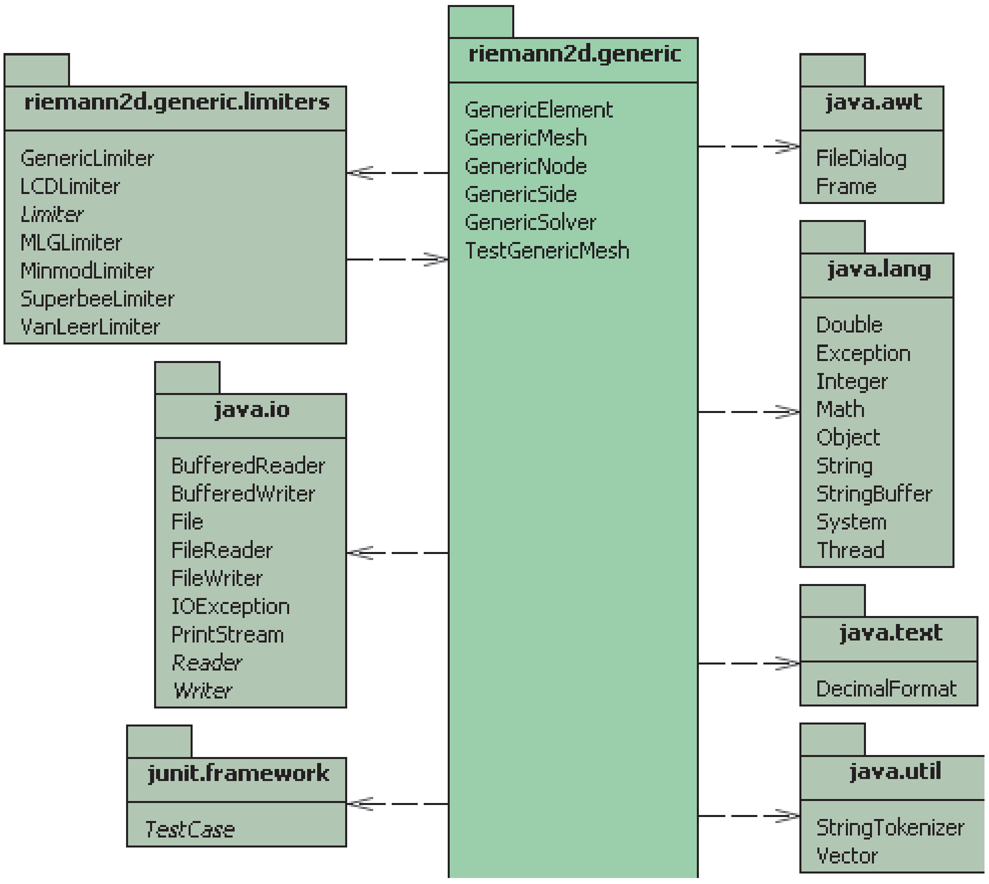


Figure 7.4: Diagram shows the classes contained within *riemann2d.generic* package and its relation with other packages. It also shows the relationship with the Java packages.

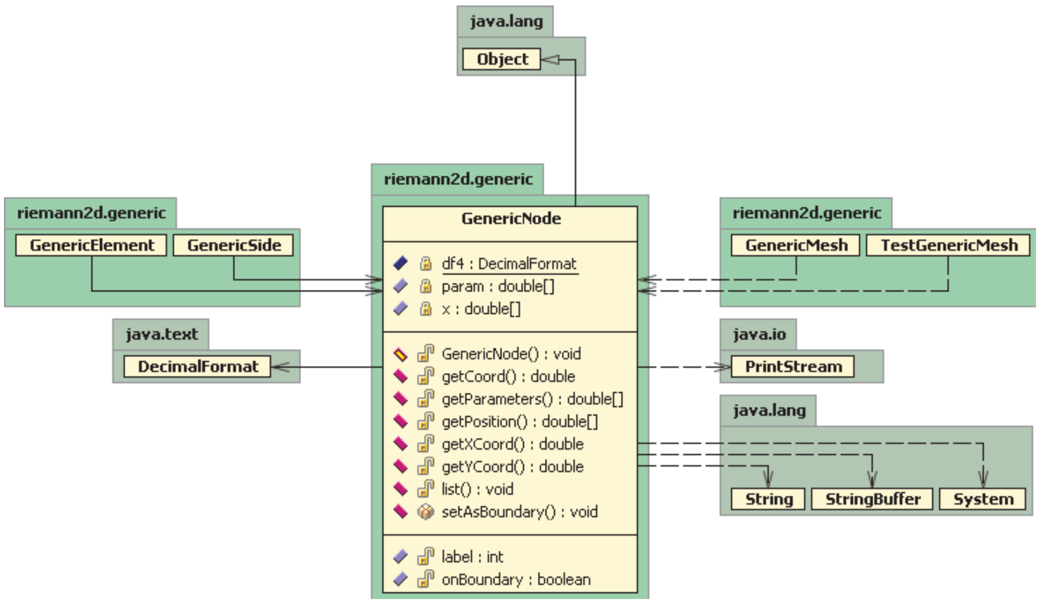


Figure 7.5: Class diagram for *GenericNode* class of the *riemann2d.generic* package. It shows the relationship of *GenericNode* class with other classes/packages of *Riemann2D* and packages of Java. This class is implemented by the *Node* class of a subclassed package to inherit all the generic properties of a node as classified in *GenericNode* Class.

difference in flux across this side, if this side is a boundary side then what type of boundary condition is applied. For more details on the variables and methods, see Figure 7.6.

GenericElement Class : It contains the information related to an element (triangular) in the 2-dimensional mesh such as the nodes and sides that make this element, label, area, center of the element, position within domain (*i.e.*, if it is on the boundary or within the domain). This element is also the control volume.

It also contains the properties (variables and methods) that are generic and required by the element to solve a hyperbolic problem. These are: the state variables of the elements, state incremental values after every computation, type of limiter used and method to calculate the values at the sides of the triangle, the method to write values for the output and the method for setting up the ghost element to supply the values at the boundary. For more details on the variables and methods, see Figure 7.7.

GenericMesh Class : This class works as an interface for the input file and the model. It contains the code which is used to call the input file, filter and sort its data and use it to make nodes, sides and elements. It also provides the basic information for each of these components to its respective classes such as the x- and y- coordinates for the node, nodes connecting the sides, nodes and elements making the elements and mesh properties for the elements. Also, it acts as the interface for the output data: this class calls different methods from other classes to compile the output data. For example, to print the output of the state variables it calls the methods for the element number (label), x- and y- coordinates of the center of the elements, the value of the state variables from the generic element of *riemann2d.generic* and the element of the *riemann2d.shallowwater* classes.

Solving an hyperbolic system: To solve an hyperbolic system of time-dependent equations on an unstructured triangular mesh, we need to subclass GenericMesh and provide proper implementations for the following four methods by overriding the same methods found in the superclass:

1. getNewNode
2. getNewElement

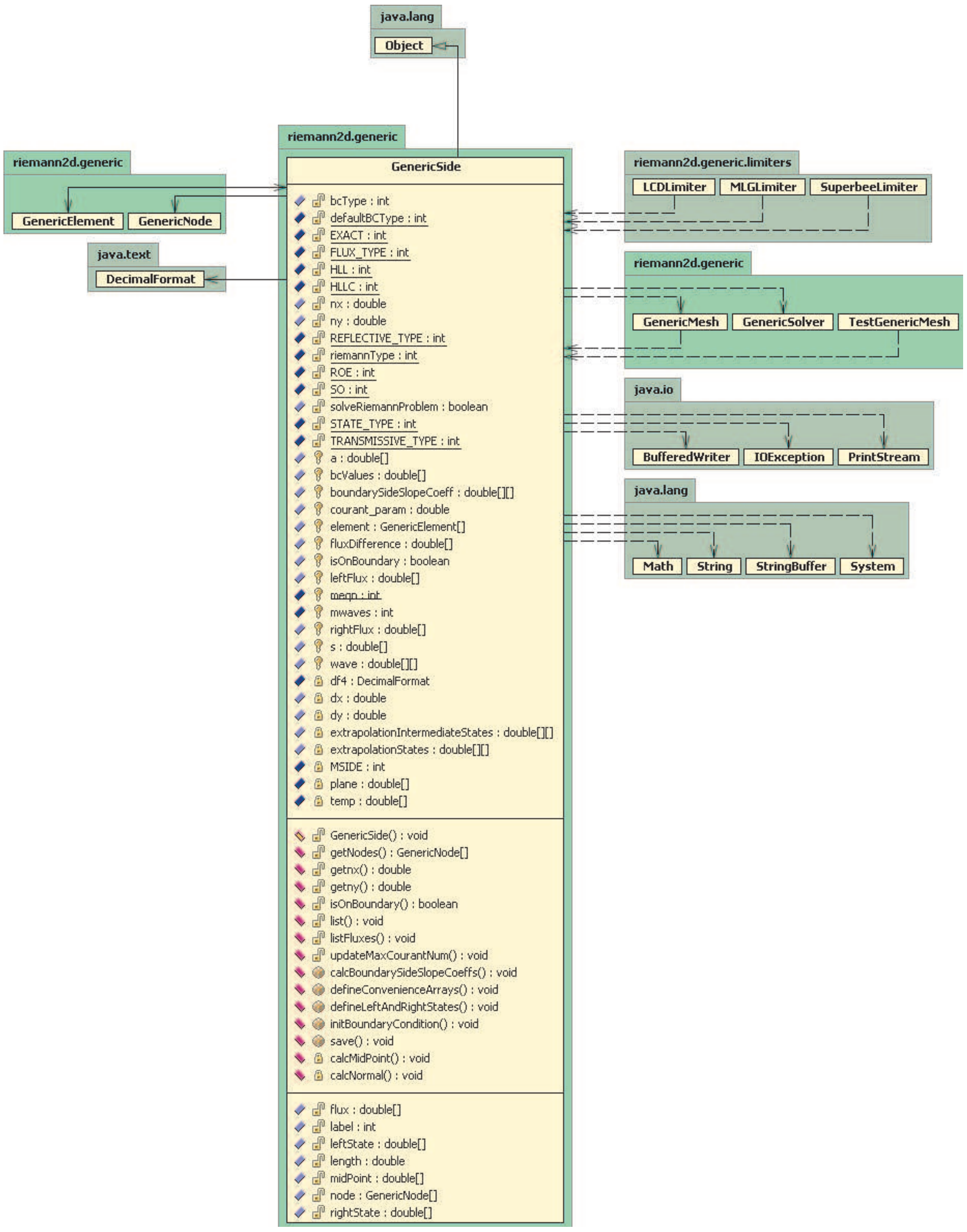


Figure 7.6: Class diagram for *GenericSide* class of the *riemann2d.generic* package. It shows the relationship of *GenericSide* class with other classes/packages of *Riemann2D* and packages of Java. This class is implemented by the *Side* class of a subclassed package to inherit all the generic properties of a side as classified in *GenericNode* class.

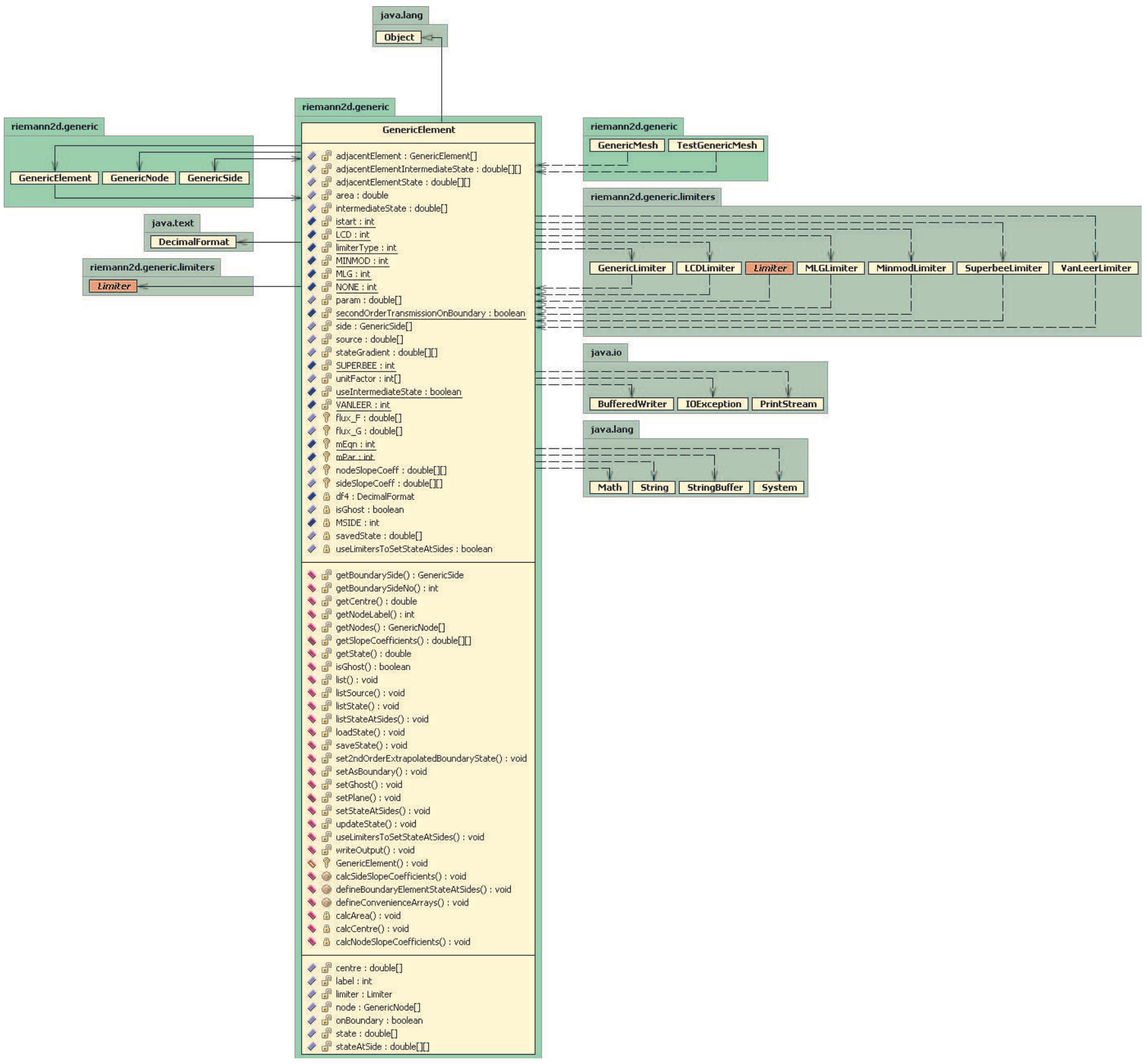


Figure 7.7: Class diagram for *GenericElement* class of the *riemann2d.generic* package. It shows the relationship of *GenericElement* class with other classes/packages of *Riemann2D* and the packages of Java. This class is implemented by the *Element* class of the subclassed package to inherit all the generic properties of an element as classified in *GenericElement* class.

3. getNewSide
4. execute

The first three methods supplies GenericMesh with nodes, elements and sides it needs to assemble the mesh. As a default, GenericMesh uses the GenericNode, GenericElement and GenericSide classes, but this provides just a skeleton without much functionality. Additional functionality is provided by subclassing the GenericNode, GenericElement and GenericSide classes and supplying these classes to GenericMesh via the getNewNode, getNewElement and getNewSide methods.

The final method facilitates the solution of the time-dependent hyperbolic system by providing a mechanism for time-stepping the solution. For more details on variables and methods, see Figure 7.7.

GenericSolver Class : It contains the generic code for solving any Riemann problem in the required sequence. In the outer output loop of *solve* method, there is an inner loop. An implementation of the abstract GenericMesh method execute() is required to timestep the solution in the inner loop. A printout at each timestep occurs over the outer loop. Output occurs via mesh.writeOutput() at the end of each loop. For more details on the variables and methods, see Figure 7.9.

7.3.2 *riemann2d.limiter* Package

The *riemann2d.limiter* package contains the classes that are used to limit the slope of the state variables. Although this represents the generic part of the code, it is kept separate in a package. The reason behind this is that all the limiters have a common way of performing their calculations and hence the codes also look alike. If all the codes of this package are kept in the generic package it will make it bulky and more complicated.

One of the purposes of using object-oriented technology is to simplify things. Object-oriented design is more of an art than a science. At one stage in the development it was realised that separating the limiters in different package would not only help to simplify the code structure, but it would also be useful for adding a number of limiters with little effort. This also helps in better understanding of the limiters.

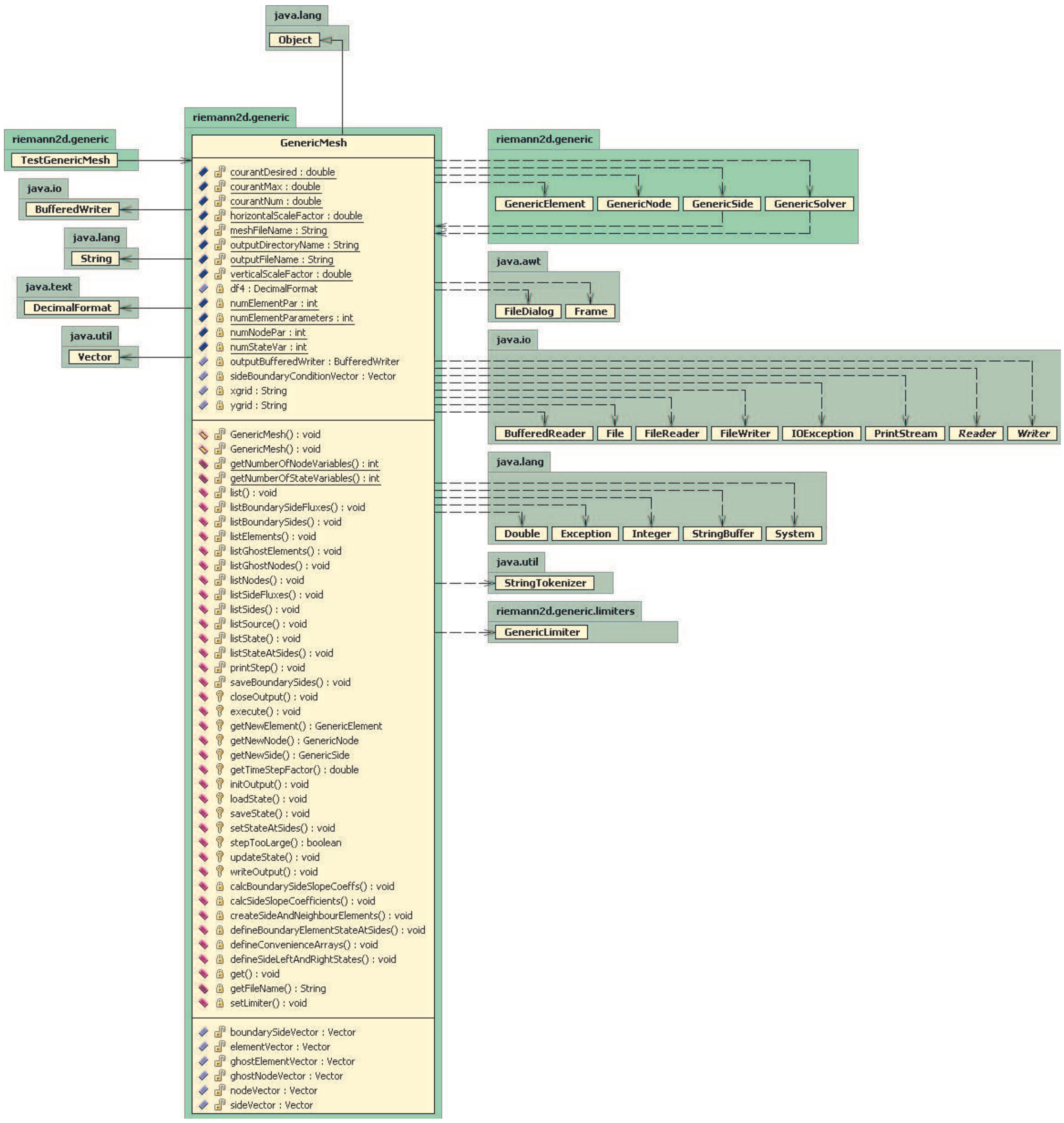


Figure 7.8: Class diagram for *GenericMesh* class of the *riemann2d.generic* package. It shows the relationship of *GenericMesh* class with other classes/packages of *Riemann2D* and the packages of Java. This class is implemented by the *Mesh* class of the subclassed package to inherit all the generic properties of a mesh as classified in *GenericMesh* class.

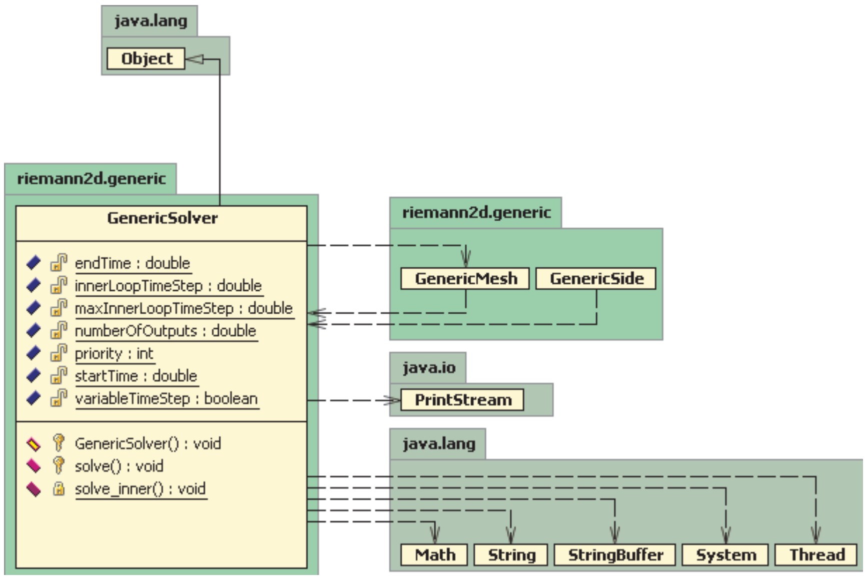


Figure 7.9: Class diagram for *GenericSolver* class of the *riemann2d.generic* package. It shows the relationship of *GenericSolver* class with other classes/packages of *Riemann2D* and the packages of Java. This class is implemented by the *Solver* class of a subclassed package to inherit all the generic properties of a solver as classified in *GenericSolver*Class.

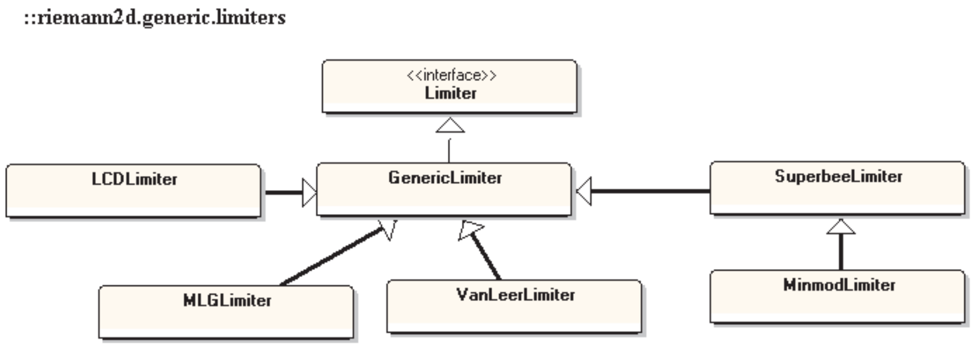


Figure 7.10: Inheritance and associations among the classes in *riemann2d.limiter* package. The detailed UML diagram for few of these classes can be found in Figures 7.13 - 7.17

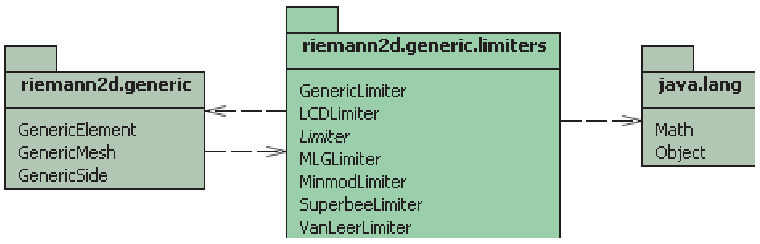


Figure 7.11: Diagram shows the classes contained within the *riemann2d.limiter* package and its relation with other packages. It also shows the relationship with the Java packages.

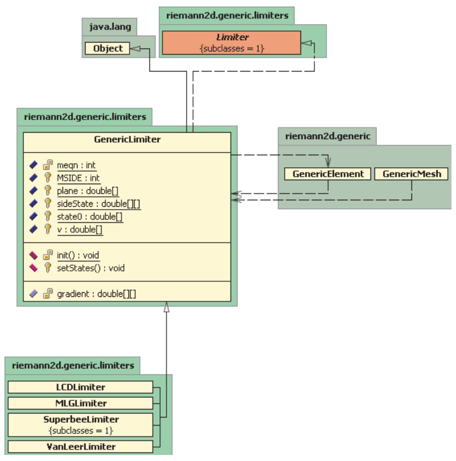


Figure 7.12: Class diagram for the *GenericLimiter* class of the *riemann2d.limiter* package. It shows the relationship of the *GenericLimiter* class with other classes/packages of *Riemann2D* and the Java packages. This class is the superclass for other classes of different limiters in the *riemann2d.limiter* package. It has common properties of any limiter class.

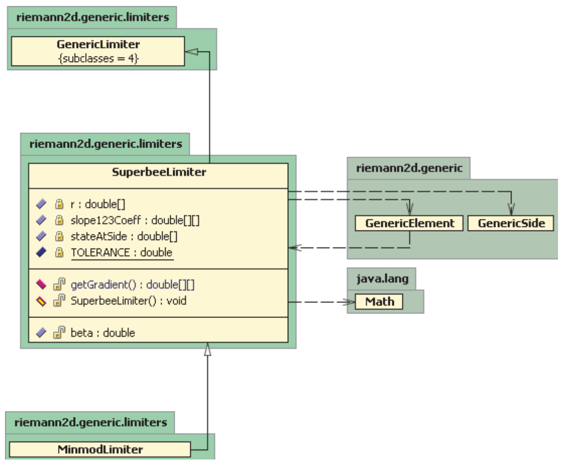


Figure 7.13: Class diagram for the *SuperbeeLimiter* class of the *riemann2d.limiter* package. It shows the relationship of the *SuperbeeLimiter* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericLimiter* (see Figure 7.12) class as a superclass and hence inherits its properties.

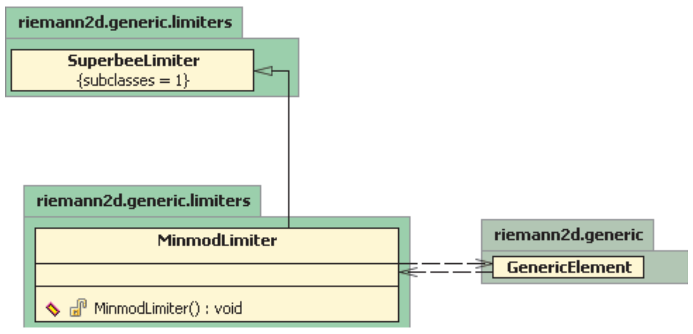


Figure 7.14: Class diagram for the *MinmodLimiter* class of the *riemann2d.limiter* package. It shows the relationship of the *MinmodLimiter* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *SuperbeeLimiter* (see Figure 7.13) class as a superclass and hence inherits its properties.

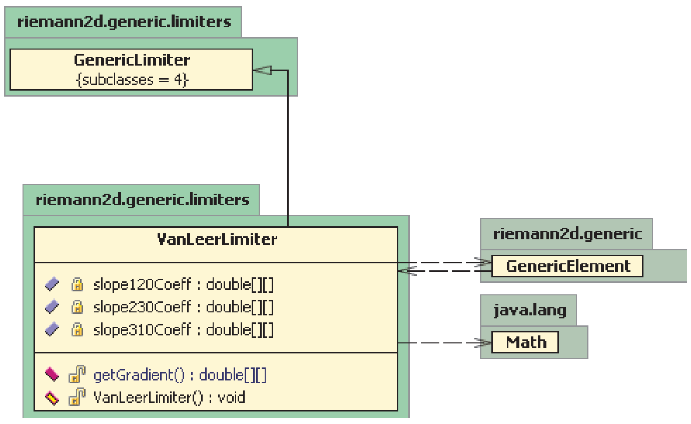

riemann2d.generic.limiters
GenericLimiter
{subclasses = 4}
riemann2d.generic.limiters
VanLeerLimiter
slope120Coeff : double[][]
slope230Coeff : double[][]
slope310Coeff : double[][]
getGradient() : double[][]
VanLeerLimiter() : void
riemann2d.generic
GenericElement
java.lang
Math

Figure 7.15: Class diagram for the *VanleerLimiter* class of the *riemann2d.limiter* package. It shows the relationship of the *VanleerLimiter* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericLimiter* (see Figure 7.12) class as a superclass and hence inherits its properties.

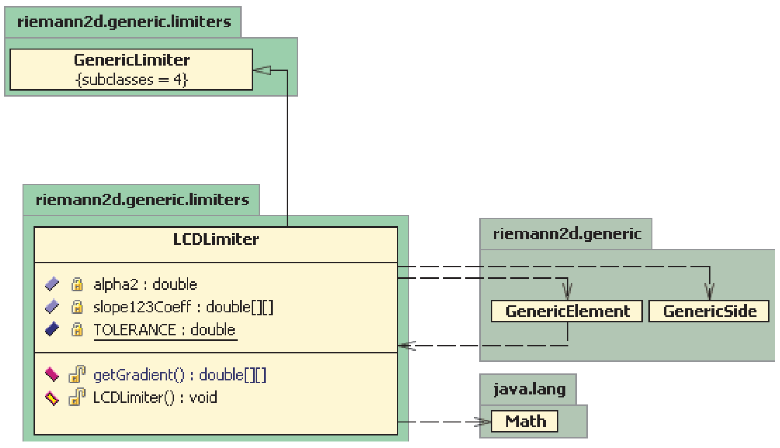


Figure 7.16: Class diagram for the *LCDLimiter* class of the *riemann2d.limiter* package. It shows the relationship of the *LCDLimiter* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericLimiter* (see Figure 7.12) class as a superclass and hence inherits its properties.

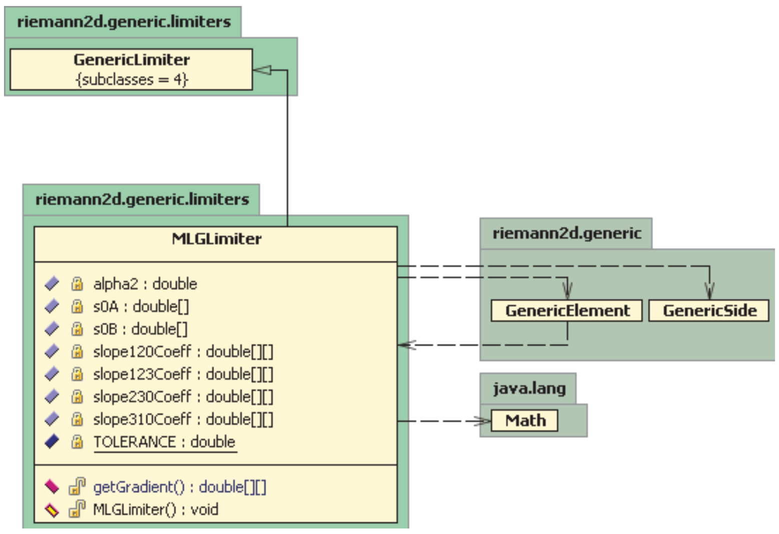

riemann2d.generic.limiters
GenericLimiter
{subclasses = 4}
riemann2d.generic.limiters
MLGLimiter
alpha2 : double
s0A : double[]
s0B : double[]
slope120Coeff : double[][]
slope123Coeff : double[][]
slope230Coeff : double[][]
slope310Coeff : double[][]
TOLERANCE : double
getGradient() : double[][]
MLGLimiter() : void
riemann2d.generic
GenericElement
GenericSide
java.lang
Math

Figure 7.17: Class diagram for the *MLGLimiter* class of the *riemann2d.limiter* package. It shows the relationship of the *MLGLimiter* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericLimiter* (see Figure 7.12) class as a superclass and hence inherits its properties.

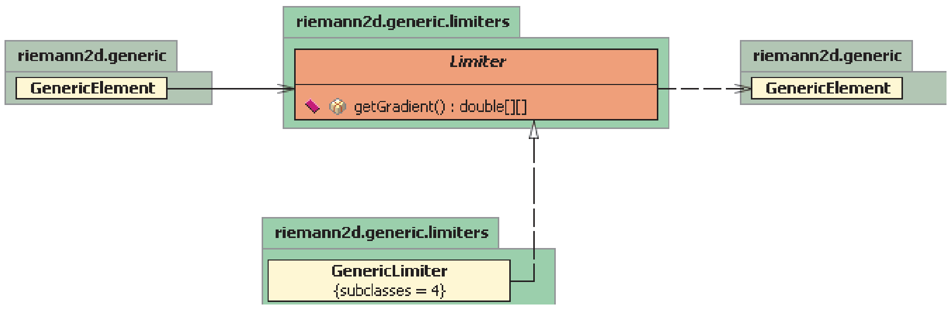


Figure 7.18: Diagram for the *Limiter* interface of the *riemann2d.limiter* package. An interface is a contract in the form of a collection of methods and constant declarations. When a class implements the *Limiter* interface, it has to implement all of the methods declared in the *Limiter* interface.

The classes in the *riemann2d.limiter* package are *GenericLimiter*, *Superbee*, *Minmod*, *Vanleer*, *LCDLimiter* and *MLGLimiter*. As the name suggests, these classes hold the variables and methods used by these limiters. The *GenericLimiter* is the superclass which is extended by other classes. The idea behind having this superclass is to classify the common properties (variables and methods) used by all these limiters. The mathematical expressions of the limiters can be found in Chapter 4 (section 5.9.2). Also in this package there is an interface called *Limiter*, which is implemented by the GenericElement class of the *riemann2d.generic* package. As the subclasses in this package have same the functionality (but a different way of calculation) it can be seen in Figures 7.13, 7.15, 7.16 and 7.17 that the subclasses all have similar UML diagram. The *MinmodLimiter* class is subclass of the *SuperbeeLimiter* class because the only difference between Superbee and Minmod Limiter is a constant as described in Chapter 4 (section 5.9.2). Here also the benefit of object-oriented design can be seen; the *MinmodLimiter* class is a subclass of the *SuperbeeLimiter* class which is a subclass of the *GenericLimiter* class, therefore the *MinmodLimiter* class inherits the properties from both the *SuperbeeLimiter* class and *GenericLimiter*.

Most limiters subclass the *GenericLimiter* class. This class essentially implements the no limiter case, so it is used when limiters are not applied. Hence, the states will be made constant across the element.

7.3.3 *riemann2d.shallowwater* Package

riemann2d.shallowwater package contains the classes which are used to solve the shallow water problem. The classes in this package are *Node*, *Side*, *Element*, *Mesh* and *Solver*. These classes are the subclass of the *GenericNode*, *GenericSide*, *GenericElement*, *GenericMesh* and *GenericSolver* of *riemann2d.generic*. This means that the subclasses of this package inherit the properties (variables and methods) from their superclass. This package also provides an insight for other developers to understand the subclassing process for a particular hyperbolic problem. The codes in this package are such that it is only useful to solve shallow water problems and the majority of hyperbolic problems don't need this code.
A brief description of the classes in *riemann2d.shallowwater* package are as follows:

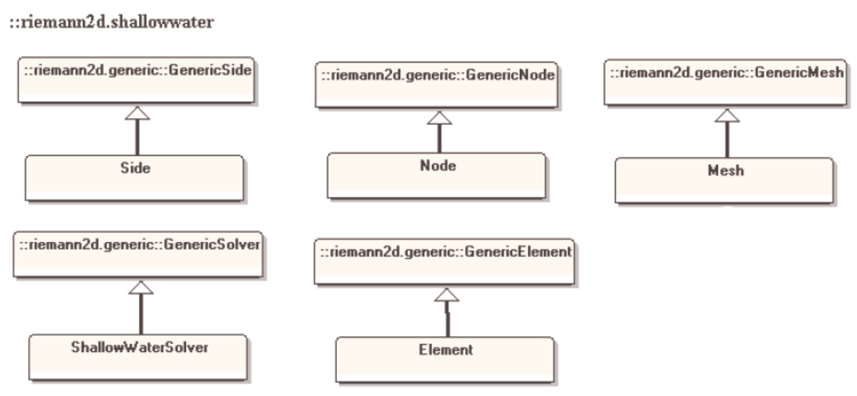

::riemann2d.shallowwater
::riemann2d.generic::GenericSide
Side
::riemann2d.generic::GenericNode
Node
::riemann2d.generic::GenericMesh
Mesh
::riemann2d.generic::GenericSolver
ShallowWaterSolver
::riemann2d.generic::GenericElement
Element

Figure 7.19: Inheritance and associations among the classes in *riemann2d.shallowwater* package. The detailed UML diagram for few of these classes can be found in Figures 7.5 - 7.25

Node Class : It only stores parameters supplied from the mesh file. For more details see Figure 7.21.

Side Class : It contains the code for solving Riemann problem across the side. There are different Riemann solvers added in this class. A few of them have been added in this thesis (see section 6.6), whereas some others are still under development for the future use. The main reason for having a Riemann solver in this class is because of the state variables which are different for different hyperbolic problems. It also calculates the maximum Courant number based on the values across the sides. For more details see Figure 7.22.

Element Class : It contains the codes for the calculations such as bed & surface slopes, fluxes and energy. For more details please see figure 7.23.

Mesh Class : It contains the code for executing the process of solving the shallow water problem. It overrides the methods of the *GenericMesh* class as discussed earlier (see section 7.3.1).

ShallowWaterSolver Class : It contains the code for solving the shallow water probelms, where it extends and only calls the solve method of the *GenericSolver* from *riemann2d.generic* package. For more details see Figure 7.25.

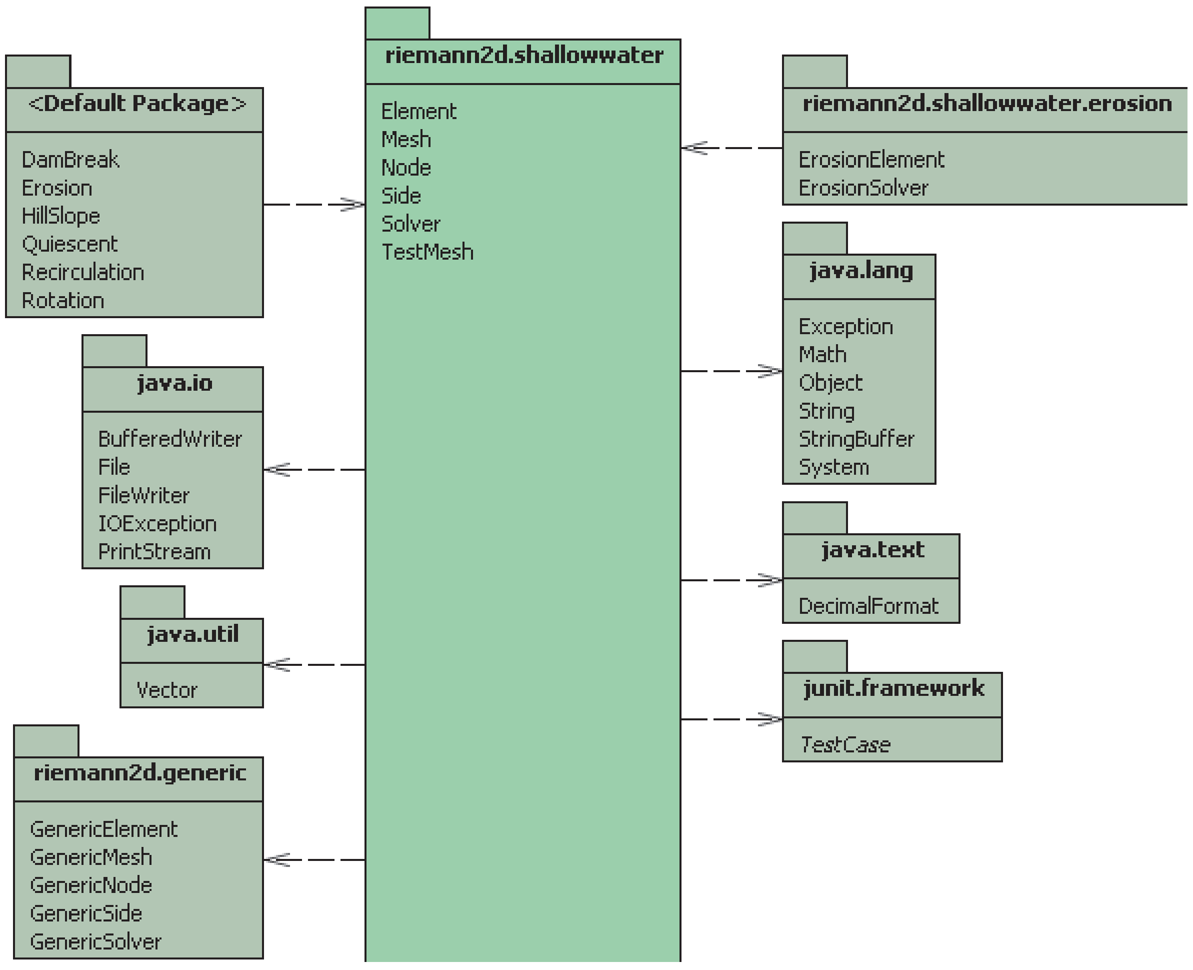


Figure 7.20: Diagram shows the classes contained within the *riemann2d.shallowwater* package and its relation with other packages. It also shows the relationship with the Java packages.

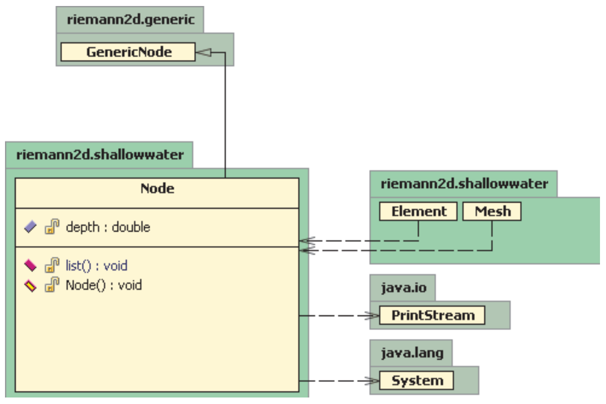


Figure 7.21: Class diagram for the *Node* class of the *riemann2d.shallowwater* package. It shows the relationship of the *Node* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericNode* (see Figure 7.5) class as a superclass and hence inherits its properties.

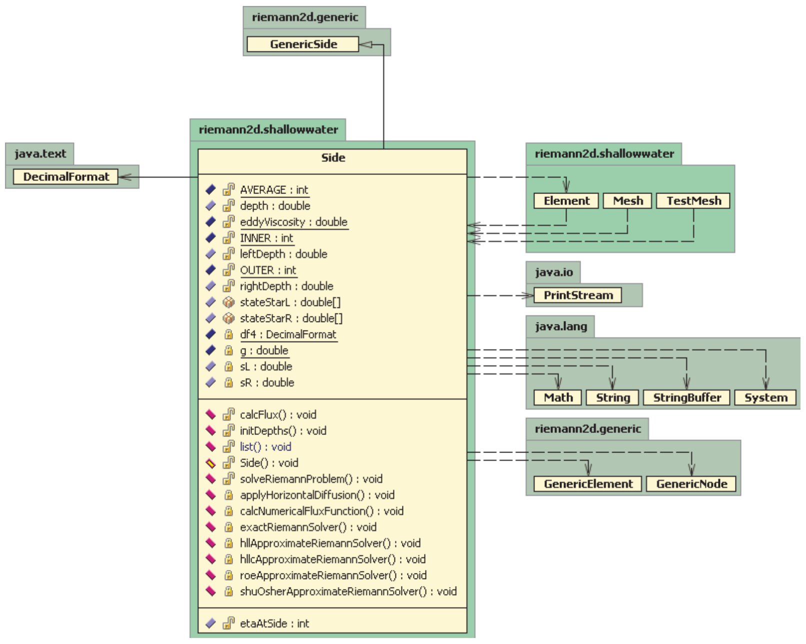


Figure 7.22: Class diagram for the *Side* class of the *riemann2d.shallowwater* package. It shows the relationship of the *Side* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericSide* (see Figure 7.6) class as a superclass and hence inherits its properties.

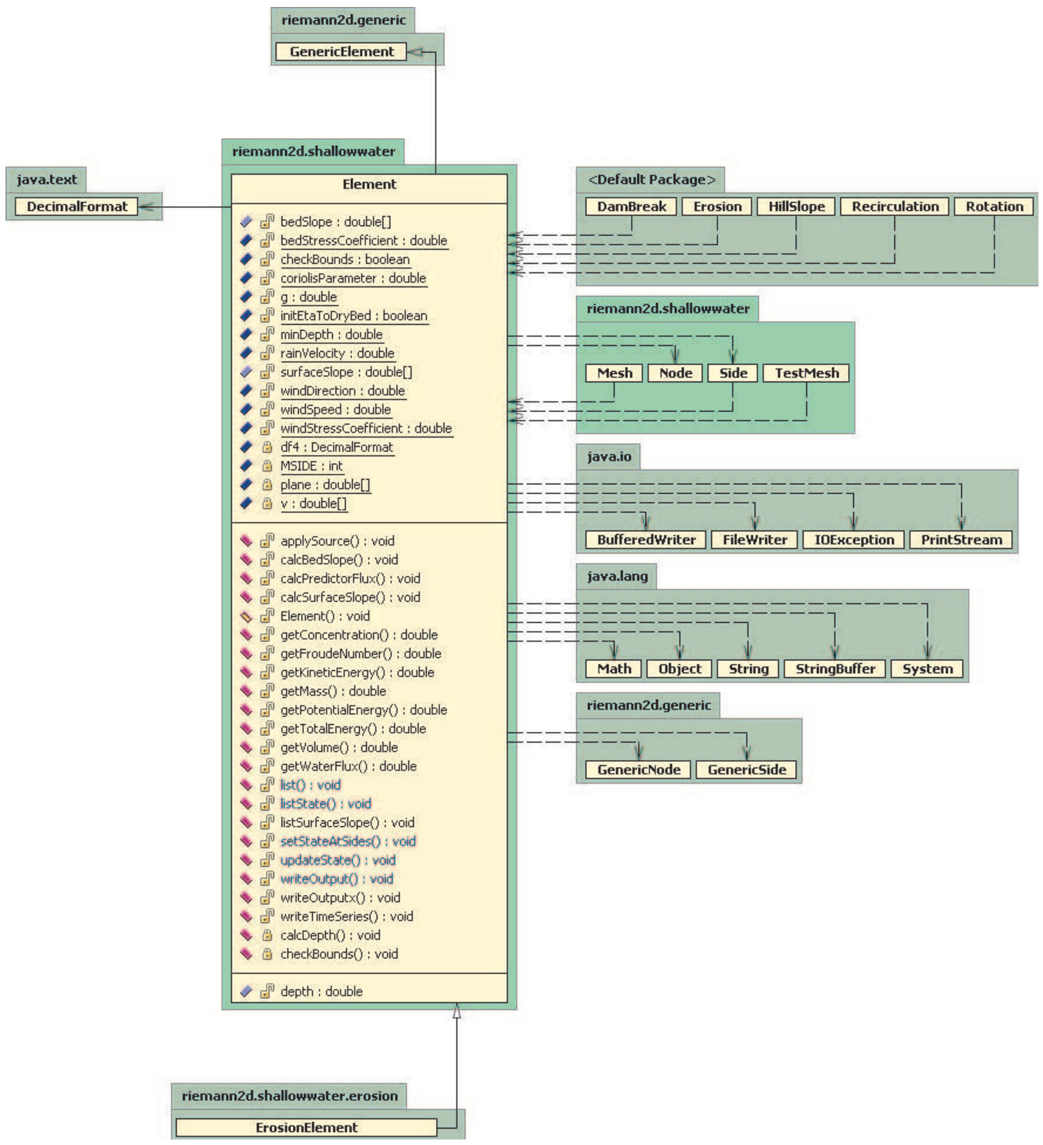


Figure 7.23: Class diagram for the *Element* class of the *riemann2d.shallowwater* package. It shows the relationship of the *Element* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericElement* (see Figure 7.7) class as a superclass and hence inherits its properties.

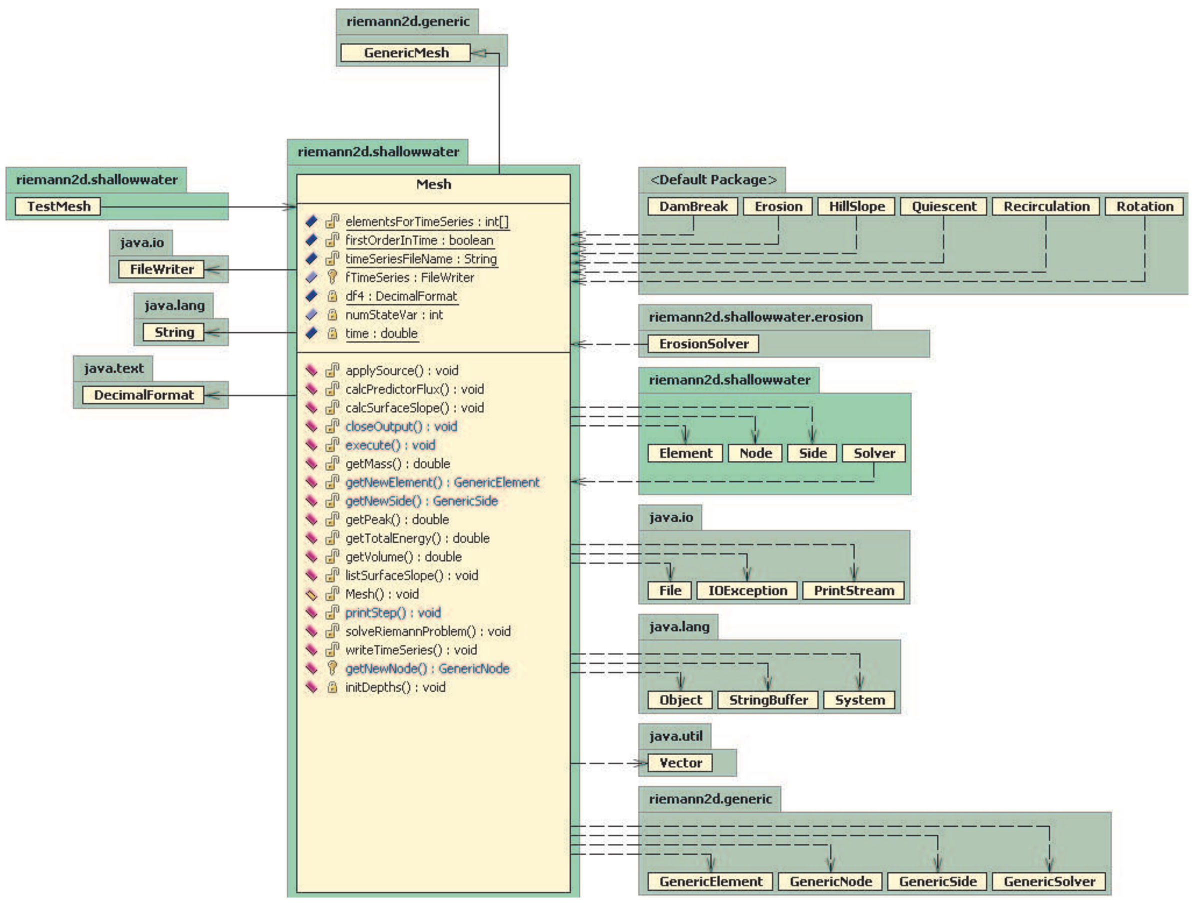


Figure 7.24: Class diagram for the *Mesh* class of the *riemann2d.shallowwater* package. It shows the relationship of the *Mesh* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericMesh* (see Figure 7.8) class as a superclass and hence inherits its properties.

configuration Classes These classes are used to provide the configuration for solving different type of problems. The benefit of this is that it will avoid any manual changes to the initialised values in the main classes by simply overriding them.

7.3.4 *riemann2d.shallowwater.erosion* Package

Erosion model is not a part of this work, however, it is presented here to show the extensibility of the model and explain the benefit of object-oriented design. As discussed earlier, different hyperbolic systems can subclass the generic superclass for solving a variety of hyperbolic systems. Similarly, these system can be further extended by subclassing them. This is done similar to the case where shallow water model extends generic package. The erosion model here only subclass the *Element* and provide further parameters (variables and methods) required to solve the problem. This reduces the effort and make it much simpler to solve different problem within same system. For more details on the design see Figures 7.28, 7.29 and 7.27.

7.4 *Riemann2D* Development

This section aims to give a brief introduction to the development of *Riemann2D* . This section is intended to provide information for those readers who would carry forward the development of *Riemann2D* or who wish to develop similar model.

7.4.1 *Riemann2D* Development Process

A *software development process* is a structure imposed on the development of a software product [`http://en.wikipedia.org/`]. Synonyms include *software life cycle* and *software process*. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

The development process of *Riemann2D* followed the *Iterative and Incremental development model*. The basic idea behind iterative enhancement was to develop *Riemann2D* incrementally, to take advantage of what was learned during the development of earlier, incremental, deliverable versions of the system. Learning came from both the development and use of the system. Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the

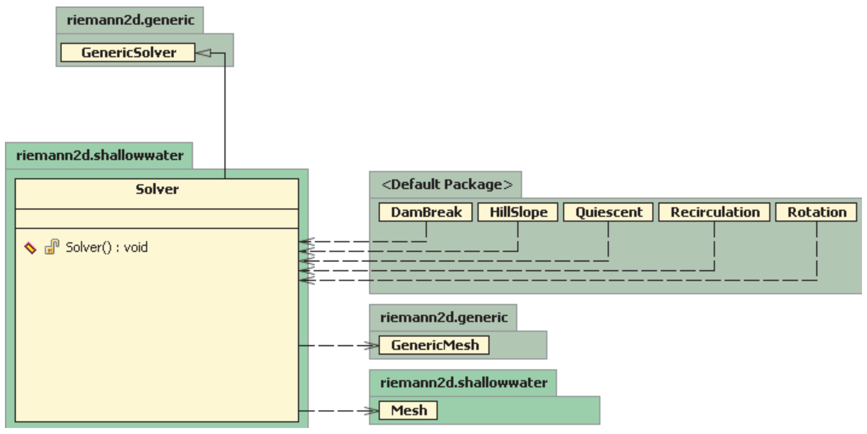


Figure 7.25: Class Diagram for the *ShallowWaterSolver* class of the *riemann2d.shallowwater* package. It shows the relationship of the *Element* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericSolver* (see Figure 7.9) class as a superclass and hence inherits its properties.

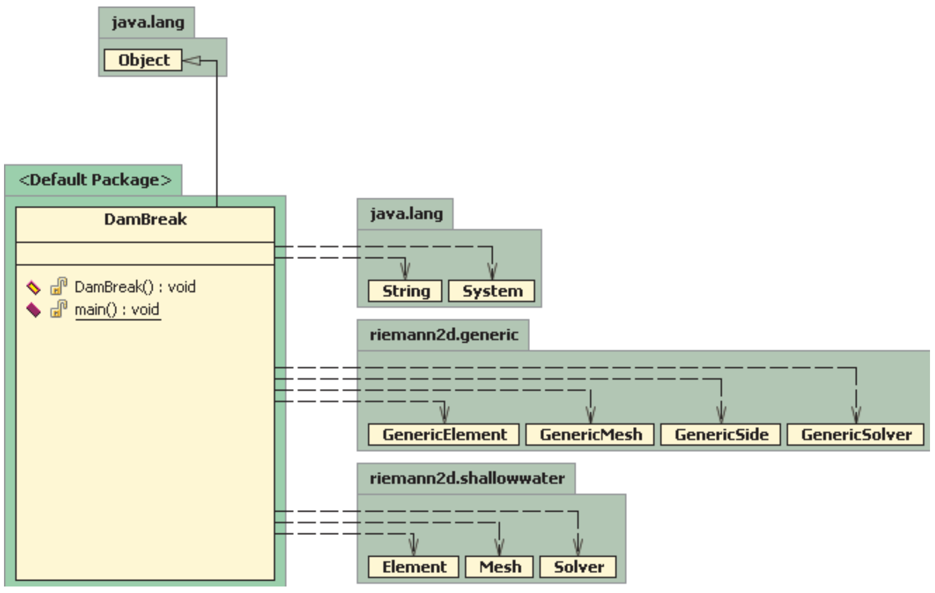


Figure 7.26: Class diagram for the *DamBreakTest* class of the *riemann2d.shallowwater* package. It shows the relationship of the *DamBreakTest* class with other classes/packages of *Riemann2D* and the Java packages

.

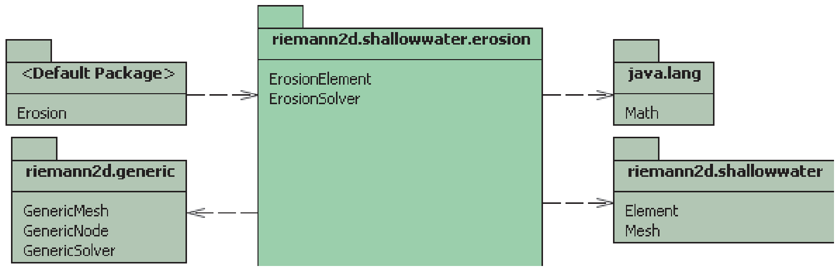


Figure 7.27: Diagram shows the classes contained within the *riemann2d.shallowwater.erosion* package and its relation with other packages. It also shows the relationship with the Java packages.

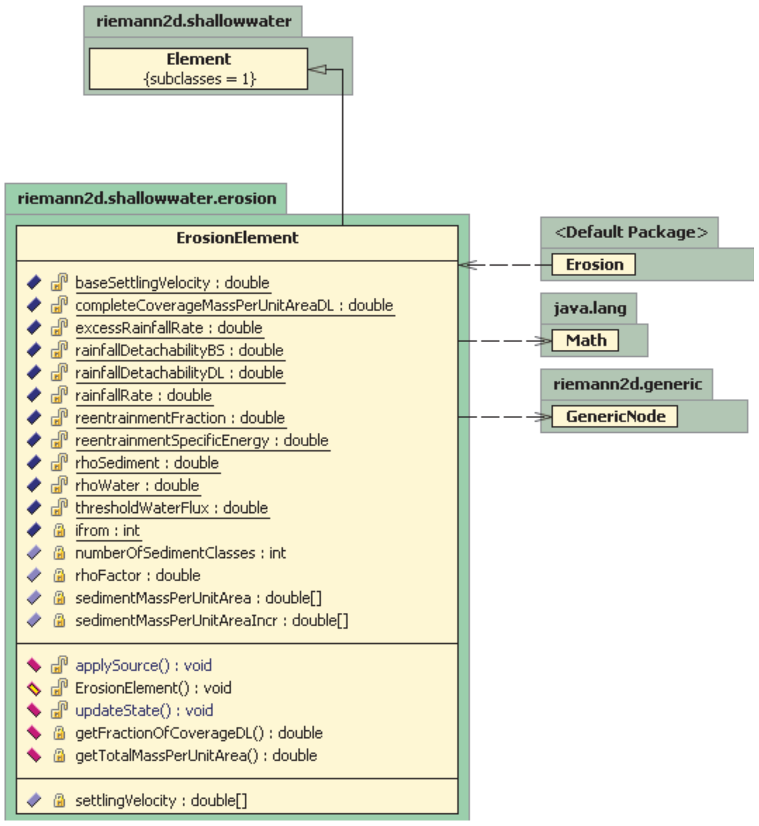

riemann2d.shallowwater
Element
{subclasses = 1}
riemann2d.shallowwater.erosion
ErosionElement
baseSettlingVelocity : double
completeCoverageMassPerUnitAreaDL : double
excessRainfallRate : double
rainfallDetachabilityBS : double
rainfallDetachabilityDL : double
rainfallRate : double
reentrainmentFraction : double
reentrainmentSpecificEnergy : double
rhoSediment : double
rhoWater : double
thresholdWaterFlux : double
ifrom : int
numberOfSedimentClasses : int
rhoFactor : double
sedimentMassPerUnitArea : double[]
sedimentMassPerUnitAreaIncr : double[]
applySource() : void
ErosionElement() : void
updateState() : void
getFractionOfCoverageDL() : double
getTotalMassPerUnitArea() : double
settlingVelocity : double[]
<Default Package>
Erosion
java.lang
Math
riemann2d.generic
GenericNode

Figure 7.28: Class diagram for the *ErosionElement* class of the *riemann2d.shallowwater.erosion* package. It shows the relationship of the *ErosionElement* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *Element* class as a superclass and hence inherits its properties.

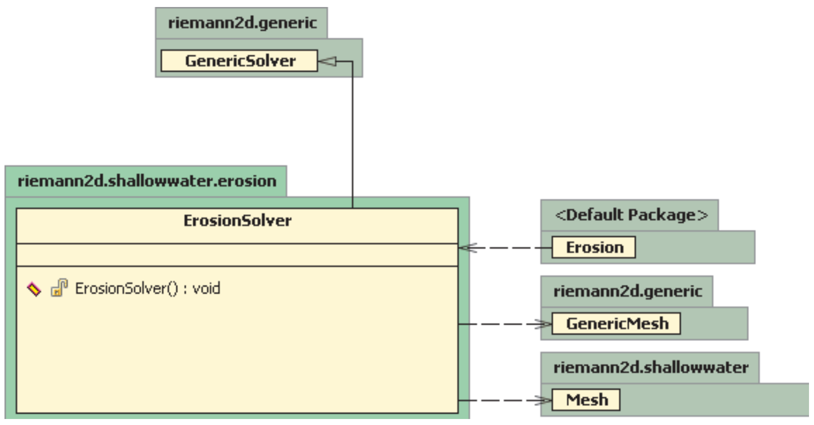


Figure 7.29: Class diagram for *EorsionSolver* class of the *riemann2d.shallowwater.erosion* package. It shows the relationship of *ErosionSolver* class with other classes/packages of *Riemann2D* and the Java packages. This class extends the *GenericSolver* class as a superclass and hence inherits its properties.

evolving sequence of versions until the full system was implemented. At each iteration, design modifications were made and new functional capabilities were added.

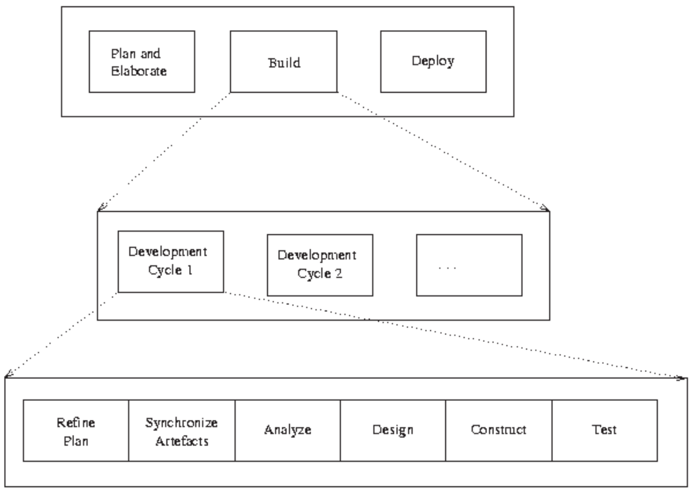


Figure 7.30: Iterative Development Cycles used for *Riemann2D* development.

Figure 7.30 shows the development cycle for each iterative step. Many online resources and books can be found on the software development process and different models. A few of the references on *iterative development* are Larman (2004), Gamma and Larman (2005), Kruchten (2004) and Bittner and Spence (2006).

7.4.2 Java IDE - Borland JBuilder

Borland JBuilder[1] is one of today's leading Java IDE[2] (Integrated Development Environment). The main features of JBuilder that helped in the development of *Riemann2D* and which also provides a huge opportunity for its future growth are as follows:

1. JBuilder provided a variety of tools for fast application development: wizards for automatically generating code, code audits for finding code problems early and coding shortcuts and templates for creating and correcting code.

2. JBuilder uses one window to perform most of the development functions: editing, visual designing, navigating, browsing, compiling, debugging, and other operations. This helped in easy handling of the process in the code development and less confusion.

3. JBuilder provided local version control and many ways to compare files, view differences between different files and between different versions of the same file, and to manage, merge, and revert file differences.

4. Additional capabilities, such as refactoring, UML code visualization, and integration with application lifecycle management tools, allowed to effectively work from the beginning of the project to its completion.

7.4.3 CVS & *Riemann2D* Development

In the early days of the *Riemann2D* development, a CVS (Concurrent Versions System) repository was set up on a departmental web server, called Civil Engineering Java Repository (see Figure 7.31). The code was committed to this repository when significant changes were made to the code). This ensured that the latest version of the code was always placed on the CVS server and there was no need to make backups on the local disk. This saved lot of development time and was a more organised approach toward the development. CVS was extremely helpful during code reviews with the

[1]A range of literature is available on the official site of Borland [`http://info.borland.com/techpubs/jbuilder/`].

[2]Java's other well known IDEs are from IBM (Eclipse), JetBrains (IntelliJ IDEA) and NetBeans.

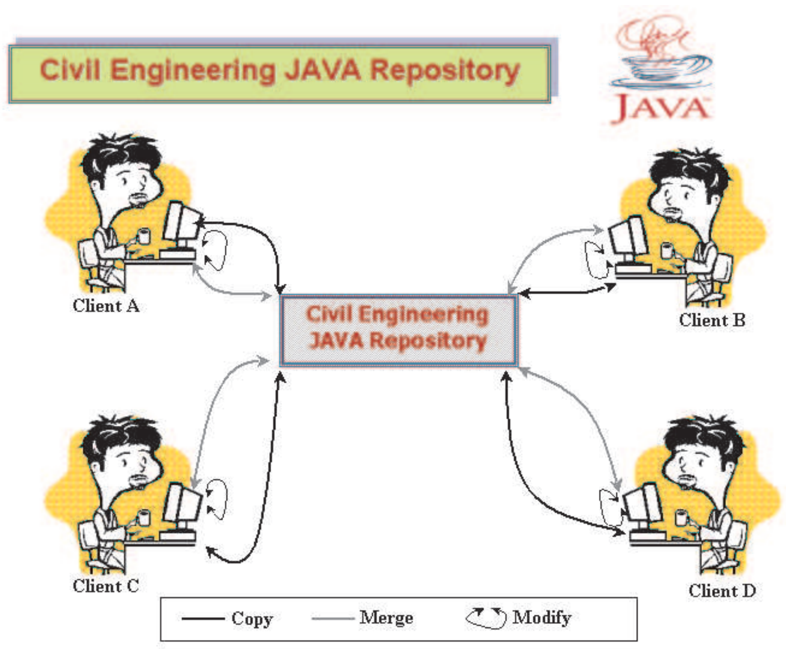

Civil Engineering JAVA Repository
JAVA
Client A
Client B
Civil Engineering JAVA Repository
Client C
Client D
Copy
Merge
Modify

Figure 7.31: CVS & *Riemann2D* Development: The above figure shows the relation and interaction between various people using the repository.

supervisor. It ensured that the latest copy of the code was reviewed and then where possible minor changes were made immediately and changes committed to the repository. In many instances, the code was reviewed by checking out the code from the repository, making changes if necessary and immediately updating the repository for further development. The great advantage in doing this was all these actions were performed remotely, through both internet and intranet. This ascertained timely reviews, corrections and thus saving total development time.

7.5 Conclusions

This chapter has shown that the object-oriented design is more than developing programs in modern languages, it is a new way of thinking about designing and realizing numerical modelling for engineering and scientific applications. The advantages that this design achieves by using object-oriented approach to model hyperbolic systems are as follows:

1. *Reusability*: A new class inherits the capabilities of the class from which it is derived, but new features can be added. This increases model flexibility because reusability makes it simpler and easier to modify and extend the functionality and capabilities of the existing code. Object-oriented design (OOD) produces software modules that can be plugged into one another, which allows creation of new programs.

2. *Faster Development*:Reusing code allows lower lifecycle maintenance costs, higher-quality design implementations and stronger development, which ultimately results in faster development.

3. *Increased Quality*: Increases in quality are largely a by-product of this program reuse. If 90% of a new application consists of proven, existing components, then only the remaining 10% of the code has to be tested from scratch. That observation implies an order-of-magnitude reduction in defects.

4. *Modular Architecture*: Object-oriented systems have a natural structure for modular design: objects, subsystems, framework, and so on. Thus, OOD systems are easier to modify. OOD systems can be altered in fundamental ways without ever breaking up since changes are neatly encapsulated.

5. *Better Mapping to the Problem Domain*: This is the biggest advantage of OOD, particularly when the project maps to the real world. Whether objects represent shallow water problem, Euler problem or any other real system, they can provide a clean, self-contained implication which fits naturally into human thought processes.

CHAPTER 8

Riemann2D Applied to Shallow Water Equations

8.1 Introduction

In this work *Riemann2D* is extended for solving shallow water problem and presented in this chapter.

The objective of this chapter is to present:

- Test cases for two-dimensional shallow water problem;
- Configuration and discussion for each test case;
- Comparison of the *Riemann2D* results with analytical, numerical and experimental results.

8.2 Model Verification and Validation

Models are mathematical representations of mechanisms that govern natural phenomena. Mathematical modelling has become an indispensable tool *via* computerised decision support systems for policy makers and researchers to provide ways to express the scientific knowledge, to lead to new discoveries, and/or to challenge old dogmas

[Tedeschi (2006)]. For these models to be accepted, model validation and verification have evolved as essential parts of the model development process. Where,

- **Validation** is the process of ensuring that a computer program will perform in the manner intended by its designers and implementors.
- **Verification**, on the other hand, is the process of assessing the goodness of fit to the characteristics of the models empirical referents. It is utilised in the comparison of the conceptual model to the computer representation that implements that conception.

Model validation and verification can be undertaken in many different pretexts; however, in this work it embraces the methods of assessing the following criteria: numerical accuracy, capability, reproducibility and adaptability.

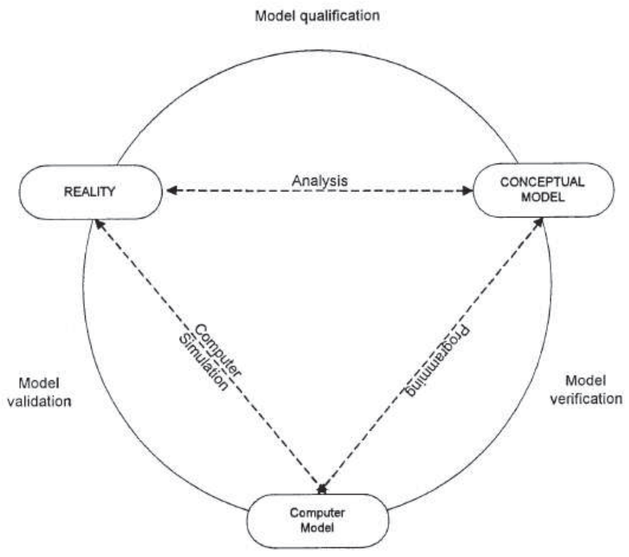


Figure 8.1: The validation of simulation models: The above figure shows the relationship between modelling and reality [Schleisinger (1979)].

Figure 8.1 shows a framework for validating simulation models proposed by Schleisinger (1979). Analysis generates a conceptual mathematical model. Model qualification determines the adequacy of this model. The next step produces a computer program. Model verification confirms that the computerised model represents the conceptual model within specified limits of accuracy using carefully chosen test cases. The final stage, validation, tests the input-output transformation of the simulation by analysis. The strength of this framework is that it combines three different mechanisms for establishing confidence in a simulation.

For this work, theory given in the previous chapters provides the conceptual background of *Riemann2D* . Whereas, validation and verification of *Riemann2D* presented in this chapter satisfies the above framework and outlines its strength.

A total of eight test specifications are prepared[1] and presented as part of this chapter. These test specifications are:

1. One-dimensional dam break problem;
2. Two-dimensional circular dam break;
3. Two-dimensional partial dam break;
4. Rotation test for shallow water equations;
5. Dam break in a converging-diverging channel;
6. Dam break in a channel with 90° bend;
7. Two-dimensional dam break experiment of Fraccarollo and Toro;
8. Oblique hydraulic jump; and

Above tests are aimed at assessing *Riemann2D* against a wide range of criteria, which are:

1. **Numerical accuracy**: The numerical accuracy can be assessed if an analytical solution exists for the physical situation/configuration that is being modelled. Dam break modelling is one of the vital tests that is conducted for model validation. Unfortunately, analytical solutions does not exists for two-dimensional dam break cases. However, the present model is assessed against the one-dimensional exact Riemann solver (Test case 1: section 8.3 against CLAWPACK, LeVeque (2002)), which is widely tested and accepted against the analytical solutions. One advantage of assessing *Riemann2D* results against the exact solver is that, exact solver gives numerical solutions for one-dimensional cases, which also provides the extent to which two-dimensional numerical models should achieve in modelling one-dimensional problems. Test cases 4 and 8 are tested against the

[1]These test specifications are built upon the strengths of the previous works published in a number of journals, articles and research work.

analytical solutions and hence comes under this category (see sections 8.10 and 8.6).

2. **Capability**: The capability of a model can be assessed objectively by testing the most commonly required features of a model. For this the capability of the model has been proposed by the following "can do" tests. Test cases 2 and 3 comes under this category (see sections 8.4 and 8.5).

3. **Reproducibility**: Reproducibility can be tested by a series of comparison tests i.e. numerical results compared with experimental or real world datasets. Test cases 5, 6 and 7 comes under this category (see sections 8.7, 8.8 and 8.9).

4. **Adaptability**: A model may excel at some or all of the above-mentioned tests, however without the ability to be adaptable or to have suitable form and function the user will potentially be hindered in its use. Any such assessment is subjective as users have different levels of experience and technical knowledge. However, the object-oriented design of the *Riemann2D* satisfies this criteria. Also, all the test are performed only using the configuration classes of the model (see description on classes in section 7.3.3). This shows that the adaptability of the *Riemann2D* , where all the above tests are conducted without changing the main code. Extensibility of the *Riemann2D* (example given for erosion problem) satisfies the criteria of easy adaptability for future research.

Note: One limitation of mesh generator (Argus MeshMaker) used in this work is that it can not create triangles with nodes on the interior boundary. Figure 8.2 (a) and (c) shows the typical mesh generated by the Argus MeshMaker. It is obvious that this can introduce significant error in the simulation results. Therefore, $x-$ and $y-$ coordinates of the nodes in the domain, for each test case, are manipulated to lay them on the interior boundaries. Figure 8.2 (b) and (d), shows the results after manipulation.

8.3 One-Dimensional Dam Break Problem

The dam-break problem has been widely studied in the literature by many different experimental, theoretical and numerical methods. Under the assumptions of isothermal and incompressible flow, the configurations are considered as shown in Figure 8.3.

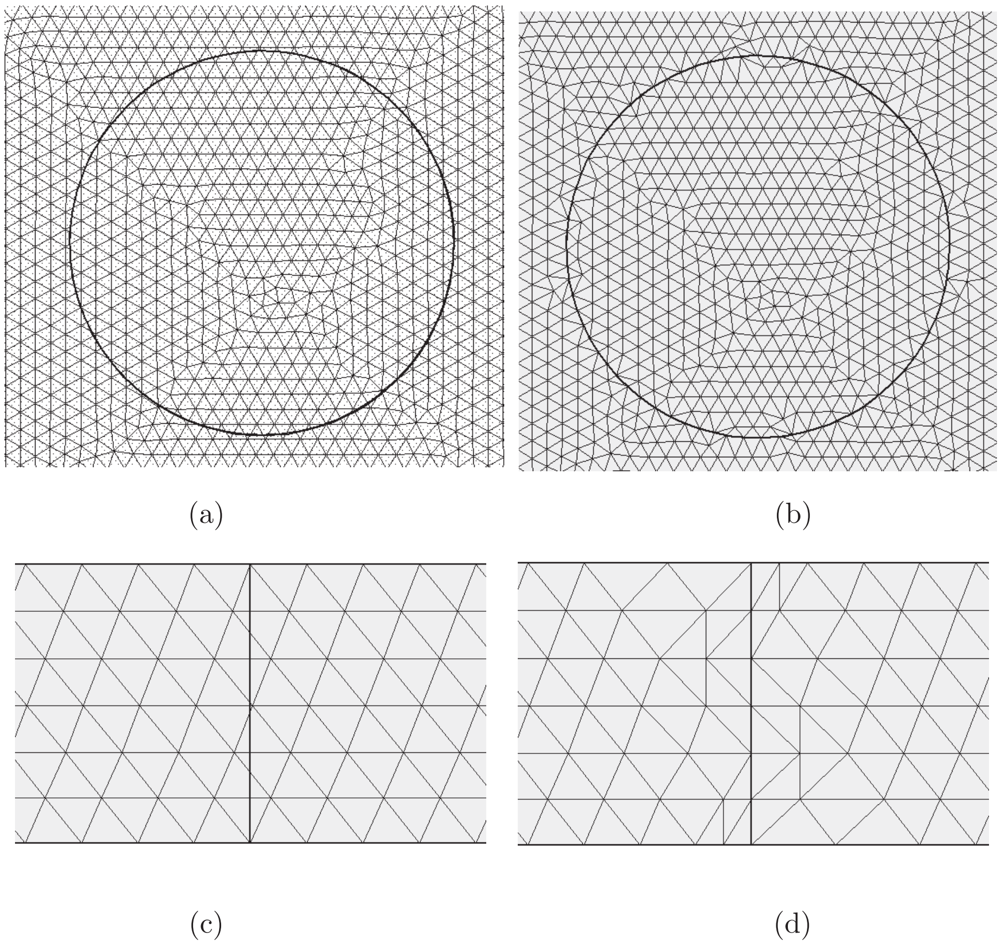


Figure 8.2: Sample mesh diagrams. (a) and (c) shows mesh with interior boundaries before manipulation. (b) and (d) shows mesh with interior boundaries after manipulation.

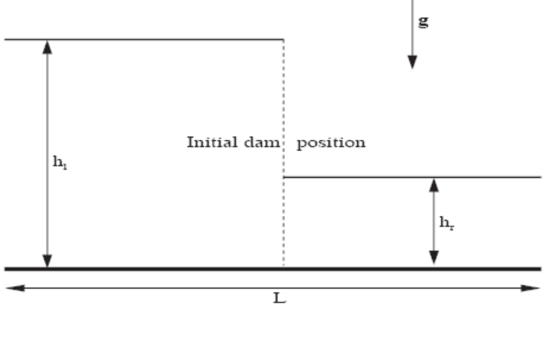


Figure 8.3: Definition sketch of one-dimensional dam break problem.

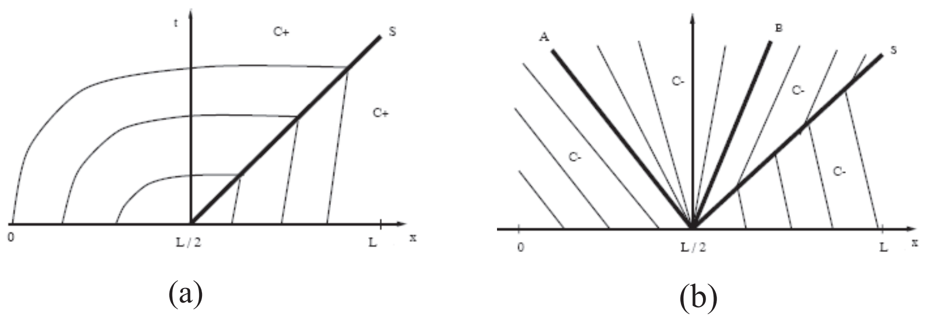


Figure 8.4: Dam-break flow on wet bottom. (a): description of the characteristic curves $C_+ = u + \sqrt{gh_s}$ in the plane$(x,\, t)$. Initially, the dam-break occurs at $x = 0$. A shock wave S appears. (b): description of the characteristic curves $C_- = u - \sqrt{gh_s}$ in the plane $(x,\, t)$. A rarefaction wave is generated between A and B.

One-dimensional dam-break test case is very interesting because it allows to validate the numerical simulations compared to analytical solution, experimental data and unsteady theoretical solutions. The problem emphasises the influence of the gravity, the surface tension and the viscosity. Shock and rarefaction waves appear during the wave breaking over the downstream water layer (see Figure 8.4). In certain cases, it is observed the developmental of a water jet leading to free surface shearing, stretching and droplet ejection. The strong deformations of the interface and the unsteady character of the flow confer on the test case a reference point of view to validate shock capturing methods used in the present work as well as numerical approximations.

8.3.1 Aim of Test

The aim of this test is to:

1. study the ability of *Riemann2D* in simulating the one-dimensional dam break problem;
2. study the numerical accuracy of different approximate Riemann solver (Roes', HLL and SO) with different limiters. The results are compared with the exact solution of Riemann problem;
3. study the behaviour of the *Riemann2D* in different direction;

4. present the particulars for developing and undertaking this test with *Riemann2D* and discuss the associated results.

8.3.2 Test Configuration

Usually the test cases for one-dimensional problems have width to length much less than 1. This doesn't allow to see the broader picture of the problem. Therefore, for an indepth assessment of the *Riemann2D* , the computational domain is chosen to be 50m $\times$ 50m. The numerical mesh consists of 7585 nodes and 14848 triangular elements (see Figure 8.5). Full breach of dam is considered *i.e.*, the breach is across full width of the channel, which is situated at the center of the channel, $x = 25$ m. All other boundaries are considered as reflective type (see section 4.5.2). The bed is considered as horizontal and the thickness of the dam wall is negligible. At the instant of breaking of the dam, water is released through the breach, forming a positive wave propagating downstream and a negative wave spreading upstream.

Figures 8.6 to 8.11 show the plot of height (h) and momentum (hu) versus distance in $x-$ direction for different approximate solvers with limiters. All the measurements are in meters (m). The numerical mesh configuration for these cases is shown in Figure 8.5(a).

To investigate the effect of direction ($x-$ and $y-$) on these solvers, simulation for four different initial conditions were done keeping the same number of element (see Figure 8.5). Figure 8.12 shows the comparison of these simulation results.

Note:

- Initial Condition: Upstream and downstream condition for each case is shown in figure 8.12.
- Boundary Condition: Reflective type on all exterior boundaries.
- Riemann Solver and Limiters: Details are given in each figure captions. See Figures 8.6 to 8.11.
- Courant Number: Desired - 0.25, Maximum - 0.5;
- Time step: Variable;

- Output Time: All results are plotted 4 seconds after dam break.
- Order in Time: First;
- Exact Solution is obtained from *Riemann1D*[1]. No Limiter is used and the cell size is $50/1200 = 0.0417$.

[1] *Riemann1D* was developed in the initial stage of the research (April, 2003 to August, 2003) as object-oriented version of CLAWPACK one-dimensional shallow water solver [see LeVeque (2002)]. Riemann1D was thoroughly tested against CLAWPACK results

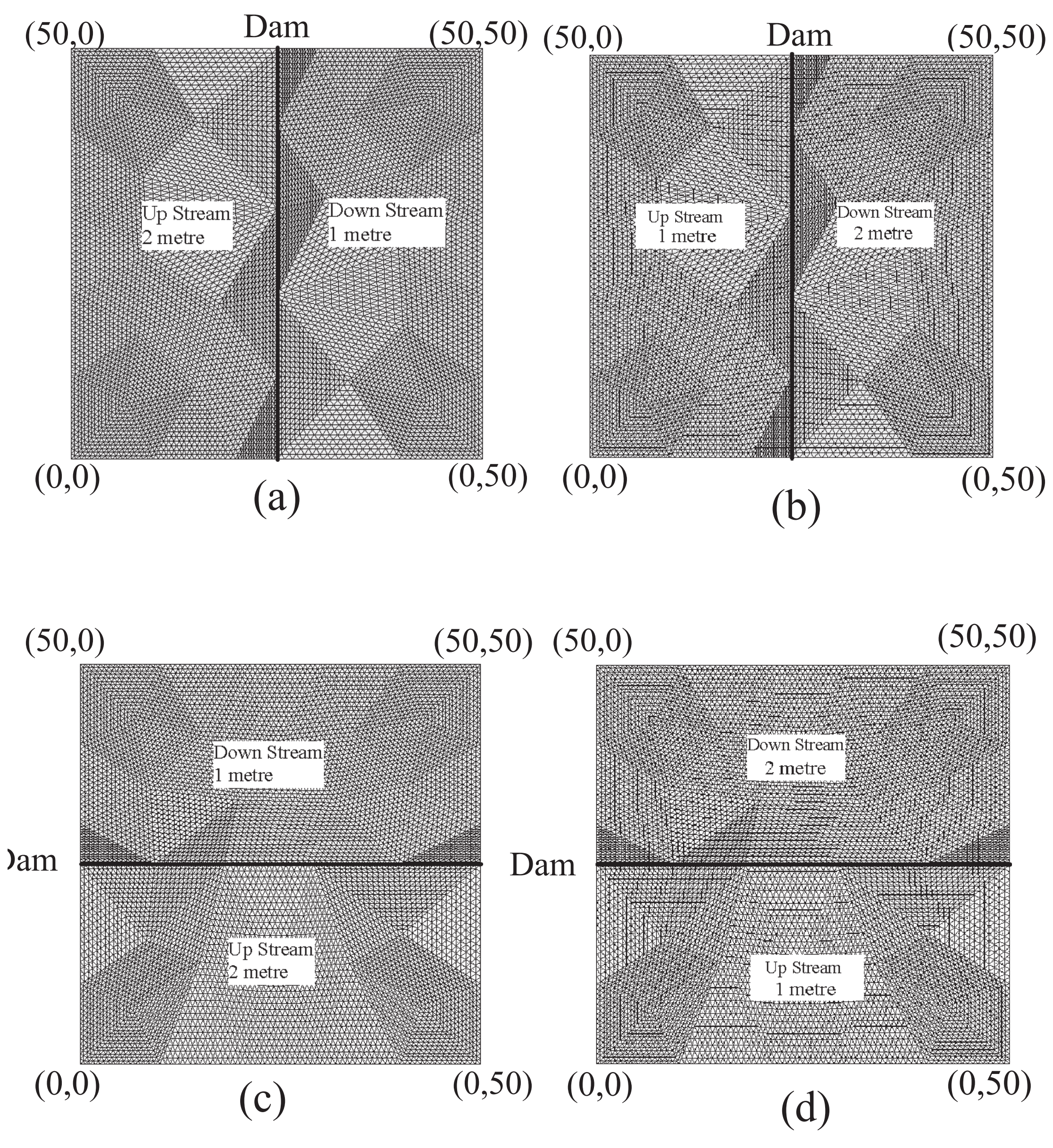
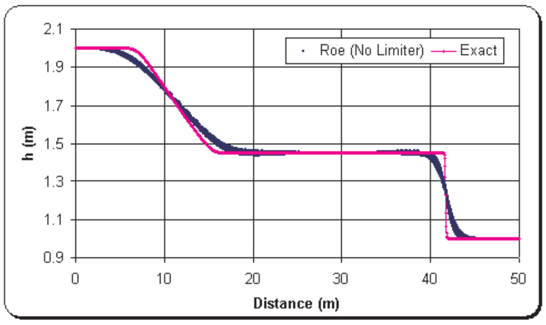
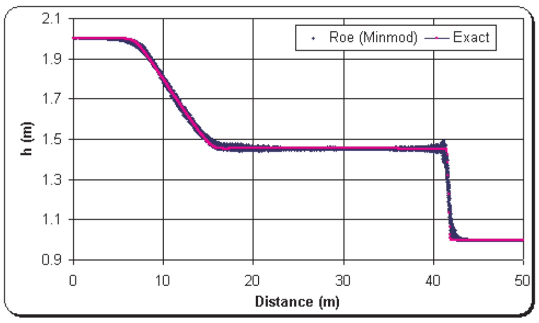


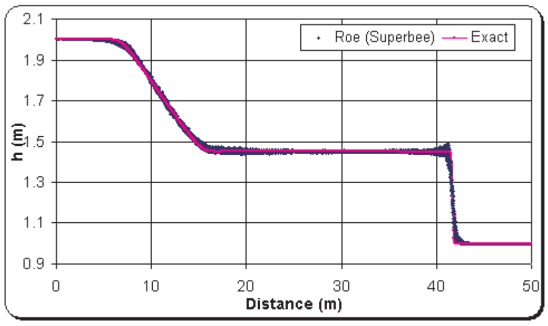
Figure 8.5: Numerical mesh configuration for one-dimensional dam break problem: The above figure shows four condition for dam break simulation. (a) Dam Break condition in positive $x-$ direction (b) Dam Break condition in negative $x-$ direction (c) Dam Break condition in positive $y-$ direction (d) Dam Break condition in negative $y-$ direction. Note: All the dimensions in the above figure are in meter.



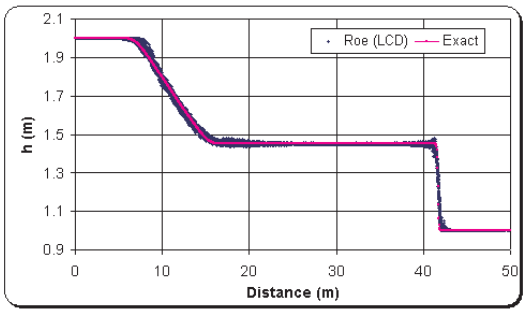
(a)



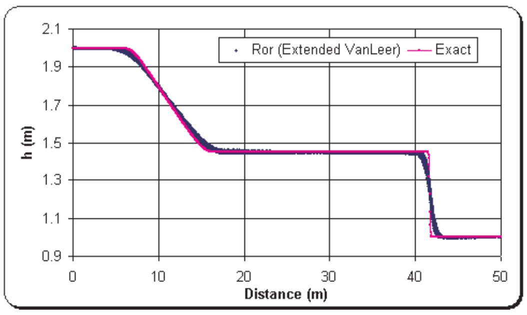
(b)



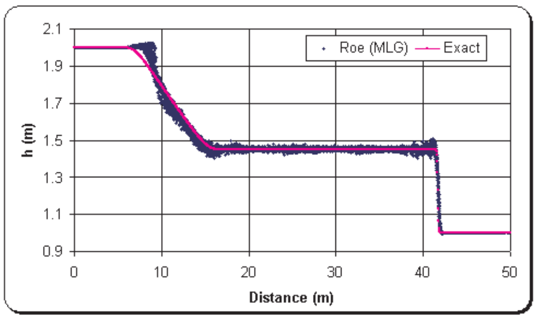
(c)



(d)

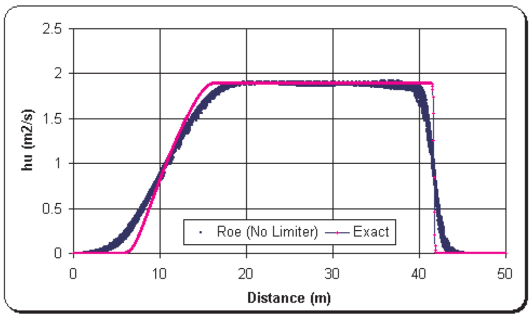


(e)

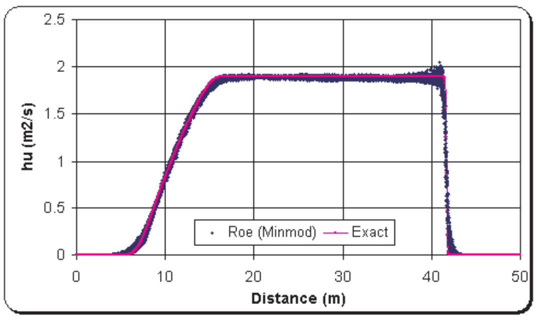


(f)

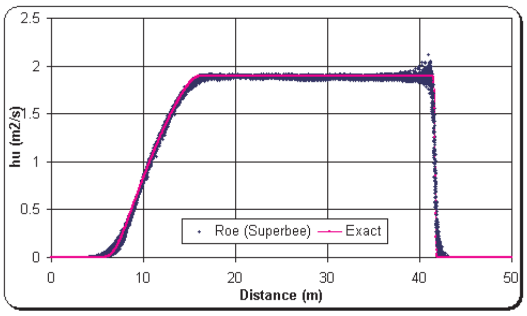
Figure 8.6: **Depth Vs Distance:** One-dimensional dam break simulation (Mesh configuration: Figure 8.5(a)) using Roe's approximate Riemann solver. The above figure shows the comparison between Roe's approximate solver (used in *Riemann2D*) with different limiters and one-dimensional exact solver [using Clawpack (see LeVeque (2002))]. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 4s after dam break.



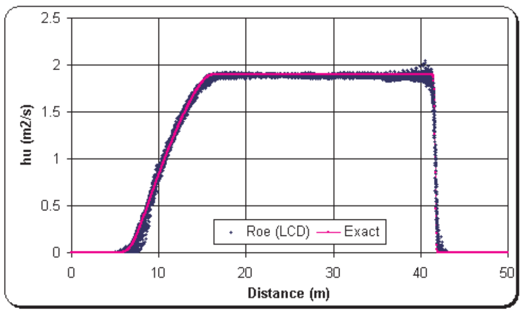
(a)



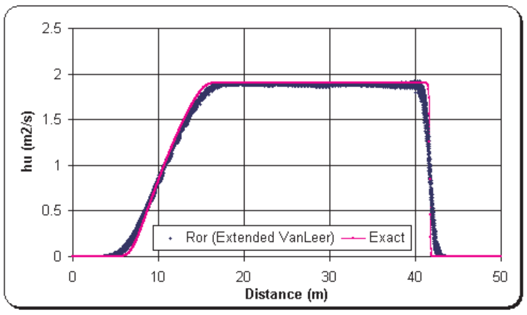
(b)



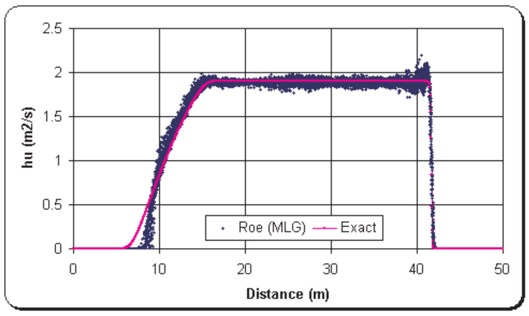
(c)



(d)

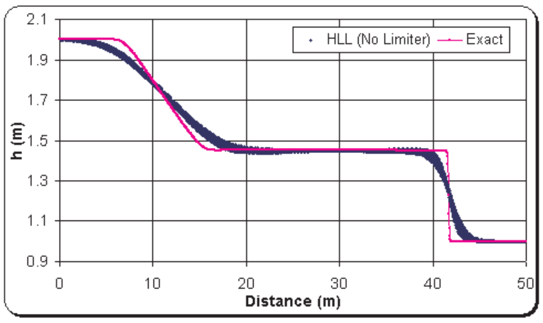


(e)

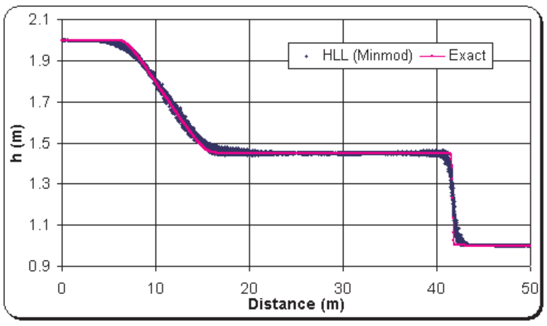


(f)

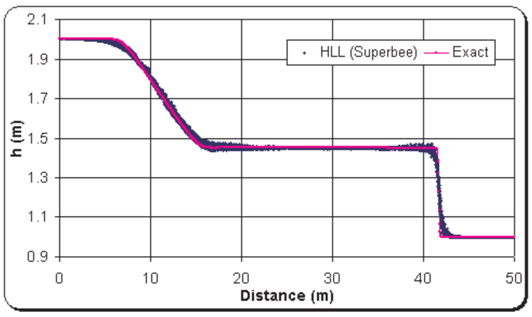
Figure 8.7: **Momentum Vs Distance:** One-dimensional dam break simulation (Mesh configuration: Figure 8.5(a)) using Roe's approximate Riemann solver. The above figure shows the comparison between Roe's approximate solver (used in *Riemann2D*) with different limiters and one-dimensional exact solver [using Clawpack (see LeVeque (2002))]. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 4s after dam break.



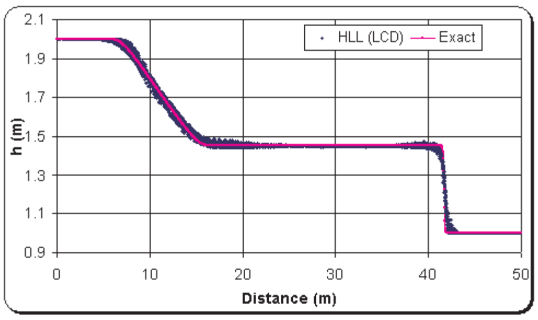
(a)



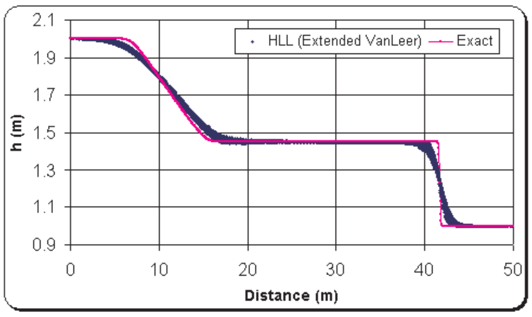
(b)



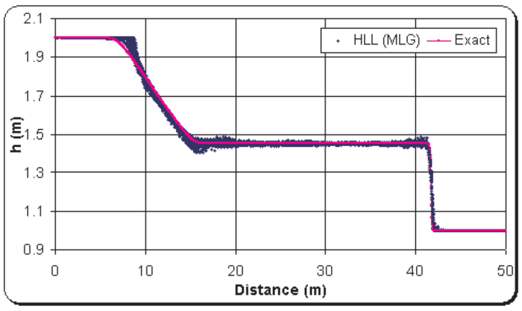
(c)



(d)

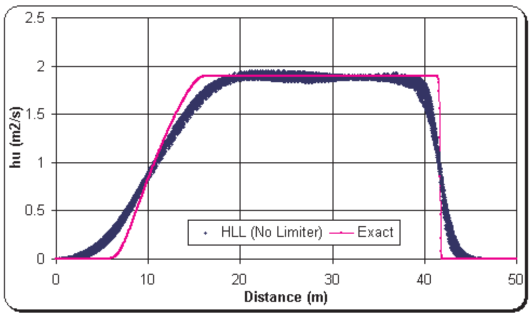


(e)

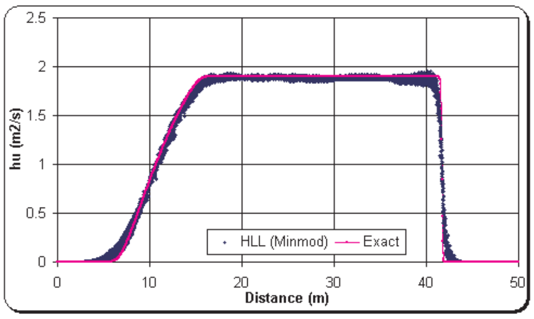


(f)

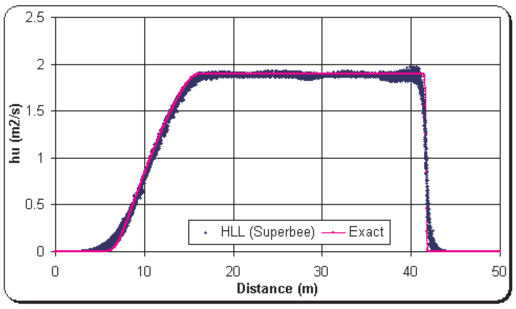
Figure 8.8: **Depth Vs Distance:** One-dimensional dam break simulation (Mesh configuration: Figure 8.5(a)) using HLL approximate Riemann solver. The above figure shows the comparison between HLL approximate solver (used in *Riemann2D*) with different limiters and one-dimensional exact solver [using Clawpack (see LeVeque (2002))]. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 4s after dam break.



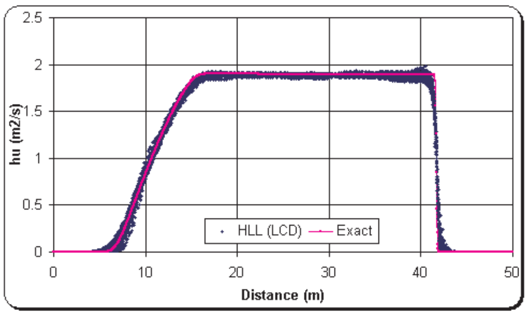
(a)



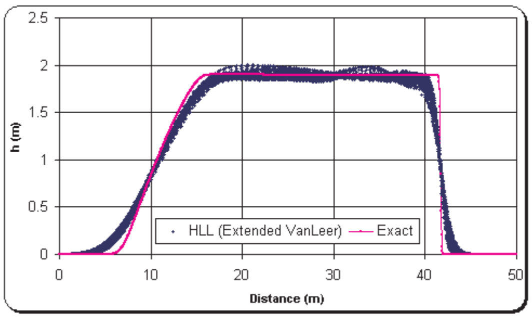
(b)



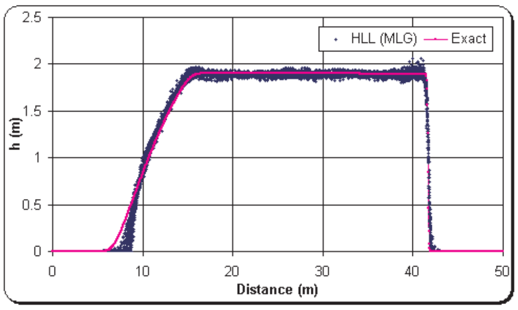
(c)



(d)

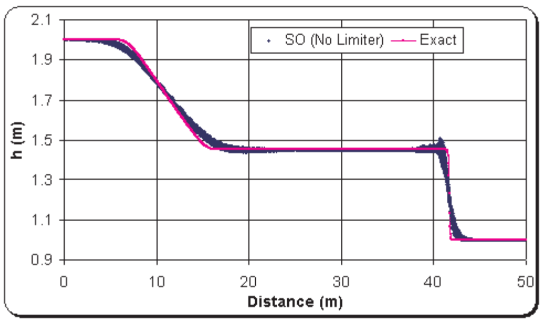


(e)

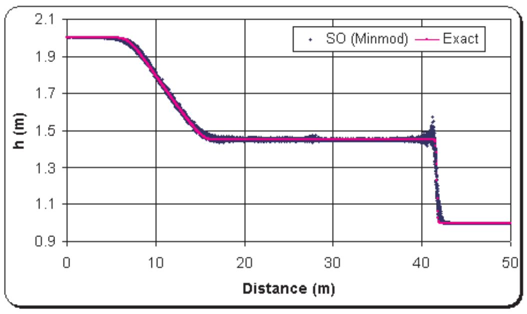


(f)

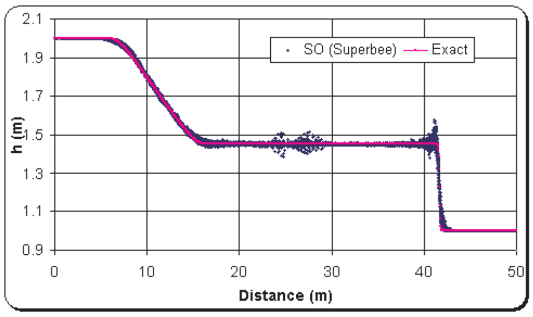
Figure 8.9: **Momentum Vs Distance:** One-dimensional dam break simulation (Mesh configuration: Figure 8.5(a)) using HLL approximate Riemann solver. The above figure shows the comparison between Roe's approximate solver (used in *Riemann2D*) with different limiters and one-dimensional exact solver [using Clawpack (see LeVeque (2002))]. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 4s after dam break.



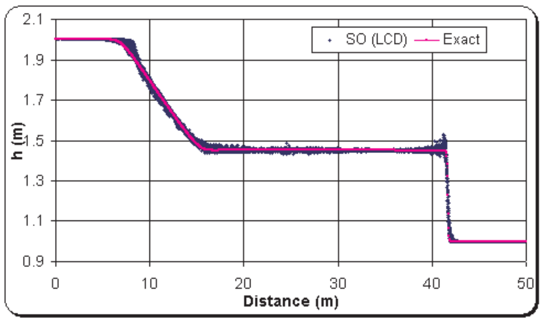
(a)



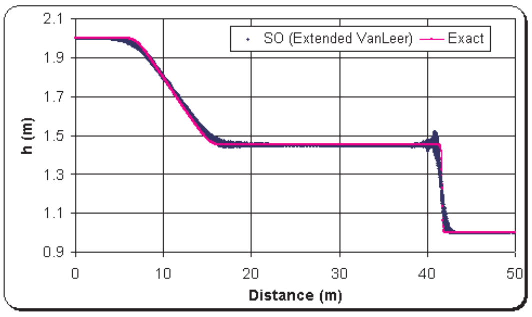
(b)



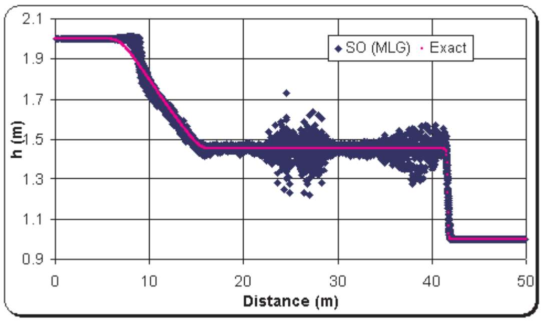
(c)



(d)



(e)



(f)

Figure 8.10: **Depth Vs Distance:** One-dimensional dam break simulation (Mesh configuration: Figure 8.5(a)) using Shu and Oshers' (SO) approximate Riemann solver. The above figure shows the comparison between Roe's approximate solver (used in *Riemann2D*) with different limiters and one-dimensional exact solver [using Clawpack (see LeVeque (2002))]. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 4s after dam break.

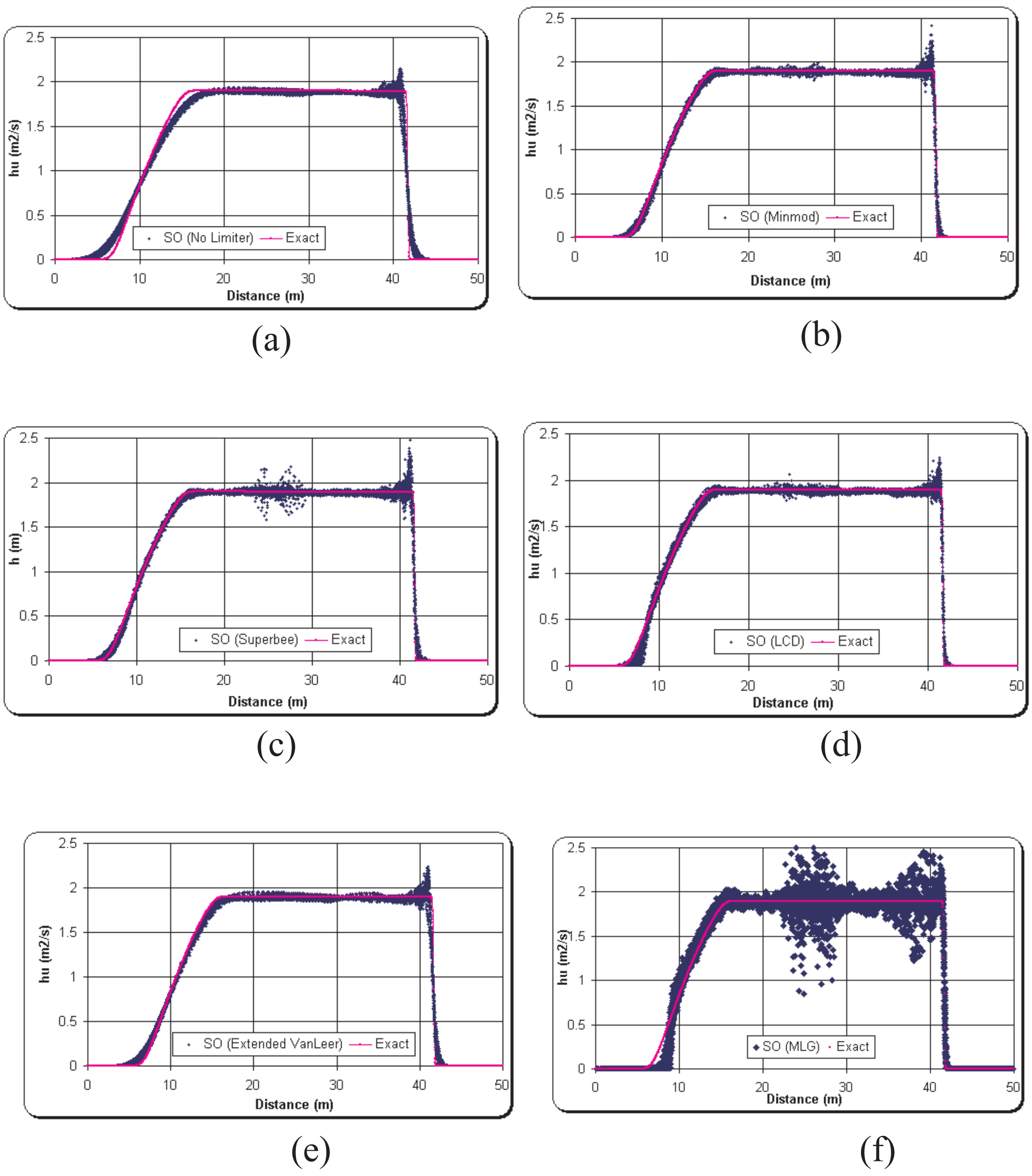


Figure 8.11: **Momentum Vs Distance:** One-dimensional dam break simulation (Mesh configuration: Figure 8.5(a)) using Shu and Oshers' (SO) approximate Riemann solver. The above figure shows the comparison between Roe's approximate solver (used in *Riemann2D*) with different limiters and one-dimensional exact solver [using Clawpack (see LeVeque (2002))]. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 4s after dam break.

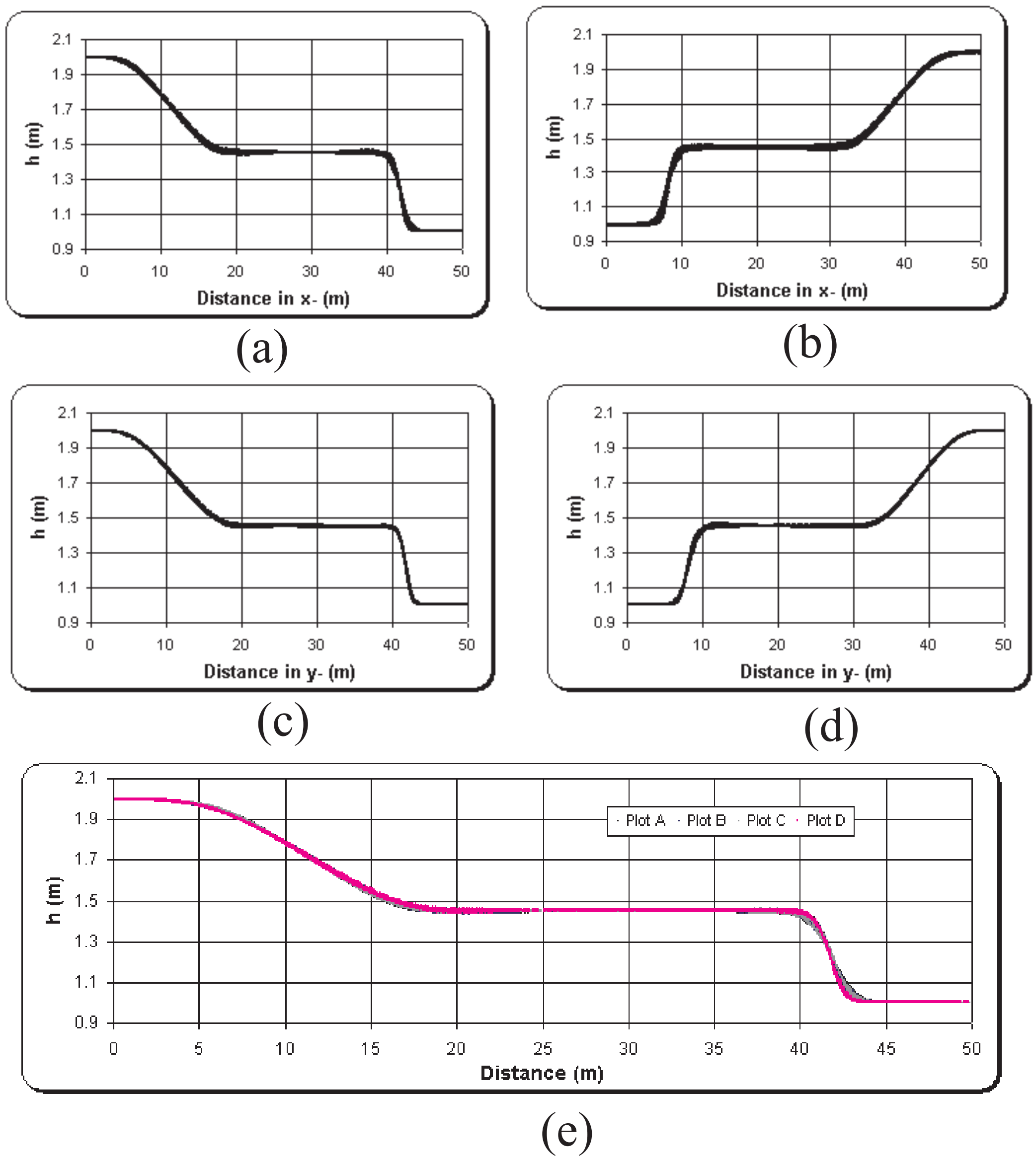


Figure 8.12: One-dimensional dam break simulation. All the above case is simulated using Roe solver without any limiter. Simulation for:
(a) mesh configuration in Figure 8.5(a);
(b) mesh configuration in Figure 8.5(b);
(c) mesh configuration in Figure 8.5(c);
(d) mesh configuration in Figure 8.5(d); and
(e) Comparison of all the four cases by rotating the axis for cases (b) and (d). Note: Plot at time = 4s after dam break.

8.3.3 Observations

The test is an investigation of the capability of the approximate Riemann solvers with limiters to simulate the behaviour of the hydraulic condition in one-dimensional channel. All the figures use the *scatter* plot (MS Excel), which helps to see properties (h and hu) for every element present within the domain. As a result several notable difference between various combinations of approximate solution and limiters, and the exact solution have been highlighted. These observations can be summarised as follows:

1. Roes' approximate solver (see Figures 8.6 and 8.7):

    - Without use of any limiter, as expected, the Roes' solver fails to capture the shock. However, it gives a smooth profile for h and hu near the shock. This emphases the need for limiters for capturing shocks.
    - The minmod, superbee and LCD limiters, which are widely used by many researchers (see section 2.4.2), captures the shocks and gives good results compare to the exact solution. But, these limiters show the signs of oscillation at the tip of the shock. However, it should be noted that these oscillation are visible because of the width of the channel. Also, the deviation of the properties (h and hu) at the tip of the shock is present only in fewer elements compared to the total number of elements across the width of the channel. LCD, which is of same rank (1) as minmod and superbee, shows better shock capturing ability compared to other rank 1 limiters.
    - The extended version of one-dimensional VanLeer limiter shows better shock capturing ability compared to no limiter case. But, it is found to have inferior shock capturing ability compared to other limiters for this test case. However, an advantage of extended-VanLeer over other limiters is that it keeps the solution smooth thought out the profile.
    - The MLG limiter is found to capture the shock but it fails to capture the refracted wave. It also produce small oscillation through out the profile. From the profiles it can be seen that MLG limiter is trying to limit the slope to greater extend, which could be the reason for these oscillations.

2. HLL approximate solver (see Figures 8.8 and 8.9):
   The observations for limiters with HLL solver are not much different than those with Roes'. There are few other important observations:
   - All the limiters seems to give smooth profile with HLL approximate solver compared to working with Roe's approximate solver.
   - But the performance of these limiters with HLL approximate solver is inferior compared to those with Roes'.
3. SO approximate solver (see Figures 8.10 and 8.11)
   - Given the performance of the SO approximate solver without any limiter; minmod, LCD and extended-VanLeer gives decent results.
   - Superbee limiter can be seen producing oscillation near the tip of the shock as well as the original position of the Dam.
   - The MLG limiter can be concluded to be a total failure. It is interesting to note that the behaviour of MLG and superbee limiter resembles each other. This could be because of the fact that the MLG is an extended version of superbee limiter [Batten *et al.* (1996)].
4. To confirm the isotropic behaviour of the solvers, results for same problem with different direction (see Figure 8.5) were compared. Figure 8.12 is illustrates the perfect isotropic behaviour of Roes' solver (without any limiter). Similar results were obtained for other solver in combination with limiters.

8.4 Two-Dimensional Circular Dam Break

Another important benchmark example is the one presented in many literature including, Alcrudo and Garcia-Navarro (1993), Gottardi and Venutelli (2004), and Delis and Katsaounis (2005). It involves the breaking of a circular dam, and it is an important test example for the analysis and performance of the presented Riemann solvers and limiters when solving complex shallow flow problems, especially for symmetry. This is like a two-dimensional Riemann problem for the two-dimensional shallow water equations [Delis and Katsaounis (2005)]. The circular dam-break bore waves will spread and propagate radially and symmetrically as the water drains from the deepest region.

Then there is a transition from subcritical to supercritical flow (see Figure 8.13 (d)). In this test case, we consider a circular dam in idealized and horizontal bottom and study the time evolution of the shock waves associated with the breaking of the dam.

8.4.1 Aim of the test

The aim of this test is to:

1. study the ability of the *Riemann2D* to simulate the case of radial symmetry.
2. study the numerical ability of approximate solvers (Roes', HLL and SO) and limiters in handling transitional (sub-to-super and super-to-sub critical) flow condition. In absence of analytical and experimental data results are compared to solutions publications in a number of journals (see Figure 8.20); and
3. present the particulars for developing and undertaking the test with *Riemann2D* and discuss the associated results.

8.4.2 Test Configuration

Initially, the physical model is that of two regions of still water separated by a cylindrical wall (with radius 11 m) centered in a 50×50m square domain. The wall (assumed to be of negligible thickness) is then assumed to be removed completely and no slope or friction is considered. The numerical mesh consists of 1609 nodes and 3072 triangular elements (see Figure 8.13 (a)). The initial conditions are $u = v = 0$ throughout the domain, whereas the water depth inside and outside the dam is respectively, $h_0 = 10$m and $h_0 = 1$m (see Figure 8.13 (b)). The plot of the free surface elevation at $t = 0.69$s is shown in Figures 8.14 to 8.19.

Note:

- Initial Condition: Water depth in the circular dam = 10m, outside dam = 1m.
- Boundary Condition: Reflective type on all exterior boundaries.
- Riemann Solver and Limiters: Details are given in each figure captions. See Figures 8.14 to 8.19.

- Courant Number: Desired - 0.25, Maximum - 0.5;
- Time step: Variable;
- Output Time: All results are plotted 0.69 seconds after dam break.
- Order in Time: First;
- The mesh shown in 3D perspective view (see Figures 8.15, 8.17 and 8.19) are only for better visual effect. They are not the numerical mesh for the problem.

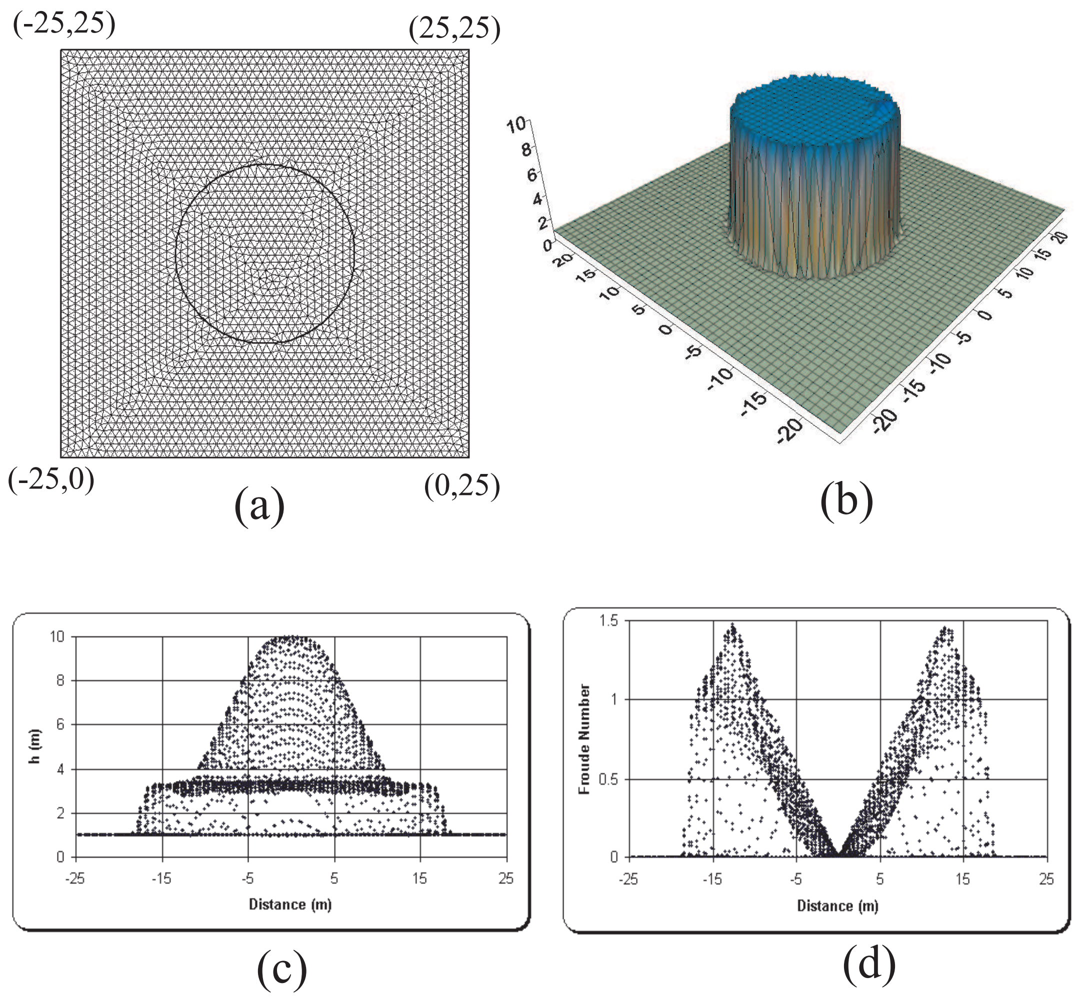


Figure 8.13: Two-dimensional circular dam break configuration: The above figure shows (a) numerical mesh and plan view of the problem, (b) 3D perspective view of the initial condition, (c) scatter plot of the water height in each element of the domain at $t = 0.69$ s (using Roes' approximate solver without limiter), and (d) scatter plot of the Froude number for each element of the domain at $t = 0.69$ s (using Roes' approximate solver without limiter). It can be seen that this is problem of two-dimensional transitional (sub-to-super and super-to-sub critical) flow condition.

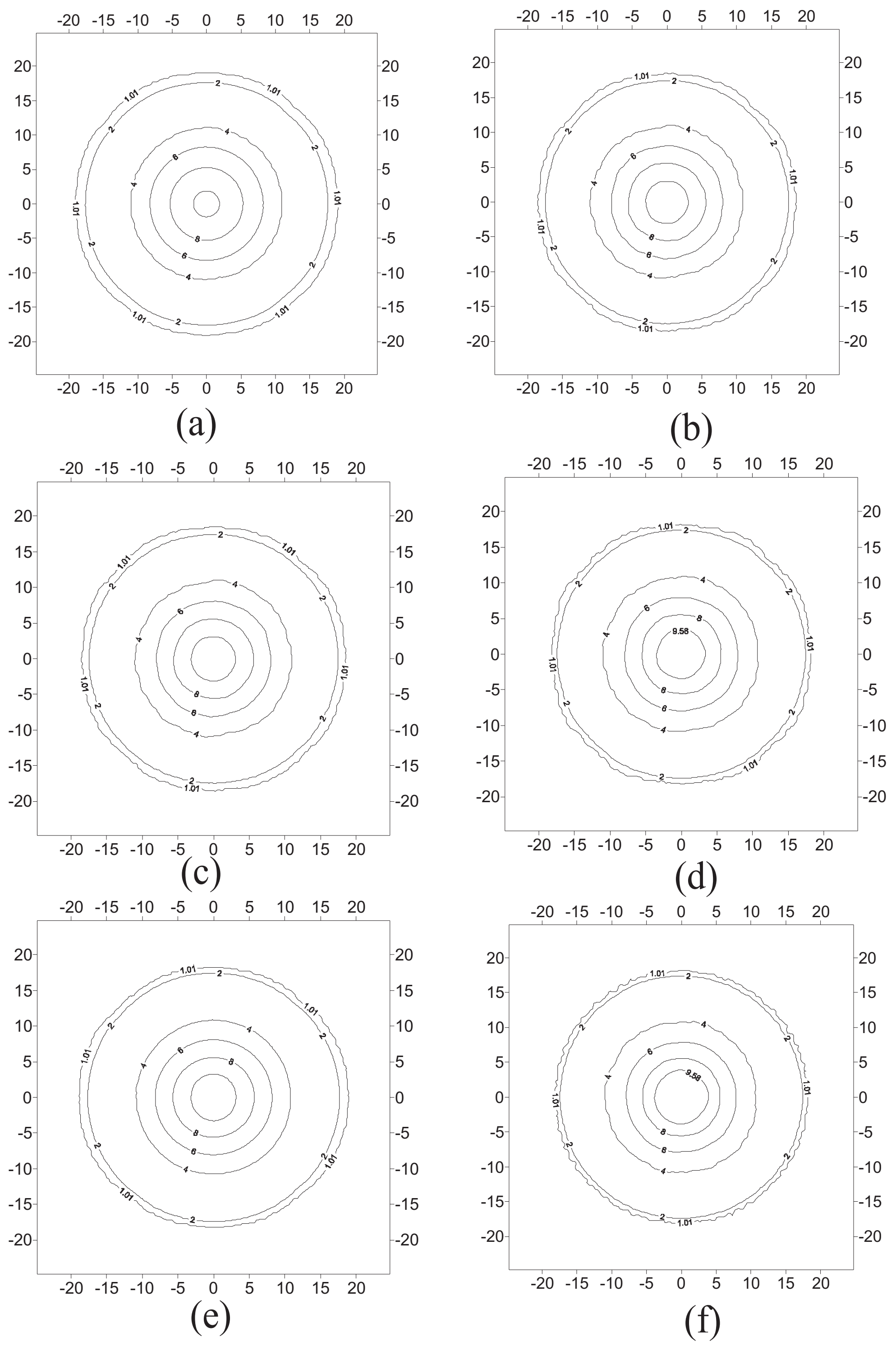


Figure 8.14: Two-dimensional circular dam break simulation (Mesh configuration: Figure 8.13) using Roe's approximate Riemann solver. The above figure shows depth contour plots for dam break situation. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 0.69s after dam break.

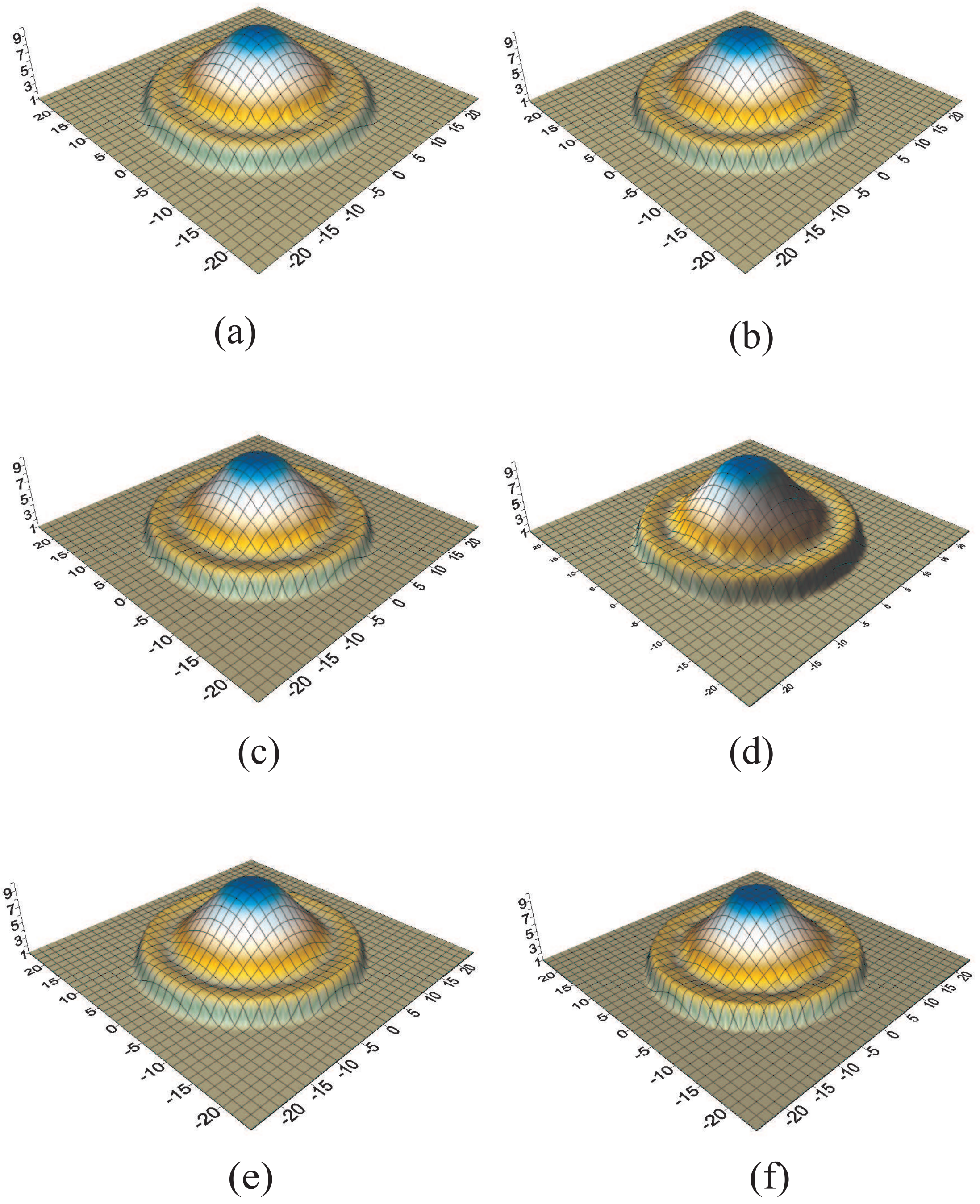


Figure 8.15: Two-dimensional circular dam break simulation (Mesh configuration: Figure 8.13) using Roe's approximate Riemann solver. The above figure shows 3D perspective view of the dam break situation. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 0.69s after dam break.

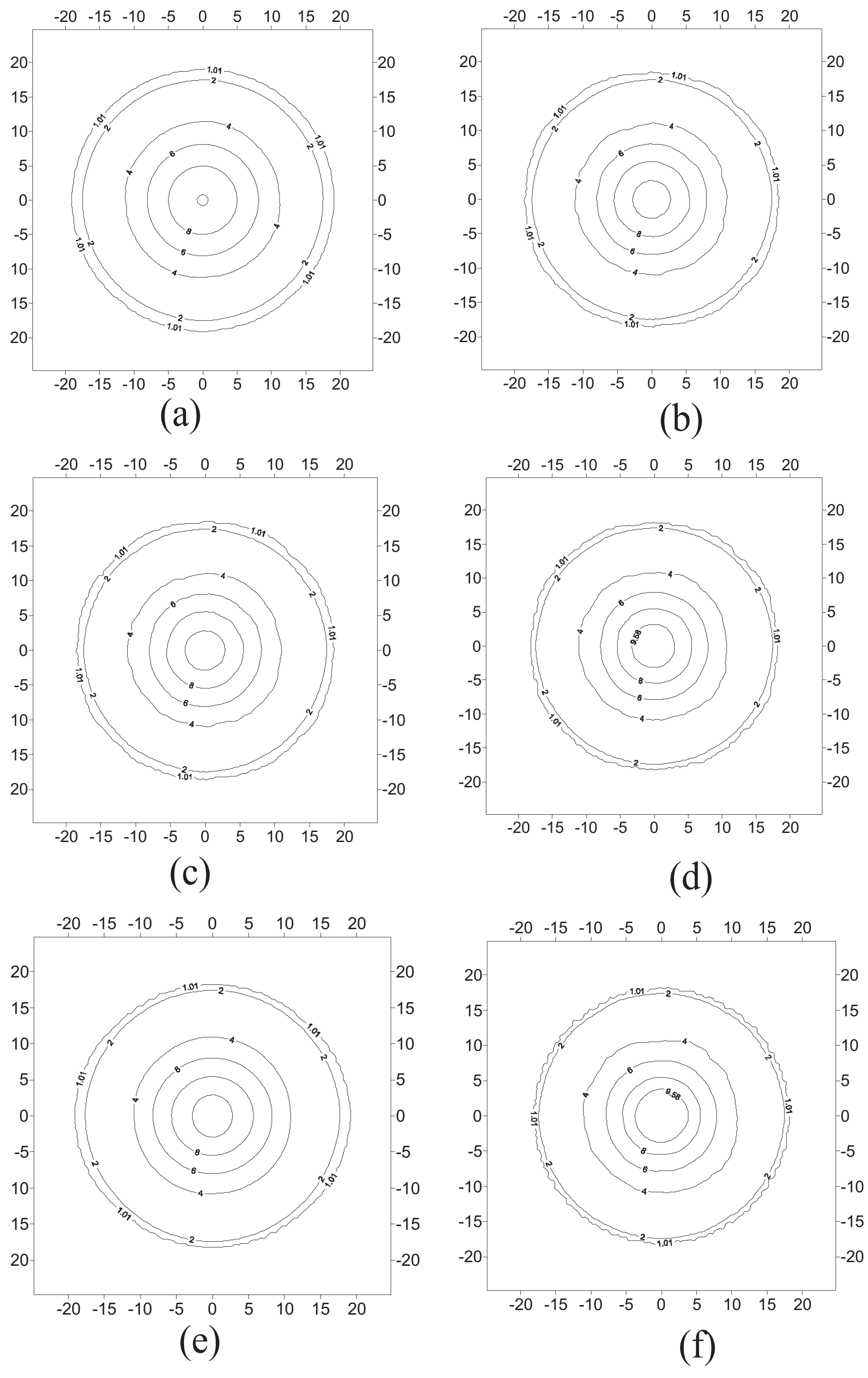
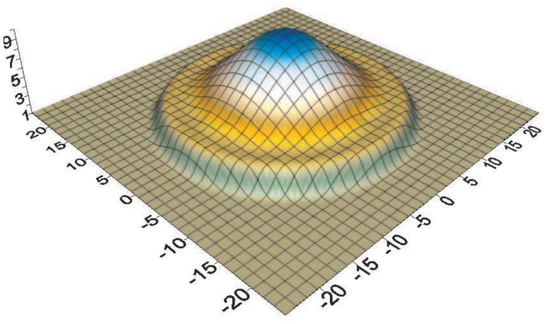
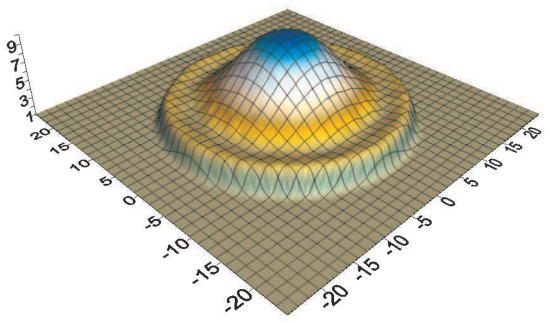


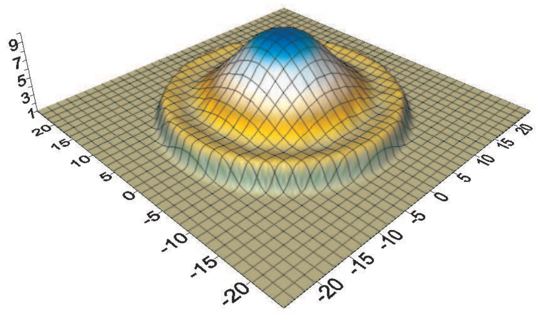
Figure 8.16: Two-dimensional circular dam break simulation (Mesh configuration: Figure 8.13) using HLL approximate Riemann solver. The above figure shows depth contour plots for dam break situation. Limiters used with HLL solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 0.69s after dam break.



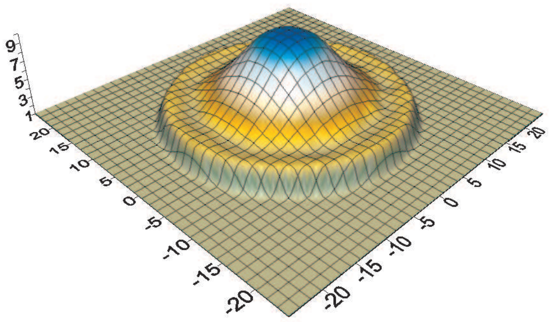
(a)



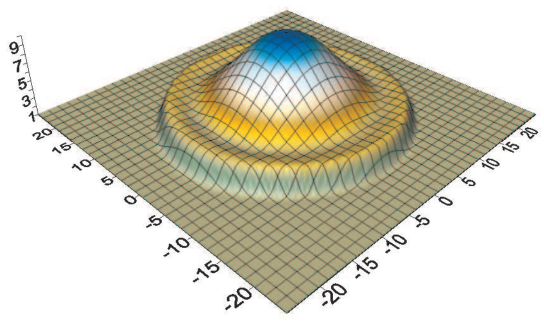
(b)



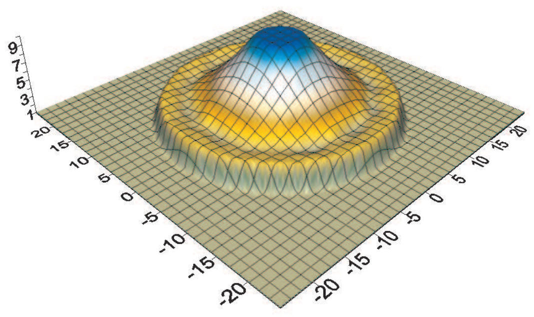
(c)



(d)



(e)



(f)

Figure 8.17: Two-dimensional circular dam break simulation (Mesh configuration: Figure 8.13) using HLL approximate Riemann solver. The above figure shows 3D perspective view of the dam break situation. Limiters used with HLL solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 0.69s after dam break.

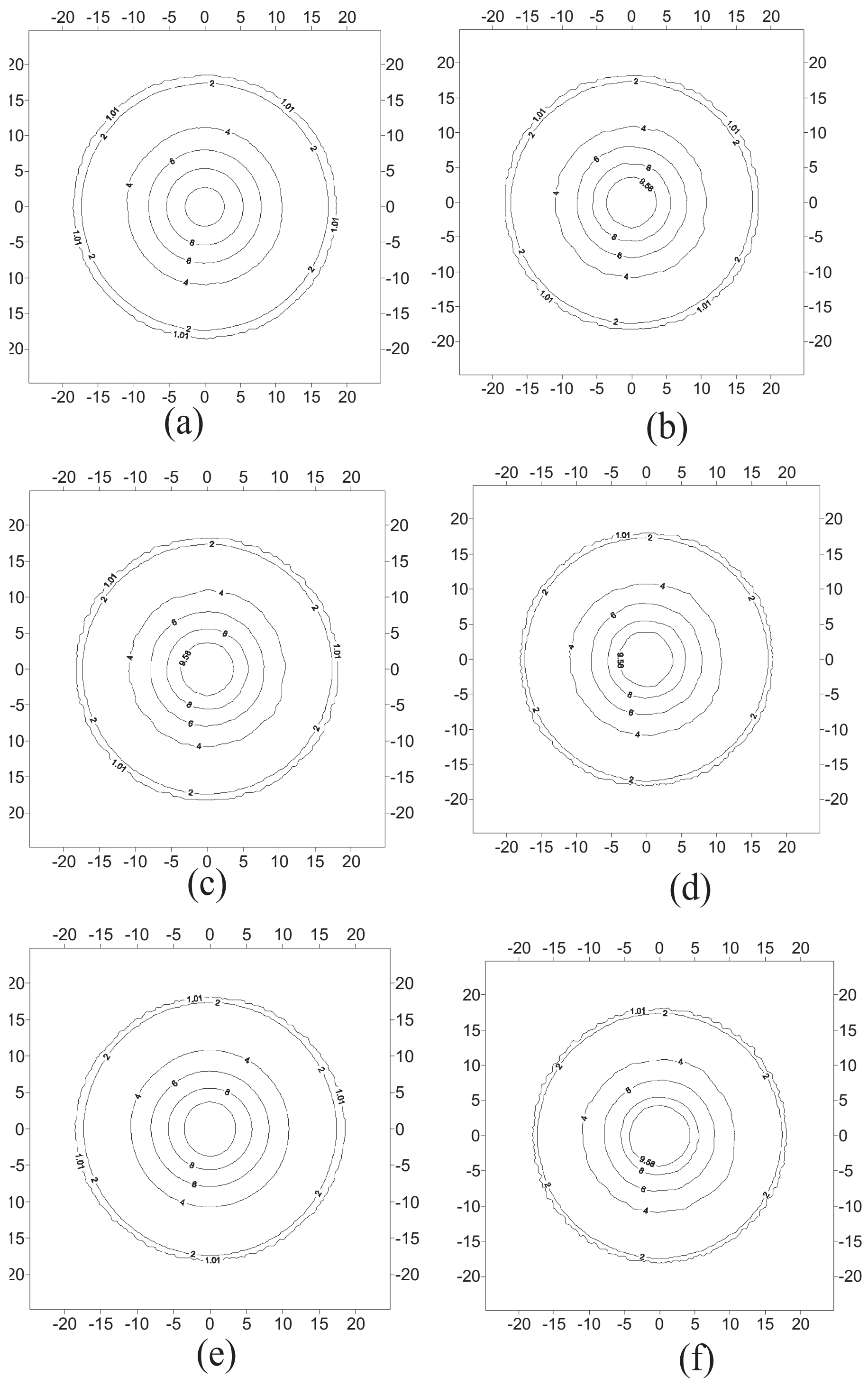


Figure 8.18: Two-dimensional circular dam break simulation (Mesh configuration: Figure 8.13) using Shu and Osher's (SO) approximate Riemann solver. The above figure shows depth contour plots for dam break situation. Limiters used with Shu and Osher's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 0.69s after dam break.

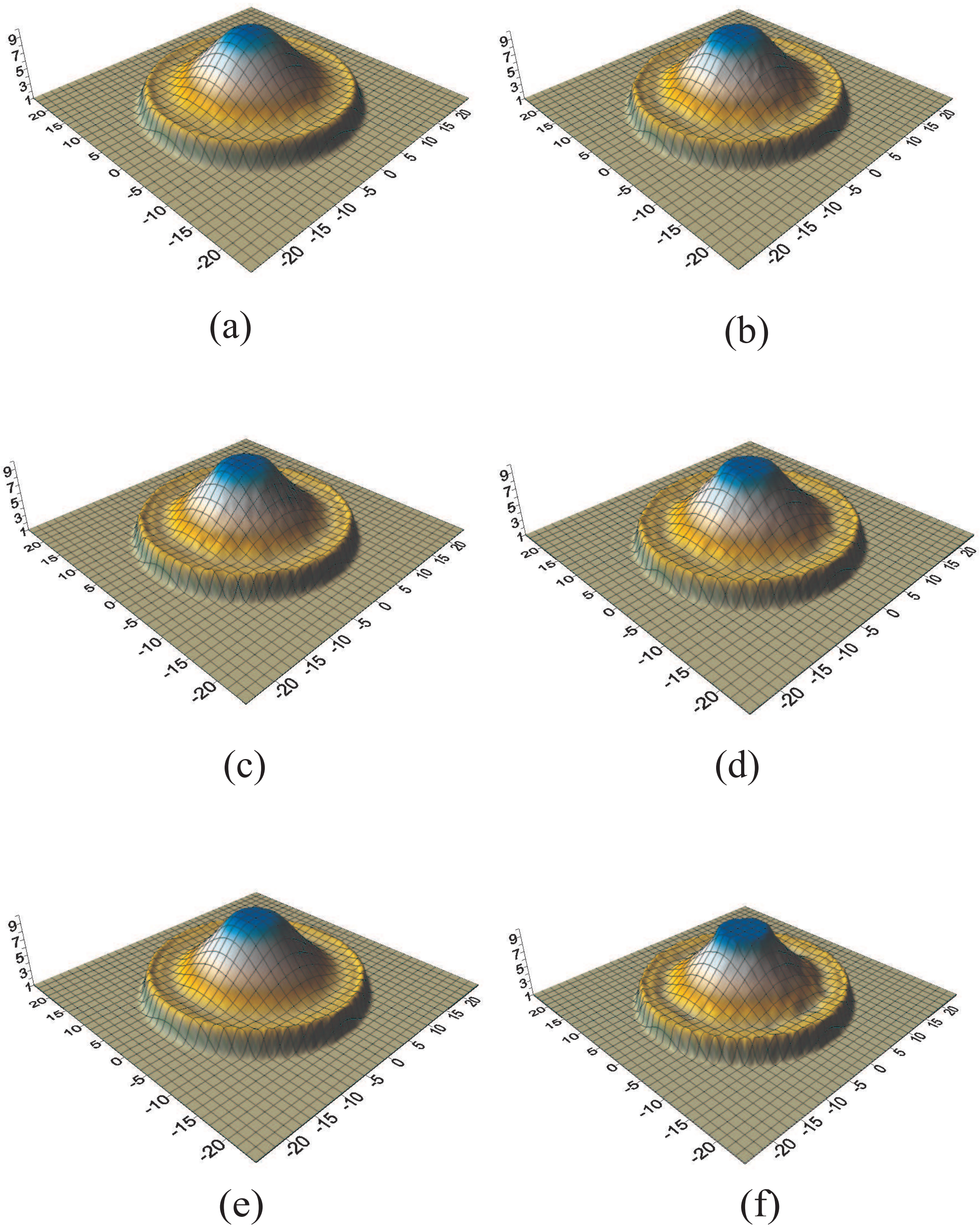
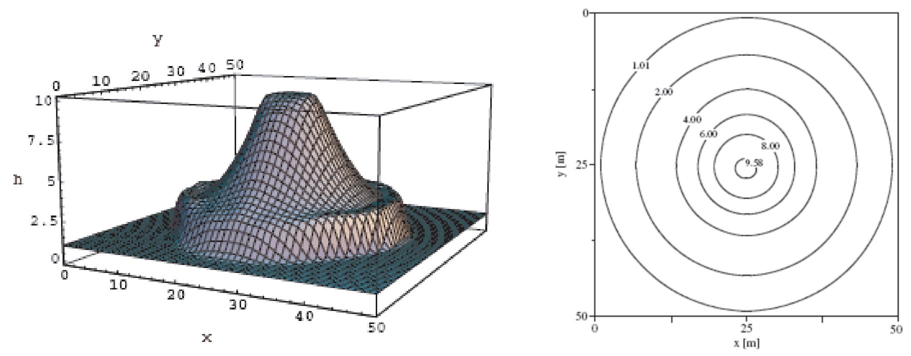
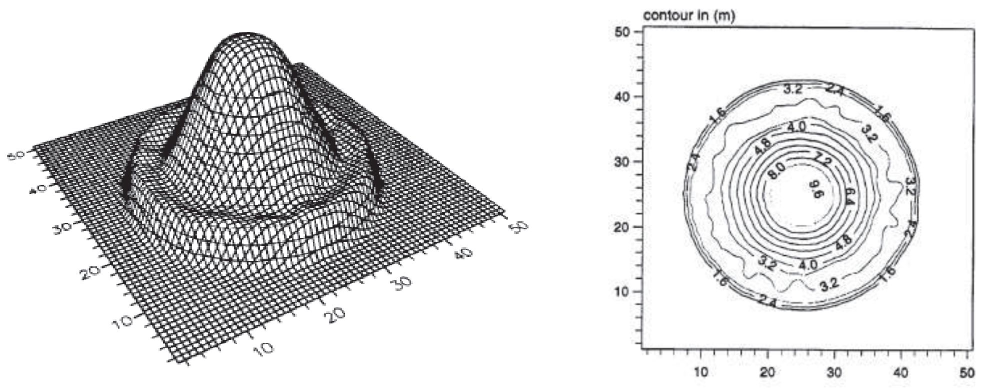


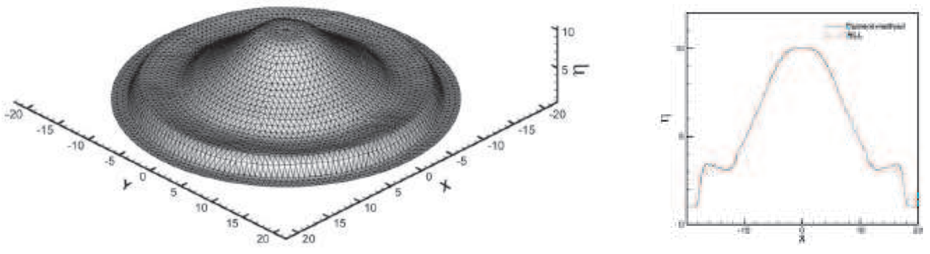
Figure 8.19: Two-dimensional circular dam break simulation (Mesh configuration: Figure 8.13) using Shu and Osher's (SO) approximate Riemann solver. The above figure shows 3D perspective view of the dam break situation. Limiters used with Shu and Osher's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 0.69s after dam break.



(a)



(b)



(c)

Figure 8.20: 3D surface view and water depth contour published in various literature (a) Gottardi and Venutelli (2004) (b) Anastasiou and Chan (1997) and (c) Mohamadian *et al.* (2005). Note: All the initial condition and boundary conditions are same as used in the present test case. Time = 0.69s

8.4.3 Observations

This test is an extreme investigation of the capability of the approximate Riemann solvers with limiters to simulate the transitional flow condition.

It can be clearly seen that for all the combination of approximate Riemann solver and limiters, the waves spread uniformly and symmetrically, with the radial symmetry slightly distorted by the effects of the triangular elements due to the inability to represent a circle on a triangular mesh, but otherwise the solution gives very good results and agrees very well with those shown in Figure 8.20. The results of this test can also be seen in Delis and Katsaounis (2005), Alcrudo and Garcia-Navarro (1993) and Mingham and Causon (1998). However, few comments can be made on the presented results:

1. The extended-VanLeer can be seen giving very smooth results, compared to other limiters, for all the solvers.
2. The MLG limiter gives most distorted result, of all other limiters, for all the solvers.
3. It can be concluded that all the combination of approximate solver and limiter gives sharp gradient at the shock. This can be noted from the Figures 8.14 to 8.19 that the contour of 2m and 1.01m is very near to each other.

8.5 Two-Dimensional Partial Dam Break

The two-dimensional hypothetical problem used here is the one presented by Fennema and Chaudhry (1990), Anastasiou and Chan (1997) and Delis and Katsaounis (2005).

8.5.1 Aim of the test

The aim of this test is to:

1. study the ability of the *Riemann2D* to simulate different front wave propagations, with particular attention to the 2D aspects of the flow;
2. study the numerical accuracy of approximate solvers (Roes', HLL and SO) and limiters compared to solutions produced in different publications; and

3. present the particulars for developing and undertaking the test with *Riemann2D* and discuss the associated results.

8.5.2 Test Configuration

For this problem the dam, located in the center of a region, is assumed to partially fail instantaneously. The bottom is frictionless (see Figure 8.21). The water depth upstream of the dam is $h_u = 10$m and downstream is assumed to be either $h_d = 5$m. The computational domain is a 200×200m. The numerical mesh consists of 7381 nodes and 15232 triangular elements. The breach is 75 m in length, which has distances of 30m from the left bank and 95 m from the right. The boundary conditions at $x = 0$ and $x = 200$m are assumed to be transmissive and all other boundaries are considered as reflective. At the instant of breaking of the dam, water is released through the breach, forming a positive wave propagating downstream and a negative wave spreading upstream. When $h_d = 5$m the flow is subcritical everywhere.
Note:

- Initial Condition: Water in upstream = 10 m, and downstream = 5 m.
- Boundary Condition: Reflective type on all exterior boundaries.
- Riemann Solver and Limiters: Details are given in each figure captions. See Figures 8.22 to 8.27.
- Courant Number: Desired - 0.25, Maximum - 0.5;
- Time step: Variable;
- Output Time: All results are plotted 7.2 seconds after dam break.
- Order in Time: First;
- The mesh shown in 3D perspective view (see Figures 8.23, 8.25 and 8.27) are only for better visual effect. They are not numerical mesh for the problem.

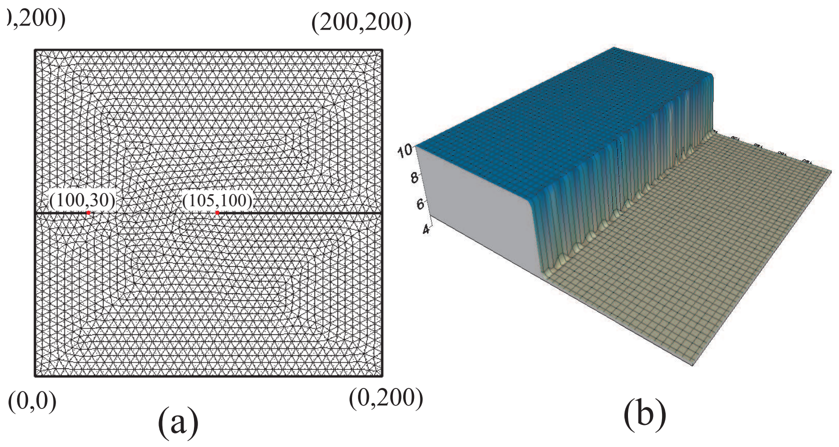


Figure 8.21: Two-dimensional partial dam break configuration: The above figure shows (a) numerical mesh and plan view of the problem, and (b) 3D perspective view of the initial condition

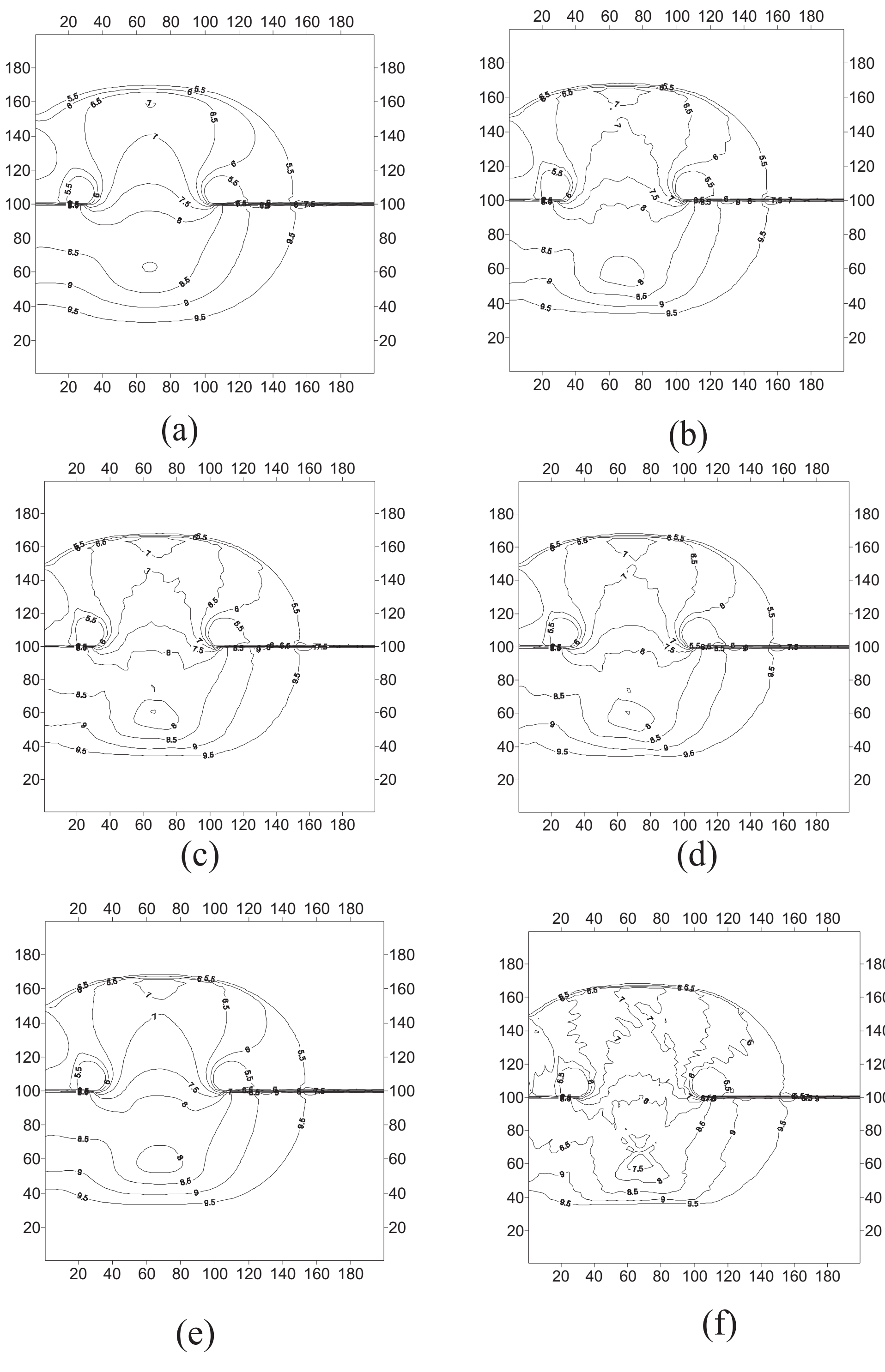


Figure 8.22: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using Roe's approximate Riemann solver. The above figure shows depth contour plots for the dam break situation. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 7.2s after dam break.

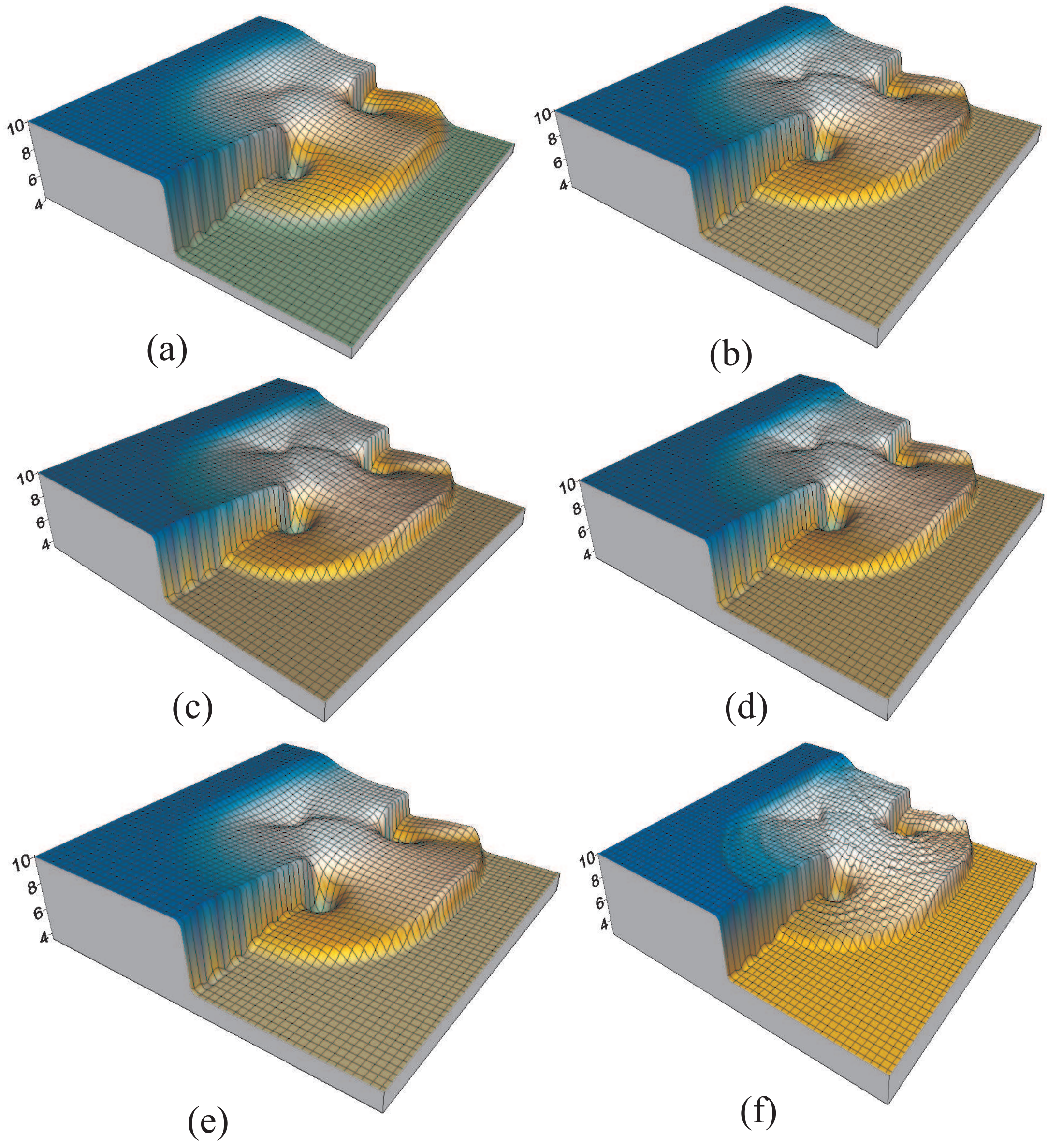


Figure 8.23: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using Roe's approximate Riemann solver. The above figure shows 3D perspective view of the dam break situation. Limiters used with Roe's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 7.2s after dam break.

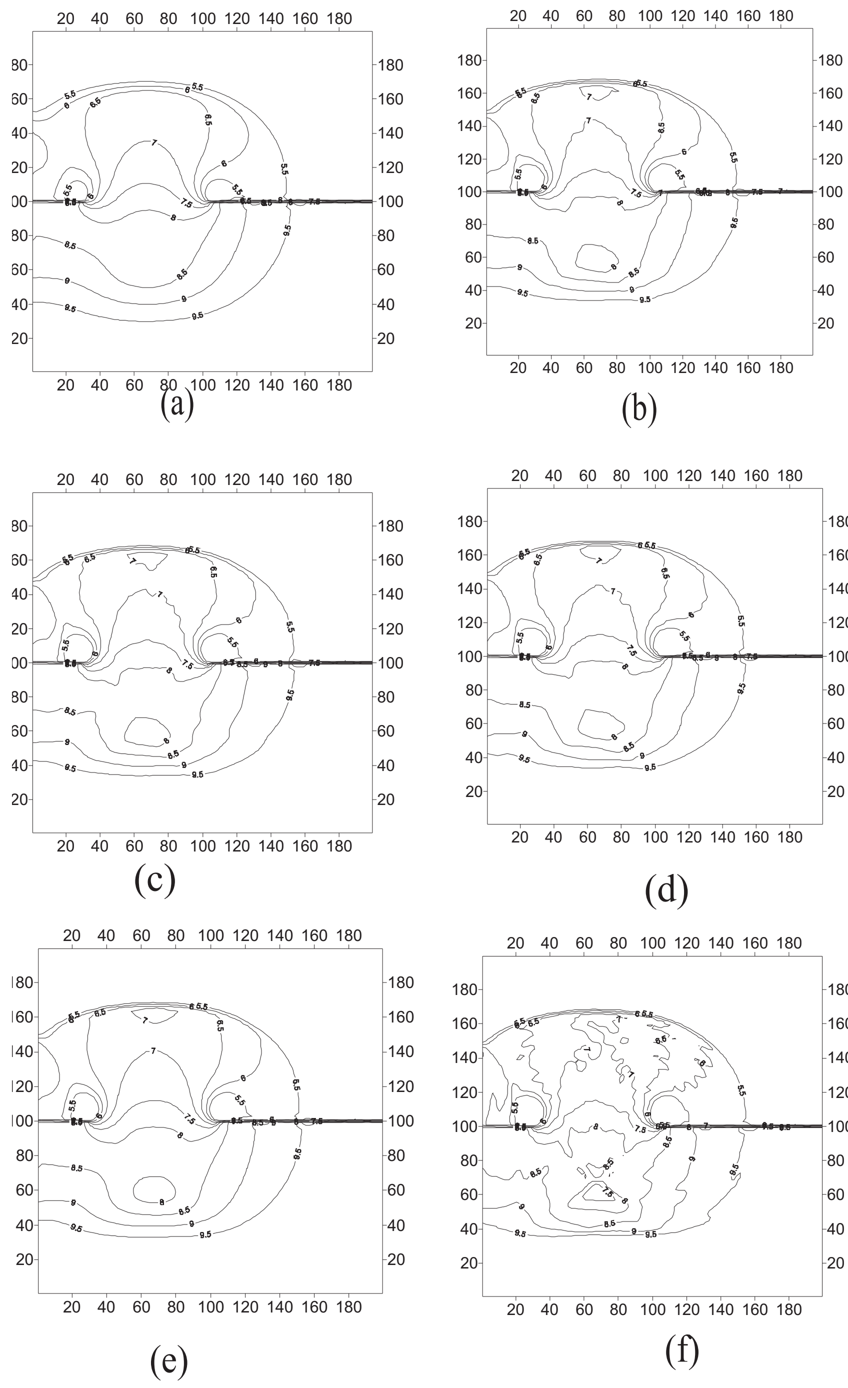


Figure 8.24: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using HLL approximate Riemann solver. The above figure shows depth contour plots for the dam break situation. Limiters used with HLL solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 7.2s after dam break.

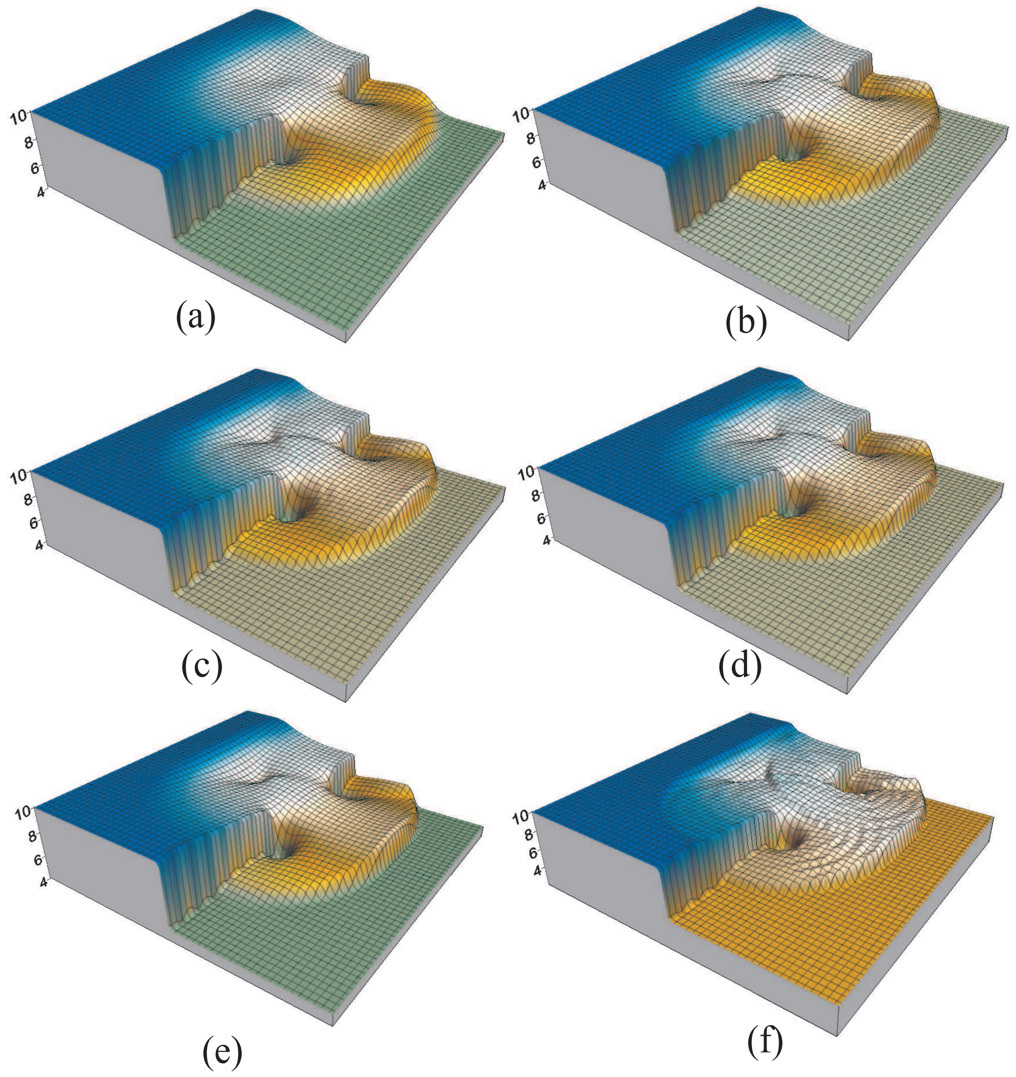


Figure 8.25: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using HLL approximate Riemann solver. The above figure shows 3D perspective view of the dam break situation. Limiters used with HLL solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 7.2s after dam break.

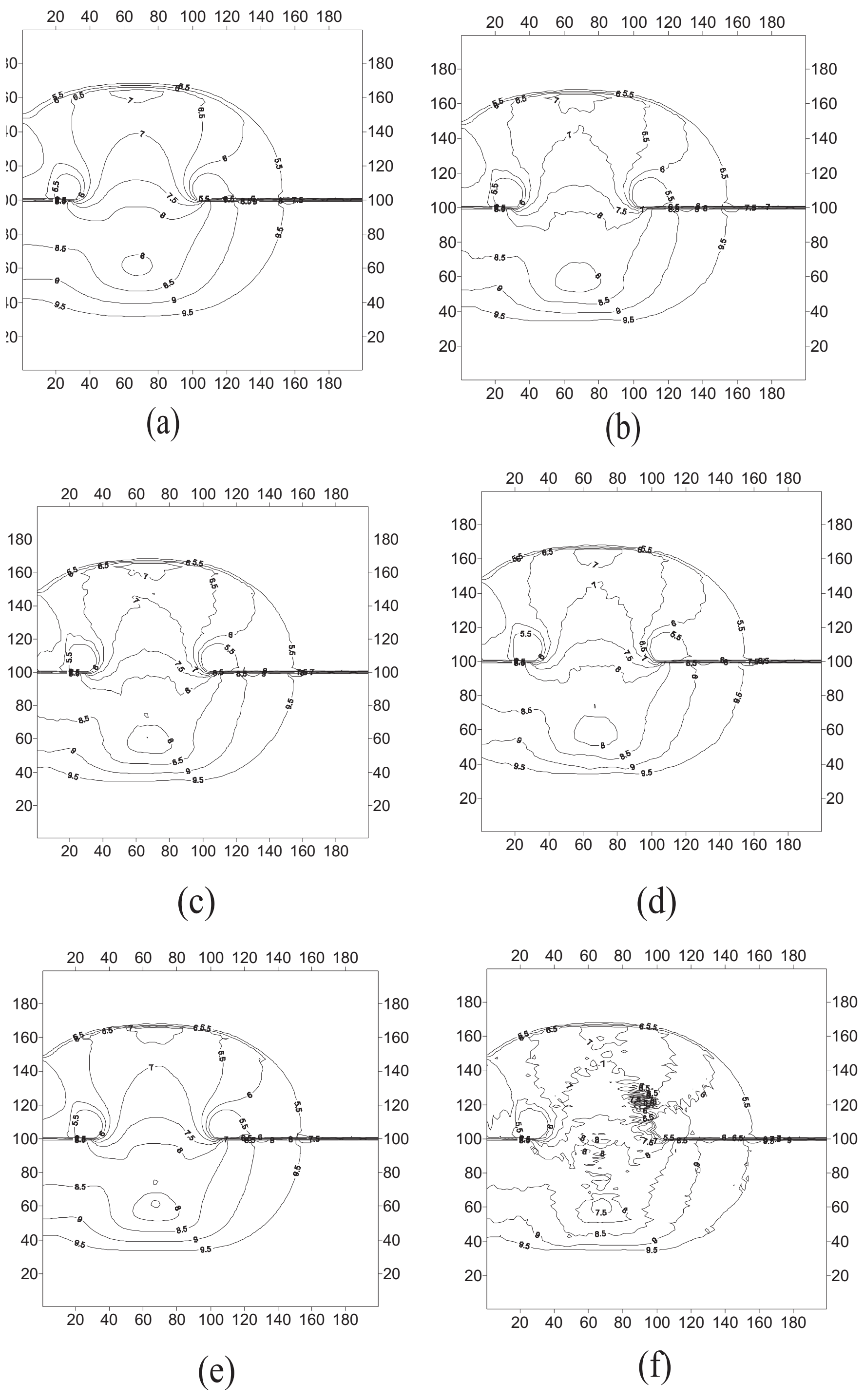


Figure 8.26: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using Shu & Osher's approximate Riemann solver. The above figure shows depth contour plots for the dam break situation. Limiters used with Shu & Osher's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 7.2s after dam break.

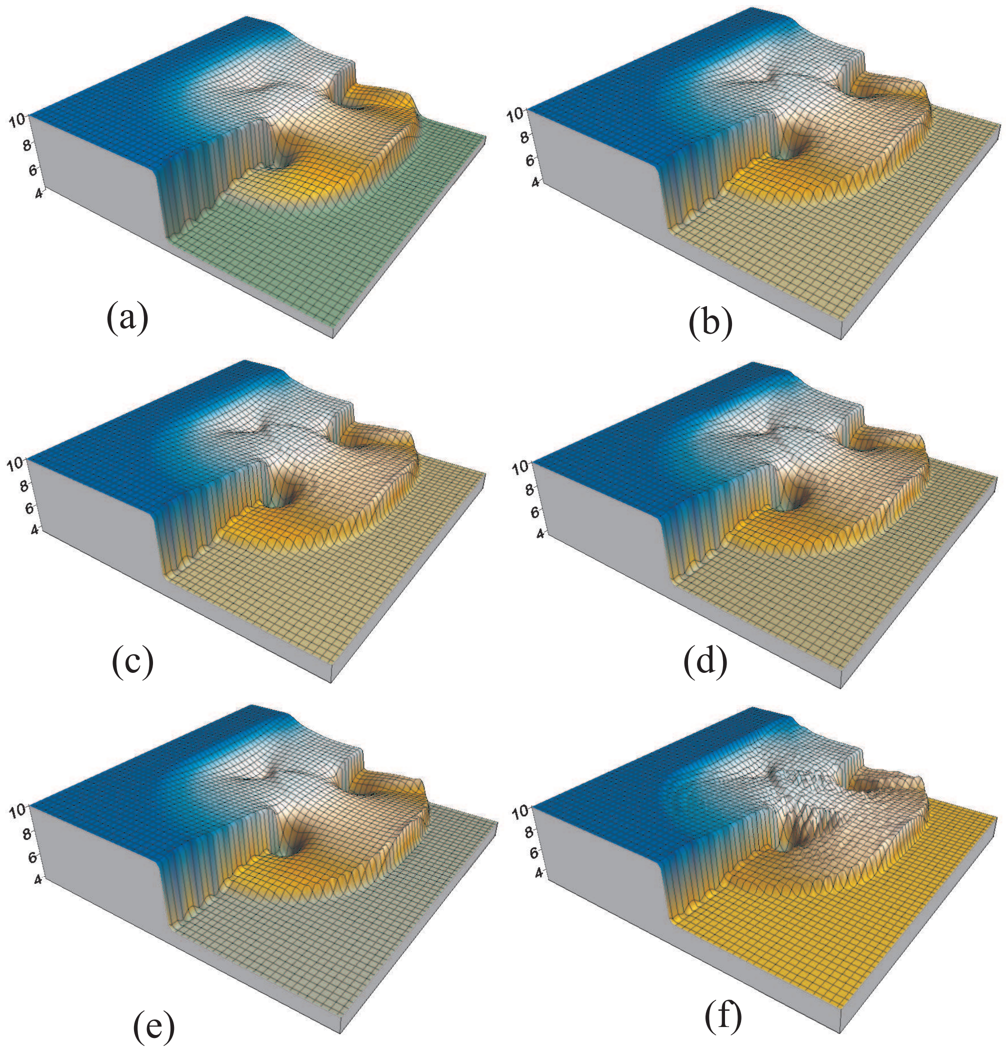
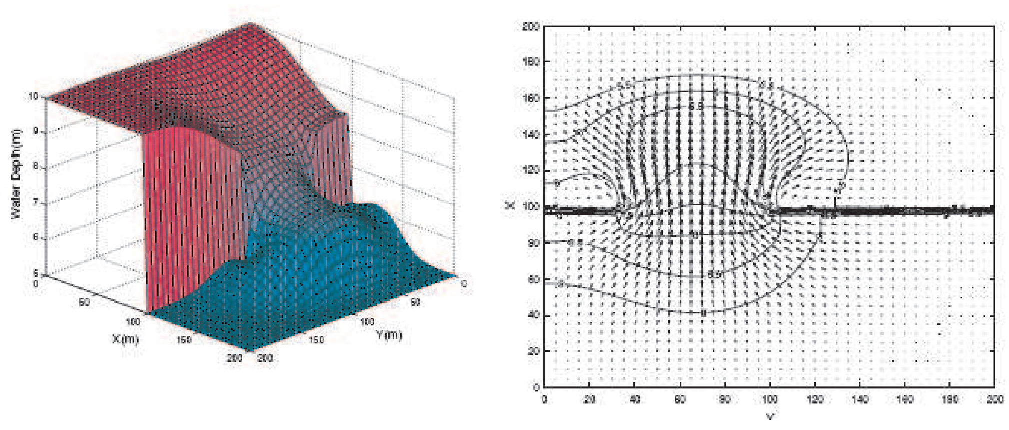
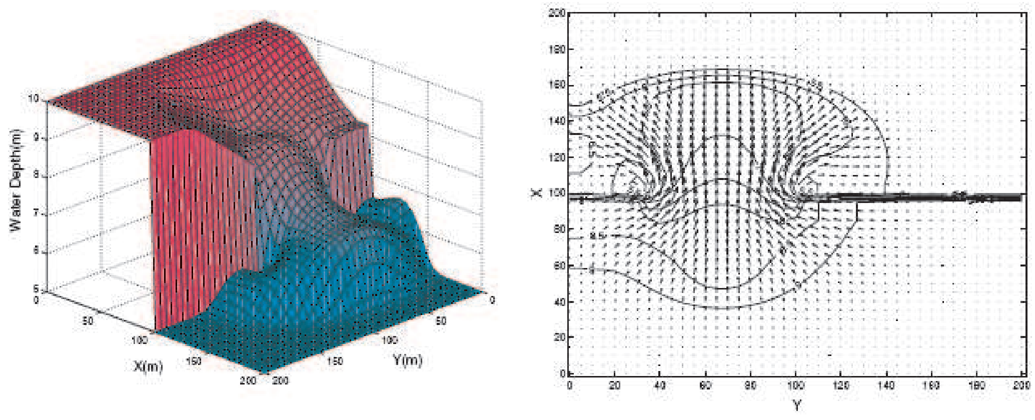


Figure 8.27: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using Shu & Osher's approximate Riemann solver. The above figure shows 3D perspective view of the dam break situation. Limiters used with Shu & Osher's solver are: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (e) Extended VanLeer (f) MLG. Note: Plot at time = 7.2s after dam break.

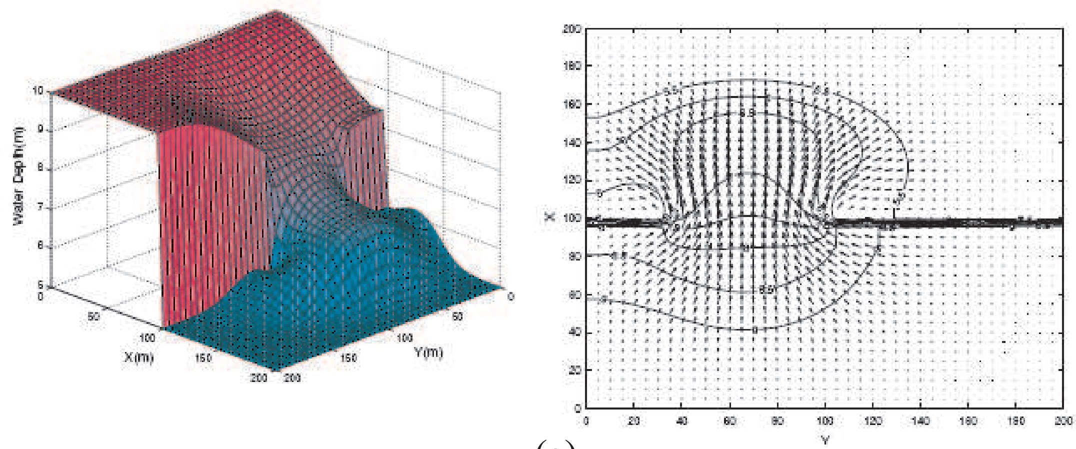


(a)

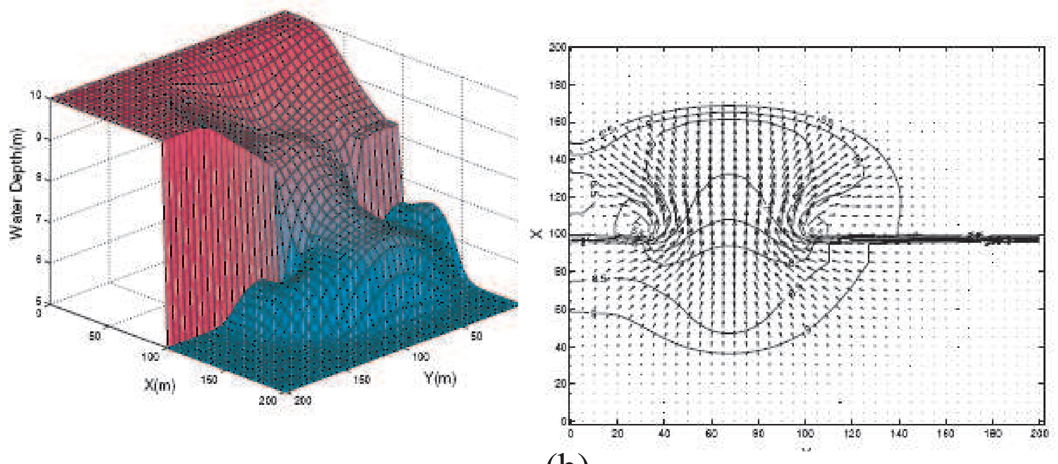


(b)

Figure 8.28: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using relaxation schemes. The above figure shows 3D perspective view of the dam break situation and depth contour plots for the dam break situation. (a)the upwind relaxation scheme (b) the MUSCL relaxation scheme (Minmod limiter) Note: Plot at time = 7.2s after dam break. Results Published: Delis and Katsaounis (2005)



(a)



(b)

Figure 8.29: Two-dimensional partial dam break simulation (Mesh configuration: Figure 8.21) using relaxation schemes. The above figure shows 3D perspective view of the dam break situation and depth contour plots for the dam break situation. (a)the upwind relaxation scheme (Superbee limiter) (b) the MUSCL relaxation scheme (VanLeer limiter) Note: Plot at time = 7.2s after dam break. Results Published: Delis and Katsaounis (2005)

8.5.3 Observations

This test is an investigation of the capability of the approximate Riemann solvers with limiters to capture the effect of full two-dimensional wave development.

It can be clearly seen that for all the combination of approximate Riemann solvers and limiters, gives very good results and agrees very well with those in Fennema and Chaudhry (1990), Anastasiou and Chan (1997) and Delis and Katsaounis (2005). However, few comments can be made on the presented results:

1. The extended-VanLeer can be seen giving very smooth contours, compared to other limiters, for all the solvers.

2. The MLG limiter gives most distorted result, of all other limiters, for all the solvers.

3. It can be concluded that all the combination of approximate solver and limiter gives sharp gradient at the shock fronts. It is interesting to note that the results presented here with limiters (No Limiter, Minmod, Superbee and extended-VanLeer) with Roes', HLL and SO approximate solver handles shock more effectively compared to same limiters used for relaxation schemes (see Figure 8.28 and 8.29).

8.6 Rotation Test for Shallow Water Equations

This test case involves a rotating cubical-shaped scalar field in evaluation of the numerical accuracy of various high-order limiters. The scaler field can be viewed as solute, whose properties doesn't change with water contact. The mathematical expression eigen structure for the solutes is given in section 6.5.0.1.

8.6.1 Aim of the test

The aim of this test is to:

1. study the numerical accuracy of various limiters for the advection of a scalar field;

2. study the effect of mesh size on the behaviour of limiters; and

3. present the particulars for developing and undertaking the test with *Riemann2D* and discuss the associated results.

8.6.2 Test Configuration

In this example a scalar cubical-shaped (2.5m×2.5m×1m) field is advected around by a constant rotating velocity field in a 10m×10m square computational domain. The angular frequency (ω) of the velocity field is set to be 0.1rotation/s and the center of the cube is set to be 5m from the center of the computations domain. The cube has a square top face of 2.5m×2.5m. The definition of the problem domain and the initial condition of the scalar field are shown in Figure 8.30. For this test case three level of numerical mesh is used. The computational domain was triangulated by 512, 2048 and 8192 elements (see Figure 8.31). Each level contains four times the elements in the previous level.

Note:

- Initial Condition: Solute concentration 1 (see Figure 8.30;
- Boundary Condition: Reflective type on all exterior boundaries.
- Riemann Solver and Limiters: Roe's approximate solver with different limiters as shown in Figures 8.32 to 8.43;
- Courant Number: Desired - 0.25, Maximum - 0.5;
- Time step: Variable;
- Order in Time: First;
- The results shown in the Figures 8.32 to 8.43 are only for one complete rotation of the scaler field. The time taken for this is: $2\pi/$ 0.1=62.83185s. This is numerical time and therefore presented to fifth decimal place.
- The mesh refinement is done on the initial numerical mesh of 512 elements (see Figure 8.31 (a)) by using "automatic mesh refine tool" of ArgusMeshMaker.
- The mesh shown in 3Dperspective view (see Figures 8.33, 8.37 and 8.41) are only for better visual effect. They are not numerical mesh for the problem.
- Figures 8.34, 8.38 and 8.42 are shown for comparision, therefore it doesn't represent any physical quantity on $x-$ scale.

- Other details for each figures are given in their respective captions.

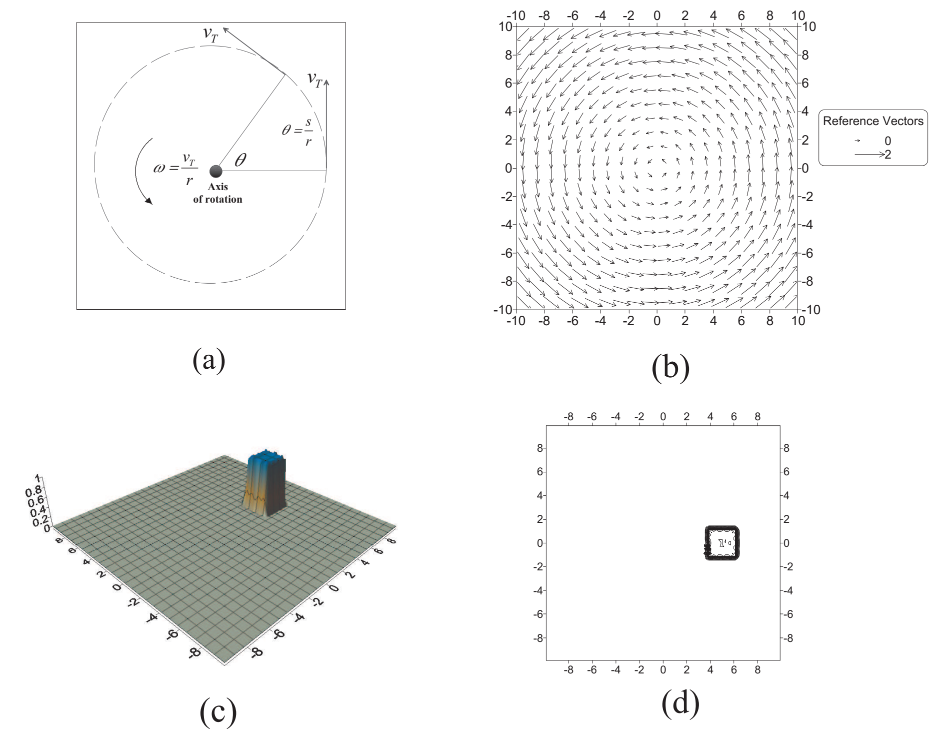


Figure 8.30: Rotation test configuration: (a) Diagram showing the angular velocity applied at a point in the domain, (b) Velocity vector constantly applied in the domain, (c) 3D perspective view of the initial condition of the rotation test, and (d) Initial condition of the concentration contour

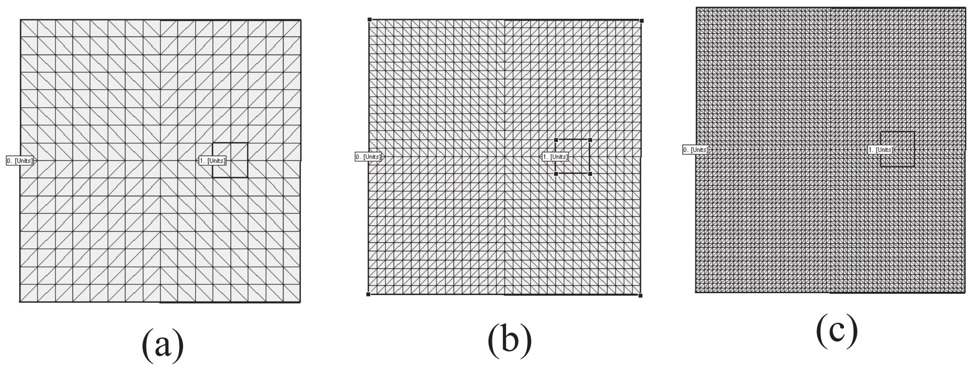


Figure 8.31: Rotation test mesh configuration: Test is conducted on three level of mesh for same domain and initial condition (a) 512 elements (b) 2048 elements (c) 8192 elements

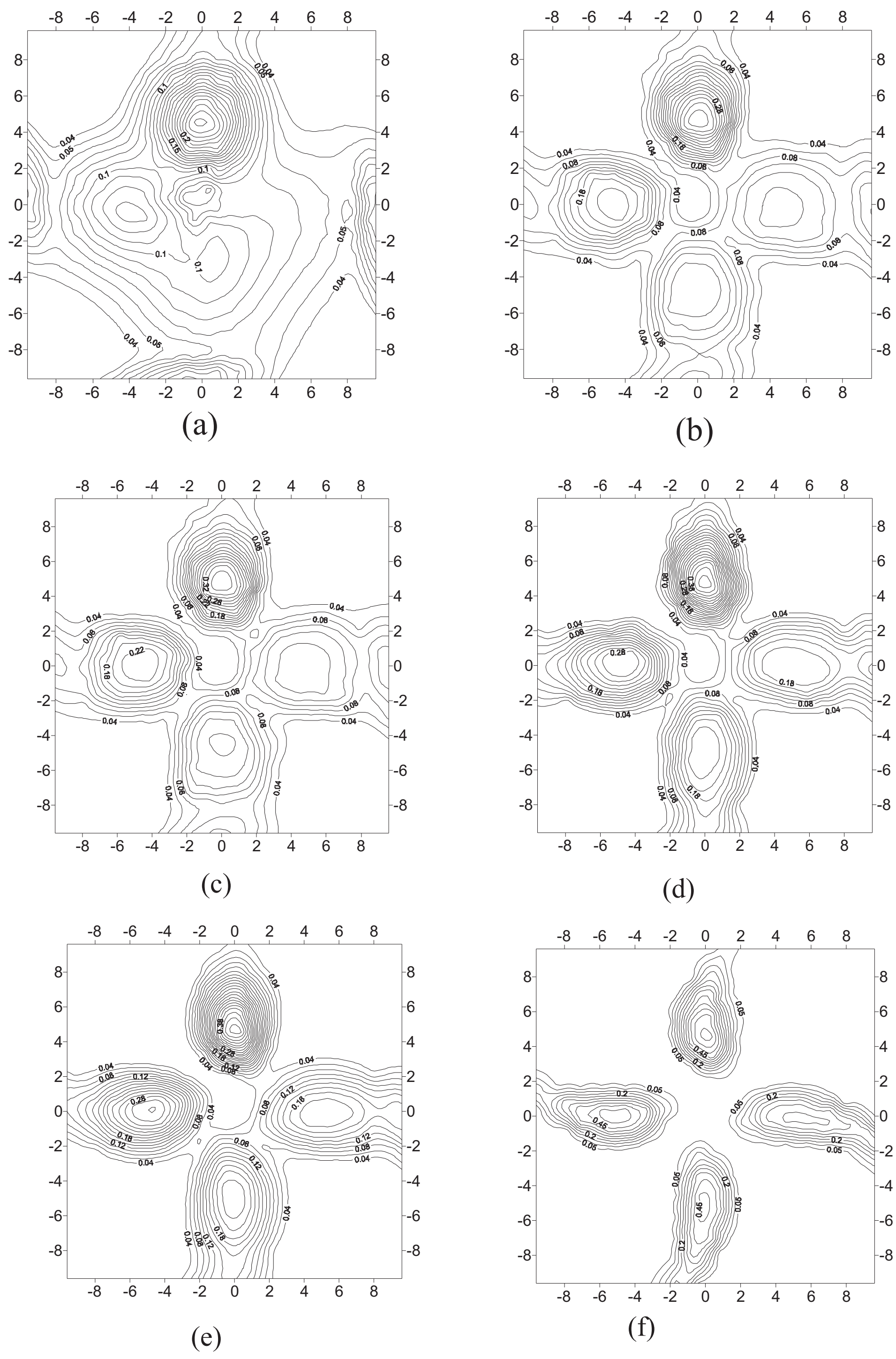


Figure 8.32: Rotation test: Concentration contours diagram for domain with 512 elements mesh (Figure 8.31). Each figure above shows the concentration contours at quarter rotation in anti-clockwise direction. Corresponding time of travel are 15.70796 s (12'O clock), 31.41593 s (9'O clock), 47.12389 s (6'O clock) and 62.83185 s (3'O clock). Limiters used for rotation test: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (d) Extended VanLeer (e) MLG.

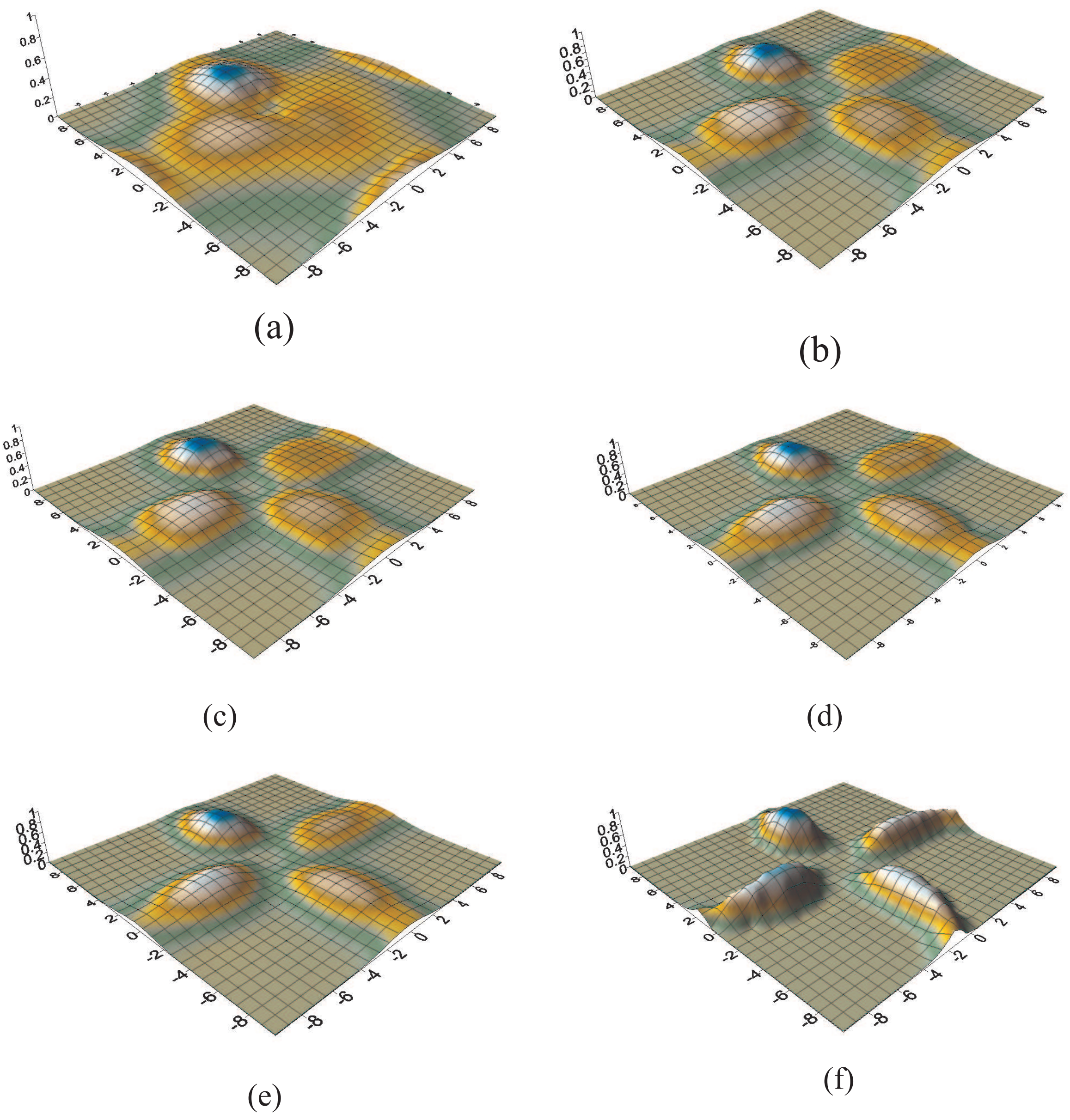


Figure 8.33: Rotation test: 3D perspective view for domain with 512 elements mesh (Figure 8.31). Each figure above shows the concentration surface at quarter rotation in anti-clockwise direction. Corresponding time of travel are 15.70796 s (12'O clock), 31.41593 s (9'O clock), 47.12389 s (6'O clock) and 62.83185 s (3'O clock). Limiters used for rotation test: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (d) Extended VanLeer (e) MLG.

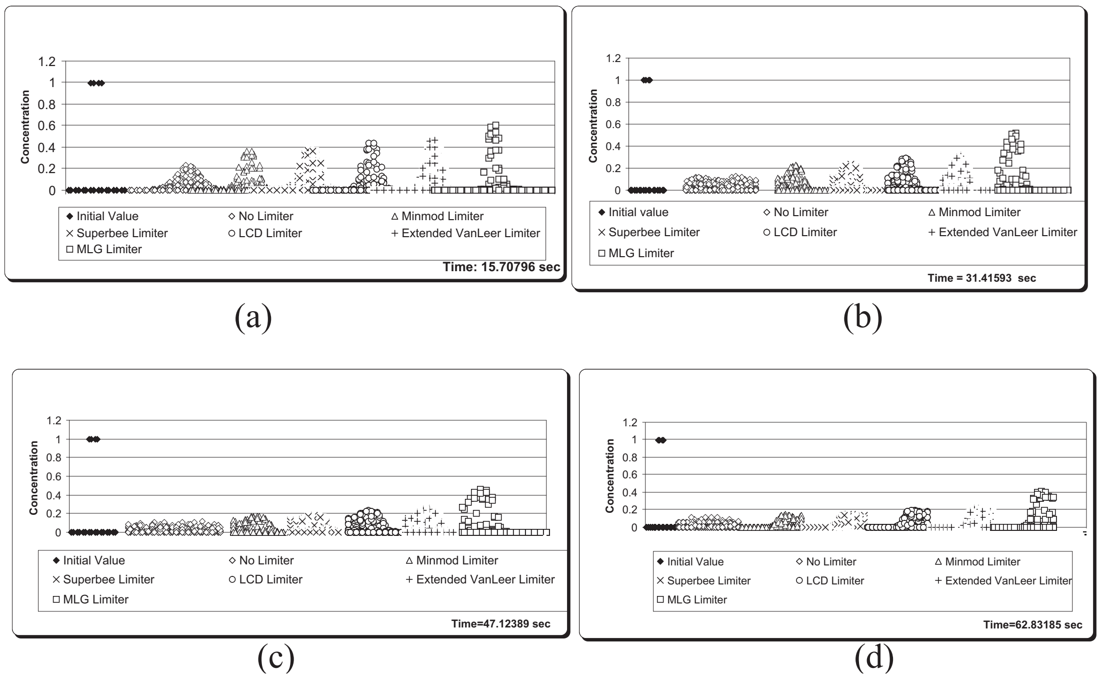


Figure 8.34: Rotation test: Concentration depth comparison diagram for domain with 512 elements mesh (Figure 8.31). Each figure above shows the comparison of the concentration depth at quarter rotation in anti-clockwise direction. Corresponding time of travel are (a) 15.70796 s (12'O clock), (b) 31.41593 s (9'O clock), (c) 47.12389 s (6'O clock) and (d) 62.83185 s (3'O clock). Limiters used for rotation tests are No Limiter, Minmod, Superbee, LCD, Extended VanLeer and MLG.

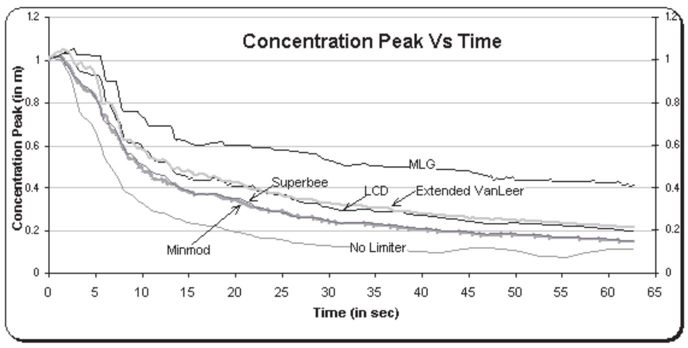


Figure 8.35: Rotation test: Time series comparison diagram for domain with 512 elements mesh (Figure 8.31). Limiters used for rotation tests are No Limiter, Minmod, Superbee, LCD, Extended VanLeer and MLG.

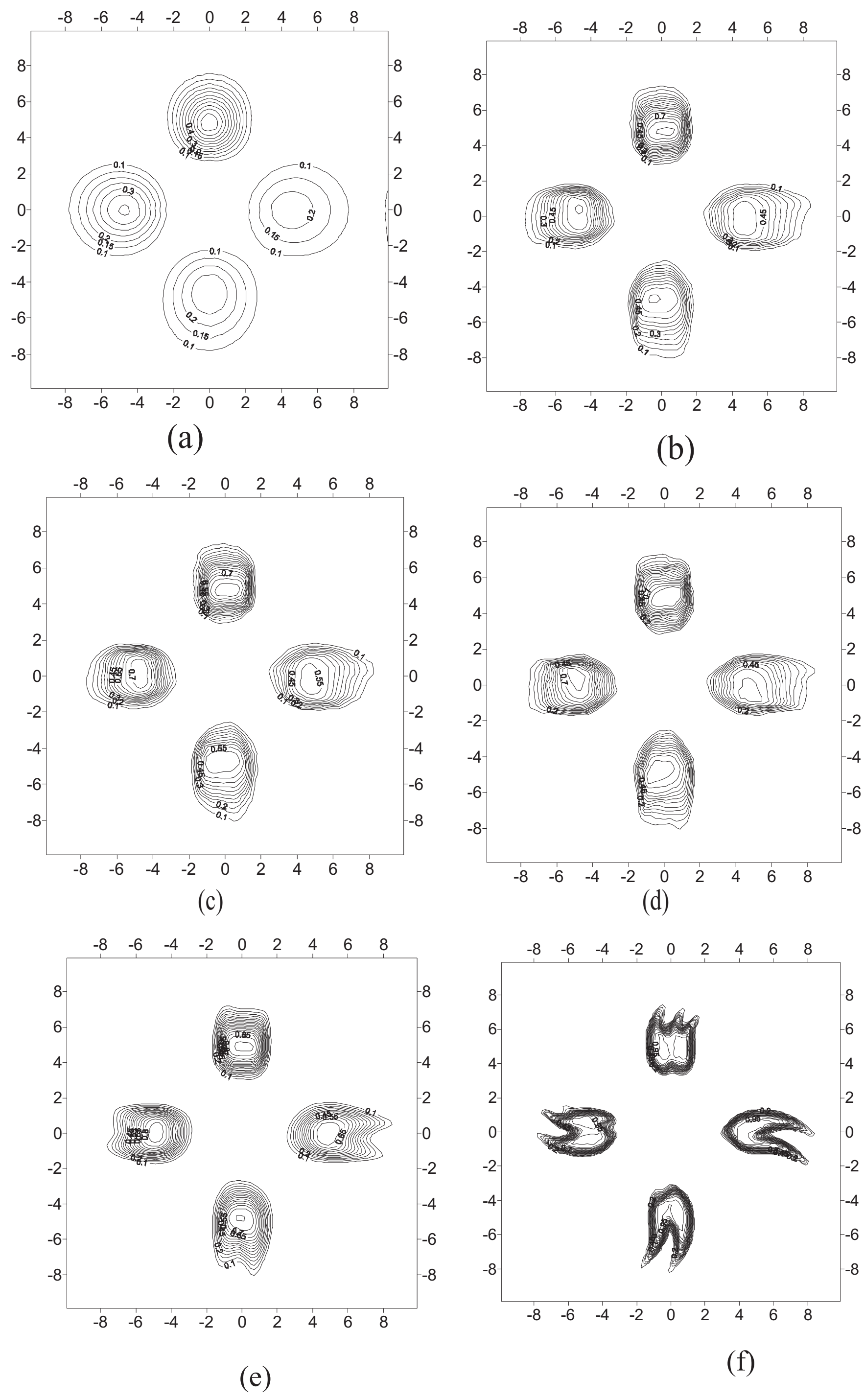


Figure 8.36: Rotation test: Concentration contours diagram for domain with 2048 elements mesh (Figure 8.31). Each figure above shows the concentration contours at quarter rotation in anti-clockwise direction. Corresponding time of travel are 15.70796 s (12'O clock), 31.41593 s (9'O clock), 47.12389 s (6'O clock) and 62.83185 s (3'O clock). Limiters used for rotation test: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (d) Extended VanLeer (e) MLG.

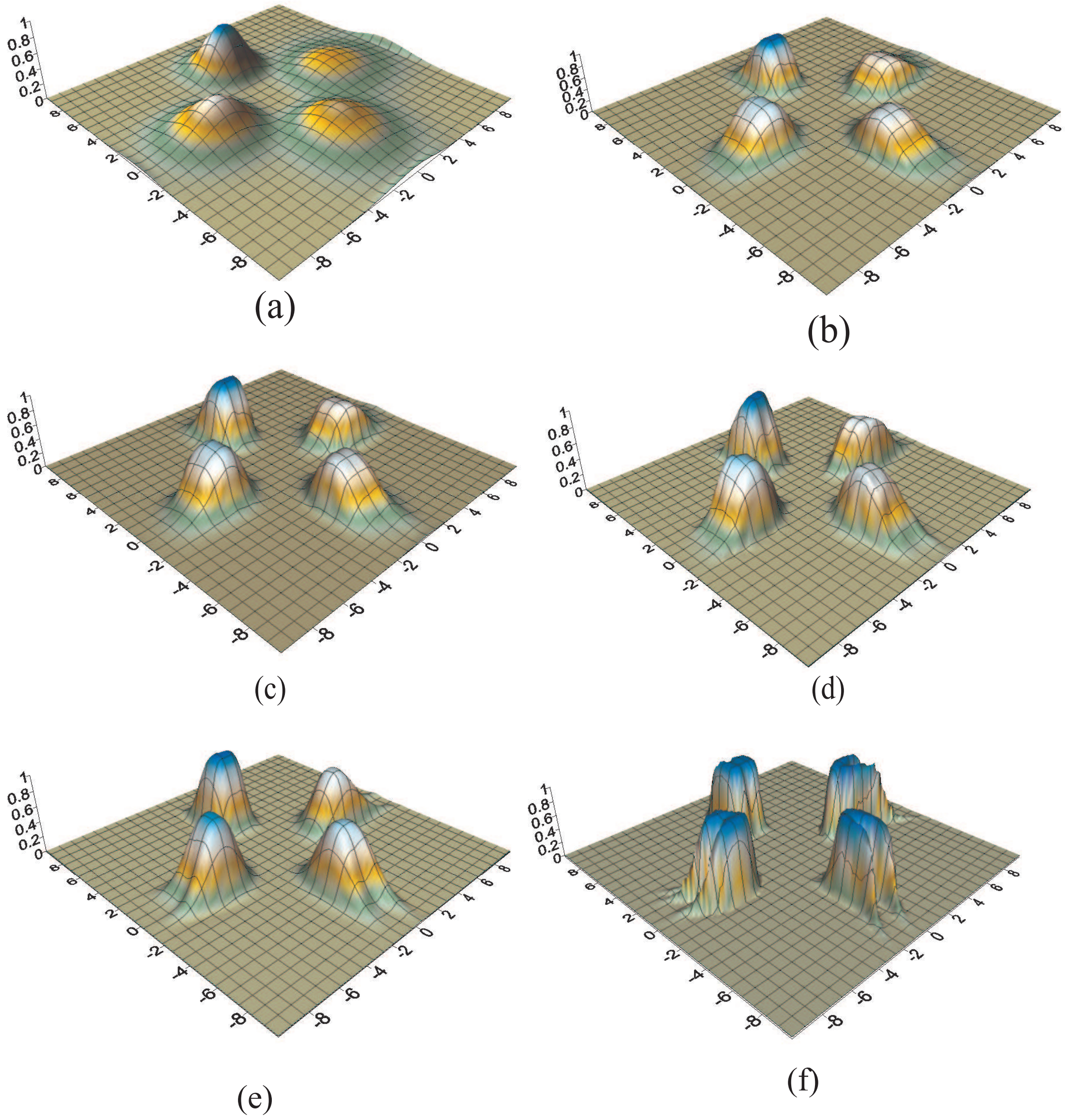
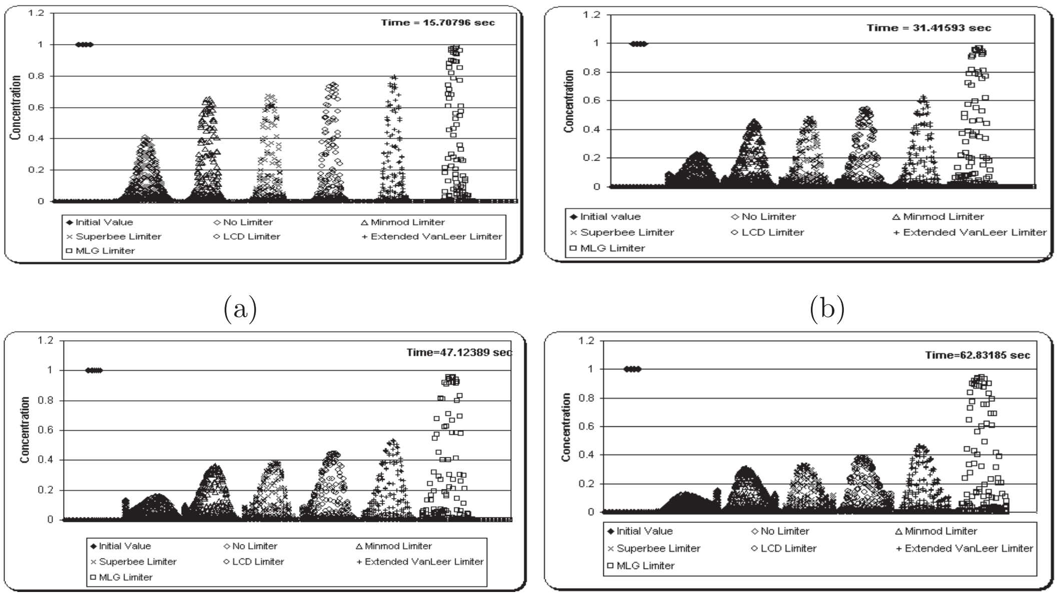


Figure 8.37: Rotation test: 3D perspective view for domain with 2048 elements mesh (Figure 8.31). Each figure above shows the concentration surface at quarter rotation in anti-clockwise direction. Corresponding time of travel are 15.70796 s (12'O clock), 31.41593 s (9'O clock), 47.12389 s (6'O clock) and 62.83185 s (3'O clock). Limiters used for rotation test: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (d) Extended VanLeer (e) MLG.



(a) (b)

(c) (d)

Figure 8.38: Rotation test: Concentration depth comparison diagram for domain with 2048 elements mesh (Figure 8.31). Each figure above shows the comparison of the concentration depth at quarter rotation in anti-clockwise direction. Corresponding time of travel are (a) 15.70796 s (12'O clock), (b) 31.41593 s (9'O clock), (c) 47.12389 s (6'O clock) and (d) 62.83185 s (3'O clock). Limiters used for rotation tests are No Limiter, Minmod, Superbee, LCD, Extended VanLeer and MLG.

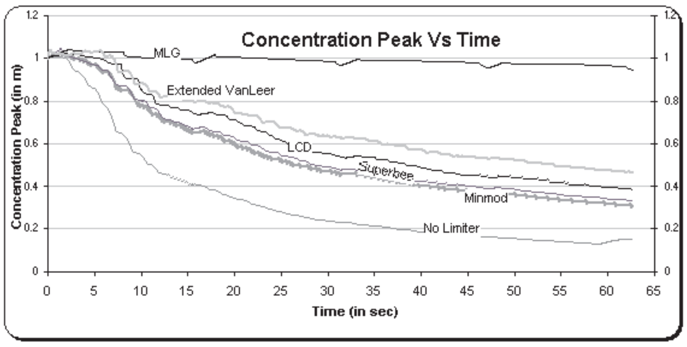


Figure 8.39: Rotation test: Time series comparison diagram for domain with 2048 elements mesh (Figure 8.31). Limiters used for rotation tests are No Limiter, Minmod, Superbee, LCD, Extended VanLeer and MLG.

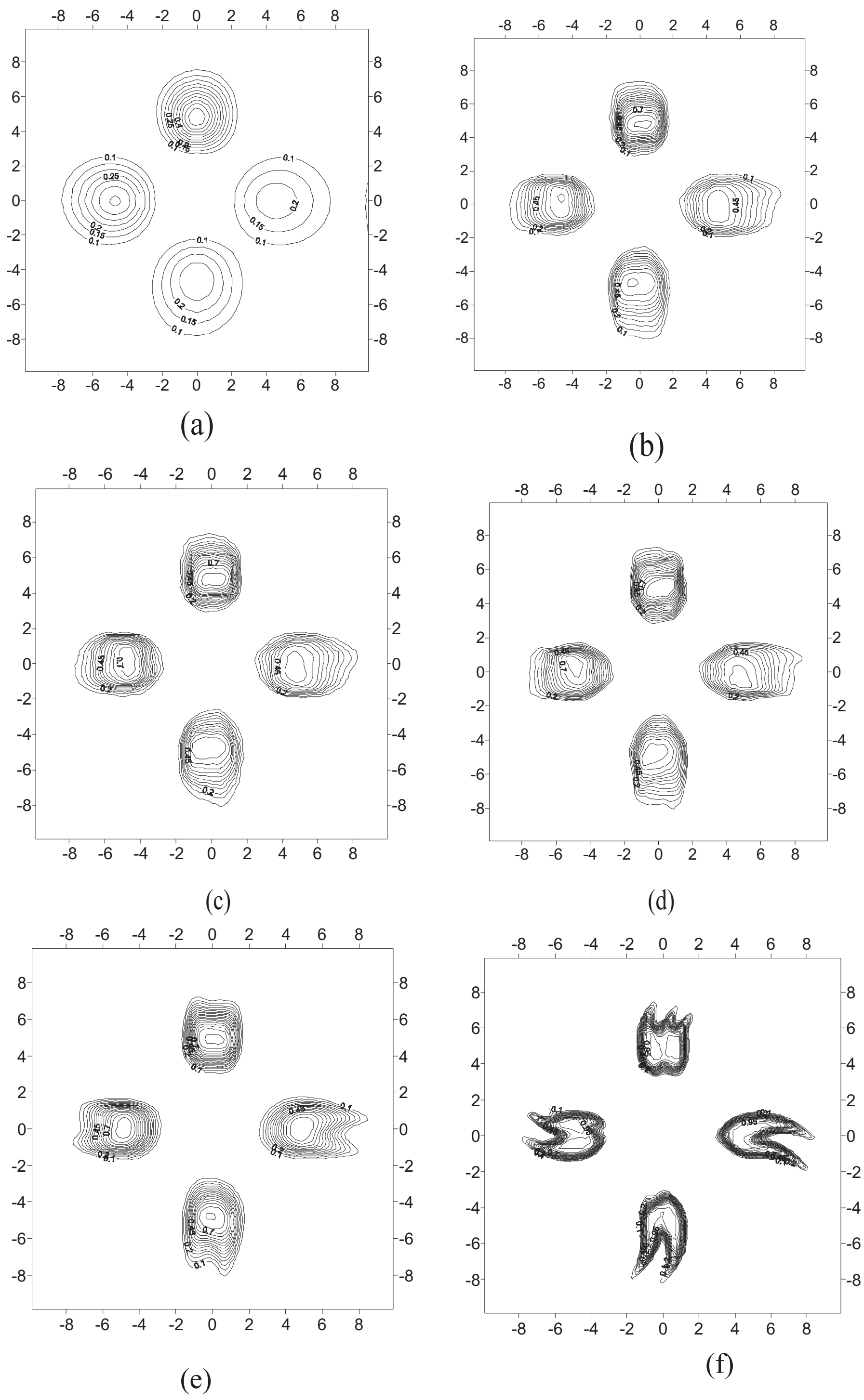


Figure 8.40: Rotation test: Concentration contours diagram for domain with 8192 elements mesh (Figure 8.31). Each figure above shows the concentration contours at quarter rotation in anti-clockwise direction. Corresponding time of travel are 15.70796 s (12'O clock), 31.41593 s (9'O clock), 47.12389 s (6'O clock) and 62.83185 s (3'O clock). Limiters used for rotation test: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (d) Extended VanLeer (e) MLG.

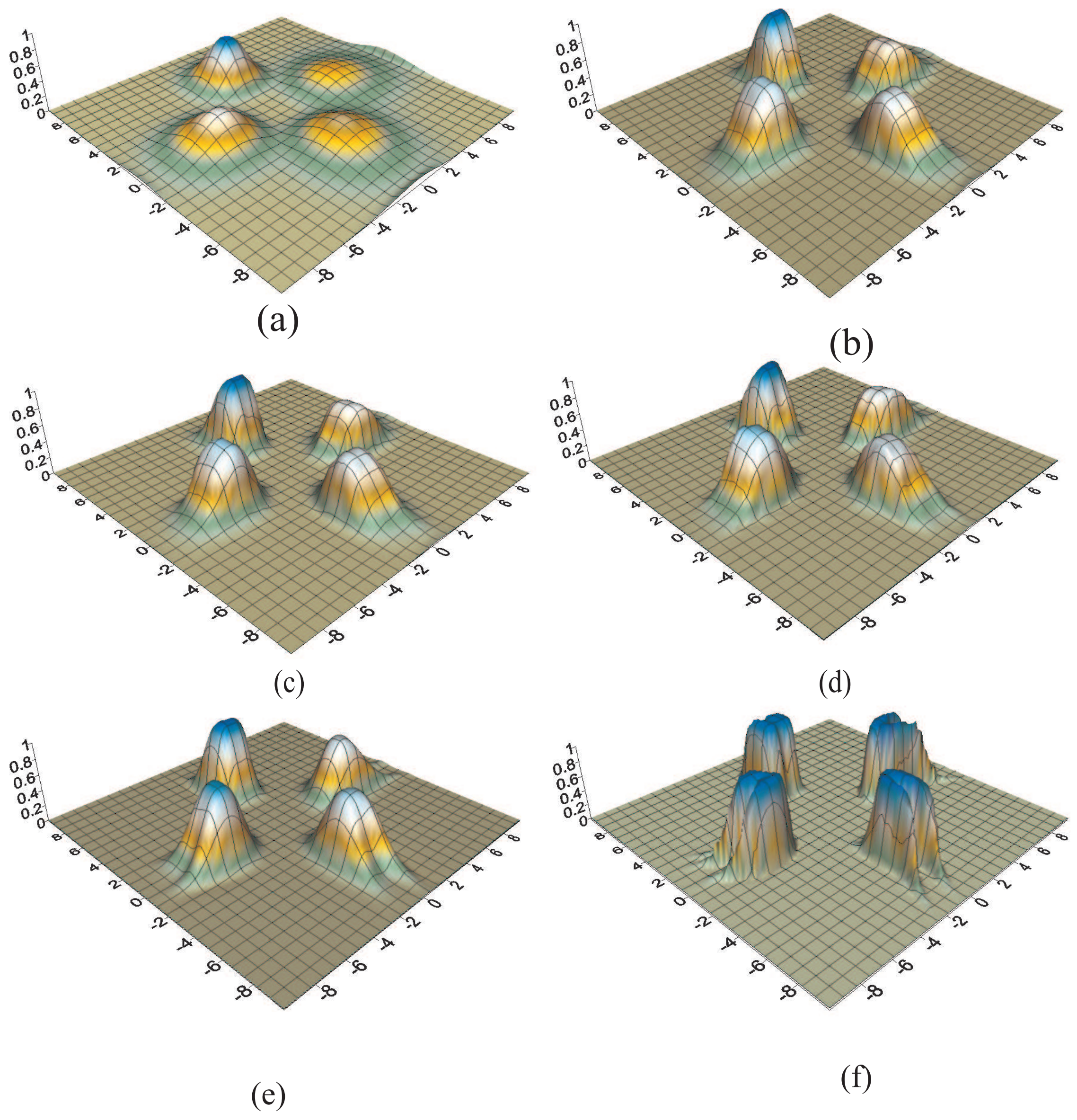


Figure 8.41: Rotation test: 3D perspective view for domain with 8192 elements mesh (Figure 8.31). Each figure above shows the concentration surface at quarter rotation in anti-clockwise direction. Corresponding time of travel are 15.70796 s (12'O clock), 31.41593 s (9'O clock), 47.12389 s (6'O clock) and 62.83185 s (3'O clock). Limiters used for rotation test: (a) No Limiter (b) Minmod (c) Superbee (d) LCD (d) Extended VanLeer (e) MLG.

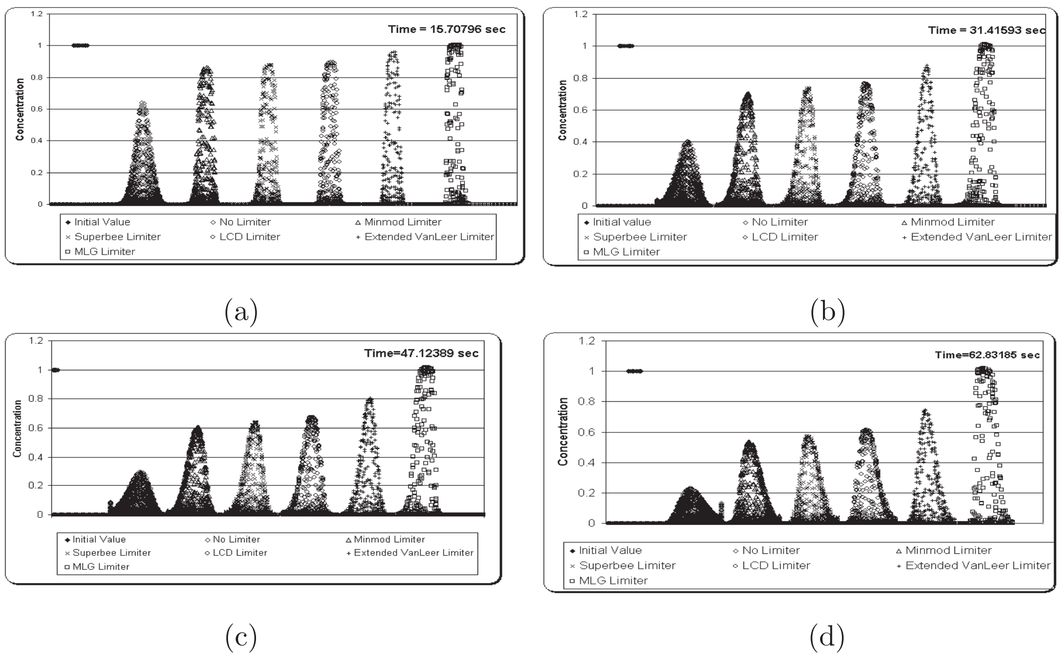


Figure 8.42: Rotation test: Concentration depth comparison diagram for domain with 8192 elements mesh (Figure 8.31). Each figure above shows the comparison of the concentration depth at quarter rotation in anti-clockwise direction. Corresponding time of travel are (a) 15.70796 s (12'O clock), (b) 31.41593 s (9'O clock), (c) 47.12389 s (6'O clock) and (d) 62.83185 s (3'O clock). Limiters used for rotation tests are No Limiter, Minmod, Superbee, LCD, Extended VanLeer and MLG.

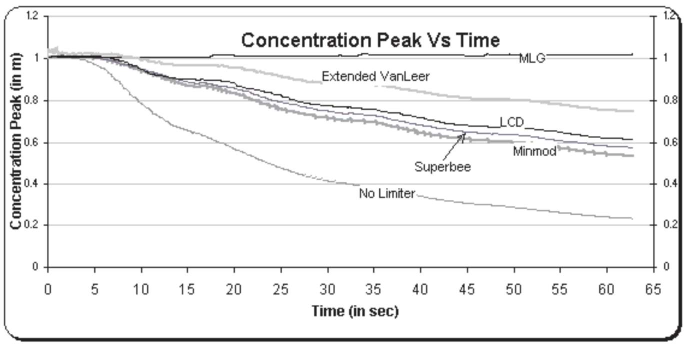


Figure 8.43: Rotation test: Time series comparison diagram for domain with 8192 elements mesh (Figure 8.31). Limiters used for rotation tests are No Limiter, Minmod, Superbee, LCD, Extended VanLeer and MLG.

8.6.3 Observations

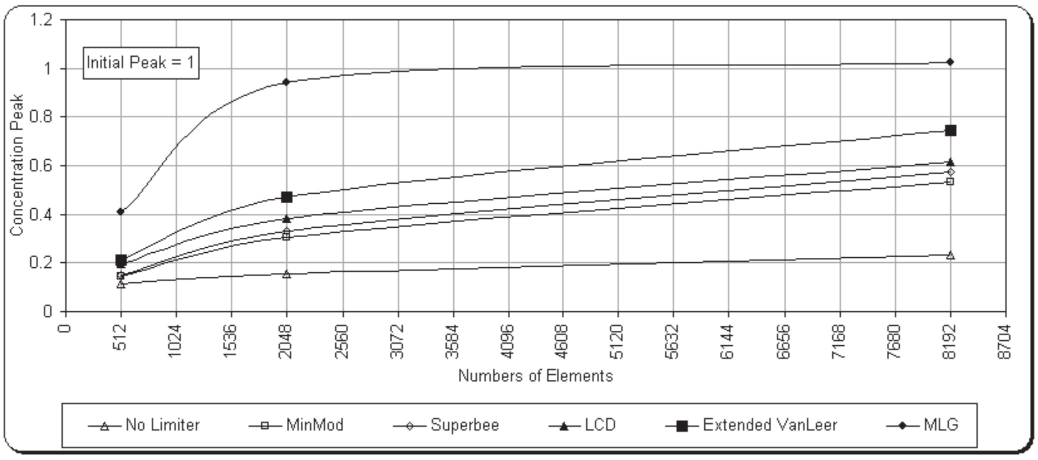
This test is an investigation of the capability of the limiters for handling numerical diffusion at different level of mesh refinement.

This test case give very interesting results for the comparison of the limiters. In this test case each limiter should (ideally) bring the scalar field in same shape (Cube: 2.5m×2.5m×1m) after one full complete rotation. Because of numerical diffusion the peak of the cube falls and the surface area is distorted. It should be noted that total volume of the scaler quantity is same through out the domain. Before discussion on the results the rank (discussed in section 5.9.2) of the limiters should be noted:

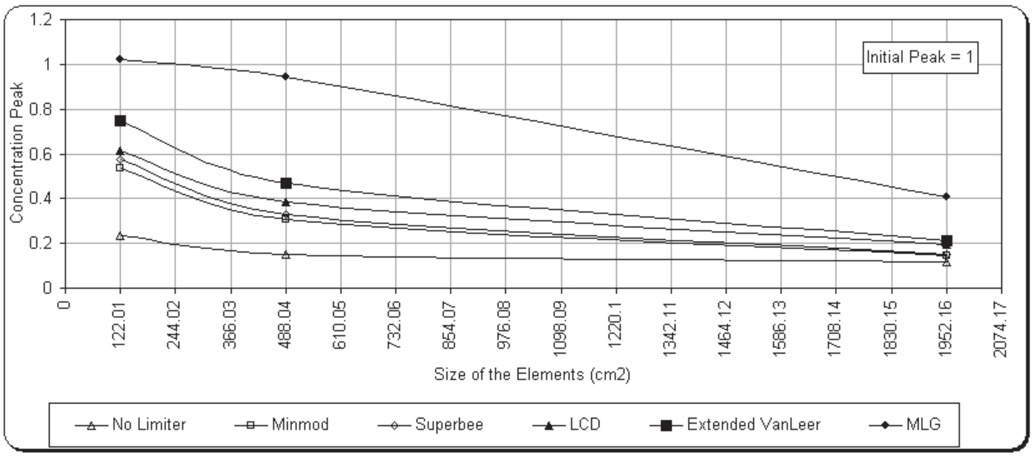
- No Limiter is rank 0;
- Minmod, superbee and LCD are rank 1;
- Extended-VanLeer is rank 3; and
- MLG is rank 4.

Based on the Figures 8.32 to 8.43 following comments can be made:

- Mesh with 512 elements (see Figures 8.31 (a), 8.32,8.33, 8.34 and 8.35):
  - Because of the coarse mesh, the limiters give very poor results for the rotation test.
  - This shows that the numerical diffusion is more, as expected, for lower density mesh.
  - However, the performance of MLG limiter outstands among others. Extended-VanLeer and LCD gives the similar time-series curve. Whereas, superbee and minmod time-series matches with each other.
- Mesh with 2048 elements (see Figures 8.31 (b), 8.36,8.37, 8.38 and 8.39):
  - Because the mesh is refined to four times the previous case, the limiters' performance has improved.
  - This shows that the numerical diffusion decreases, as the mesh is refined.



(a)



(b)

Figure 8.44: Comparison of limiters for concentration of peak after one full rotation (a) *Vs* number of elements in the mesh (b) *Vs* size of the elements (cm^2) [assuming that the element size in same in each mesh, see Figure 8.31 (a), (b) and (c)].

Note: Initial peak is 1 unit. The information is available only for mesh size 512, 2048 and 8192. Therefore, the line joining three point is a smooth curve passing through these points and they do not represent true value of peak for corresponding size of mesh (or number of elements).

  - It is interesting to see that performance of MLG has improved to greatest extend and it shows, little fall (5.69%) in the peak. However, distortion in the shape of the contours can be seen.
  - Similarly, other limiters shows sign of improvement over the previous case. But at this level of mesh refinement, extended-VanLeer, which is rank 3, starts showing better performance over the rank 1 limiters. Also superbee is slightly improved than minmod.
  - In general, the performance of limiters shows much improvement over the case of "no limiter".

- Mesh with 8192 elements (see Figures 8.31 (c), 8.40,8.41, 8.42 and 8.43):
  - This is the finest mesh for this test. As expected, all the limiters shows improvement over the previous cases.
  - For this level of mesh refinement a clear demarcation can be seen between the results (Figure 8.43) of the time-series curve.

- Performance of limiters with mesh refinement (see Figure 8.44): This finding is based on only three level of mesh refinements, but the level of refinement is 16 times (=8192/512) the initial size. Therefore it provides enough information in drawing few conclusions:
  - The MLG limiter is found to be less diffusive among all the limiters and at every level of mesh refinement.
  - The MLG limiter produce better results at second level of mesh refinement (2048 elements) compared to all other limiters at third level of refinement (8192 elements).
  - The performance of rank 1 limiter, *i.e.*, minmod, superbee and LCD tends to give results similar to each other with little difference. However, LCD shows better performance over superbee, which in turn shown better performance over minmod.
  - The extended-VanLeer limiter being rank 3 limiter shows the performance between rank 4 (MLG) and rank 1 limiters.

– Assuming that the size of the elements in each mesh of Figure 8.31, Figure 8.44 (b) give rough idea on the relation between size of element *vs* peak (or numerical diffusion).

8.7 Dam Break in a Converging-Diverging Channel

This test case together with the experimental data and test specifications is supplied on the website of Environmental Agency[1]. The test was conducted as a part of project called Concerted Action on Dam-break Modelling (CADAM[2]).

8.7.1 Aim of Test

The aim of this test is to:

1. study the ability of *Riemann2D* in using different Riemann solvers and limiters to replicate the behaviour of a surge wave, caused by the sudden collapse of a large body of water, in a channel with a local constriction and expansion;
2. compare the numerical results against laboratory results obtained in the European Commission's CADAM project; and
3. present the particulars for developing and undertaking the test with *Riemann2D* and discuss the associated results.

8.7.2 The Laboratory Configuration of the CADAM Project

The model comprises of a horizontal 0.5m wide channel of uniform rectangular cross-section. The overall length of the channel was set at 19.30m, with the first 6.10m of the channel at the upstream end specified as the reservoir. A removable sluice gate was built into the channel to retain the water within the reservoir, the gate being removed in approximately 0.2s to simulate the break of the dam. A constriction was located 7.70m downstream of the sluice gate. The constriction was given an overall length of 1.4m, the first and last 0.2m of which were tapered at 45∘ angles to the channel walls.

[1]`http://environment.gov.uk/commondata/105385/w5_105_tr_1_882583.pdf`

[2]CADAM was an EC funded Concerted Action Programme looking at dam-break modelling and application. February 1998 - January 2000. `http://www.hrwallingford.co.uk/projects/CADAM/CADAM/index.html`

The middle width of the constriction of 0.1m therefore remains uniform for 1.0m in length, as illustrated in Figure 8.45.

The initial conditions for the test were set at 0.3m depth of water in the reservoir upstream from the gate, and 0.003m in the channel downstream from the gate. The experimental data for the test was obtained by removing the gate, and measuring the depth and velocity of flow at the benchmarking stations S1 to S4 (see Figure 8.45). S1 is located 1.0m upstream of the dam gate, S2, 6.10m downstream of the dam gate, S3, 8.80m downstream of the dam gate, and S4, 10.50m downstream of the dam gate. The measurements were taken every 0.04s and recorded up to a simulation time of 10.00s.

8.7.3 Test Configuration

The computational domain set same as specified in Figure 8.46. The numerical mesh consists of 1362 nodes and 2308 triangular elements. The boundary conditions at $x = 0$ is considered as reflective, whereas at $x = 19.3$m it is assumed to be transmissive and all other boundaries are considered as reflective. In absence of model for gate, the dam is assumed to fail instantaneously. This is in contrast to the laboratory conditions, where gate is removed in 0.2s. At the instant of breaking of the dam, water is released through the breach, forming a positive wave propagating downstream and a negative wave spreading upstream.

Note:

- Courant Number for all the cases is 0.2; and
- All the simulation results are for first order in time.

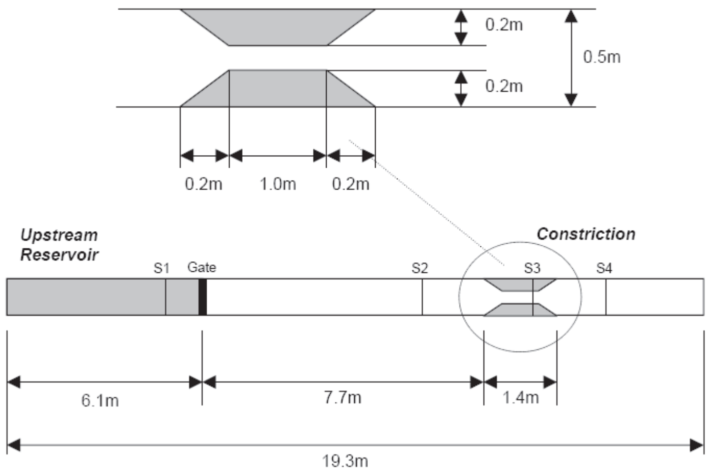


Figure 8.45: Schematic diagram of channel with local constriction. The laboratory model consists of a 19.30 m long rectangular channel with a removable sluice gate at 6.10 m and a constriction at 13.8 m. The constriction is 0.1 m wide and 1.0 m long. Also shown are the gauging stations S1 to S4.

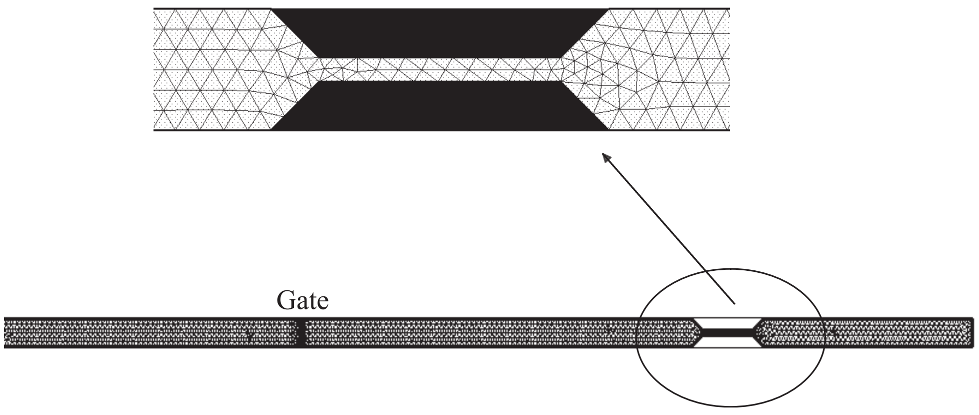
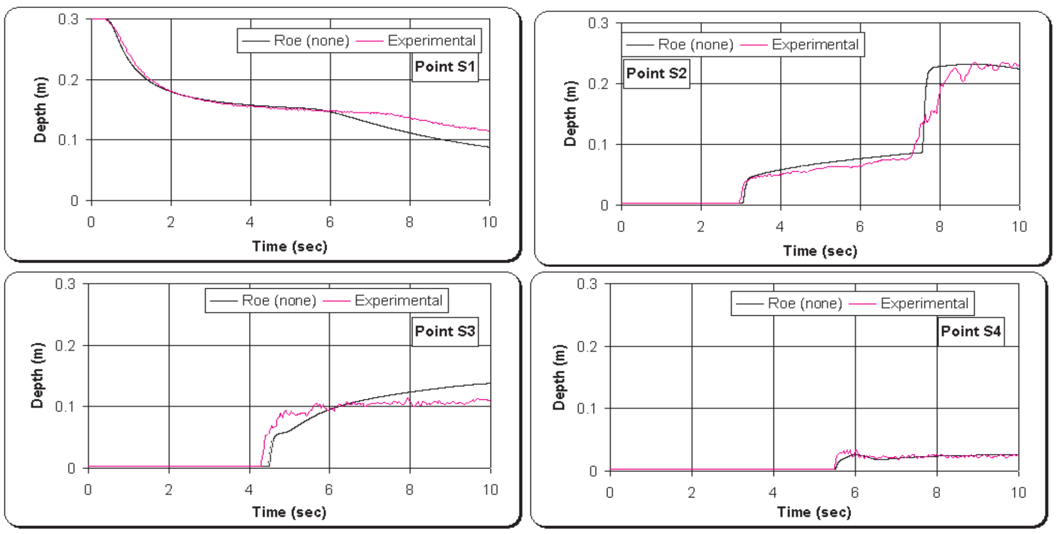
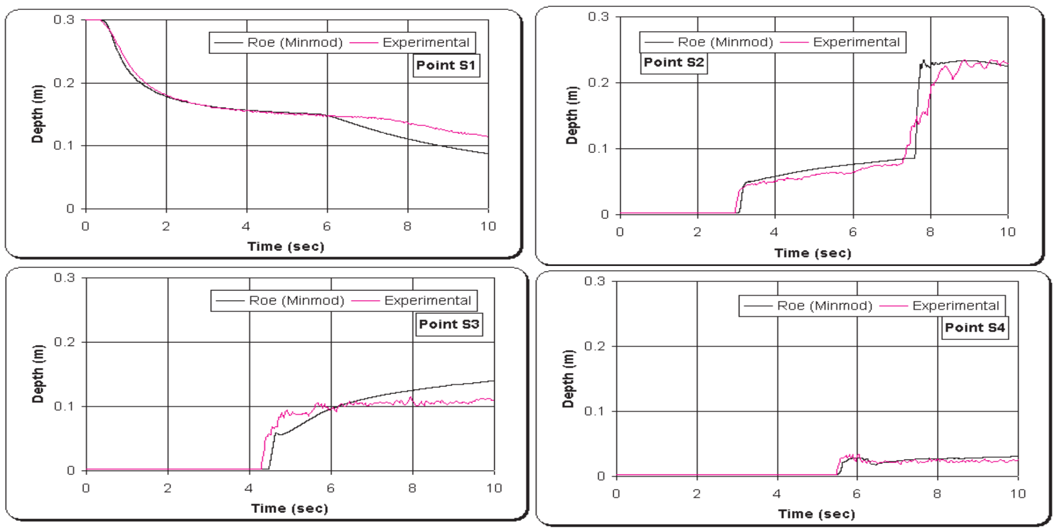


Figure 8.46: Mesh diagram of channel with local constriction. The laboratory model dimensions are given in the Figure 8.45.

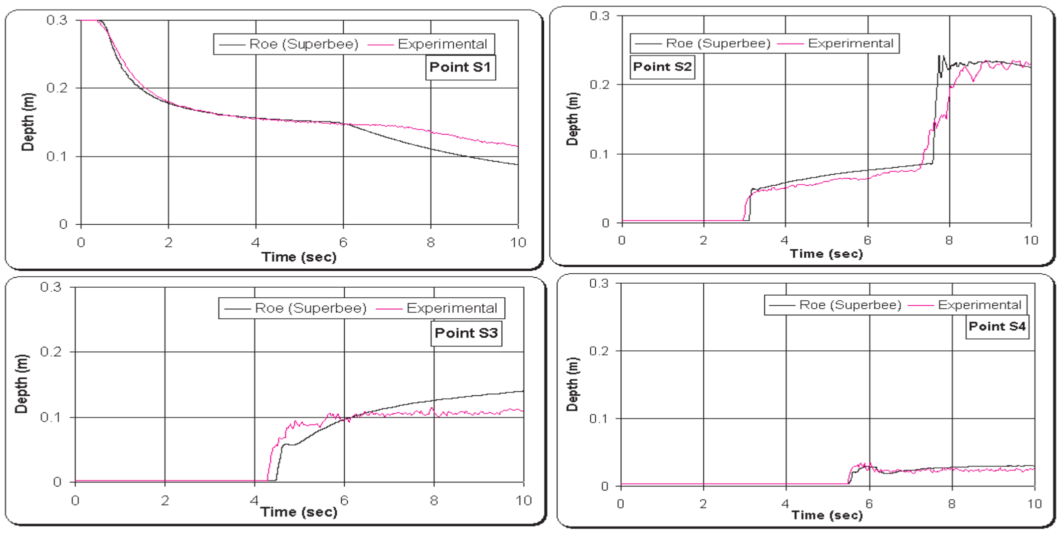


(a)

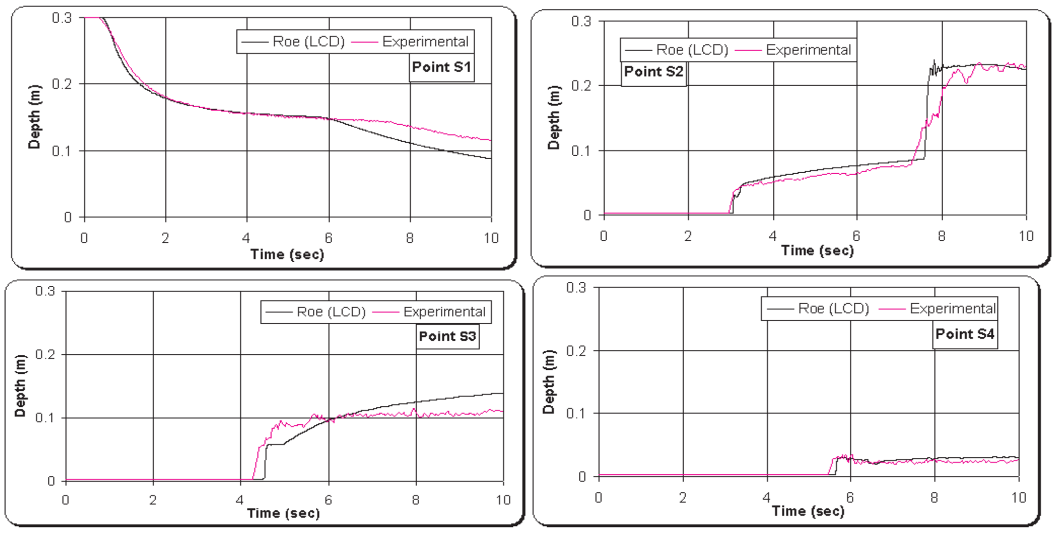


(b)

Figure 8.47: Comparison between experimental data and numerical results using Roe's approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Limiters used with Roe's solver: (a) No Limiter (b) Minmod.

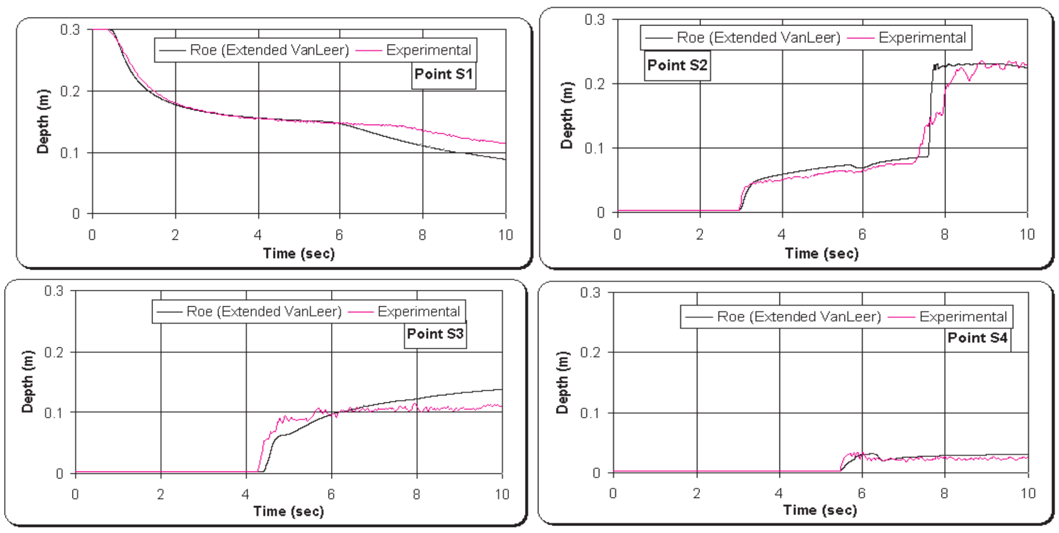


(a)

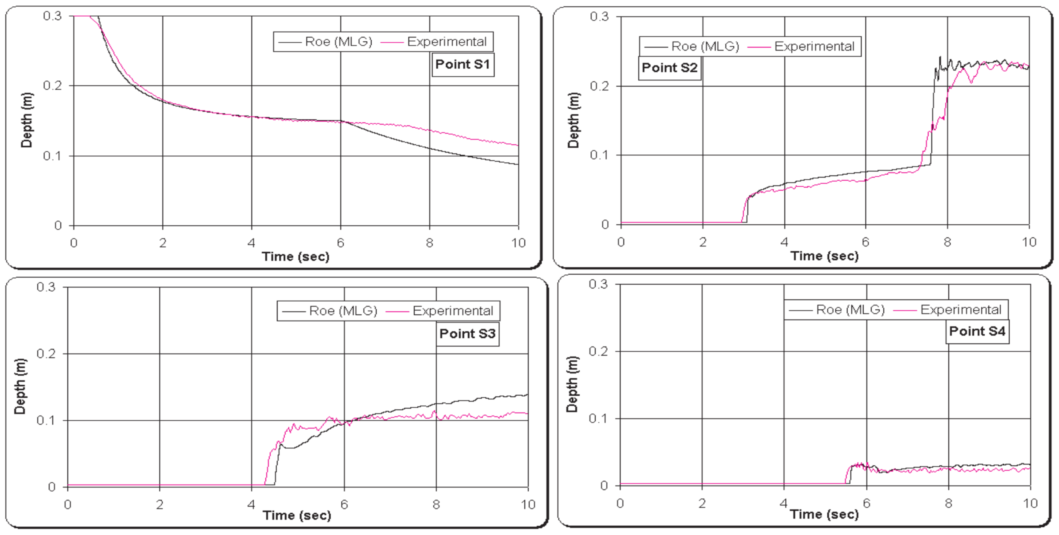


(b)

Figure 8.48: Comparison between experimental data and numerical results using Roe's approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Limiters used with Roe's solver: (a) Superbee (b) LCD.

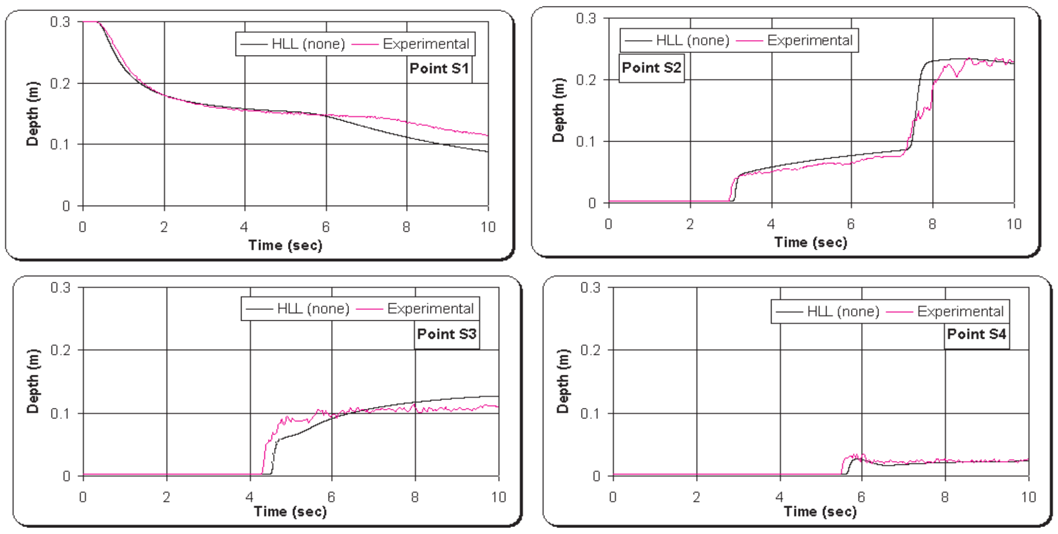


(a)

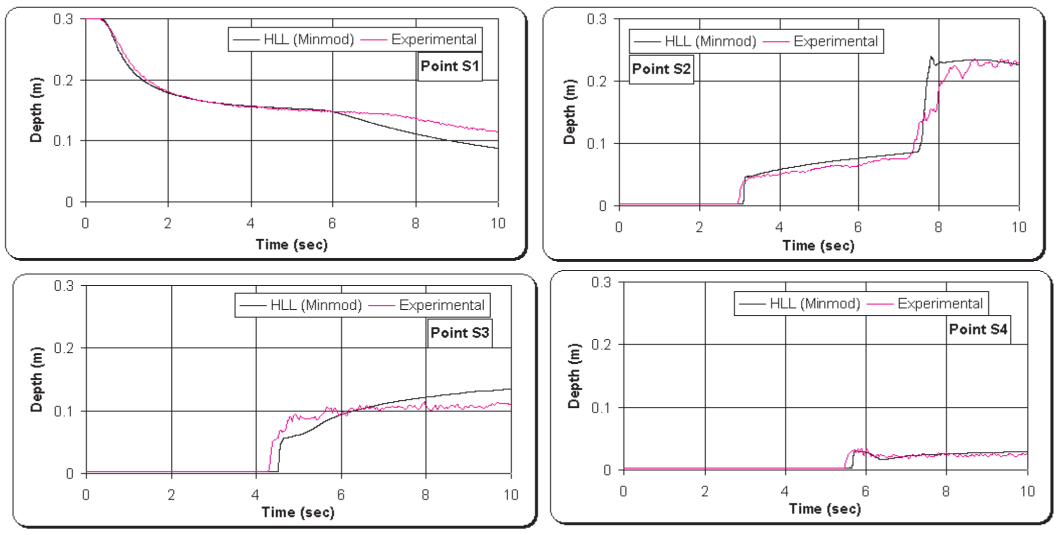


(b)

Figure 8.49: Comparison between experimental data and numerical results using Roe's approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Limiters used with Roe's solver: (a) Extended VanLeer (b) MLG.

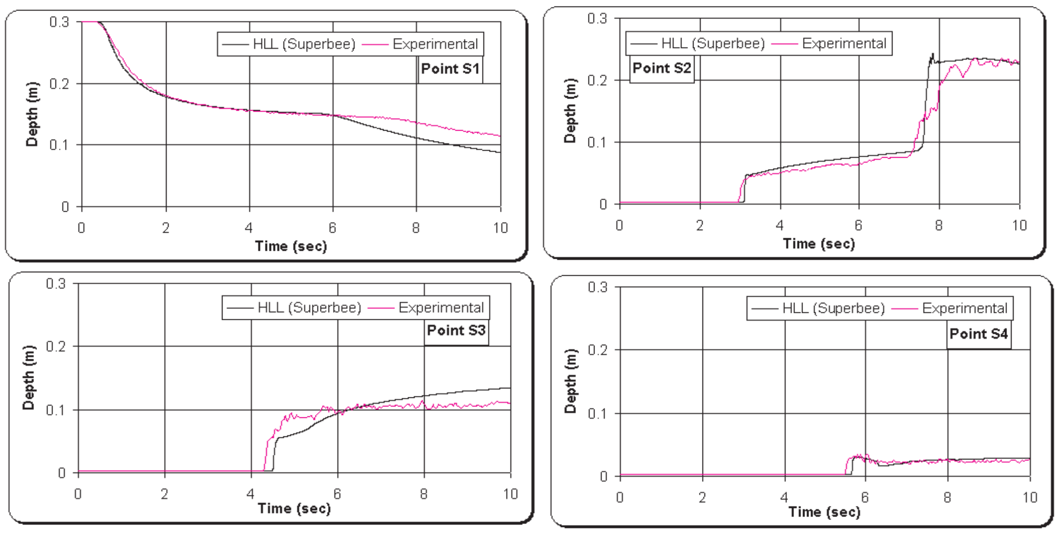


(a)

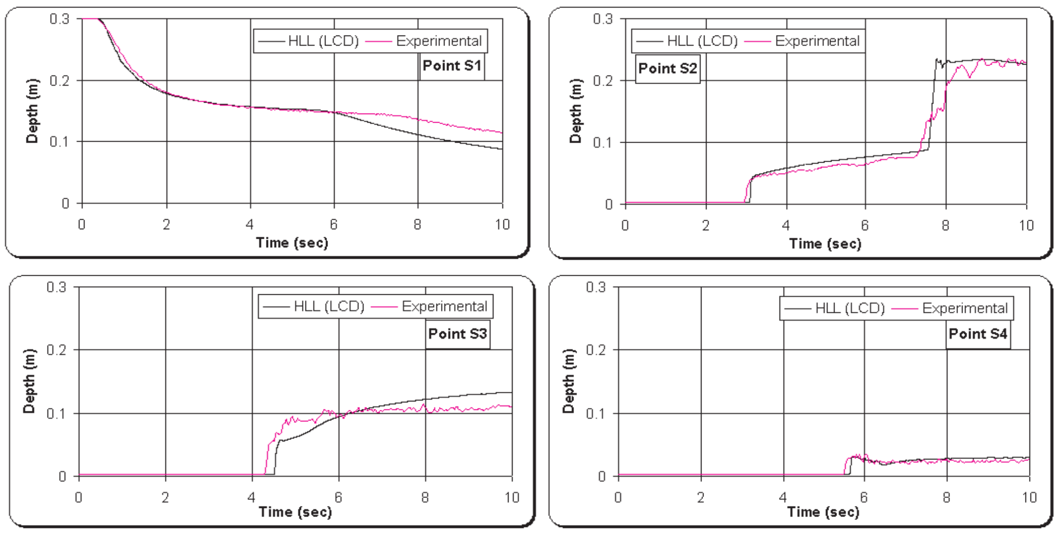


(b)

Figure 8.50: Comparison between experimental data and numerical results using HLL approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Limiters used with HLL solver: (a) No Limiter (b) Minmod.

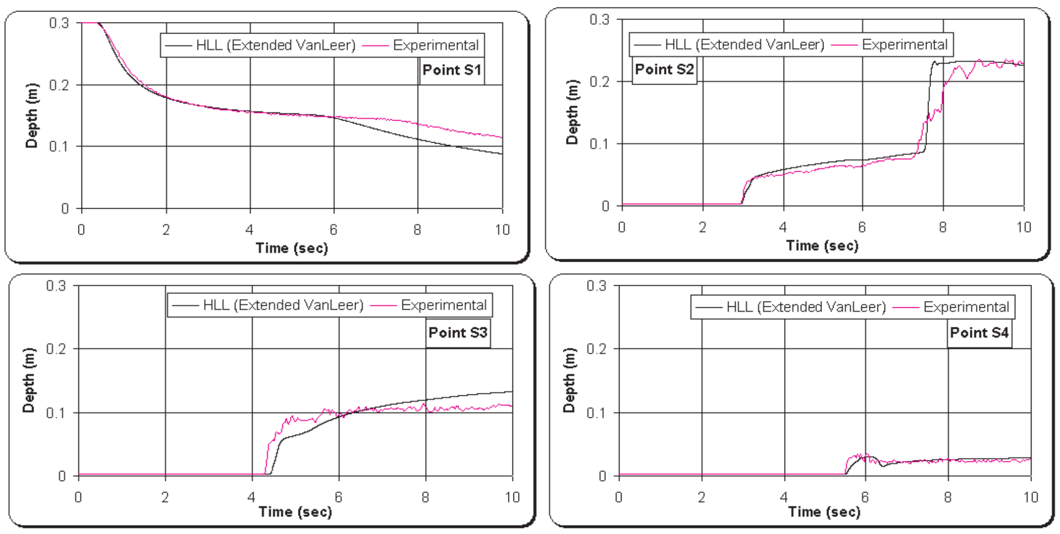


(a)

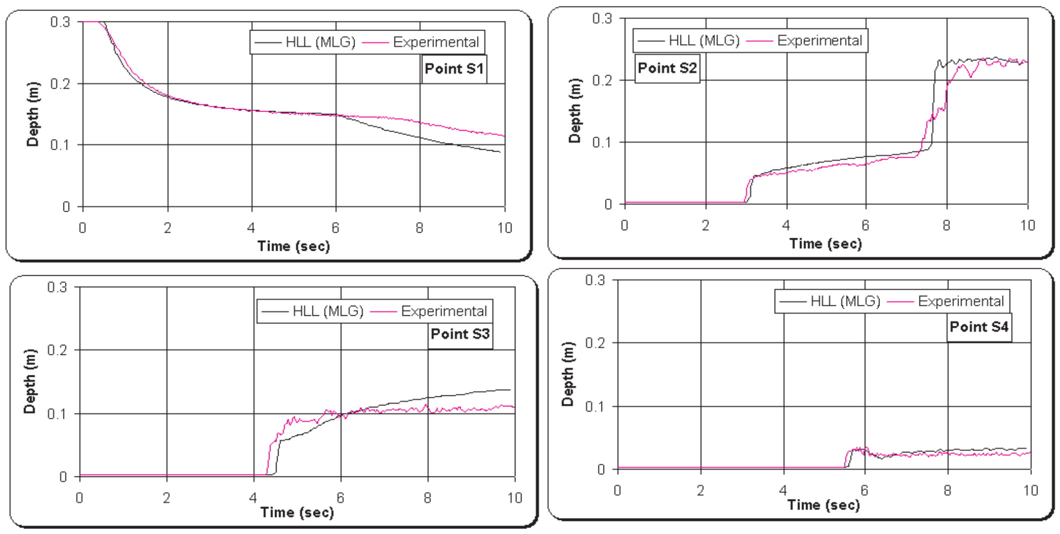


(b)

Figure 8.51: Comparison between experimental data and numerical results using HLL approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Limiters used with HLL solver: (a) Superbee (b) LCD.

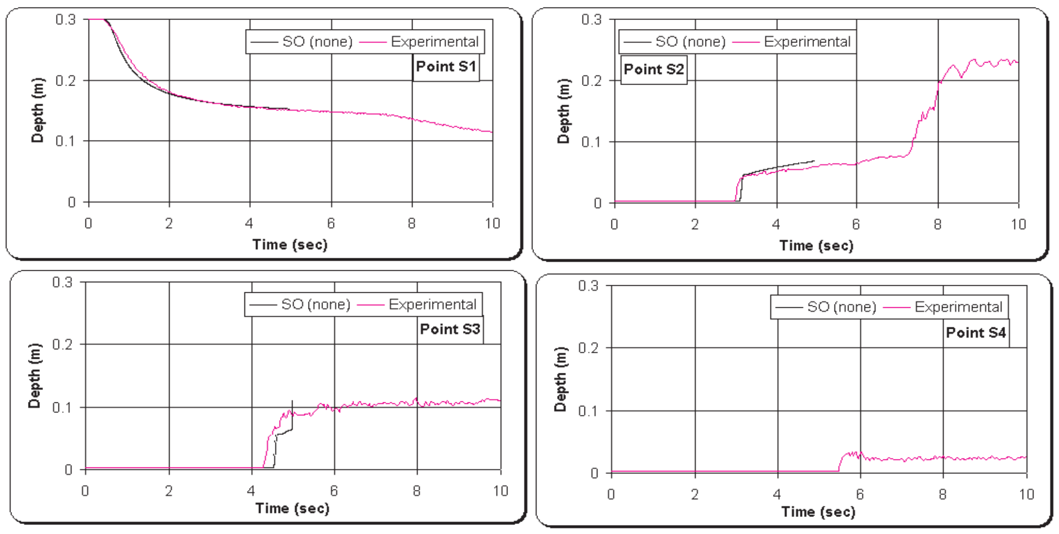


(a)

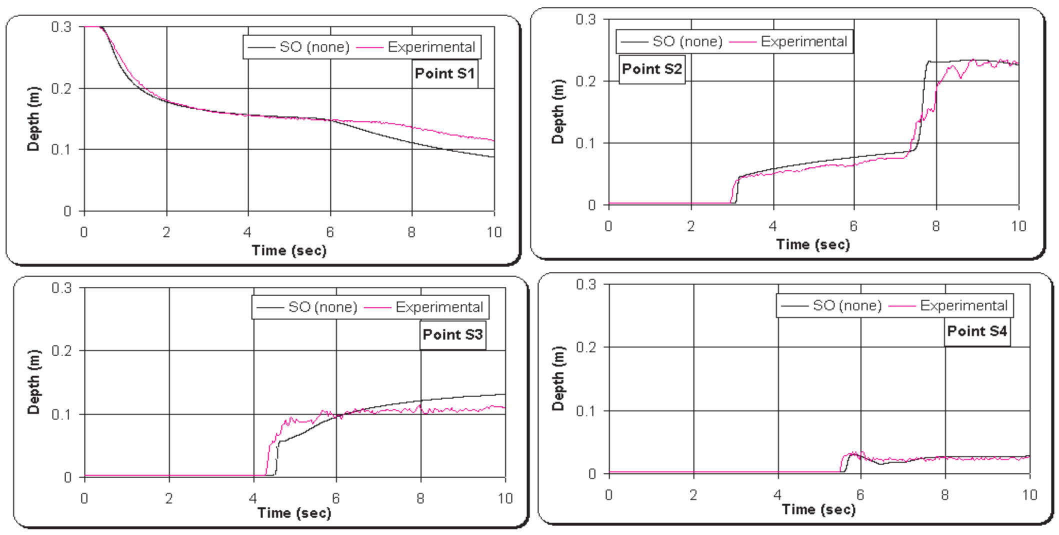


(b)

Figure 8.52: Comparison between experimental data and numerical results using HLL approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Limiters used with HLL solver: (a) Extended VanLeer (b) MLG.



(a)



(b)

Figure 8.53: Comparison between experimental data and numerical results using SO approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Limiters used with SO solver: (a) No Limiter fails for assumption: minimum depth = 10^{-6} (b) No Limiter with assumption for minimum depth = 10^{-4}.

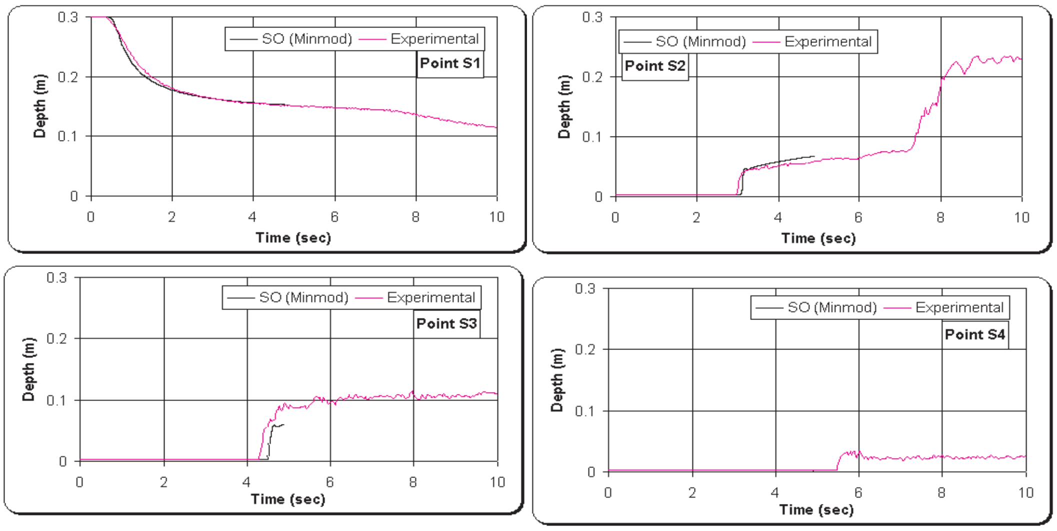


Figure 8.54: Comparison between experimental data and numerical results using SO approximate solver and Minmod limiter on the time evolution during 10s of the water depth at the gauging points S1, S2, S3 and S4. Minmod limiter fails for assumption: minimum depth = 10E-4.

8.7.4 Observations

The modelling approach required for this test has provided a true real life test for the *Riemann2D* . This test has been an extreme investigation of the capabilities of the *Riemann2D* and there are several notable areas, summarised as below:

- Roes' approximate solver (see Figures 8.47 to 8.49):
  - The initial depth of the water in downstream is 0.003 which is very small compared to upstream water depth 0.3. This means, application of limiters to the model may cause the oscillation at the tip of the shock. This oscillation might develop a dry zone near the shock. Therefore a minimum depth of 10^{-6}m is imposed to the domain. This depth is significantly small to cause any inaccuracy into the solution.
  - Gauging Station S1: The numerical and experimental results have perfect match, for all the limiters, upto time evolution of 6 seconds. After this the numerical results shows faster receding water level. This can be explained

by the behaviour of water profile at S3 gauging station, where water level rises more than the experimental data.

  - Gauging Station S2: For all the limiters there is good match between the numerical and experimental results, except a little disagreement of the results at time 7.5s (approximately).
  - Gauging Station S3: It can be seen that there is a disagreement between the numerical and experimental results, in spite of good agreement at other gauging stations. Although similar results have been found in various literatures, where authors have used different numerical schemes.
  - Gauging Station S4: This station shows very good match of the numerical and experimental results.

- HLL approximate solver (see Figures 8.50 to 8.52):
  - The numerical results for all the limiters using HLL solver is very similar to the behaviour to those using Roes' approximate solver.
- SO approximate solver (see Figures 8.53 and 8.54):
  - SO has shown oscillations at the tip of the shock in one-dimensional tests(see section 8.3). Therefore, for same reason as Roes' unstability in dry zone, a minimum depth of 10^{-6}m is assigned. It can be seen in Figure 8.53 (a) that the numerical results match with the experimental results till time = 5s (approximately). It should be noted that time = 5s is the time at which water enters the narrow section of the channel (near to gauging station S3). It can be argued that the behaviour of the water wave till it enters the converging section is one-dimensional. After this two-dimensional effect comes into play. Therefore, due to complex (two-dimensional) behaviour of the flow and oscillatory nature of the SO solver, it becomes unstable after time = 5s.
  - Due to the above stated reason the minimum depth of the water is increased to 10^{-4}m. It is found that rising the minimum water level make the SO solver stable. The results shown in Figure 8.53 (b) is very similar to others using Roes' and HLL solvers.

  - One-dimensional tests have shown that the limiters increase the oscillation for SO at the tip of the shock (see section 8.3). Therefore, when limiters are applied to SO and minimum depth is set to 10^{-4}m it give unstable results after time = 5s, *i.e.*, when two-dimensional behaviour is prominent (see Figure 8.54). Any depth (minimum depth) set more than 10^{-4}m is not worth for this case, because the downstream depth is 0.003m.

- For all the results obtained in this test case, it is important to note that the numerical results are similar. The only question has been, if the solver and limiter is stable enough to produce results. Therefore, it can be said that for stable combination of solver and limiter it doesn't not make much difference which solver and which limiter is used for simulation, at least for small scale laboratory test cases).

8.8 Dam Break in a Channel with 90° Bend

This test case together with the test specifications is supplied on the website of HR Wallingford[1]. The test was conducted as a part of project called, Concerted Action on Dam-break Modelling (CADAM[2]).

When a permanent flow moves around a curve, the water level rises at the outer bank and a corresponding lowering occurs at the inner bank. In rapidly varying flows around a sharp bend, a bore can form and lead to important local head losses that will affect the progression of the water downstream from the bend. In a dam-break case especially, the water storms in the downstream valley and it is important to be able to properly simulate the propagation of the front. The shape of the valley has a great influence on this propagation velocity and bends can slow down the front but also cause an increase in the water level upstream or even lead to the formation of an upstream travelling bore [Frazao *et al.* (1998)].

The dam-break flow in a channel with a 90○ bend shows important two-dimensional features in the bend. However, besides the bend zone and out of the severe transient

[1]`http://www.hrwallingford.co.uk/projects/CADAM/CADAM/index.html`

[2]CADAM was an EC funded Concerted Action Programme looking at dam-break modelling and application. February 1998 - January 2000.

phase during the strong reflection process, the flow can be reasonably assumed as one-dimensional in both upstream and downstream channel reaches. It is thus attractive to test two-dimensional model for this test case. The bed depth is assumed to be dry in downstream, therefore it can also be used to test the ability of the solvers in modelling dry bed cases in one-dimensional and two-dimensional cases.

8.8.1 The Laboratory Configuration of the CADAM Project

The shape of the channel is shown in Figure 8.55. The initial water level in the upstream reservoir is 20cm (or 0.2m) above the channel bed level. The channel downstream is dry in a first set of measurements.

The upstream reservoir has dimensions of 2.44m×32.39m, the channel is rectangular, 0.495m wide, with glass walls, the upstream reach is about 4m long and the downstream reach, after the bend, is about 3m long. The channel bed level is 0.33m above the upstream reservoir bed level. The downstream end of the channel is open (free chute). The initial water level in the reservoir is 0.25m above the channel bottom.

The channel was equipped with six gauges (see Table 8.1) recording the time evolution of the water level. Several set of measurement campaigns showed a very good reproducibility in both the dry and wet bed cases.

8.8.2 Aim of Test

The aim of this test is to:

1. study the ability of *Riemann2D* in using different Riemann solvers to replicate the transition flow (one- to two- and two- to one- dimensional);
2. study the ability to simulate the dry bed cases;
3. compare the numerical results against laboratory results obtained in the European Commission's CADAM project; and
4. present the particulars for developing and undertaking the test with *Riemann2D* and discuss the associated results.

Note:

- Courant Number for all the cases is 0.2;

- All the simulation results are for first order in time; and
- Test results are shown only for “No limiter” cases.

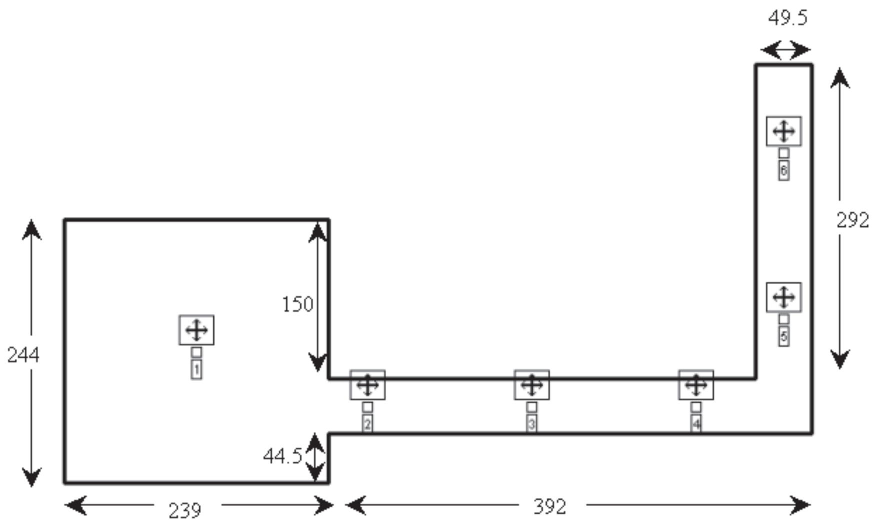


Figure 8.55: Schematic diagram of channel with 90° bend. The laboratory model consists of a long rectangular channel with a removable sluice gate at the entrance. Also shown are the gauging stations G1 to G6. Note: Dimensions are in mm.

Table 8.1: Position of the gauging points in the 90° bend channel, the origin of the axes being in the lower left corner. Note: The name of the station are same as Fraccarollo and Toro (1995).

Station	x [m]	y[m]
G1	1.19	1.20
G2	2.74	0.69
G3	4.24	0.69
G4	5.74	0.69
G5	6.56	1.51
G6	6.56	3.01

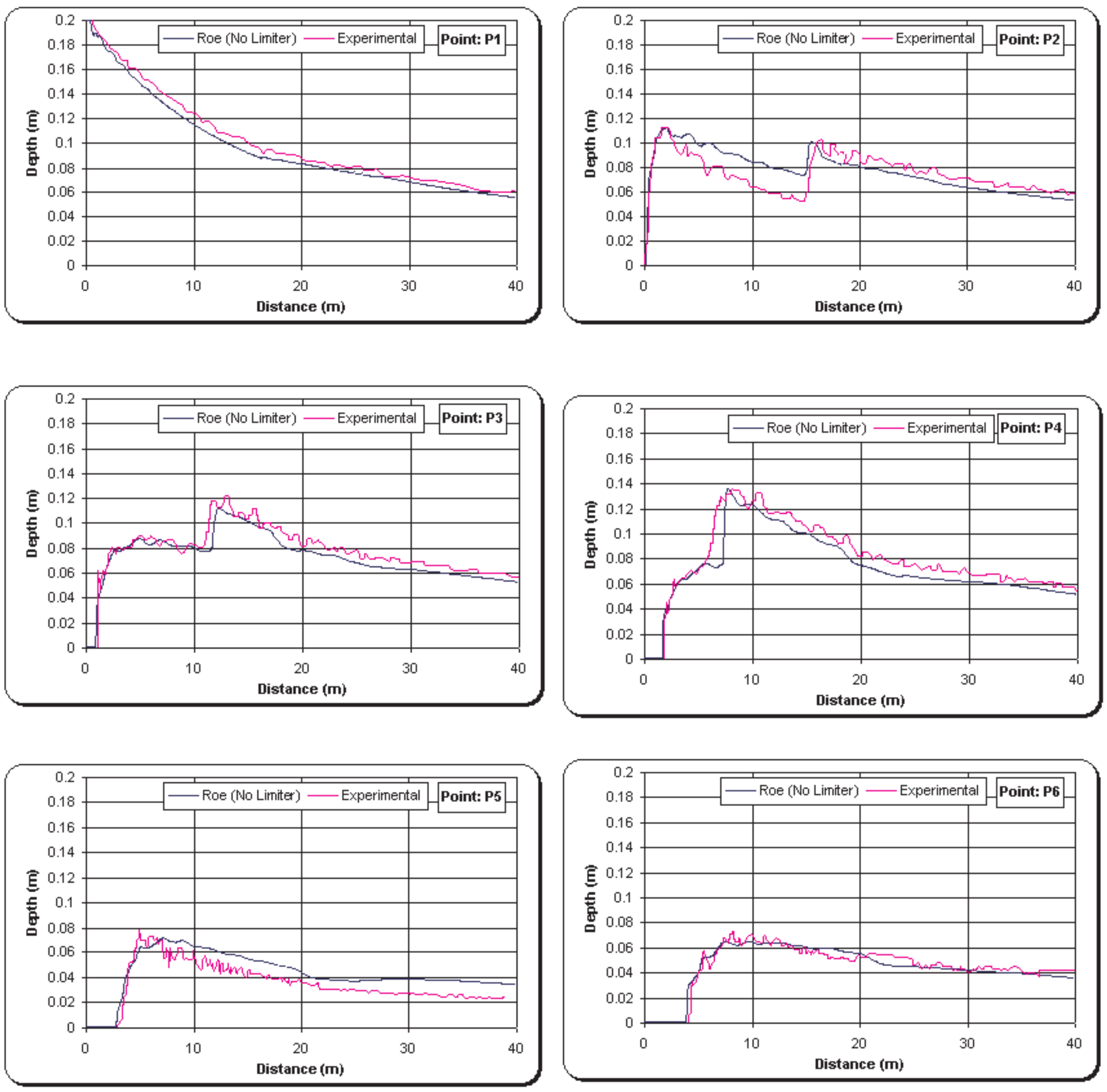


Figure 8.56: Comparison between experimental data and numerical results using HLL approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3, S4, S5 and S6.

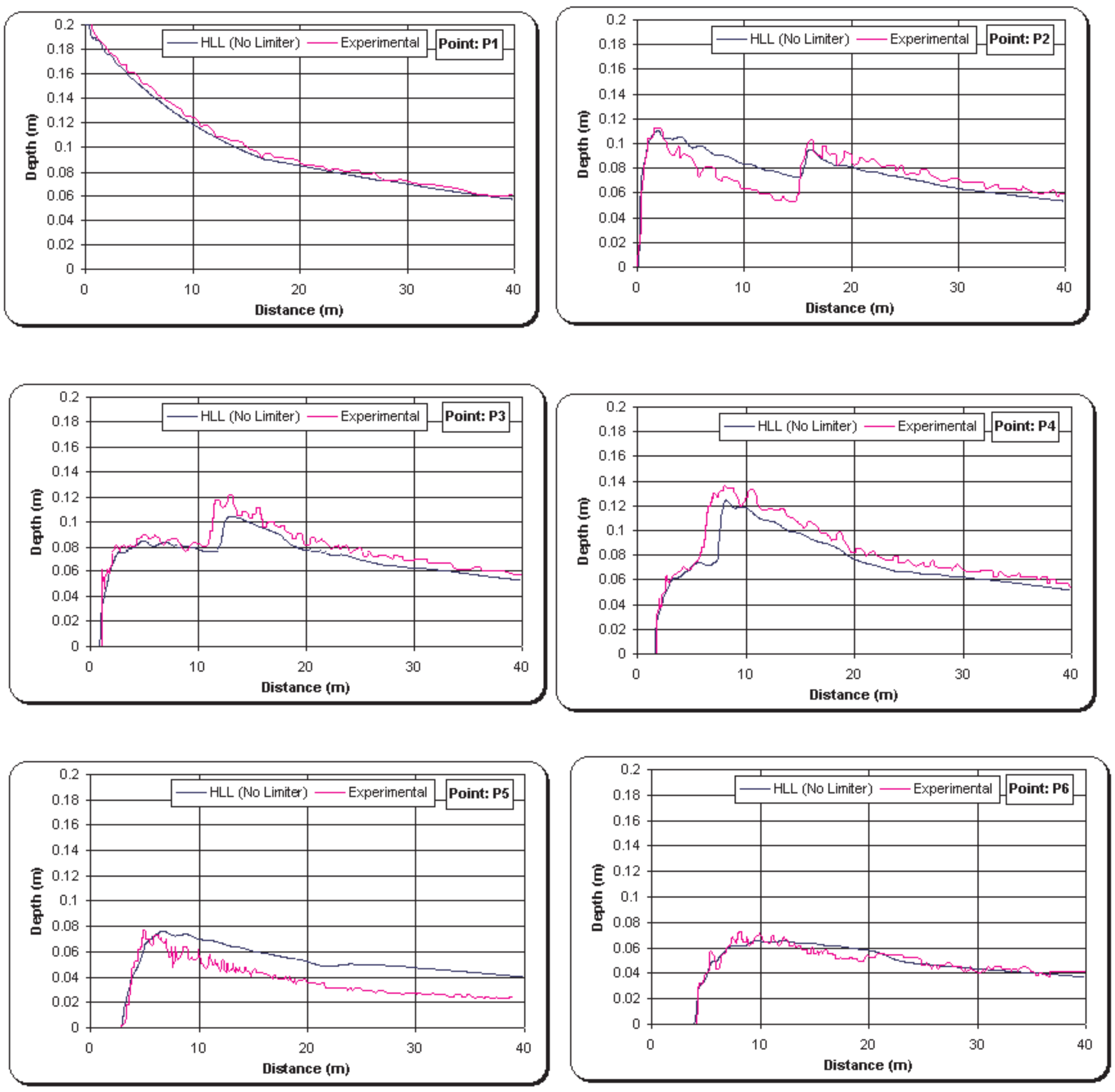


Figure 8.57: Comparison between experimental data and numerical results using HLL approximate solver on the time evolution during 10s of the water depth at the gauging points S1, S2, S3, S4, S5 and S6.

8.8.3 Observations

In the upstream reach of the channel, the flow is mainly 1D, while it is clearly 2D in the downstream reach. Due to the contraction at the entrance of the channel, there are some local 2D features near the gate, but at some distance (60-70cm approx.)downstream from the gate, these effects vanish. In the upstream reservoir, non symmetric features can also be observed, as a consequence of the non centred position of the gate : the circular emptying front does not reach all the walls at the same time, producing non symmetric reflections and oscillations. However, this does not seem to have a great influence on the flow in the channel, especially during the first seconds when the strongest phenomenon consists of the dam break wave. After 15s (approximately), the receding bore formed by the reflection in the bend reaches the reservoir and disappears. The flow then becomes almost permanent, i.e. the general shape of the water surface is preserved but the water level decreases progressively. Some notable observations related to solvers are:

- Roes' and HLL approximate solver (see Figures 8.56 and 8.57):
  - The results from both the solvers are very similar to each other.
  - Minimum depth assumption for Roes' solver is 10^{-4}m, because assumption of 10^{-6}m makes the solver unstable at the bend. This is in contrast to the test case in the section 8.7. The possible reason for this could be complicated behaviour of water flow at the bend, compared to complexity at the converging and diverging section in the previous test case (section 8.7).
  - Gauge G1 represents the emptying curve of the reservoir. The water level evolution is correct, which means that the numerical model computes the right discharge coming into the channel.
  - The arrival times of the receding reflected front (abrupt increase of the water level) at gauge G4, G3 and G2 can be seen. After the bore has reached these gauges, the time evolution of the water level is similar to the one in the reservoir, and represents the progressive emptying of the system.
  - One differences between the numerical model results and the measurements are in the arrival times of the bore, a bit late at G3 and G4, the approximate difference being 1s.

- The main difference between the numerical and experimental results is at G5, where the numerical water level is higher than the experimental water level. This means that the numerical model does not predict the correct height of the water at the changing angles. This may be because of physical diffusion at the bend.

- SO approximate solver completely fails to simulate this test case.

8.9 Two-Dimensional Dam Break Experiment of Fraccarollo and Toro

Due to some disagreement between the numerical and experimental results in the previous test cases (see sections 8.7 and 8.8), this test case is considered. This test case is a simple two-dimensional test case with a dry flood-plane (see Figure 8.58), whereas in the previous test cases it was transition between one- and two- dimensional flows.

8.9.1 Aim of Test

The aim of this test is to:

1. study the ability of *Riemann2D* to replicate the dry bed flow condition;
2. compare the numerical results against the experimental results; and
3. present the particulars for developing and undertaking the test with *Riemann2D* and discuss the associated results.

8.9.2 The Laboratory Configuration

The experimental flume, shown in Figure 8.58, is 2m wide and 3m long, 1m of which is occupied by the reservoir [see Fraccarollo and Toro (1995) and Gottardi and Venutelli (2004)]. The width of the gate, symmetrically centred, is 0.40m. The flood-plain boundaries are all open. In the simulation model velocities normal to closed boundary are taken equal to zero, whereas, in the outflow open boundary longitudinal gradients of u, v and h were assumed to zero.

The initial depth in the reservoir was taken 0.60 m, and the floodplain was considered dry. The position of gauging points are given in table 8.2.

Note:

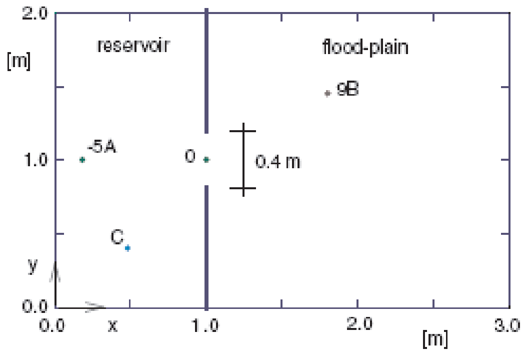
- Courant Number for all the cases is 0.2;
- All the simulation results are for first order in time;
- Test results are shown only for No limiter cases; and
- Bed surface is horizontal through out the domain.

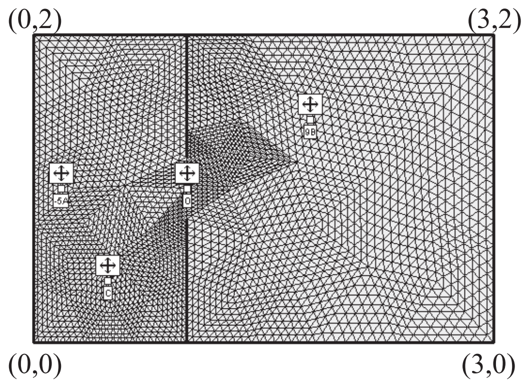
8.9.3 Observations

There is excellent agreement between the numerical and experimental results (see Figure 8.59). Only exception is at gauging station “Point O” where the depth of water, for first 2 seconds, is less for the numerical results compared to experimental results. It is possible that the effect of wall friction initial reduces the velocity which cause the water level to rise and this effect disappears. It should be noted that there is no source term for the wall friction.

Table 8.2: Position of the gauging points in the reservoir and flood-plain, the origin of the axes being in the lower left corner

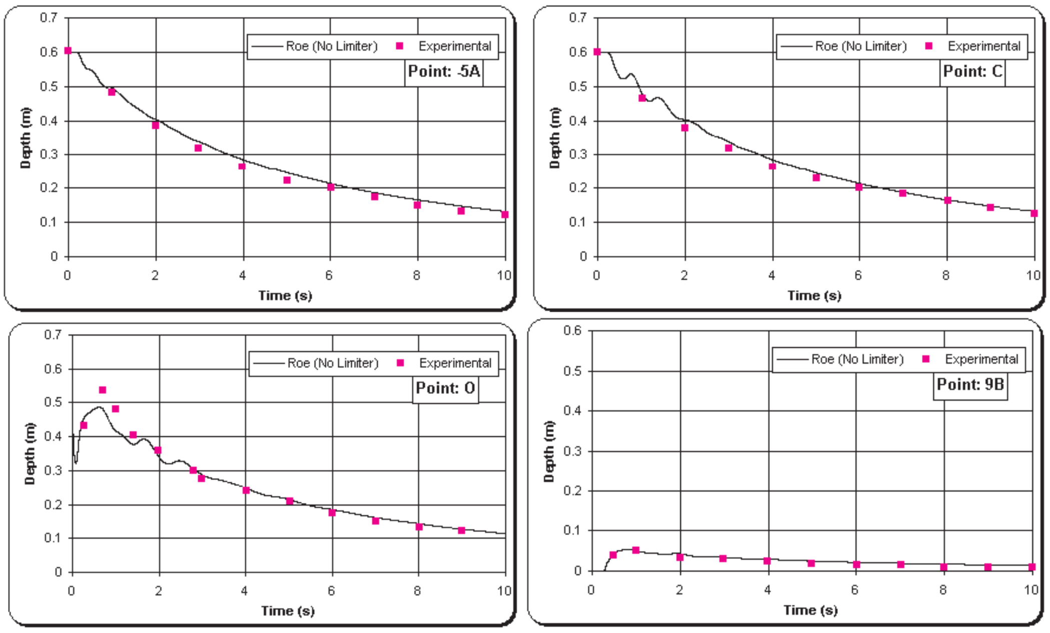
Station	x [m]	y [m]
-5A	0.18	1
C	0.48	0.40
O	1	1
9B	1.802	1.45



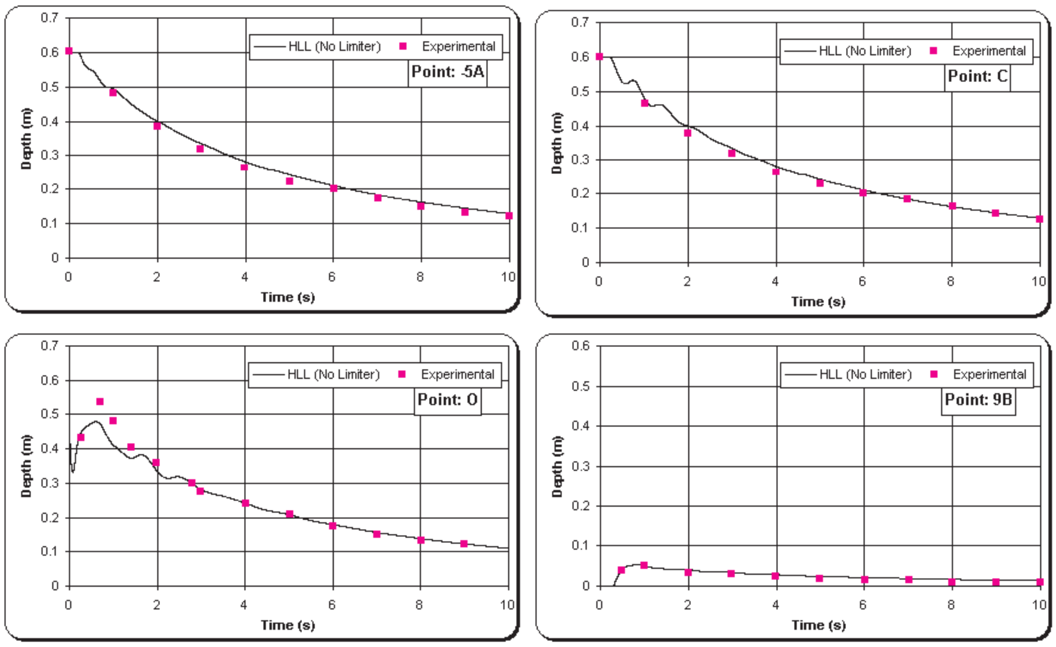


(a) (b)

Figure 8.58: (a) Schematic diagram of experimental reservoir and flood-plain, (b) Numerical mesh for the domain.



(a)



(b)

Figure 8.59: Comparison between experimental data and numerical results using approximate solver on the time evolution during 10s of the water depth at the gauging points -5A, C, O and 9B. (a) Roes' (b) HLL

8.10 Oblique Hydraulic Jump

A further inviscid flow test case is examined which corresponds to an oblique hydraulic jump problem for which an analytical solution is available. Supercritical channel flow subject to a sudden change in cross-section can lead to the formation of hydraulic jumps (shocks) and negative jumps (rarefaction waves). In the present test case the channel configurations used by Chippada *et al.* (1998) and Zienkiewicz and Ortiz (1995) is adopted. For the channel flows, the inlet supercritical Froude number F, is defined as follows:

$$F = \frac{U}{\sqrt{gH}} \tag{8.1}$$

where U and H are the mean velocity and depth of the fluid at the inlet, respectively.

8.10.1 Aim of Test

The aim of this test is to:

1. study the ability of the *Riemann2D* simulate the hydraulic jump of correct height and angle due to oblique angle at a section;
2. present the particulars for developing and undertaking the test with *Riemann2D* and discuss the associated results.

8.10.2 Test Configuration

The problem domain and numerical mesh is shown in Figure 8.60 (b). An oblique hydraulic jump is induced in the flow field by a supercritical flow which is deflected by a converging wall. A schematic diagram of the induced shock front is given in Figure 8.60 (a), where α is the angle formed by the shock front with the x-axis.

The test case simulated is supercritical flow ($F > 1$) encountering a sudden change of cross-section through a boundary wall constriction on one side wall of the channel. On the other side the flow remains unbounded. The domain definition of the channel along with the numerical mesh is shown in Figure 8.60. This type of channel geometry leads to the formation of a stationary hydraulic jump originating at the point of constriction. In the supercritical case the channel has constant bathymetric depth.

Numerical simulations were performed with a constriction angle of 15° and inlet supercritical Froude numbers $F = 2.5$. The numerical mesh consists of 1609 nodes and 3072 triangular elements.

The steady state water depths for the inlet Froude numbers is considered as 1 m.

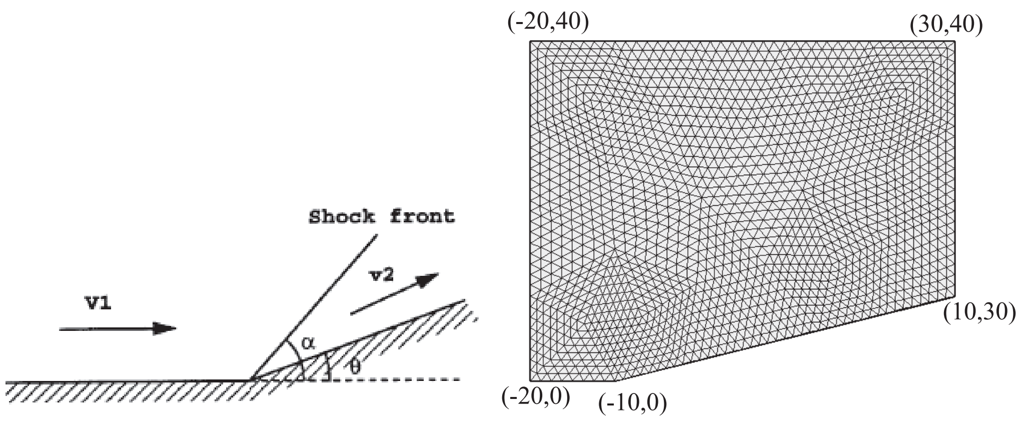


Figure 8.60: (a) Schematic diagram showing oblique shock front in supercritical flow, (b) Mesh.

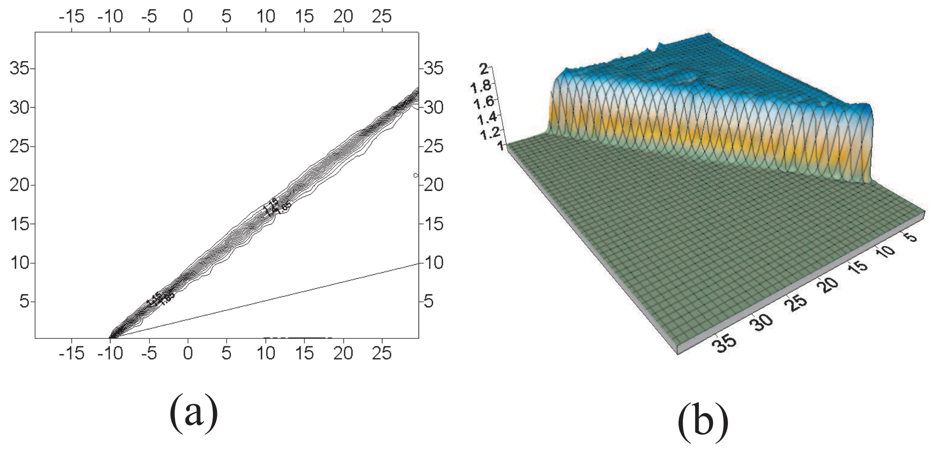


Figure 8.61: Contour of the water depth

8.10.3 Observations

The method is seen to be monotonic with no oscillations across the shock. The width of the shock is seen to span only two to four cell divisions. Neglecting the viscous dissipation, this problem can be solved analytically [Ippen (1951) and Chippada *et al.* (1998)]. The analytical angle, α is 39.58 and the numerical angle is 39.52. This can seen as an excellent agreement of the analytical and numerical results.

8.11 Conclusions

In this chapter *Riemann2D* is applied to solve a wide range of shallow water test problems. The observations made in this chapter can be summarised as follows:

- The test on **one-dimensional dam break** problem in section 8.3, shows that the Roes', HLL and SO approximate Riemann solver using different limiters (Minmod, Superbee, LCD ,Extended VanLeer and MLG) are capable of simulating the dam break problem. The results vary with the use of limiters. However, SO approximate Riemann Solver produce oscillation in the solution. In this test the width of the dam is same as the length of the dam *i.e.*, 50m, and the results are plotted for each element. Therefore, this test provides more reliable comparison compared to previously published results. Also, the scale of plot are large, which gives closer picture of the compared results.

- The test on **two-dimensional circular dam break** problem in section 8.4, shows that the Roes', HLL and SO approximate Riemann solver using different limiters (Minmod, Superbee, LCD ,Extended VanLeer and MLG) are capable of simulating the circular dam break problem. The results shows that all the combination of approximate Riemann solver and limiter produce stable and good results. This case is a test of:

  - extreme situation where the ratio of water depth on two side of the dam is 1:10;
  - the symmetry of the result for radial flow; and
  - the stability of solution in a trans-critical flow *i.e.*, super- to sub- to super-critical flow.

  For all the above, *Riemann2D* give good results compared to other published results.

- Following the success of the above cases, another test case on **two-dimensional dam break** (section 8.5) is done to assess the ability of the model to simulate different front wave propagation. This test was simulated using the Roes', HLL and SO approximate Riemann solver in combination with different limiters (Minmod, Superbee, LCD, Extended VanLeer and MLG). All the results for this case are stable and give good results compared to previously published results.

- **Rotation test** (section 8.6) of a pile of solute particles in a constant rotating velocity field gives some interesting results:

  - The numerical diffusion depends on the mesh density for a particular problem. For a coarse mesh, the behaviour of limiter is similar. However, when the mesh density is increased, effect of different limiters can be clearly seen.
  - The behaviour of limiters is dependent of the rank of the limiter *i.e.*, the number of triangles required by the limiters. Higher the rank better the result.
  - In modelling a real world shallow water problem, the source terms are calibrated against the experimental data. This calibrated source term is the balance between actual value of source term and the loss due to numerical diffusion. As seen in the Figure 8.44, MLG limiter produce negligible numerical diffusion at certain mesh density and mesh size. Therefore, it can be concluded that if the mesh size are known at which numerical diffusion is zero for MLG limiter, then actual value of the source terms can be calculated for a real world problem. This could be an important contribution to the shallow water modelling problems, however, further testing is required to thoroughly verify the findings through more test cases.

- The results in the previous test cases were compared either with the exact solution, analytical solution or the previously published results. Therefore, to asses the ability of *Riemann2D* to model the real test cases. The *Riemann2D* was modelled to simulate the experimental setup of the CADAM project for a **dam break in a converging-diverging channel** (section 8.7). The results from this test case shows the ability of the *Riemann2D* to model experimental set-up, with little discrepancy. Important finding in this case were:

  - due to very less depth of water (0.003m) in the downstream, some Riemann solver with certain limiters failed to simulate the problem; and
  - Combination of Riemann solver and limiters produce nearly similar results for this test case.

- A test on a **dam break in a channel with** 90° **bend** (section 8.8) was done to:
  - test the model to simulate the dry bed dam break test. Roe's and HLL approximate solver successfully produced good results. However, SO approximate solver completely fails to simulate the problem.
  - asses the ability of the model to simulate transitional flow from 1d- to 2d- to 1d-flows.
  - study the behaviour of the model due to hydraulic jump due to change in the profile of the cross-section.

- Another experimental test case, based on work of Fraccarollo and Toro (1995), is presented to show the ability of the *Riemann2D* to simulate a **dam break in a two-dimensional reservoir** (section 8.9).

- At last, the model is tested for an oblique hydraulic jump in a converging channel, for which the model give excellent result.

CHAPTER 9

Summary and Future Work

9.1 Attractions of the Present Work

The main attraction of the presented work lie in the fact that the development were made in the framework of a new, generic modelling system, which makes immediate application possible in shallow water problems. However, similar models for solving problems in various related fields of hyperbolic systems (e.g., gas dynamics, acoustics, elastodynamics, optics, geophysics, and biomechanics to name a few.) can be easily developed.

Each chapter in this thesis present features that has significantly contributed to the model development.

Object-Oriented Approach (Chapter 3): A new and different is approach presented in this chapter, which is used for developing *Riemann2D*. It helps to classify the code of the model into various classes, which makes the development process faster, and easier to maintain and debug. One important aspect of this approach is that it is generic in nature and could be adopted for developing other CFD models as well.

Finite Volume Methods for Hyperbolic Problems (Chapter 4): This chapter helps to identify common features of various hyperbolic model and thus leads to generalisation and better understanding of hyperbolic theory. It can be seen in this chapter that most of the required code for model any hyperbolic system is present here. Key attractions of this chapter lies in the:

- discussion on control volume;
- generalised CFL criteria for two-dimensional hyperbolic prolems; and
- generalised implementation of the boundary condition

High-Resolution Methods (Chapter 5): Most of the text in this chapter is adopted from previously published literatures. However, the classification of code into various classes in a separate limiter package leads to the clarity of the limiter implementation. This has led to categorise the limiters based on the numbers of triangles used, called *rank* of the limiter. The results in section 8.6 shows the relation of the *rank* of the limiter and its performance. Also, a *rank* 3 limiter is developed for 2D-triangular mesh, which is based on 1D- vanLeer limiter. This limiter fills the gap between existing *rank* 1 and *rank* 4 limiters.

Free-Surface Shallow Water Flows (Chapter 6): Here, the background and theory of the shallow water problem shows the ease of solving a hyperbolic system using the generic *Riemann2D* model. Main attraction of this chapter are:

- the discussion on the problem due to varying bathymetry and its solution;
- eigen structure for shallow water equations with solute;
- the discussion on the problem due to wetting and drying bed and its solution; and
- the possible approach to applying limiters to the bed for a movable bed problem.

Object-Oriented Design and Development (Chapter 7): The object-oriented design of the model is presented using UML diagram. The software life cycle and use of CVS provided a new approach to CFD modelling, where learning and model development goes hand-in-hand.

Riemann2D Applied to Shallow Water Equations (Chapter 8): The verification of the *Riemann2D* which is documented in this chapter, ensure the quality of the model. Furthermore, a broad range of verification cases cover most of the intended extension of the application domain of the shallow water model. The test cases allow critical assessment of the properties and the model is now capable of reproducing various physical phenomena. However, the section 8.6 on the *rotation test for shallow water equations* provides some very interesting findings:

- *Rank* of the limiter and mesh density is directly related to the performance of the limiter in terms of numerical diffusion. With the increase in the *rank* of the limiter or the mesh density, the numerical diffusion is decreased.
- Figure 8.44, shows the relationship between limiter and mesh size. The importance of this figure is that it helps to measure the effect of mesh size and limiter on the numerical diffusion.
- In modelling a real world shallow water problem, the source terms are calibrated against the experimental data. This calibrated source term is the balance between actual value of source term and the loss due to numerical diffusion. As seen in the Figure 8.44, MLG limiter produce negligible numerical diffusion at certain mesh density and mesh size. Therefore, it can be concluded that if the mesh size are known at which numerical diffusion is zero for MLG limiter, then actual value of the source terms can be calculated for a real world problem. This could be an important contribution to the shallow water modelling problems, however, further testing is required to thoroughly verify the findings through more test cases.

9.2 Realised New Development

In this work, a new object-oriented framework and model is presented that may serve as a generic foundation for solving any hyperbolic problem. This object-oriented framework and model represents a first step towards the development of generic model architectures and provides modularity, flexibility, reusability, and advantages in process evaluation previously unavailable.

The two important features that strengthen the use of object-oriented technology for *Riemann2D* development are:

1. Object-oriented technology (OOT) is widely known to promote greater flexibility and maintainability in software model development, and is widely popular in

large-scale software engineering. Hence, adopting OOT to research models would be significant achievement.

2. Object-oriented programming is easier to learn for those new to computer programming than previous approaches, and that the object-oriented approach is often simpler to develop and to maintain, lending itself to more direct analysis, coding, and understanding of complex situations and procedures than other programming methods. This would encourage new researchers to adopted and reuse the previously developed model.

The idea behind object-oriented framework and model development for *Riemann2D* is the model may be seen as comprising a collection of individual units (or model) that act on each other, as opposed to a traditional view in which a computer model (program) may be seen as a collection of functions, or simply as a list of instructions to the computer. Each object is capable of receiving messages, processing data, and sending messages to other objects. The design of the *Riemann2D* has following important features:

- **Universality:** From the point of view of model universality, the application domain of the developed model is much wider than previously developed models. Any number of developers/users should be able to implement the *Riemann2D* for solving various hyperbolic problems.
- **User-Friendly:** The code is written in such a way that new developers should be able to overwrite, if required, the functionalities of the *Riemann2D* for their individual needs, without making any change to the generic part of the model. Thus it will serve the purpose of every user, while safe-guarding the core model. The naming convention in this model is so simple that the new developers/users with less knowledge of object-oriented technology or the background of the generic model should also be able to to adopt it for their own purpose.
- **Reusability** A principal goal of object-oriented design of the *Riemann2D* is to make the model as reusable as possible: to have it serve many different situations and applications so that you can avoid reimplementing, even if in only slightly different form, something that's already been done. Reusability is influenced by a variety of different factors, including:

- how reliable and bug-free the code is;
- how clear the documentation is;
- how simple and straightforward the programming interface is; and
- how efficiently the code performs its tasks

9.3 Recommended Further Developments

Further development of the model presented, *Riemann2D* can be categories into three major areas:

1. **Solving shallow water problems**

   - Integration with Geographic Information Systems (GIS): GIS is an organised collection of computer hardware, software, geographic data, and personnel designed to efficiently capture, store, update, manipulate, analyze, and display all forms of geographically referenced information. It combine relational databases with spatial interpretation and outputs often in form of maps. Integrating *Riemann2D* with GIS would help *Riemann2D* to use the GIS database to retrieve and store information. The output from the *Riemann2D*, stored in GIS database, could be easily analysed for impact assessment studies of real world problems. Few example would be: impact assessment study of tsunami disaster, overland flooding in rivers and dam-break.
   - Movable bed problems: The approach presented in chapter 6, can be adopted and further modelled to simulate movable bed problem. Application of limiter to the moving bed could be a huge success in modelling the bed movements.
   - Further extension of the model: The shallow water model could be extended for soil erosion models as shown in chapters 3 and 7.
   - Further study on MLG limiter: As seen in Figure 8.44 and suggested in section 9.1, further study can help to develop a method in which actual value of the source terms for a real world problems can be calculated.

2. **Multi-physics model development**

- Adding more hyperbolic problem models: The main objective of the design of *Riemann2D* in this work has been to facilitate the development of multi-physics model, which can be used to solve a variety of hyperbolic problems. The implementation of shallow water model can be followed to develop similar extension models for other hyperbolic problems.
- Further development of the generic model: The development process of *Riemann2D* has been through studying requirements of a variety of hyperbolic problems. However, with addition of other hyperbolic extension models, new things can be learned and thus improvement can be made to the existing generic code.

3. **Test bed for numerical schemes**

   - As shown in this work, a number of Riemann solvers and limiters can be added to the model. The generalisation of Riemann solvers and limiters could help to test them on new hyperbolic problems. Also, new Riemann solver and limiters could be developed based on the knowledge of the existing Riemann solver and limiters.

APPENDIX A

Glossary of Object-Oriented & Computing Terms

Abstraction: A simplified representation of something that is potentially quite complex. It is often not necessary to know the exact details of how something works, is represented or is implemented, because we can still make use of it in its simplified form. Object-oriented design often involves finding the right level of abstraction at which to work when modeling real-life objects. If the level is too high, then not enough detail will be captured. If the level is too low, then a program could be more complex and difficult to create and understand than it needs to be.

Abstract Method: A method with the abstract reserved word in its header. An abstract method has no method body. Methods defined in an interface are always abstract. The body of an abstract method must be defined in a sub class of an abstract class, or the body of a class implementing an interface.

ActiveX: ActiveX is a complex technology. The term itself applies to a set of object-oriented technologies and tools typically used to develop networking and Internet-aware software.

Application: Often used simply as a synonym for program. However, in Java, the term is particularly used of programs with a Graphical User Interface (GUI) that are not applets.

Array: A fixed-size object that can hold zero or more items of the array's declared type.

Behaviour: The methods of a class implement its behavior. A particular object's behavior is a combination of the method definitions of its class and the current state of the object.

Class: A programming language concept that allows data and methods to be grouped together. The class concept is fundamental to the notion of an object-oriented programming language. The methods of a class define the set of permitted operations on the class's data (its attributes). This close tie between data and operations means that an instance of a class - an object - is responsible for responding to messages received via its defining class's methods.

Code: Code written by a programmer in a high-level language and readable by people but not computers. Source code must be converted to object code or machine language before a computer can read or execute the program.

COM: A software architecture developed by Microsoft[1] to build component-based applications. COM objects are discrete components, each with a unique identity, which expose interfaces that allow applications and other components to access their features.

Commit: A commit occurs when a user copy the changes made on the local files to the repository (the version control software takes care of knowing which files changed since the last time the two were synchronised)

Component: A unit of code that provides a set of services through well-defined interfaces and is specifically designed and packaged to be reusable. A component acts like a "black box" that hides or encapsulates the details of how the services are actually implemented. Components can be mixed and matched to form larger systems.

CORBA: Short for Common Object Request Broker Architecture, an architecture that enables pieces of programs, called objects, to communicate with one another regardless of what programming language they were written in or what operating

[1]Microsoft Corporation: `http://www.microsoft.com/`

system they're running on. CORBA was developed by an industry consortium known as the Object Management Group[1] (OMG).

DCOM: Short for Distributed Component Object Model, an extension of the Component Object Model (COM) that allows COM components to communicate across network boundaries.

Delphi: An application development system for Windows from Borland[2]. Introduced in 1995 and based on the object-oriented version of Pascal (Object Pascal), it includes visual programming tools and generates executable programs (.EXE files).

Dynamic Link Library (DLL): Short for Dynamic Link Library, a library of executable functions or data that can be used by a Windows application. Typically, a DLL provides one or more particular functions and a program accesses the functions by creating either a static or dynamic link to the DLL.

Encapsulation: Safeguarding the state of an objects by defining its attributes as private and channeling access to them through accessor and mutator methods.

Integrated Development Environment (IDE): A set of programs run from a single user interface. For example, programming languages often include a text editor, compiler and debugger, which are all activated and function from a common menu.

Inheritance: A feature of object-oriented programming languages in which a sub type inherits methods and variables from its super type. Inheritance is most commonly used as a synonym for class inheritance.

Instance: An object of a particular class. In programs written in the Java(TM) programming language, an instance of a class is created using the new operator followed by the class name.

Interface: A Java programming language keyword used to define a collection of method definitions and constant values. It can later be implemented by classes that define this interface with the "implements" keyword.

[1]Object Management Group: `http://www.omg.org/`

[2]Borland Software Corporation: `http://www.borland.com/`

Java: A portable high level programming language released by Sun Microsystems[1].

Method: The part of a class definition that implements some of the behavior of objects of the class. The body of the method contains declarations of local variables and statements to implement the behavior. A method receives input via its arguments, if any, and may return a result if it has not been declared as void.

Object: An instance of a particular class. In general, any number of objects may be constructed from a class definition (see singleton, however). The class to which an object belongs defines the general characteristics of all instances of that class. Within those characteristics, an object will behave according to the current state of its attributes and environment.

Object-Oriented Language: Programming languages such as C++ and Java that allow the solution to a problem to be expressed in terms of objects which belong to classes.

Package: A named grouping of classes and interfaces that provides a package namespace. Classes, interfaces and class members without an explicit public, protected or private access modifier access!modifier have package visibility. Public classes and interfaces may be imported into other packages via an import statement.

Polymorphism: The ability of an object reference to be used as if it referred to an object with different forms. Polymorphism in Java results from both class inheritance and interface inheritance. The apparently different forms often result from the static type of the variable in which the reference is stored.

Private: A Java programming language keyword used in a method or variable declaration. It signifies that the method or variable can only be accessed by other elements of its class.

Protected: A Java programming language keyword used in a method or variable declaration. It signifies that the method or variable can only be accessed by elements residing in its class, subclasses, or classes in the same package.

[1]`http://java.sun.com/`

Public: A Java(TM) programming language keyword used in a method or variable declaration. It signifies that the method or variable can be accessed by elements residing in other classes.

Repository: The repository is where the files are stored, often on a server.

References

Abbott, M.B. (1966). *An introduction to the method of characteristics*. Elsevier, New York. 2.3.2

Abbott, M.B. and Basco, D.R. (1989). *Computational fluid dynamics an introduction for engineers*. Longman, London. 2.6

Abbott, M.B. and Verwey, A. (1970). Four-point method of characteristics. *Journal of the Hydraulics Division*, **96**, 2549–2564. 2.3.2

Ajayi, A.E. (2004). *Surface runoff and infiltration processes in the volta basin, West Africa: Observation and modeling*. PhD Thesis, University of Bonn. 2.3.2

Alcrudo, F. and Benkhaldoun, F. (2001). Exact solutions to the riemann problem of the shallow-water equations with a bottom step. *Computers & Fluids*, **30**, 643–671. 2.5.2

Alcrudo, F. and Garcia-Navarro, P. (1993). A high-resolution godunov-type scheme in finite volumes for the 2d shallow-water equations. *International Journal of Numerical Methods in Fluids*, **16**, 489–505. 2.3.4, 2.5.1, 8.4, 8.4.3

Alcrudo, F., Garcia-Navarro, P. and Saviron, J.M. (1992). Flux difference splitting for 1d open channel flow equations. *International Journal of Numerical Methods in Fluids*, **14**, 1009–1018. 2.5.1

Alfredsen, K. (2000). Simulation of human impacts on flood regimes - the design of an object-oriented integration framework. *Hydroinformatics 2000, Iowa*. 2.6

Ambrosi, D. (1995). Approximation of shallow water equations by roe's approximate riemann solver. *International Journal of Numerical Methods in Fluids*, **20**, 157–168. 2.5.2

AMES, W.F. (1992). *Numerical methods for partial differential equations (3rd edition)*. Academic Press, Inc., Boston. 2.2.1

ANASTASIOU, K. AND CHAN, C.T. (1997). Solution of the 2d shallow water equations using the finite volume method on unstructured triangular meshes. *International Journal of Numerical Methods in Fluids*, **24**, 1225–1245. (document), 2.3.4, 2.4.2, 2.5, 2.5.1, 8.20, 8.5, 8.5.3

ANDERSON, W.K. (1994). A grid generation and flow solution method for euler equations on unstructured grids. *Journal of computational Physics*, **110**, 23–38. 2.4.2

ARORA, M. AND ROE, P.L. (1997). A well-behaved tvd limiter for high-resolution calculations of unsteady flow. *Journal Of Computational Physics*, **132**, 3–11. 2.4.2

BARTH, T.J. (1993). *Recent developments in high order k-exact reconstruction on unstructured meshes*. AIAA (0668). 2.3.4

BATTEN, P., LAMBERT, C. AND CAUSON, D.M. (1996). Positively conservative high-resolution convection schemes for unstructured elements. *International Journal for Numerical Methods in Engineering*, **39**, 1821–1838. 2.4.2, 5.9.1, 5.9.2.5, 3

BENEDICT, D.R., ALISTAIR, G.L.B. AND PAUL, H.T. (2003). Mathematical balancing of flux gradient and source terms prior to using roe's approximate riemann solver. *Journal of Computational Physics*, **192**, 422–451. 2.5.1, 2.5.2

BENNETT, S., MCROBB, S. AND FARMER, R. (2005). *Object-oriented systems analysis and design using UML*. McGraw Hill Higher Education. 2

BERMUDEZ, A. AND VAZQUEZ, M.E. (1994). Upwind methods for hyperbolic conservation laws with source terms. *Computers & Fluids*, **23**, 1049–1071. 2.5.2, 5.6, 6.7

BERZINS, M. (2001). Modified mass matrices and positivity preservation for hyperbolic and parabolic pdes. *Communications in Numerical Methods in Engineering*, **17**, 659–666. 2.3.2

BILLETT, S. AND TORO, E. (1997). On waf-type schemes for multidimensional hyperbolic conservation laws. *Journal of computational Physics*, **130**, 1–24. 5.9

Bittner, K. and Spence, I. (2006). *Managing iterative software development projects*. Addison Wesley. 7.4.1

Booch, G. (1991). *Object oriented design with applications*. Benjamin / Cummings. 2

Booch, G. (1993). *Object-oriented analysis and design with applications*. Addison-Wesley. 2

Book, D.L., Boris, J.P. and Zalesak, S.T. (1981). *Flux-corrected transport, in finite difference techniques for vectofised fluid dynamics calculations*. D.L. Book (ed.), Sprlnger-Verlag, New York. 2.4.1

Boris, J.P. and Book, D.L. (1973). Flux-corrected transport i. shasta, a transport algorithm that works. *Journal of Computational Physics*, **11**, 38–69. 2.4.2, 5.9

Boris, J.P. and Book, D.L. (1976). Flux corrected transport iii, minimal error fct algorithms. *Journal of Computational Physics*, **20**, 397–431. 2.4.2

Bradford, S.F. and Sanders, B.F. (2002). Finite-volume model for shallow-water flooding of arbitrary topography. *Journal of Hydraulic Engineering*, **128**, 289–298. 2.3.4

Breuss, M. (2005). An analysis of the influence of data extrema on some first and second order central approximations of hyperbolic conservation laws. *Mathematical Modelling and Numerical Analysis*, **39**, 965–994. 5.5.1

Brezis, H. (1988). Partial differential equations in the 20th century. *Advances in Mathematics*, **135**, 76–144. 2.2.1

Brio, M. and Wu, C.C. (1988). An upwind differencing scheme for the equations of ideal magnetohydrodynamics. *Journal of Computational Physics*, **75**, 400–422. 4.3.3

Brooks, A. and Hughes, T. (1982). Streamline upwind/petrovgalerkin formulations for convection dominated flows with particular emphasis on the incompressible navierstokes equations. *Computer Methods in Applied Mechanics and Engineering*, **32**, 199–259. 2.3.4

Brown, D. (1997). *Introduction to object-oriented analysis: objects in plain english*. John Wiley and Sons. 3.3

Bruner, S. (1996). *Parallelization of the Euler equations on unstructured grids*. Ph.D. thesis, Virginia Polytechnic Institute and State University, Department of Aerospace engineering. 2.4.2

Burguete, J. and Garcia-Navarro, P. (2001). Efficient construction of high-resolution tvd conservative schemes for equations with source terms: application to shallow water flows. *International Journal for Numerical Methods in Fluids*, **37**, 209–248. 2.5.2

Caleffi, V., Valiani, A. and Zanni, A. (2003). Finite volume method for simulating extreme flood events in natural channels. *Journal of Hydraulic Research*, **41**, 167–177. 2.4.2, 2.5.1, 6.6.2, 6.6.2

Calle, J.L.D., Devloo, P.R.B. and Gomes, S.M. (2005). Stabilized discontinuous galerkin method for hyperbolic equations. *Computer Methods in Applied Mechanics and Engineering*, **194**, 1861–1874. 2.3.4

Canuto, C., Hussaini, M.Y., Quarteroni, A. and Zang, T. (1998). *Spectral methods in fluid dynamics*. Springer-Verlag, New York. 2.4.3

Carpenter, M.H., Gottlieb, D. and Abarbanel, S. (1993). The stability of numerical boundary treatments for compact high-order finite-difference schemes. *Journal of Computational Physics*, **108**, 272–295. 2.3.2

Cate, T.H., Lin, H.X. and Mynett, A.E. (1998). A study on integrating software packages. *Hydroinformatics 1998, Balkema, Rotterdam*, 457–464. 2.6

Chainais-Hillairet, C. (1999). Finite volume schemes for a nonlinear hyperbolic equation: Convergence towards the entropy solution and error estimates. *ESAIM: Mathematical Modelling and Numerical Analysis*, **33**, 129–156. 2.4.3

Chan, C.T. and Anastasiou, K. (1999). Solution of incompressible flows with or without a free surface using the finite volume method on unstructured triangular meshes. *International Journal for Numerical Methods in Fluids*, **29**, 35–57. 2.5

Chen, G.Q. (2000). Shock capturing and related numerical methods in computational fluid dynamics. *Proceedings of 15th Algorithmy, Vysoke Tatry, Slovak*, **70**, 51–74. 2.4.3, 5.2

CHEN, G.Q., DU, Q. AND TADMOR, E. (1993). Spectral viscosity approximations to multidimensional scalar conservation laws. *Mathematics of Computation*, **6**, 629–643. 2.4.3

CHIPPADA, S., DAWSON, C., MARTINEZ, M. AND WHEELER, M. (1998). A godunov-type finite volume method for water equations the system of shallow water equations. *Computer Methods in Applied Mechanics and Engineering*, **151**, 105–129. 8.10, 8.10.3

CLARKE, J.F., KARNI, S., QUIRK, J.J., SIMMONS, L.G., ROE, P.L. AND TORO, E.F. (1993). Numerical computation of two-dimensional, unsteady detonation waves in high energy solids. *Journal of Computational Physics*, **106**, 215–233. 4.3.3

COCKBURN, B. AND SHU, C.W. (2001). Rungekutta discontinuous galerkin method for convection-dominated problems. *Journal of of Scientific Computing*, **16**, 173–261. 2.3.4

COCKBURN, B., LIN, S.Y. AND SHU, C.W. (1989). Tvb rungekutta local projection discontinuous galerkin finite element method for conservation laws iii: one dimensional systems,. *Journal of Computational Physics*, **84**, 90–113. 2.3.2

COLELLA, P. AND WOODWARD, P. (1984). The piecewise parabolic method (ppm) for gas dynamical simulations. *Journal of Computational Physics*, **54**, 174–201. 2.4.3, 5.9

COURANT, R., FRIEDRICHS, K. AND LEWY, H. (1928). ber die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, **100**, 32–74. 4.4.5

COURANT, R., ISAACSON, E. AND REES, M. (1952). On the solution of nonlinear hyperbolic differential equations by finite difference. *Communications on Pure & Applied Mathematics*, **5**, 243–255. 2.3.2, 4.3.1

CRUTCHFIELD, W.Y. AND WELCOME, M.L. (1993). Object-oriented implementation of adaptive mesh refinement algorithms. *Scientific Programming*, **2**, 145–156. 2.6

CUNGE, J.A., HOLLY, F.M. AND VERWEY, A. (1980). *Practical aspects of computational river hydraulics*. Pitman, London. 2.6

Dafermos, C.M., LeFloch, P.G. and Toro, E.F. (2003). Nonlinear hyperbolic waves in phase dynamics and astrophysics, report from the organisers. *Isaac Newton Institute for Mathematical Sciences*, **27 January to 11 July**, URL (cited on 09 March 2006): http://www.newton.cam.ac.uk/reports/0203/npa.html. 2.3.1

Darwish, M.S. and Moukalled, F. (2003). Tvd schemes for unstructured grids. *International Journal of Heat & Mass Transfer*, **46**, 599–611. 2.4.1

Davis, S.F. (1988). Simplified second-order godunov-type methods. *SIAM Journal on Scientific Computing*, **9**, 445–473. 4.3.4

Davis, S.F. (1992). An interface tracking method for hyperbolic systems of conservation laws. *Applied Numerical Mathematics*, **10**, 447–472. 2.3

Deckers, F. (1994). A geohydrological information system based on object-oriented technology. *Hydroinformatics 94, Balkema, Rotterdam*, 253–260. 2.6

Delis, A.I. and Katsaounis, T. (2005)). Numerical solution of the two-dimensional shallow water equations by the application of relaxation methods. *Applied Mathematical Modelling*, **29**, 754–783. (document), 8.4, 8.4.3, 8.5, 8.28, 8.29, 8.5.3

Dubois, F. and Mehlman, G. (1993). A non parametric entropy fix for roe's method. *AIAA Journal*, **31**, 199–200. 4.3.3

Dubois, P. (1996). *Object technology for scientific computing - object-oriented numerical software in Eiffel and C*. Prentice Hall. 2

Durran, D.R. (1999). *Numerical methods for wave equations in geophysical fluid dynamics*. Springer-Verlag: New-York. 2.4

Edenhofer, J. and Schmitz, G. (1981). An implicit method of characteristics for the solution of hyperbolic initial-boundary value problems and its convergence. *Computing*, **26**, 257–264. 2.3.2

Einfeldt, B. (1988). On godunov-type method for gas dynamics. *SIAM Journal on Numerical Analysis*, **25**, 294–318. 4.3.4

Einfeldt, B., D.Muntz, C., Roe, P.L. and Sjogreen, B. (1991). On godunov-type methods near low densities. *Journal of Computational Physics*, **92**, 273–295. 4.3.3

Fennema, R.J. and Chaudhry, M.H. (1990). Explicit methods for 2d transient free-surface flows. *Journal of Hydraulic Engineering*, **116**, 1003–1014. 2.5, 8.5, 8.5.3

Ferzoui, L. and Stoufflet, B. (1989). A class of implicit upwind schemes for euler simulations with unstructured grids. *Journal of Computational Physics*, **84**, 174–206. 2.4.2

Fraccarollo, L. and Toro, E.F. (1995). Experimental and numerical assessment of the shallow water model for two-dimensional dam-break type problems. *Journal of Hydraulic Research*, **33**, 843–864. (document), 8.1, 8.9.2, 8.11

Frazao, S.S., Sillen, X. and Zech, Y. (1998). Dam-break flow through sharp bends physical model and 2d boltzmann model validation. *Proceedings of the CADAM meeting Wallingford, United Kingdom, March 1998*. 8.8

Frink, N.T. (1992). Upwind schemes for solving the euler equations on unstructured tetrahedral meshes. *AIAA Journal*, **3**, 70–77. 2.4.2

Fujihara, M. and Borthwick, A.G.L. (2000). Godunov-type solution of curvilinear shallow water equations. *Journal of Hydraulic Engineering*, **126**, 827–836. 2.5.1

Fursenko, A.A., Shaov, D.M., Timofeev, E.V. and Voinovich, P.A. (1993). High resolution schemes in unstructured grids in transient shocked flow simulation. *Proceedings of the 13th International Conference On Numerical Methods in Fluid Dynamics,*, 250–255. 2.4.2

Gamma, E. and Larman, C. (2005). *Applying UML and patterns, an introduction to object-Oriented analysis and design and iterative development*. Addison-Wesley. 7.4.1

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design patterns : Elements of reusable object-oriented software*. Addison-Wesley Professional Computing Series, Reading, MA. 2

Garcia-Navarro, P. and Vazquez-Cendon, M.E. (2000). On numerical treatment of the source terms in the shallow water equations. *Computers & Fluids*, **29**, 951–979. 2.5.2

Gascon, L. and Coberan, J. (2001). Construction of second-order tvd schemes for nonhomogeneous hyperbolic conservation laws. *Journal of Computational Physics*, **172**, 261–297. 2.5.1, 2.5.2

George, D.L. (2004). *Numerical approximation of the nonlinear shallow water equations with topography and dry beds: A Godunov-type scheme*. Master of Science Thesis. (document), 4.3.4, 6.4, 6.5

Ghidaglia, J.M., Kumbarob, A. and LeCoq, G. (2001). On the numerical solution to two-fluid models via a cell-centered finite volume method. *European journal of mechanics. B, Fluids*, **20**, 841–867. 2.3.4

Ghidaoui, M.S., Deng, J.Q., Gray, W.G. and Xu, K. (2001). A boltzmann based model for open channel flows. *International Journal for Numerical Methods in Fluids*, **35**, 449–494. 6.7

Giraud, L. and Manzini, G. (1996). Parallel implementation of 2d explicit euler solvers. *Journal of computational Physics*, **123**, 111–118. 4.3.3

Glaister, P. (1988). Approximate riemann solutions of the shallow water equations. *Journal of Hydraulic. Research*, **26**, 293–306. 2.5.1, 6.6.1, 6.7

Glass, I.I. (1974). *Shock waves and man*. The University of Toronto Press, Toronto. 2.4.3

Glaz, H. and Liu, T.P. (1984). The asymptotic analysis of wave interactions and numerical calculation of transonic nozzle flow. *Advances in Applied Mathematics*, **5**, 111–146. 2.4.3

Glimm, J. (1965). Solutions in the large for nonlinear hyperbolic systems of equations. *Communications in Pure and Applied Mathematics*, **18**, 697–715. 2.4.3

Glimm, J., Isaacson, E., Marchesin, D. and McBryan, O. (1981). Front tracking for hyperbolic systems. *Advances in Applied Mathematics*, **2**, 173–215. 2.3

Glimm, J., Klingenberg, C., McBryan, O., Plohr, B., Sharp, D. and Yaniv, S. (1985). Front trackingand two-dimensional riemann problems. *Advances in Applied Mathematics*, **6**, 259–290. 2.4.3

Godlewski, E. and Raviart, P.A. (1996). *Numerical approximation of hyperbolic systems of conservation laws*. Springer, New York. 2.3, 2.4, 4.3.1, 4.3.2

Godunov, S.K. (1959). A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *(Russian) Matem. Sbornik (N.S.)*, **47**, 271–306. 2.3, 2.4.3, 2.5.1, 4.3.1

Goldberg, D.E. and Wylie, B. (1983). Characteristic method using time-line interpolations. *Journal of Hydraulic Engineering*, **109**, 670–683. 5.7

Gottardi, G. and Venutelli, M. (2004). Central scheme for two-dimensional dambreak flow simulation. *Advances in Water Resources*, **27**, 259–268. (document), 8.4, 8.20, 8.9.2

Gray, M. and Roberts, R. (1997). Object-based programming in fortran 90. *Computer in Physics*, **11**, 355–361. 3.3

Grove, J.W. (1994). Applications of front tracking to the simulation of shock refractions and unstable mixing. *Applied Numerical Mathematics: Transactions of IMACS*, **14**, 213–237. 2.3

Guinot, V. (2002). The time-line interpolation method for large-time-step godunov-type schemes. *Journal of Computational Physics*, **117**, 394–417. 5.7

Guinot, V. (2003). *Godunov-type schemes, An introduction for engineers*. Elsevier Science. Elsevier. 5.7

Harpham, C. (1997). *Linear partial differential equations*. `http://www-staff.lboro.ac.uk/~macah/LPDE/LPDE.html`. 2.2.2

Harten, A. (1983). High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics*, **49**, 357–393. 2.3, 2.4.1, 2.5.1, 2.5.2

Harten, A. and Hyman, J.M. (1983). Self-adjusting grid methods for one-dimensional hyperbolic conservation laws. *Journal of Computational Physics*, **50**, 235–269. 4.3.3

Harten, A., Lax, P.D. and van Leer, B. (1983). On upstream differencing and godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, **25**, 35–61. 4.3.4, 4.3.4

Harten, A., Engquist, B., Osher, S. and Chakravarthy, S. (1987). Uniformly high order accurate non-oscillatory schemes iii. *Journal of Computational Physics*, **71**, 231–303. 2.4.3

Harvey, D., Han, D. and Cluckie, I. (2002). A blueprint for next-generation modelling software. *Proceedings of the Fifth International Conference on Hydroinformatics, Cardiff, UK*, **2**, 1276–1281. 1.1, 2.6

Helzel, C., Berger, M.J. and LeVeque, R.J. (2005). A high-resolution rotated grid method for conservation laws with embedded geometries. *SIAM Journal on Scientific Computing*, **26**, 785–809. 4.4.1

Hirsch, C. (1990). *Numerical computation of internal and external flows*. Wiley J. and Sons. 2.3.3, 2.4.2, 5.9.2.2

Holnicki, P. (1996). A piecewise-quintic interpolation scheme. *Journal of Computational Physics*, **127**, 316–329. 2.4.1

Horstmann, C. (2005). *Object-oriented design and patterns*. John Wiley & Sons Inc. 2

Hu, K., Mingham, C.G. and Causon, D.M. (1998). Abore-capturing finite volume method for open-channel flows. *International Journal of Numerical Methods in Fluids*, **28**, 1241–1261. 2.3.4

Hubbard, M. and Garcia-Navarro, P. (2000). Flux difference splitting and the balancing of source terms and flux gradients. *Journal of Computational Physics*, **165**, 89–125. 2.5.2, 6.7

Ichikawa, Y., Tachikawa, Y., Takara, K. and Shiiba, M. (2000). Object-oriented hydrological modelling system. *Hydroinformatics 2000, Iowa*. 2.6

IPPEN, A.T. (1951). Mechanics of supercritical flow. *Transactions,ASCE*, **116**, 268–295. 8.10.3

JI-WEN, W. AND RU-XUN, L. (2000). A comparative study of finite volume methods on unstructured meshes for simulation of 2d shallow water wave problems. *Mathematics and Computers in Simulation*, **53**, 171–184. 2.3.4

JIANG, G.S., LEVY, D., LIN, C.T., OSHER, S. AND TADMOR, E. (1998). High-resolution non-oscillatory central schemes with non-staggered grids for hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, **35**, 2147–2168. 5.5.1

KASHIWA, B.A. AND RAUENZAHN, R.M. (1994). A multimaterial formalism. *Numerical Methods in Multiphase Flows, ASME*, 149–157. 2.3.4

KROGER, T. AND LUKACOVA-MEDVID'OVA, M. (2005). An evolution galerkin scheme for the shallow water magnetohydrodynamic equations in two space dimensions. *Journal of Computational Physics*, **26**, 122–149. 2.3.4

KRONER, D. (1997). *Numerical schemes for conservation laws*. WileyTeubner, Chichester/Stuttgart. 2.3

KRONER, D., NOELLE, S. AND ROKYTA, M. (1995). Convergence of higher order upwind finite volume schemes on unstructured grids for scalar conservation laws in several space dimensions. *Mathematics of Computation*, **71**, 527–560. 4.3.2

KRUCHTEN, P. (2004). *Software architecture and iterative development*. Benjamin Cummings. 7.4.1

KULIKOVSKII, A.G., POGORELOV, N.V. AND SEMENOV, A.Y. (2002). *Mathematical aspects of numerical solutions of hyperbolic systems*. Monographs and Surveys in Pure and Applied Mathematics,. 4.3.1

KURGANOV, A. AND LEVY, D. (2000). A third-order semi-discrete central scheme for conservation laws and convection-diffusion equation. *SIAM Journal on Scientific Computing*, **22**, 1461–1488. 5.5.1

KURGANOV, A. AND PETROVA, G. (2000). Central schemes and contact discontinuities. *Mathematical Modelling and Numerical Analysis*, **34**, 1259–1275. 5.5.1

Kurganov, A. and Tadmor, E. (2000). New high-resolution central schemes for nonlinear conservation laws and convectiondiffusion equations. *Journal of Computational Physics*, **160**, 241–282. 4.3.2

Kurganov, A. and Tadmor, E. (2002). Solution of two-dimensional riemann problems for gas dynamics without riemann problem solvers. *Numerical Methods for Partial Differential Equations*, **18**, 584–608. 4.3

Kutija, V. (1998). Use of object-oriented programming in modelling of flow in open channel networks. *3rd International Conference on Hydroinformatics, Copenhagen.* (document), 2.6, 2.2

Laney, C.B. (1998). *Computational gasdynamics*. Cambridge University Press. 2.4

Larman, C. (2004). *Applying Uml And patterns: An introduction to object-oriented analysis and design and iterative development*. Prentice Hall. 2, 7.4.1

Larsen, L. and Gavranovic, N. (1994). Hydroinformatics: further steps into object orientation. *Journal of Irrigation and Drainage Engineering*, **32(Extra Issue)**, 195–202. 2.6

Lax, P.D. (1954). Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications in Pure and Applied Mathematics*, **7**, 159–193. 2.4.3

Lax, P.D. (1986). On dispersive difference schemes. *PhysicaD, North-Holland, Amsterdam*, **18**, 250–254. 2.4.3

Lax, P.D. and Wendorff, B. (1960). Systems of conservation laws. *Communications on Pure and Applied Mathematics*, **13**, 217–237. 2.4.3

Lee, J.H.W., Peraire, J. and Zienkiewicz, O.C. (1987). The characteristic galerkin method for advection dominated problems - an assessment. *Computer Methods in Applied Mechanics and Engineering*, **61**, 359–369. 2.3.2

Leonard, B.P. (1979). A stable and accurate convective modeling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, **19**, 59–98. 2.4, 2.4.1

Leonard, B.P. (1991). The ultimate conservative difference scheme applied to unsteady one-dimensional advection. *Computer Methods in Applied Mechanics and Engineering*, **88**, 17–74. 2.4, 2.4.1

LeVeque, R. and Shyue, K.M. (1995). One-dimensional front tracking based on high resolution wave propagation methods. *SIAM Journal on Scientific Computing*, **16**, 348–377. 2.3

LeVeque, R.J. (1992). *Numerical methods for conservation laws*. Birkhauser Verlag, Basel, Boston, New York. 2.3, 4.4.3, 5.8

LeVeque, R.J. (1998). Balancing source terms and flux gradients in high-resolution godunov methods: the quasisteady wave-propagation algorithm. *Journal of Computational Physics*, **146**, 346–365. 2.5.2, 5.6, 6.7

LeVeque, R.J. (2002). *Finite volume method for hyperbolic problems*. Cambridge Univ. Press. (document), 1.3.1, 2.3, 2.3.3, 2.4, 2.4.1, 2.4.2, 2.4.3, 1, 4.2, 4.2.2, 4.3.1, 4.3.1, 4.3.2, 4.4.1, 5.3.1, 5.3.1, 5.2, 5.7.1, 5.4, 1, 1, 8.6, 8.7, 8.8, 8.9, 8.10, 8.11

Lie, K.A. and Noelle, S. (2003). On the artificial compression method for second-order nonoscillatory central difference schemes for systems of conservation laws. *SIAM Journal on Scientific Computation*, **24**, 1157–1174. 5.5.1

Liggett, J.A. (1994). *Fluid mechanics*. McGraw Hill International Editions. 2.3.3, 2.5.1

Lin, S.J. and Rood, R.B. (1996). Multidimensional flux-form semi-lagrangian transport schemes. *Monthly Weather Review*, **124**, 2046–2070. 2.4

Lin, S.J. and Rood, R.B. (1997). An explicit flux-form semi- lagrangian shallow-water model on the sphere. *Q. J. R. Meteorological Society*, **123**, 2477–2498. 2.4

Lin, S.J., Chao, W., Sud, Y. and Walker, G. (1994). A class of the vanleer transport schemes and its applications to the moisture transport in a general circulation model. *Monthly Weather Review*, **122**, 1575–1593. 2.4

Lipnikov, K. and Shashkov, M. (2006). The error-minimization-based strategy for moving mesh methods. *Communications In Computational Physics*, **1**, 53–80. 2.4.2

Liskov, B. and Wing, J. (1994). A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, **16**, 1811–1841. 3.4.2

Liu, X.D. and Tadmor, E. (1998). Third order non-oscillatory central scheme for hyperbolic conservation laws. *Numerische Mathematik 79 (1998)*, **79**, 397–425. 5.5.1

Lohner, R., Morgan, K., Peraire, J. and Vahdati, M. (1987). Finite element flux-corrected transport (fem-fct) for the euler and navier-stokes equations. *International Journal for Numerical Methods in Fluids*, **7**, 1093–1109. 2.4.2

Lomax, H., Pulliam, T.H. and Zingg, D.W. (1999). *Fundamentals of computational fluid dynamics*. `http://www.fm.vslib.cz/~kmo/czech/vyuka/trp/book_cfd`. 2.3.2

MacCormack, R.W. (1969). The effect of viscosity in hyperbolicity impact cratering. *AIAA Paper*, 69–354. 2.4.3

Mao, D.K. (2000). Toward front tracking based on conservation in two space dimensions. *SIAM Journal on Scientific Computing*, **22**, 113–151. 2.3

Marti, J.M. and Muller, E. (2003). Numerical hydrodynamics in special relativity. *Numerical Hydrodynamics in Special Relativity, Living Reviews Relativity*, **6**, URL (cited on 09 March 2006): http://www.livingreviews.org/lrr–2003–7. 4.2.2

Martin, R.C. (2002). *Agile software development, principles, patterns, and practices*. Prentice Hall. 3.4.2

Marx, Y.P. (1994). Time integration for the unsteady incompressible navier-stokes equations. *Journal of Computational Physics*, **112**, 182–209. 4.3.3

Mingham, C.G. and Causon, D.M. (1998). Calculation of unsteady bore diffraction using a high resultion finite volume method. *Journal of Hydraulic Research*, **38**, 49–56. 2.3.2, 2.5, 2.5.1, 8.4.3

Miyoshi, T. and Kusano, K. (2005). A multi-state hll approximate riemann solver for ideal magnetohydrodynamics. *Journal of Computational Physics*, **208**, 315–344. 4.3.4

Mohamadian, A., Roux, D.Y.L., Tajrishi, M. and Mazaheri, K. (2005). A mass conservative scheme for simulating shallow flows over variable topographies using unstructured grid. *Advances in Water Resources*, **28**, 523–539. (document), 2.5.1, 2.5.2, 6.7.1, 2, 8.20

Murray, M.G. (2003). *An investigation into object-oriented approaches for hydroinformatics tools*. PhD Thesis, University of Newcastle. 3.4

Namin, M.M., Lin, B. and Falconer, R.A. (2004). Modellling estuarine and coastal flows using an unstructured triangular finite volume algorithm. *Advances in Water Resources*, **27**, 1179–1197. (document), 4.6, 4.4.2, 6.6.3

Navarro, P.G., Hubbard, M.E. and Priestley, A. (1995). Genuinely multidimensional upwinding for 2d shallow water equations. *Journal of Computational Physics*, **121**, 79–93. 5.6

Nessyahu, H. and Tadmor, E. (1990). Non-oscillatory central differencing for hyperbolic conservation laws. *Journal of Computational Physics*, **87**, 408–463. 5.5.1

Nordstro, J. and Gong, J. (2006). A stable hybrid method for hyperbolic problems. *Journal of Computational Physics*, **212**, 436–453. 2.2.3, 2.3.4

Nujic, M. (1995). Efficient implementation of non-oscillatory scheme for the computation of free-surface flows. *Journal of Hydraulic Research*, **33**, 101–111. 2.5, 2.5.1, 2.5.2, 6.6.3

Osher, S. and Solomon, F. (1982). Upwind difference schemes for hyperbolic conservation laws. *Mathematics of Computation*, **38**, 339–374. 2.5.1

Perla, J. and Ponnambalam, K. (1994). Multi-reservoir system simulation design using c++ and object-oriented programming. *Proceeding of ASCE National Conference on Hydraulic Engineering*, **2**, 1030–1043. 2.6

Preiss, B.R. (1998). *Data structures and algorithms with object-oriented design patterns in C++*. John Wiley & Sons. 3.2

Pudasaini, S.P. (2003). *Dynamics of flow avalanches over curved and twisted channels*. PhD Thesis, TU Darmstadt. 5.5.1

QIU, J. AND SHU, C.W. (2002). On the construction, comparison, and local characteristic decompositions for high order central weno schemes. *Journal of Computational Physics 183 (2002)*, **183**, 187–209. 5.5.1

QUIRK, J.J. (1994). An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two-dimensional bodies. *Computers and Fluids*, **23**, 125–142. 5.9.3

RASULOV, M., KARAGULER, T. AND SINSOYSAL, B. (2004). Finite difference method for solving boundary initial value problem of a system hyperbolic equations in a class of discontinuous functions. *Applied Mathematics and Computation*, **149**, 47–63. 2.3.2

RECKTENWALD, G.W. (2004). *Classification of physical problems and partial differential equations*. `http://www.me.pdx.edu/~gerry/class/ME448/weekly/pdf/PDEintro.pdf`. (document), 2.1

RENTON, A. AND PALMER, R. (2005). *A polace to stay, a place to live: The Oxfam shelter report (14 December 2005)*. Oxfam International. (document), 6.1

RICHTMYER, R.D. AND MORTON, K.W. (1967). *Difference methods for initial-value problems*. Wiley-Interscience, New York. 2.4.3, 5.3.2

ROE, P.L. (1981). Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, **43**, 357–372. 2.3.2, 2.5.1, 4.3.1, 4.3.3, 4.3.3

ROE, P.L. (1992). Sonic flux forumulae. *SIAM Journal on Scientific Computing*, **13**, 611–630. 4.3.3

ROE, P.L. AND PIKE, J. (1984). Efficient construction and utilization of approximate riemann solution. *Computing methods in Applied Science and Engineering (North-Holland)*. 4.3.3, 6.6.1

ROE, P.L. AND SIDILKOVER, D. (1992). Optimum positive linear schemes for advection in two- and three-dimensions. *Journal on Numerical Analysis*, **26**, 1542–1568. 2.4.2

ROOD, R.B. (1987). Numerical advection algorithms and their role in atmospheric transport and chemistry models. *Reviews of Geophysics*, **25**, 71–100. 2.4

Rossmanith, J.A. (2006). A wave propagation method for hyperbolic systems on the sphere. *Journal of Computational Physics*, **213**, 629–658. 2.3.4

Ruland, P. and Rouve, G. (1994). Advantages of object-oriented gis for the integration of hydraulic models. *Hydroinformatics 1994, Balkema, Rotterdam*, 253–260. 2.6

Sainsaulieu, L. (1995). Finite volume approximation of two-phase fluid flows based on an approximate roe-type riemann solver. *Journal of computational Physics*, **121**, 1–28. 4.3.3

Savic, S. and Holly, F.M. (1993). Dambreak flood waves computed by modified godunov method. *Journal of Hydraulic Research*, **31**, 187–204. 5.7

Schleisinger, S. (1979). Terminology for model credibility. *Simulation*, **32**, 103–104. (document), 8.1, 8.2

Schwab, C. (1998). *p- and hp- Finite Element MethodsTheory and Applications in Solid and Fluid Mechanics*. Oxford Science Publications, Clarendon Press, Oxford. 2.3.2

Shane, R.M., Zagona, E.A., McIntosh, D., Fulp, T.J. and Goranflo, H.M. (1996). Project object. *Civil Engineering, ASCE*, **1**, 61–63. 2.6

Sheu, T.W.H., Lee, P.H. and Lin, R.K. (2003). Development of a high resolution hyperbolic model on quadratic elements. *Computer Methods in Applied Mechanics and Engineering*, **192**, 5037–5056. 2.3.2

Shu, C.W. and Osher, S. (1988). Efficient implementation of non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, **77**, 439–471. 2.5.2, 4.3.5, 4.3.5, 4.3.5, 6.6.3

Simpson, D. (1967). *A Numerical Method of Characteristics for Solving Hyperbolic Partial Differential Equations*. Michigan University Press, Ann Arbor,. 2.3.2

Singh, V.P. (1996). *Kinematic wave modeling in water resources: surfacewater hydrology*. John Wiley and Sons Ltd. 2.3.2

Smith, G.D. (1985). *Numerical solution of partial differential equations: Finite difference methods (3rd edition)*. Oxford University Press. 2.3.2

Smolarkiewicz, P.K. (1984). A fully multidimensional positive definite advection algorithm with small implicit diffusion. *Journal of Computational Physics*, **54**, 325–362. 2.4

Solomatine, D.P. (1994). Object-orientation in hydroinformatics. *Hydroinformatics 1994, Balkema, Rotterdam*, 261–266. 2.6

Solomatine, D.P. (1996). Object-orientation in hydraulic modelling architectures. *Journal of Computing in Civil Engineering*, **4**, 125–135. 2.6

Sonar, T. (1997). On the construction of essentially non-oscillatory finite volume approximations to hyperbolic conservation laws on general triangulations: polynomial recovery, accuracy and stencil selection. *Computer Methods in Applied Mechanics and Engineering*, **140**, 157–181. 2.3.4

Souadnia, A., Soltana, F., Lesage, F. and Latifi, M.A. (2005). Some computational aspects in the simulation of hydrodynamics in a trickle-bed reactor. *Chemical Engineering and Processing*, **44**, 847–854. 2.3.4

Spekreijse, S. (1987). Multigrid solution of monotone, second order discretization of hyperbolic conservation laws. *Mathematics of Computations*, **45**, 15–21. 2.4.2

Strikwerda, J.C. (1989). *Finite difference schemes and partial differential equation*. Wadsworth and Brooks,California. 5.3.1

Swanson, R.C., Radespiel, R. and Turkel, E. (1998). On some numerical dissipation schemes. *Journal of Computational Physics*, **147**, 518–544. 2.4.2

Sweby, P.K. (1984). High resolution schemes using flux-limiters for hyperbolic conservation laws. *SIAM Journal of Numerical Analysis*, **21**, 995–1011. 2.4.1, 2.4.2

Szpilka, C.M. and Kolar, R.L. (2003). Numerical analogs to fourier and dispersion analysis: development, verification, and application to the shallow water equations. *Advances in Water Resources*, **26**, 649–662. 2.4.2

Tachikawa, Y., Ichikawa, Y., Takara, K. and Shiiba, M. (2000). Development of a macro scale distributed hydrological model using an object-oriented hydrological modelling system. *Hydroinformatics 2000, Iowa*. 2.6

Tadmor, E. and Tanner, J. (2003). An adaptive order godunov type central scheme. *Proceedings of the 9th International Conference on Hyperbolic Problems: Theory, Numerics, Applications (T. Hou & E. Tadmor eds.) Springer*, **CalTech, March 2002**. 5.5.1

Tai, Y.C., Noelle, S., Gray, J.M.N.T. and Hutter, K. (2002). Shock-capturing and front-tracking methods for granular avalanches. *Journal of Computational Physics*, **175**, 269–301. 2.4.3

Tedeschi, L.O. (2006). Assessment of the adequacy of mathematical models. *Agricultural Systems*, **89**, 225–247. 8.2

Thomas, J.W. (1995). *Finite difference schemes and partial differential equation*. Springer, New York. 5.3.1

Tomicic, B. and Yde, L. (1998). Integrated software for an integrated management and planning of urban drainage and wastewater systems. *Hydroinformatics 1998, Balkema, Rotterdam*, 465–471. 2.6

Toro, E.F. (1995). Direct riemann solver for the time-dependent euler equations. *Shock Waves*, **5**, 75–80. 5.9

Toro, E.F. (1999). *Riemann solvers and numerical methods for fluid dynamics*. Springer-Verlag, Berlin Heidelberg. 1.3.1, 2.3, 2.4.3, 1, 4.2.1, 4.2.2, 4.3, 4.3.1, 4.3.3, 4.3.4, 1, 4.5.4, 5.2, 5.7, 5.9.3, 6.6.2

Toro, E.F. (2001). *Shock-capturing methods for free-surface shallow slows*. Springer-Verlag, Berlin Heidelberg. 1, 4.3.1, 5.2, 1, 6.5, 6.6.1, 6.6.2

Toro, E.F. and Titarev, V.A. (2006). Musta fluxes for systems of conservation laws. *Journal of Computational Physics*, **216**, 403–429. 4.3.1

Toro, E.F., Spruce, M. and Speares, W. (1994). Restoration of the contact surface in the hll-riemann solver. *Shock Waves*, **4**, 25–34. 4.3.4

Trong, T.B. (2000). A parallel, finite-volume algorithm for large-eddy simulation of turbulent flows. *Computers & Fluids*, **29**, 877–915. 2.3.4

Tseng, M.S. (1999). Explicit finite volume non-oscillatory schemes for 2d transient free-surface flows. *International Journal of Numerical Methods in Fluids*, **30**, 831–843. 2.3.4

Valiani, A., Caleffi, V. and Zanni, A. (1999). *Finite volume scheme for 2D shallow-water equations application To a flood event in the Toce river*. CADAM. 2.3.4

Valiani, A., Caleffi, V. and Zanni, A. (2002). *Case Study: Malpasset dam-break simulation using a 2D finite volume method*, vol. 128. 2.5.1

Vanecek, S., Verwey, A. and Abbott, M. (1994). An exercise in object-orientation for water hammer and water distribution simulation in pipe networks,. *Hydroinformatics 94, Balkema, Rotterdam*, 267–272. 3

vanLeer, B. (1974). Towards the ultimate conservative difference scheme ii. monotonocity and conservation combined in a second order scheme. *Journal of Computational Physics*, **14**, 361–370. 4.3.1, 5.9.2.4

vanLeer, B. (1977a). Towards the ultimate conservative difference scheme iii. upstream-centered finite-difference schemes for ideal compressible flow. *Journal of Computational Physics*, **23**, 263–275. 5.9

vanLeer, B. (1977b). Towards the ultimate conservative difference scheme iv. a new approach to numerical convection. *Journal of Computational Physics*, **23**, 276–299. 2.4, 2.5.1, 5.9

vanLeer, B. (1979). Towards the ultimate conservative difference scheme v. a second order sequel to godunov's method. *Journal of Computational Physics*, **32**, 101–136. 2.3, 2.4.1, 2.4.2, 2.4.3, 5.1, 5.9, 5.9.3

Vazquez-Cendon, M. (1999). Improved treatment of source terms in upwind schemes for the shallow water equations in channels with irregular geometry. *Journal of Computational Physics*, **148**, 497–526. 2.5.2

Venkatakrishnan, V. (1993). Implicit solvers for unstructured meshes. *Journal of computational Physics*, **105**, 83–91. 2.4.2

Venkatakrishnan, V. (1994). Parallel implicit unstructured grid euler solvers. *AIAA Journal*, **32**, 1985–1991. 2.4.2

Venkatakrishnan, V. (1996). Perspective on unstructured grid solvers. *AIAA Journal*, **34**, 533–547. 2.4.2

Venkatakrishnan, V. and Barth, T.J. (1989). Application of direct solvers to unstructured meshes for the euler and navier-stokes equations using upwind schemes. *AIAA-paper*, **89**. 2.4.2

Vollmer, D.B. (2003). *Adaptive mesh refinement using subdivision of unstructured elements for conservation laws*. Master Thesis, Department of Mathematics, University of Reading. 2.4.2, 5.9, 5.9.2.3, 5.9.2.5

von Neumann, J. and Richtmyer, R.D. (1950). A method for the numerical calculation of hydrodynamical shocks. *Journal of Applied Physics*, **21**, 232–237. 2.4.3, 5.3.2

Vukicevic, T., Steyskal, M. and Hecht, M. (2001). Properties of advection algorithms in the context of variational data assimilation. *Monthly Weather Review*, **129**. 2.4

Vukovic, S. and Sopta, L. (2002). Eno and weno schemes with the exact conservation property for one-dimensional shallow-water equations. *Journal of Computational Physics*, **,179**, 593–621. 2.5.2, 5.6

Wang, J.S., Ni, H.G. and He, Y.S. (2000). Finite-difference tvd scheme for computation of dam break problems. *Journal of Hydraulic Engineering*, **126**, 253–262. 2.4.2

Wang, Z.J. and Liu, Y. (2002). Spectral (finite) volume method for conservation laws on unstructured grids ii. extension to two-dimensional scalar equation. *Journal of Computational Physics*, **179**, 665–697. 2.4.2

Winterfeldt, D.V. (1980). Structuring decision problems for decision analysis. *Acta Psychologica*, **45**, 71–93. 1

Woodward, P.R. and Colella, P. (1984). The simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics*, **54**, 115–173. 5.7

Xing, Y. and Shu, C.W. (2006). High order well-balanced finite volume weno schemes and discontinuous galerkin methods for a class of hyperbolic systems with source terms. *Journal of Computational Physics*, **214**, 567–598. 2.3.2

Xu, K. (2002). A well-balanced gas-kinetic scheme for the shallow-water equations with source terms. *Journal of Computational Physics*, **178**, 533–562. 2.5.2, 6.7

Yee, H.C. (1987). Construction of explicit and implicit symmetric tvd schemes and their applications. *Journal of Computational Physics*, **68**, 151–179. 2.3

Zalesak, S.T. (1979). Fully multidimensional flux corrected transport algorithms for fluids. *Journal of Computational Physics*, **31**, 335–362. 2.4.2

Zhang, X.D., Trepanier, J.Y. and Camarero, R. (2000). A posteriori error estimation for finite-volume solutions of hyperbolic conservation laws. *Computer Methods in Applied Mechanics and Engineering*, **185**, 1–19. 2.3.4

Zhao, D.H., Shen, H.W., Tabious, G.Q., Lai, J.S. and Tan, W.Y. (1994). Finite-volume two-dimensional unsteady flow model for river basins. *Journal of Hydraulic Engineering*, **120**, 864–883. 2.3.4, 2.5

Zhao, D.H., Shen, H.W., Lai, J.S. and III, G.Q.T. (1996). Approximate riemann solvers in fvm for 2d hydraulic shock modelling. *Journal of Hydraulic Engineering*, **122**, 692–702. 2.5

Zhou, J.G., Causon, D.M., Mingham, C.G. and Ingram, D.M. (2001). The surface gradient method for the treatment of source terms in the shallow water equations. *Journal of Computational Physics*, **168**, 1–25. 2.5.2, 6.7, 6.7.1, 2

Zienkiewicz, O.C. and Ortiz, P. (1995). A split-characteristic based finite element model for the shallow water equations. *International Journal of Numerical Methods in Fluids*, **20**, 1061–1080. 8.10