Object-oriented software development applied to adaptive resolution control in two-fluid models

by

Jan-Hendrik Kruger

B.Ing. (University of Pretoria) 1997
M.Ing. (Potchefstroom University for CHE) 2000

Thesis submitted for the degree of

Philosophiae Doctor

in

Mechanical Engineering

in the

School of Mechanical and Materials Engineering

of the

North-West University, Potchefstroom

Promoter:

Professor C. G. de K. du Toit

Potchefstroom
2004

Abstract

The accuracy of two-phase flow simulations is mainly determined by the mathematical formulations and resolution of the computational mesh used in the numerical model. Selective mesh refinement improves the accuracy of results while maintaining low resource requirements. This study introduces arbitrees, a new recursive grid design that allows hierarchical refinement and coarsening to be applied on uniform and arbitrary grids.

An object-oriented simulation framework was developed that allows the mathematical equations to be written directly in the code, providing direct control over the behaviour of the flow model. Two-fluid models based on the volume of fluid method are implemented and coupled to the arbitree grid to investigate adaptive refinement in two-fluid simulations.

Test cases for shape advection in oblique, rotating and shear flow were used to validate the interface tracking capabilities of the adaptive system. The adaptive two-fluid methodology was tested by modelling the collapse of a liquid column. In all cases, the predicted results exhibited encouraging agreement with analytical solutions or experimental measurements.

Opsomming

Die akkuraatheid van twee-fase vloei simulasies word hoofsaaklik bepaal deur die wiskundige formulerings en die fynheid van die berekeningsrooster van die numeriese model. Selektiewe roosterverfyning verbeter die akkuraatheid van die oplossing en beperk die gebruik van rekenaarhulpbronne. Tydens hierdie studie is "arbitrees" ontwikkel, 'n nuwe rekursiewe roosterontwerp wat hierargiese verfyning en vergrowwing op uniforme of arbitrêre roosters toelaat.

'n Objek-georiënteerde simulasie raamwerk is ontwikkel wat mens in staat stel om wiskundige vergelykings direk in die program as kode te skryf en dus direkte beheer oor die model verleen. Twee-vloeier modelle is m.b.v. hierdie raamwerk ontwikkel en gekoppel aan die "arbitree" rooster objek sodat die effek van lokale verfyning op twee-vloeier simulasies ondersoek kon word.

Toetsgevalle is voltooi waar die beweging van sekere geometriese vorms in a skuins-, roterende- en skuifvloeiveld bereken is. Hierdie resultate is gebruik om die intervlak-volg-vermoë van die verfynings algoritme te toets. Die twee-fase vloei stelsel is getoets deur die klassieke probleem van 'n dam wat breek op te los. In al die gevalle was daar belowende ooreenkomste tussen die voorspelde resultate en analitiese oplossings of eksperimentele resultate.

Acknowledgements

I bring praise to the Lord for the opportunity to complete this project and for supporting me through the difficult times I experienced during the course of this study. Such a prolonged study cannot be completed without His Divine inspiration.

My heartfelt gratitude to my family and prayer group for their loving support, patience and motivation during this period.

I would like to thank my promoter, Prof. du Toit, for his guidance during the project and for financial support from the North-West University.

I am indebted to Louis le Grange who invested a lot of effort in the development of the FTK simulation framework.

My appreciation goes to Dr. Onno Ubbink for the insightful discussions on two-fluid modelling.

Finally, thanks goes to Rufus, Christo, Johan, Robert, Riaan, Peet, Quinten, Philip and Nico who shared countless cups of coffee and their fellowship as colleagues and dear friends with me.

Jan-Hendrik

Potchefstroom, Dec 2004

Contents

List of Figures

List of Tables

Nomenclature

Roman Letters (lower case)

a_P	central coefficient
a_{nb}	matrix coefficient of neighbour cell
c	Courant number
$\vec{d}$	vector between point P and the neighbour N
$d\vec{S}$	general area vector
f	face, point in the centre of a face
$\vec{g}$	gravitational acceleration
$\vec{k}$	non-orthogonal correction vector
k_γ	constant in the CICSAM convection scheme
$\vec{n}$	normal vector to the interface
n	number of faces of a control volume, generic counter
t	time
$\vec{x}$	coordinate vector of arbitrary point in the domain
x	x-coordinate of arbitrary point in the domain
y	y-coordinate of arbitrary point in the domain

Roman Letters (upper case)

$\vec{A}$	face area vector, pointing outwards from owner cell
$\vec{D}$	orthogonal part of the face vector
E	solution error
$\vec{F}_C$	convection boundary flux

$\vec{F}_D$	diffusion boundary flux
F	volumetric flux
$\vec{H}(\vec{U})$	transport part
$\vec{I}$	unit tensor
$\mathcal{L}$	linear interpolation factor
N	point in the centre of the neighbouring control volume
P	point in the centre of the control volume, pressure
Q_V	volumetric source
$\vec{Q}_S$	surface source
S_{α_P}	source term of the discretized indicator function
$\vec{S}_{\vec{U}_P}$	source term of the discretized momentum equation
$\vec{T}$	stress tensor for a Newtonian fluid
$\vec{U}$	velocity
V	vertex
V, Vol	volume
$\vec{U}$	velocity vector

Greek Letters

α	volume fraction, indicator function
β	CICSAM weighting factor for face value prediction
γ_f	CICSAM scheme weighting factor
δ	small, finite thickness
δt	small, finite time step
ϵ	limiter in refinement criterium
θ_f	angle between the normal to the interface and $\vec{d}$
μ	dynamic viscosity
ρ	density
ϕ	general variable

∂V	surface area of control volume
ζ	Correction factor in transfer equation

Super and subscripts

ϕ^T	transpose of a vector
$\tilde{\phi}$	normalized values
ϕ'	correction value
ϕ^*	guessed or predicted value
$\phi^\dagger$	pseudo value
ϕ^a	analytical value
ϕ^n	new value
ϕ^o	original value
ϕ_1	value for fluid one
ϕ_2	value for fluid two
ϕ_A	acceptor cell
ϕ_D	donor cell
ϕ_U	upwind cell
ϕ_{avg}	average value
ϕ_b	value on boundary face
ϕ_c	value for generic cell
ϕ_f	value on inner face
ϕ_i	value for counter i
ϕ_{nb}	value of neighbouring cell
ϕ_β	value related to the β weighting factor

Abbreviations

AMR	Adaptive Mesh Refinement
API	Application Programming Interface
BiCGM	Bi-Conjugate Gradient Method

CBC	Convection Boundedness Criteria
CCM	Computational Continuum Mechanics
CFD	Computational Fluid Dynamics
CICSAM	Compressive Interface Capturing Scheme for Arbitrary Meshes
CN	Crank-Nicholson
CSF	Continuum Surface Force
FEM	Finite Element Method
FTK	Flow Toolkit
FVM	Finite Volume Method
GUI	Graphical User Interface
MAC	Marker And Cell
MVC	Model-View-Controller
NS	Navier-Stokes
NVD	Normalized Variable Diagram
PDE	Partial-Differential-Equation
PISO	Pressure implicit with splitting of operators
PLIC	Piecewise Linear Interface Construction
SIMPLE	Semi-Implicit Pressure Linked Equations
SIMPLE-C	SIMPLE-Consistent
SIMPLE-R	SIMPLE-Revised
VOF	Volume Of Fluid
UQ	Ultimate Quickest

Chapter 1

Adaptive two-fluids: the state of the union

1.1 Background

The objective of this study is the development of an object-oriented software framework that enables adaptive resolution control during the simulation of two-phase fluid problems.

Two-phase flow simulations form part of the challenging field of multi-phase flow simulations in the discipline of modern computational fluid dynamics. The accuracy of simulations is mainly determined by two components of the numerical model. The first is the mathematical formulations used by the model and the second is the resolution of the discretization as determined by the resolution of the computational mesh. Because higher resolution grids (more grid points closer together) can give more accurate results, but require more resources, it is important to utilize methods that apply selective grid refinement only in regions of interest.

This study presents the development of object-oriented simulation software to model fluid flow, heat transfer and mass transport problems and the implementation of automatic grid refinement algorithms. The object-oriented code can, amongst others, solve two-phase flow problems for arbitrary geometries. The exact behaviour of the flow model is specified by directly writing the mathematical equations in the main execution routine of the code. This allows direct and easy control over all aspects of the model, without parts being hidden away in proprietary software libraries. The numerical formulations of the code use a face-based approach on regular Cartesian coordinates with a co-located variable arrangement. The grid refinement is handled by a custom object that encapsulates all the grid operations and data management. It can dynamically manage the memory allocations and refine any specified cell and store the hierarchy of the refinement (to allow for unrefinement up to the original grid geometry).

The three technology components described above each represents a field of study with numerous developments during the years. As study fields mature, new opportunities for synthesis between different fields emerge. The following sections give an overview of developments in the core components and identify the advantages and

disadvantages. The chapter closes with a section that describes the contribution of this study and the outline of the following chapters.

1.2 Do we need redesigned simulators?

The traditional design of software used for fluid dynamics simulations focused solely on solving the flow problem, whilst turning a blind eye towards ease-of-use and extensibility of the code for research and development purposes. To implement new numerical and mathematical models in an existing code, users often had to delve deeply into the programmatic details of the code itself, wasting effort that should rather be spent on improving the models.

The mind-boggling complexity of software used as accurate, multi-physics research tools, as noted by Oldehoeft (2000), needs to be tamed. Demands placed on programmers and researchers that match their skills lead to effective development cycles. The most practical solutions use teams of developers - every scientist and programmer a specialist in his/her own field - cooperating through *Application Programming Interfaces* (APIs) within a common software framework to develop simulations.

Modern principles of object-oriented design applied to the development of simulation software improves this situation. By abstracting the mathematical formulation of the model from its actual code representation, the researcher and programmer can focus on their individual strengths: model development for the researcher and code implementation for the programmer.

To illustrate how object-oriented design principles can improve development of simulation software, some basic terminology and principles from the software universe are explained. The remainder of the chapter describes available toolkits and features that emerge from their comparison with each other. The popular mesh refinement strategies and two-phase flow models are evaluated and suitable design choices are explained.

1.3 On object-oriented design

The origin of the class

Multi-physics computations depend on the assembly into matrix form and solution of various *Partial-Differential-Equations* (PDEs) describing the physical phenomena in the simulation. There is a plethora of methods by which the PDEs can be assembled and these methodologies form the basis of the science of *Computational Fluid Dynamics* (CFD) or in a broader sense the discipline of *Computational Continuum Mechanics* (CCM). Various sources such as Versteeg & Malalasekera (1995) and Ferziger & Peric (2002) give detailed descriptions of the numerical processes.

Two distinctly different approaches can be followed when designing CCM software. These paradigms are procedural programming and object-oriented programming and are strongly tied to the computer languages (and the evolvement of the languages itself) used to develop the software. Stroustrup (1991) argues that to have any meaning, the term "object-oriented language" should mean a language that actively *supports* object-oriented programming. Conversely, procedural-based languages then actively support a procedural paradigm. A distinction can thus be made between being able to implement object-oriented features by using exotic programming techniques and using another language with the features built-in.

Stroustrup (1991) further stresses that one paradigm and feature set is not necessarily better than another, but that any language's support for a paradigm should at least heed the following:

- All features must be cleanly and elegantly integrated into the language
- Solutions can be achieved by combining existing features without extra additions
- "Special purpose" and "use once" features should be kept to a minimum
- A feature should be such that its implementation does not impose significant overheads on components that do not use the feature
- The user should only need to know about the subset of the language explicitly used to write a program

Historically the procedural paradigm developed first, with Fortran being the pioneering language. Later, inventions like C and Pascal followed in the same tradition. Object-oriented languages like Smalltalk-80 (Gamma *et al.*, 1995), C++ (Stroustrup, 1997) and Objective-C/C++ (Apple, 2002) heralded a brave new age of objectification in code. Even the venerable Fortran has been upgraded with object-oriented features in its Fortan 90 version (Oldehoeft, 2000).

In Stroustrup (1991) and Stroustrup (1999) the creator of the C++ language explains the conceptual advances that take a language from being procedural to being object-oriented. The way in which a language supports these concepts in handling the data[1] and functions[2] determines its true nature:

Procedural programming *Decide which procedures you want; use the best algorithms you can find.* The focus is on designing the process and performing the algorithms that process the data. No definite grouping of data and/or functions takes place. [Fortran]

[1]The information managed by the software. Data (also known as variables) represents built-in primitive language types and any user-defined types like classes.

[2]The operations performed on the data. Functions represent algorithms encoded as different procedures in the software.

Data hiding *Decide which modules you want; partition the program so that data is hidden in modules.* Related data and procedures are typically grouped together in source files - an effective but crude way to create modules. [Pascal, C]

Data abstraction *Decide which types you want; provide a full set of operations for each type.* By creating user-defined types, primitive data and related functions can be grouped into classes, providing an encapsulation into a logical unit of the members[3]. [C++, Obj-C]

Object-oriented programming *Decide which classes you want; provide a full set of operations for each class; make commonality explicit by using inheritance.* The most powerful extension to data abstraction is inheritance. Through this action it enables a child class to inherit members from a parent class[4]. When different classes are used in a program, common members may be defined in a parent class with the children inheriting the common members and only defining their own specific members. Otherwise all classes must store copies of the same members, increasing overheads and hindering maintenance. The litmus test of applicability of object-oriented techniques to a situation is whether or not commonality between objects can be exploited and grouped into suitable base classes. If no commonality exists, data abstraction usually suffices to represent the data. [C++, Obj-C]

Knowing what defines an object-oriented system is part of the methodology. The rest is knowing how to decompose a system into objects. Strategies and patterns are presented by Gamma *et al.* (1995) that assist system designers in identifying object types, their responsibilities and how they work together. Many factors influence the decomposition of a system: reusability, granularity, encapsulation, dependency, flexibility, performance and evolution. The experienced designer can strike a balance between the conflicting factors and supply a flexible solution. The key factors influencing object choice in simulation software are:

Reusability New components must be able to reuse existing components without altering them and breaking working code.

Granularity Components vary in size and number and suitable representations should be small enough to allow flexible use of components. Granulariy should not be so fine that computational overheads from the increased object count destroy the advantage.

[3]Classes are per definition defined by the programmer as custom representations of data and related functions grouped together. A class is seen as a unique type of data, and it is possible to have several *instantiations* of a class in one software program. Members refer to either the data or functions related to a specific class. Members may also be other classes.

[4]The action is also known as subclassing or deriving a subclass from a base class or superclass.

Encapsulation Components connect through interfaces which hide and protect their internal data representation. This simplifies maintenance and redesign of internals if necessary.

Dependency Interactions between components are strictly controlled through their interfaces - in that way components cannot be left in an undefined state after an operation on the object.

Flexibility Foundation objects in a system must be flexible enough to be reused and combined in other components. Ideally, they can be used as base classes for several higher-level objects.

Performance and granularity are often opponents in the design. To maximize performance requires as few objects as possible with as uniform internal data representations as possible.

Evolution The goal is to design systems for the realities of today and the challenges of tomorrow. When the above are properly applied, systems can be adapted and improved to meet new requirements.

We will now take a look at previous designs of object-oriented simulation toolkits and identify some useful features.

Multi-physics toolkits

Applications designed as *simulation toolkits* are gaining popularity under general users and research spesialists. Offerings cover the gamut from single-purpose procedural programs with static grids to object-oriented multi-physics software with dynamically changing grids.

Toolkit-type applications rely heavily on object-oriented design ideas and because of that are not often developed in procedural languages. The concept of modular toolkits can be found in Fortran: one such library provides an application with specialized procedures to do mesh refinement (MacNeice *et al.*, 2000). Another environment offered by Filippone *et al.* (1999) enables the construction of distributed solutions to general problems. Despite the difficulty of implementing object-oriented features, Fortran is still popular for large-scale commercial applications with fixed feature sets and a procedural legacy. While it is true that most of these applications have multi-physics capabilities - it does not readily qualify them as research toolkits as users can only use the supplied features. Researchers must be able to formulate their own equations and by supporting that freedom, toolkits are invaluable and that sets it apart from other, fixed feature offerings.

Object-oriented design ideas are easily implemented in C++ and were initially used to improve applications' internal structures. Liu *et al.* (1996) offered the capability to use general equations, recast in a special form with the option of using *Finite Volume Methods* (FVM) or *Finite Element Methods* (FEM) for automatic

discretization. As the technology matured, the way in which users interact with the toolkits started to change as well. Users could write code that mimics the actual equations, as described by Weller *et al.* (1998), Munthe & Langtangen (2000) and Ollivier-Gooch (2003). The leverage of modular design enabled the solution of generic problems ranging from fluid dynamics, described by Weller *et al.* (1998) and Boivin & Ollivier-Gooch (2004) to metal-forming and impact analysis done by Pantalé *et al.* (2004). Frameworks are also available for adaptive refinement on parallel processor machines, described by Castaños & Savage (1999b) and Filippone *et al.* (1999). Recent progress in frameworks reported by Stewart & Edwards (2004) offer a combination of object-oriented multi-physics modelling with adaptive meshing on super-scalar parallel machines. The advantages of modular toolkits in education have also been illustrated by Friedrich (1999) where the ability to easily define new simulation setups offer powerful learning incentives for students.

Discussion of features

Table 1.1 compares the key features of available frameworks. The frameworks chosen offer a representative sample of the evolution of simulation frameworks. The range and capabilities of frameworks differ, together with the computational requirements for these frameworks. Some designs are aimed at single processor machines, while others are tuned for performance on massively parallel, shared memory clusters.

A short discussion of the categories in Table 1.1 explains the different features:

FVM and FEM The discretization process transforms the equations into a suitable form for the numerical solver. Ideally the framework should be so modular that both finite volume and finite element discretizations can be mixed in a single simulation. Most of the frameworks are modular in design, but still based on a specific methodology. In general, only the largest frameworks can offer multiple discretization options.

Uniform, orthogonal grids Grids are in logically structured Cartesian form, with uniform spacing between computational nodes and local element axis that are orthogonal. All the frameworks at least have this capability in two-dimensions. The frameworks that can use unstructured grids have full three-dimensional designs.

Unstructured grids The grids can be unstructured, non-orthogonal tetrahedral or hexagonal shapes. The advanced frameworks offer a wider selection of cell types, e.g. shells and beams for FEM analysis.

Unstructured (un)refinement Commonly used for tetrahedral elements where grids are generated via energy or Delauney and Voronoi methods. By manipulating the locations of the seeding points for the grid generator, the local resolution of the grid can be controlled by regenerating elements. In the same manner,

Table 1.1: Comparison of different simulation frameworks.

	Previous frameworks									
Features	1	2	3	4	5	6	7	8	9	10
Finite volume method	·	·	●	–	·	●	●	●	●	●
Finite element method	●	●	●	-	●	●	·	●	·	·
Uniform orthogonal grids	●	●	●	●	●	●	●	●	●	●
Unstructured grids	●	·	·	●	●	●	●	●	●	●
Steady state	●	●	●	–	●	●	●	●	●	●
Transient	●	●	·	–	●	●	●	●	●	●
Unstructured (un)refinement	●	·	·	·	·	●	·	●	·	·
Hierarchical (un)refinement	·	●	●	●	·	●	·	●	·	●
Free-form PDEs	·	·	·	–	·	●	·	·	●	●
Proprietrary-format PDEs	●	●	●	–	●	·	●	●	·	·
Serial computer	·	·	●	·	●	·	●	·	●	●
Parallel shared memory	●	●	·	●	·	●	·	●	●	·
Fortran-90 language	·	●	·	●	·	●	·	·	·	·
C++ language	●	·	●	·	●	●	●	●	●	●

Legend:
● feature present
· feature unsupported
– not relevant to framework

1: PARED, Castaños & Savage (1999b)
2: Filippone *et al.* (1999)
3: AdaptC++, Liu *et al.* (1996)
4: PARAMESH, MacNeice *et al.* (2000)
5: Munthe & Langtangen (2000)
6: ACL Project, Oldehoeft (2000)
7: ANSLib, Boivin & Ollivier-Gooch (2004)
8: SIERRA, Stewart & Edwards (2004)
9: FOAM, Weller *et al.* (1998)
10: Xtree/FTK, developed in the current study

to unrefine the grid, the seeding points are moved further from each other and a coarser grid is generated. Instead of regeneration, existing elements can also be combined into larger elements of the same type. It is difficult to regain the original mesh before refinement, since elements change in quantity, size and location.

Hierarchical (un)refinement The domain is divided into a base grid containing enough cells to properly define boundaries and obtain a solution. Based on a certain refinement criterion or error estimation cells are refined-but always within the confines of the original cell. Refinement can occur recursively (i.e. smaller cells in a refined cell may be further refined) and enough levels of unrefinement always yield the original base level grid. Most toolkits only offer refinement as part of their steady-state algorithms.

Steady and transient problems With proper object-oriented designs an extension to transient models is straight-forward and this is reflected in the fact that all the frameworks, except the oldest one reviewed, have transient capabilities. Accurate inter-mesh interpolation schemes are needed when transient problems and adaptive meshing are coupled in the same simulation.

Free-form PDEs Users have the advantage of entering their equations in a form that very closely resembles the mathematical syntax and notation. This capability can either be implemented as a meta-language in the pre-processor or the object design can be such that the application is programmed with objects that mimic the equations.

Proprietrary-format PDEs Users must rewrite their equations in the specific format required by the framework. It simplifies equation handling in software, but can make it more difficult to find coding errors if the translation of equations is done incorrectly. It increases the learning curve of the framework as users are forced to learn an additional paradigm before using the framework.

Computer type The large frameworks are specifically developed for parallel cluster machines. The scale of problems that can be solved are orders of magnitude larger than is possible on serial machines (a.k.a. the humble desktop workstation). But, it requires access to suitable hardware, finances and trained researchers that can operate that scale of simulation projects. For individual scientists investigating smaller problems, such a large system is not always necessary, leaving room for simplified frameworks that can adequately operate on desktop machines.

Language Most of the frameworks are programmed in C++, due to their object-oriented nature. One instance of framework design in pure Fortran-90 was found (Oldehoeft, 2000) and another of a framework that can translate Fortran

and C++ code into its own intermediate C++ code (Stewart & Edwards, 2004). Again, the dual capability is only available on large-scale frameworks.

Suitable features

From the comparison between different frameworks in Table 1.1, a number of desirable features were identified that shaped the design of the Xtree/FTK system developed in this study. Pre-requisites for the project were that the framework initially use a finite-volume methodology with unstructured grids that can dynamically be refined and unrefined. Practicality and previous experience of the author further dictated that development is done for a serial machine environment and in C++. After factoring in these requirements, the only area left with decisions is the manner with which the mathematical environment in the software is manipulated by the user.

Analysis of the two FVM-based toolkits, ANSLib (Ollivier-Gooch, 2003) and FOAM (Weller *et al.*, 1998), indicates different approaches to the representation of equations. The first approach, used by ANSLib, is to have an internal representation of the generic transport equation. The user can tailor the generic equation for specific problems by specifying the functions that calculate the fluxed quantities across the faces. Different physics classes and field combinations can be used for multi-physics problems, and the fields have internal dependency trees that ensure the most recent values are always used in the solution process. By assembling several tailor-made equations a system of equations can be modelled. Ollivier-Gooch (2003) provides a good argument why FVM-based toolkits offer a much cleaner abstraction between physics and numerics than FEM-based ones. With the finite element methodology, researchers need to have an intimate knowledge of numerical details (e.g. how to chose correct element types for a problem) and be able to enforce their choices, to create successful simulations. In the finite volume method, problem physics enter the numerical system through the calculation of the face fluxes. The design advocated by ANSLib only requires users to be able to define the flux functions and further numerical details are automatically handled by the toolkit.

The second approach, used by FOAM, is to provide the user with operators that mimic mathematical notation and operate on scalar, vector or tensor fields. To model multi-physics, the user simply creates the new fields (scalar, vector or tensor), assemble the fields into equations by using the operators and prescribe the correct solution sequence. To solve simultaneous equations, fields are assembled into a higher-order variable (e.g. three different scalar fields are assembled as a vector object) and solved in the normal way. The advantage for the researcher is that a more natural form of the equations and algorithms can be used. New ideas can easily and accurately be investigated and errors in the formulation can readily be tracked down because of the similarity of code with the original mathematical notation.

A shortcoming indicated by the creators of ANSLib is that the formulations and current internal system design use the absolute values of fluxes on the faces. This

may seem insignificant, but it prevents the framework from operating with dynamically adapting grids where the flux direction must be taken into account at all times. The field operator designs used in FOAM already take into account directional fluxes and only requires the correct connectivity between numerical points to use dynamic meshes.

The discussion suggests that the design paradigm used by FOAM is the best candidate to meet the pre-requisites of a multi-physics FVM code capable of extension to adapting, unstructured grids. In Chapter 5 the layout and design of the FOAM-inspired object-oriented operators in FTK are presented.

After reviewing existing toolkits and making design decisions for the numerical side of the framework, the following sections describe the popular methodologies for adaptive mesh refinement and two-phase flow modelling.

1.4 Adaptive resolution control

Motivation for mesh refinement

The purpose of *Adaptive Mesh Refinement* (AMR) is to increase the accuracy of the numerical solution in selected areas of the domain without adversely effecting the total resource requirements of the problem. In fact, when properly applied, local refinement results in much more accurate solutions with orders of magnitude less computational points than uniform grids of comparitive resolution.

Computer graphics

The concept of adaptive refinement was first used in computer graphics, where it offered more effective storage solutions for primitive graphic elements. Classical graphics textbooks like Foley *et al.* (1990); Hearn & Baker (1994) describe how local refinement is used in spatial partitioning schemes, ray-tracing and other image processing techniques. Simulation codes using AMR present unique opportunities to improve the speed of visualization of datasets: Roettger *et al.* (2002), Weber *et al.* (2003) and Del-Pino (2004) describe how datasets can be analyzed, starting with the coarsest mesh elements to discard unnecessary elements (and all their subelements) before performing expensive visualization calculations.

Other engineering problems

Bohn & Thornton (1994) describes the use of AMR in robot navigation, where the space surrounding a robot is recognized through a process of laser sculpting. The laser range finder data is stored in adapted grids and processed to reconstruct a three-dimensional virtual environment. The reconstructed environment is navigated by the robot, guided by artificial intelligence. In structural analysis, refinement

is used for more accurate plasticity calculations (Barry *et al.*, 1998), improved dynamic stress modelling (Selman *et al.*, 1997), metal forming analysis (Kwak & Im, 2003) and in crack propagation analysis[5], as described by Bouchard *et al.* (2003) and Phongthanapanich & Dechaumphai (2004). The implementation of adaptive methods for magneto-hydrodynamic flow is described by Ziegler (1998) with a subsequent extension to three dimensions (Ziegler, 1999). Solvers for electromagnetic fields are available on adaptive Cartesian grids (Wang *et al.*, 2002) and on adaptive multigrids[6] (Hoppe, 2004).

Heat transfer analysis has also benefitted from local improvements in accuracy, as noted by Vierendeels *et al.* (2004). Their method is based on a multigrid approach. Radiation modelling is another heat transfer field that benefitted from local refinement. As illustrated by Jessee *et al.* (1998), the radiative transport equation can be modelled on adaptive grids. The gains in solution performance are determined by the distribution of emissive power, with the largest improvements in models with large gradients in the power. Mencinger (2004) investigated the use of adaptive models to predict melting in two-dimensional cavities. An industrial example from Cler (2002) is shock modelling for a cannon muzzle brake, where the same accuracy - using adaptive refinement with 135,000 cells - was achieved as resolving the entire domain using 133 million cells. An improvement by a factor of 1,072!

Industrial and general fluid-flow

The use of adaptive mesh refinement in fluid-flow simulations has been widely documented. In aero-dynamics, Hentschel (1998) used mesh refinement based on vorticity content to model flow over transonic delta wings. Examples are given by Baker (1997) that show how the numerical grids adapt to the shockwaves that form on a transonic wing. Oceanic flow adjacent to the shoreline can be modelled with great success by using adaptive shallow water approaches, noted by Ivanenko & Muratova (2000) and Burchard & Beckers (2004).

Industrial flows are usually inside containers or some form of restricted domain. In these situations, accurate modelling of the inside and outside boundary layers is important to obtain accurate velocity and pressure profiles and heat transfer values. According to Hyman *et al.* (2000) and Carey *et al.* (2004) grids that adapt to the boundary regions are an effective solution to this problem. In chemical and process engineering, boundary layer flow inside reactors play an important part in the outcome of a chemical reaction. Fraga (1998) modelled fixed-bed reactors using

[5] As a matter of interest, AMR crack propagation analysis shares some common traits with the modelling of an advancing two-fluid interface. In both fields, the region of interest is defined by a dynamic discontinuity in material properties. Local accuracy in numerical solutions in time and space is of utmost importance for an accurate prediction of the movement of the interface.

[6] In multigrid methods, the solution process is accelerated by using meshes of increasing complexity over the whole domain and propagating the results from fine grids to the coarser levels and back again. Adaptive refinement takes it a step further and combines the grids of different resolutions into a single grid.

AMR and obtained improvements in the calculation of the energy balance in the bed.

General fluid-flow simulations depend on the simulation of the *Navier-Stokes* (NS) equations. Various methods to solve adaptive NS equations on adaptive grids are required because methods are intrinsically linked to the different types of elements used in models. Adaptive refinement has been illustrated on triangle-tree grids by Wille (1996) and Plaza *et al.* (2000), on adaptive blocks by Pember & Bell (1995), Neeman (1996) and Stout *et al.* (1997) and on Cartesian grids (Wang, 1998). Implementations on hybrid[7] grids are presented by Kallinderis (1996), Khawaja *et al.* (2000) and Zheng & Liou (2003a) and Zheng & Liou (2003b). The hybrid grids strike a balance between the numerical efficiency and stability of structured Cartesian grids fitted to the boundary of the domain, while using unstructured elements on the inside. An interesting alternative to this approach can be found in Gerris-trees (Popinet, 2003), which use structured Cartesian cells over the whole domain and boundaries are represented with marker cells that have partial values, depending on the angle of the physical boundary at that point. Simulations of adaptive multiphase flows on regular Cartesian grids have also been presented by Greaves (2001) and Malik (2004) and on interface aligned grids by Ginzburg & Wittum (2001).

From the selection of test cases presented, it is clear that adaptive mesh refinement has unlimited potential as a technique to improve simulation accuracy and time-to-solution for simulations in almost any computational field. The following sections describe the typical refinement cycle and provide examples of different refinement strategies.

Typical refinement cycle

Grid refinement methodologies can be static or dynamic (adaptive) in nature and can be applied to both steady-state and transient problems. Static refinement means that the grid is subjected to a single refinement step based on past experience or a previous flow solution, e.g. grid refinement behind a flow obstacle to capture vortices, or a finer grid to capture a boundary layer. For the duration of the simulation the mesh is not modified again. In fact, static refinement is usually applied during the normal process of initial mesh construction and as part of the main grid.

Adaptive refinement modifies the numerical grid during the simulation process. The process of deciding where to refine is usually automatic and based on a-posteriori error estimates of fluid variables (Jasak & Gosman, 2000a) or definitive flow features like pressure gradients (Kwon & Jeong, 1996), shock waves (Lim *et al.*, 2001) or fluid interfaces (Malik, 2004).

[7]Hybrid grids can contain mixed element types, such as tetrahedral, prismatic or hexagonal shapes. The physical geometry and flow conditions of the problem determine the optimal element shape to use.

For steady-state problems, refinement is applied based on the converged solution at every pseudo time-step, while transient problems substitute the pseudo time-step with the real time-step. To better illustrate the method, typical steps are listed:

1. Construct the original grid with the coarsest resolution that accurately[8] captures the boundaries and converges to a realistic solution.
2. Obtain the initial converged solution.
3. Decide on parameters such as the refinement level of target cells and how much their neighbour cells are refined.
4. Based on certain criteria (e.g. pressure gradient, scalar concentration, numerical errors) refine the target cells and neighbours.
5. Transfer the solution from the old grid to the new grid as initial values.
6. Obtain a new solution on the refined grid.
7. Test the new solution against the criteria and decide if the refinement was sufficient. If not, repeat the previous three steps.
8. By constantly testing the solution against the criteria, the grid resolution automatically adapts to the solution characteristics by refining and unrefining cells where needed.
9. End the simulation when the specified time-step is reached and cells no longer need to be modified to satisfy the refinement criteria.

Improving refinement algorithms

Mesh refinement is a computationally expensive process. It involves the evaluation of refinement criteria, the creation of new elements and the mapping of the old solution to the new grid. In addition, the dynamic nature of the problem hinders the direct use of efficient linear data structures.

Reducing the refined cell front

The actual process of mesh refinement can be improved in several ways. The first (and most important) is to reduce the number of elements that must be refined. By reducing the number of cells to be refined, an accumulative advantage is obtained when applying other optimization steps. An example of the use of error estimates

[8]The solution on the coarse base grid must be investigated for sufficient accuracy on all the variables in the system. It is possible in two-phase flow models to accurately represent the interface on a grid that is too coarse to allow for realistic solutions of flow features. This results in transient calculations that converge to incorrect flow patterns and inaccurate interface movement.

in turbulent flow is given by Jasak & Gosman (2000c), which aggressively restricts refinement to areas of high numerical error. In the case of two-fluid modelling, instead of basing refinement solely on the interface location, the curvature can be employed for a more effective strategy. As noted by Malik (2004), an accurate interface position can still be obtained by using less cells in areas of low curvature and concentrating cells in regions of high curvature.

Remapping and reconstruction

Whenever a new grid is formed, the existing solution must be mapped between the old and new grids. The remapping strategy can be tailored to the type of grid and method of refinement. Hierarchical grids lend themselves to element-based remapping where the solution of a refined parent element is remapped to the new children elements. The new distribution of a variable is determined by the local gradient of the variable in the parent cell, as described by Muzaferija & Peric (1998), Jasak & Gosman (2000b) and Ferziger & Peric (2002).

Unstructured grids without recursive refinement are partly regenerated to obtain more optimal grids. Remapping algorithms are consequently designed to operate on the whole domain. Popular procedures for field-based remapping are based on least-squares-fitting methods, see Mavriplis (2003) or radial basis functions as developed by Li & Chen (2002). A novel idea presented by Margolin & Shashkov (2003) is to use local flux balances to obtain the new interpolated solution.

Field-based interpolation is potentially faster than element-based interpolation for fields where large numbers of elements change, because more uniform data structures can be used that can exploit more efficient computing routines. Element-based routines have the advantages that the computing effort scales with the number of elements used and that interpolation costs can be amortized during the process of refining the cell, negating the need for an explicit interpolation step. Field-based interpolation obtain smooth distributions across the whole domain, while element-based methods enforce local conservation of properties.

Parallel implementations

On large systems, parallel versions of refinement strategies are very popular, as proven by the large amount of literature on the subject. Tetrahedral elements are common in grids for finite-element based codes, making it the element of choice for development efforts on unstructured grids: Globisch (1995), Barry *et al.* (1998), Castaños & Savage (1999a) and Stewart & Edwards (2004) describe suitable techniques and frameworks. In contrast to tetrahedral unstructured meshes, hierarchical adaptive meshes almost exclusively use structured hexagonal (cube) shaped cells. The uniformity of the hexahedral elements can be exploited to provide efficient parallel implementations, such as those described by Balsara & Norton (2001) and Kohn (2001).

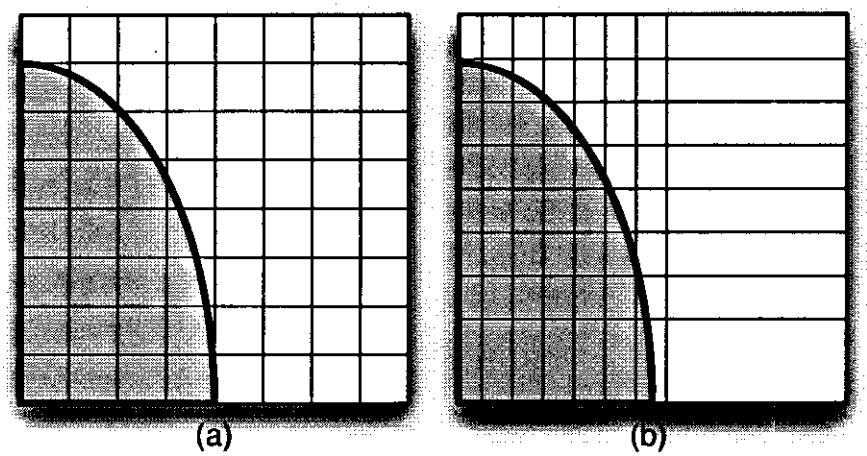


Figure 1.1: (a) Original grid (b) r-refinement moves grid points

Different adaptation strategies

Mesh movement or mesh enrichment?

Grid adaptation strategies can broadly be divided into two distinct categories (Baker, 1997). The first is mesh movement (r-refinement), where existing grid points are moved to regions where higher accuracy is required. The advantage of this method is that the total number of grid points in the domain stays the same, enabling parallel decomposition, connectivity calculations and memory allocation to be done once per simulation. The disadvantage is that since points are only moved, the elements defined by those points change and this can result in malformed meshes that introduce additional numerical errors. The second is mesh enrichment (or h-refinement), where grid points are added or subtracted according to the level of accuracy desired at a certain location in the mesh.

An example of r-refinement on a structured grid is illustrated in Fig. 1.1. Grid modifiers evaluate the distances between node points weighed against error estimates to obtain a grid that is refined in certain regions and smoothly transforms to the coarse regions. Tetrahedral elements are the common type used in unstructured meshes, as the element topology enables robust grid regeneration with automatic algorithms. Unstructured hexahedrons are more difficult to generate automatically and algorithms that generate hexahedrons are in any case based on tetrahedral elements, which are combined to obtain unstructured hexahedrons. Mesh movement on unstructured hexahedron meshes are not advised (Baker, 1997), as this easily results in badly formed meshes, especially in three dimensions.

Both structured and unstructured, tetrahedral and hexahedral meshes are good candidates for the h-refinement shown in Fig. 1.2. The grid enrichment is done by supplying seed points and generating additional elements around the seed points.

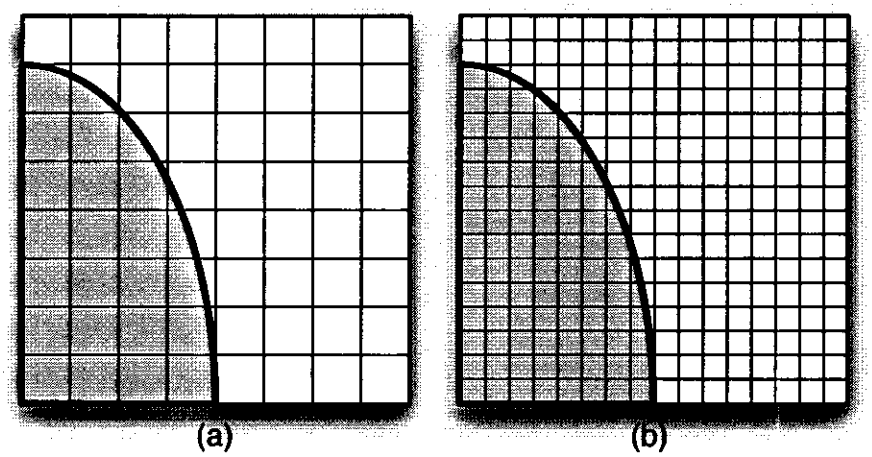


Figure 1.2: (a) Original grid (b) h-refinement adds extra elements

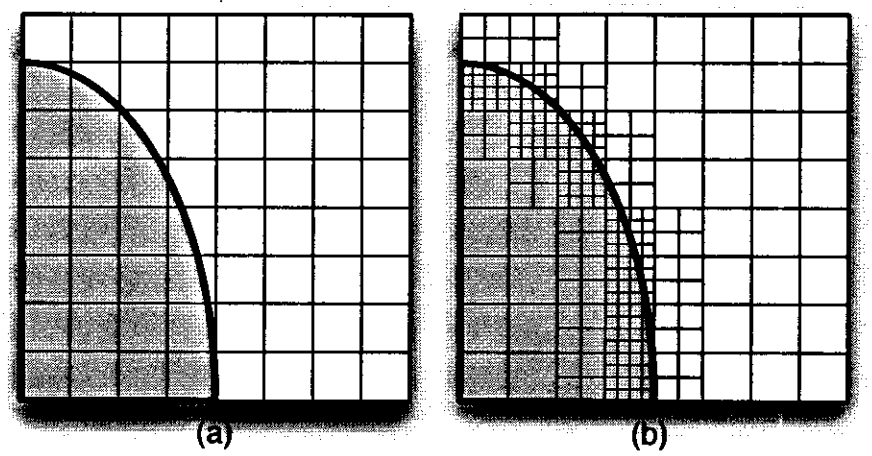


Figure 1.3: (a) Original grid (b) Adaptive block refinement with regularization

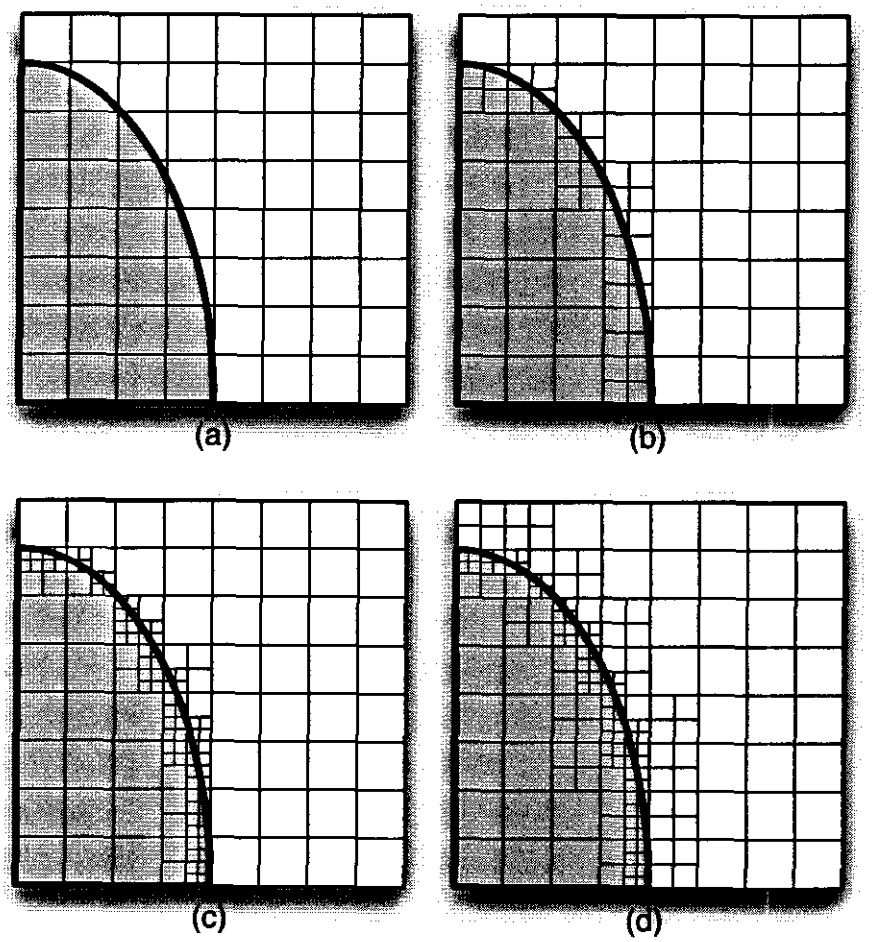


Figure 1.4: (a) Original grid (b) Single level of refinement (c) Two levels of refinement (d) Final grid after applying regularization

Adaptive block refinement

While r- and h-refinement operations are applied to the complete domain, adaptive methods only refine selected cells. When adaptive mesh blocks, shown in Fig. 1.3, are used, a cell is refined only once. Neighbouring cells are refined with a mesh block at half the resolution of the primary cells to obtain a more uniform transition between element sizes.

Hierarchical refinement

With *hierarchical (or recursive) adaptation, elements are subdivided into smaller*, similarly shaped elements. Fig. 1.4 illustrates a grid after with one level of refinement and then the same grid after two levels of refinement, but no regularization.

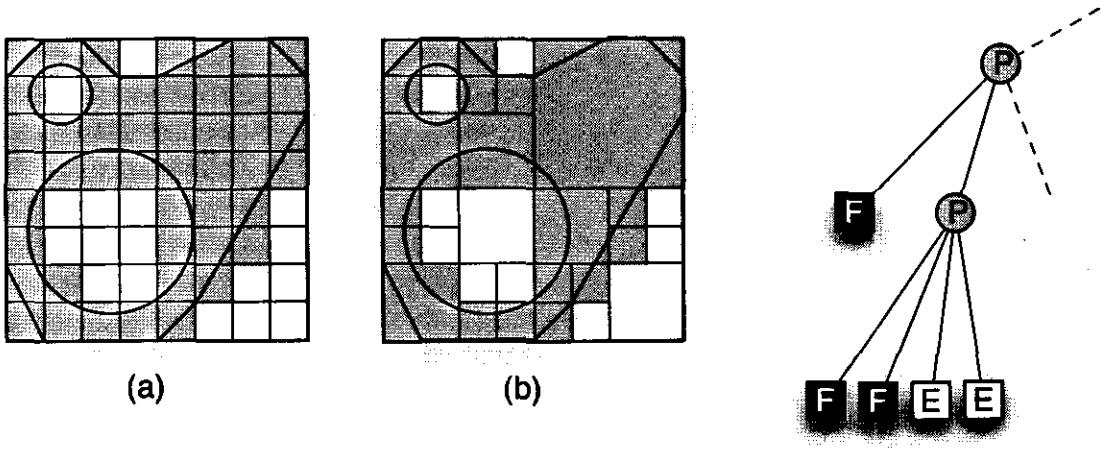


Figure 1.5: An object represented using (a) spatial-occupancy enumeration (b) a quadtree. An excerpt from the quadtree data structure for (b), where F = full, P = partially full, E = empty. (Foley *et al.*, 1990)

The last frame shows the smoothing effect of regularization. The advantages of recursion are that new element shapes are controlled by the form of the basis element and by deleting the subcells, the original grid distribution can be obtained. The biggest disadvantage is the dynamic nature of the grid, which forces parallel decomposition and connectivity between elements to be recalculated every time the grid is modified. The dynamic memory allocation also adds extra overhead to the per-iteration calculation costs.

Octree and quadtree representations

Hierarchical adaptation is very effectively handled by way of octree data representations. *Octrees* are the hierarchical variant of general spatial-occupancy enumeration and originated in the computer graphics field. Foley *et al.* (1990) describes the origin and different applications of quadtrees in image processing and octrees in object visualization. The potential of tree-based element decomposition for mesh refinement was identified and extended for use in numerical methods by various researchers like Krishnamoorthy *et al.* (1995), Yiu *et al.* (1996) and Khokhlov (1998).

The divide-and-conquer power of binary subdivision is the force that drives octrees as effective datastructures. Fig. 1.5 depicts the basic design of quadtrees and can directly be extended to octrees by subdividing volumes into octants instead of areas into quadrants. Where a quadtree represents an area, it is divided into four equal quadrants. A quadrant can have one of three states - full, empty or partial - depending on how much of the original area intersects the quadrant. If the area covers the quadrant completely, it is marked *full*; with no coverage it is marked as *empty*. If the area only covers a percentage of the quadrant, it is marked as *partial*.

Partial quadrants are subdivided and re-evaluated. The accuracy of the representation is controlled by how many levels of subdivision are allowed and the cut-off values that determine when a quadrant is full or empty. One of the most important grid operations is to determine the inter-connectivity of elements. Neighbour detection in quadtrees is accomplished by using a fixed numbering scheme and traversing the branches of the tree to find those elements that are possible neighbours. Quadrants are evaluated as neighbours based only on their numbered labels, and not on any geometric property. This makes for a fast and efficient data representation with regards to storage, but one that is structured by design and limited to trees with fixed branches (usually four or eight elements). Full descriptions of quad and octrees as datastructures and the numbering schemes used for neighbour detection can be found in Foley *et al.* (1990), Yiu *et al.* (1996), Malik (2004) and Greaves (2004).

Regularization

Regularization is the finishing touch applied to a newly refined grid. Merely adapting the region of interest by adding elements does not necessarily result in a numerically well-behaved grid. Especially as the levels of refinement increase, it happens that smaller elements are connected with much larger elements. In Fig. 1.4 the process of regularization is illustrated. This involves an examination of the new grid, and if the level of subdivision between neighbour elements differ by more than one level, the element with the lower level is refined. This process is repeated until no more elements need to be refined. By enforcing gradual changes in mesh resolution, regularization creates a smooth transitional area between the refined regions of the domain and the unrefined spaces.

Mesh refinement is a proven technique for the local improvement of numerical simulations and in this section different approaches were introduced and evaluated. The last section in this chapter reviews the current methods for modelling two-phase flows and examines ways in which adaptive refinement and two-fluid modelling can be combined.

1.5 Modelling two-phase flows

Introduction

Two-phase flows occur commonly in nature, industrial processes and most modern inventions. These range from the bubbles that stream from water aerators, a kettle boiling, water slug flow in pipes and cavitation regions in turbines to the free surface motion of fuel sloshing in a tank or waves in a harbor. Because of the rapidly changing fluid properties in a two-phase region, there are usually unwanted and unpredictable side effects. In the case of industrial equipment, partially filled pipes (Anon, 1995a:14) and cavitation in pumps (Anon, 1995b:42) are two main culprits that decrease the effectiveness of machines and increase maintenance costs. Free

surface modelling is important in the maritime industry for ship design (Azcueta *et al.*, 1999) and even influence disaster management by assisting flood prediction (Alcrudo, 1999). These examples illustrate the need to be able model two-phase flow situations accurately over a wide range of problem scales. Improved models lead to a better understanding of physical processes so that designs for equipment and structures may be improved.

Two-phase flows, and especially flows with free surfaces, is a challenging class of flows with moving boundaries. Many mehods have been proposed to model the location and orientation of the free surface (or the interface region for enclosed flows). According to Ferziger & Peric (2002) the methods may be divided into two main groups:

Interface-tracking methods treat the free surface as a sharp interface.

Interface-capturing methods do not define the interface as a sharp boundary.

Interface-tracking methods treat the free surface as a sharp interface with a definite boundary. The motion of the interface is accurately tracked by moving the boundary together with the changing interface. The boundary may be defined by massless marker particles or mesh points that are moved to align with the surface.

Interface-capturing methods do not define the interface as a sharp boundary. Instead, the combined motion of the two fluids is calculated on a fixed grid where the fluids are represented by an indicator function or massless particles. The interface is implicitly defined by those cells that contain a fraction of the indicator function or marker particles. Hybrid methods (Enright *et al.*, 2002; Li *et al.*, 1999) combine the strengths of both approaches (but with higher computing costs): the movement of the interface is calculated with a capturing method and an accurate surface is reconstructed using interface capturing.

The next section gives a short overview of different techniques and their advantages and shortcomings. For a more detailed review of these methods the reader is referred to Ubbink (1997).

Interface-tracking methods

Interfacial flow simulations depend on the accuracy of the surface representation in the model. Interface-tracking methods offer the highest accuracy of all the interface models, because discrete marker entities indicate the position of the interface at all times. There are three basic methods to define the marker entities.

Marker particles on surface

Interconnected marker particles are seeded on the interface. Fig. 1.6 illustrates this concept. The motion of the particles are calculated using a discrete Lagrangian approach and fluid motion is determined with continuum Eulerian equations. The

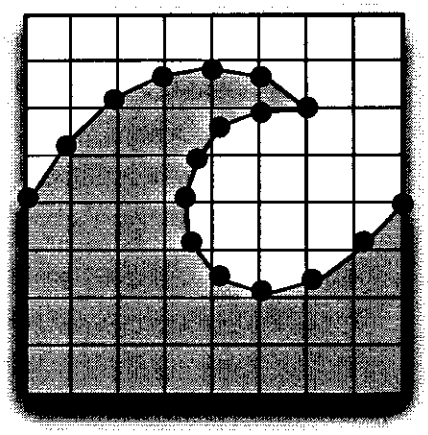


Figure 1.6: Marker particles on the surface

location of the free surface is defined by the position of the particles and by fitting a suitable curve function through the markers, a smooth representation of the interface can be obtained (Idelsohn *et al.*, 2001; Tryggvason *et al.*, 2001).

The linked particles tracking the surface place severe restrictions on the flow configurations that can be modelled with this approach. The biggest disadvantage for marker particles is that the link sequence must be preserved to retain connectivity information. Flows with fold-over regions or interfaces that rupture and merge require extensive recalculations of marker sequences. In three dimensions, the bookkeeping costs become prohibitive as a large number of particles must be stored and tracked by the algorithm.

Level set methods

The level set function is defined as a scalar property over the whole domain. The level is propagated with the fluids by solving a scalar convection equation (Gloth *et al.*, 2003). The initial level values are set to be equal to the distance from a free surface, with a positive value on one side of the interface and a negative value on the other side. At any time in the solution, the position of the free surface can be found by locating the level points with zero values. The two disadvantages of this method is that natural merging of surfaces cannot easily be predicted (Ubbink, 1997:25) and that the method is not rigorously mass-conserving. To improve the mass-conservation properties, Van der Pijl *et al.* (2003) proposed a combined volume tracking and level set method. An advantage for surface tension calculations is that the shape of the level set function is smooth over the domain, in contrast to volume tracking methods which can be discontinuous and require additional smoothing steps.

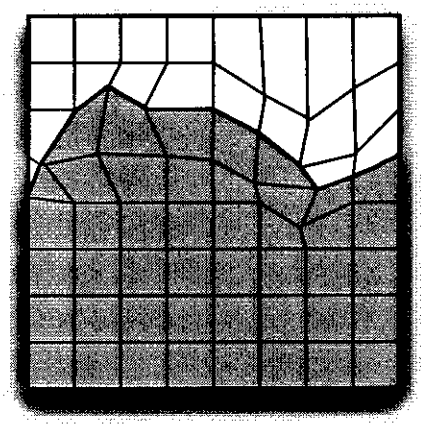


Figure 1.7: Grid points fitted to surface

Surface fitted grids

A variation on the idea of marker particles is to adapt the grid to fit the interface, as depicted in Fig. 1.7. This method is widely used for low amplitude sloshing-type problems without large interface deformations, as described by Muzaferija & Peric (1998); Souli & Zolesio (2001).

The biggest disadvantage when moving grid points it is that movement is restricted to assure mesh integrity. This hampers the simulation of flows with complex interfacial regions, as the moving grid easily distorts and forces a complete remeshing of the domain.

Interface-capturing methods

Interface-capturing methods represent multiple fluids as a single continuum over the domain. The distribution of different fluids is obtained through the use of an indicator function. The indicator function can be implemented as marker particles or a cell-stored scalar value.

Marker particle in cell

The *Marker And Cell* (MAC) method resembles the tracer particles of the surface method. Massless particles are introduced into the regions of the domain containing the fluid. Cells with no marker particles are deemed empty, as shown in Fig. 1.8. When a cell contains marker particles and lies next to an empty cell, it is assumed that the cell contains a part of the interface. An extension of the MAC method is presented by Kitagawa *et al.* (2001) for the modelling of dispersed multiphase flow. This method can successfully capture the breaking up of an interface, but is computationally expensive due to the large number of particles that must be stored.

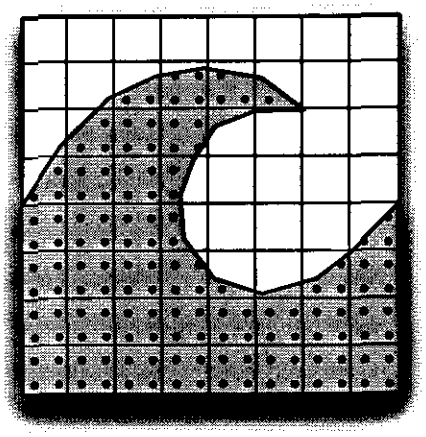
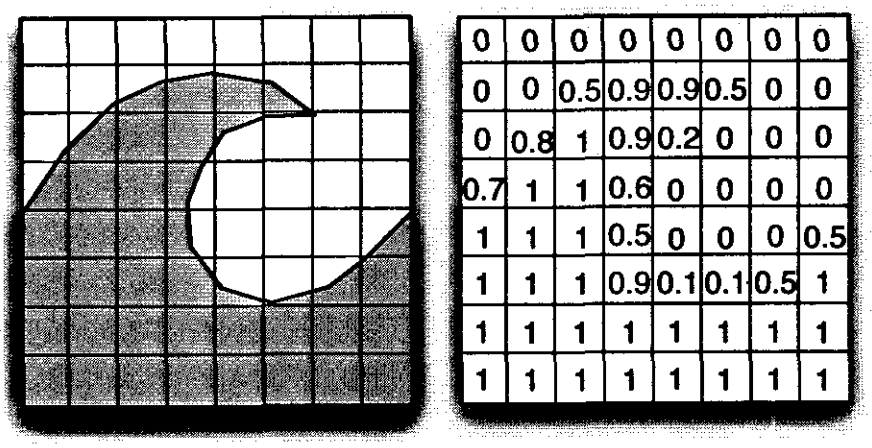


Figure 1.8: The marker-and-cell method



0	0	0	0	0	0	0	0
0	0	0.5	0.9	0.9	0.5	0	0
0	0.8	1	0.9	0.2	0	0	0
0.7	1	1	0.6	0	0	0	0
1	1	1	0.5	0	0	0	0.5
1	1	1	0.9	0.1	0.1	0.5	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Figure 1.9: Original fluid distribution and its volume fraction representation

Volume tracking

One of the most popular methods for modelling interfacial flows is the *Volume Of Fluid* (VOF) method introduced by Hirt & Nichols (1981). Advantages of this method include robust handling of complex interfaces, mass conservation, computational efficiency, ease of implementation and extensibility to multiphase flow situations. Details on the implementation of a typical volume tracking method can be found in Jasak (1996), Ubbink (1997) and Dendy *et al.* (2002).

The indicator function is a scalar value that is defined as one in the first fluid and zero in the second fluid. The indicator is subsequently used to distinguish between different fluids. When the indicator value is a fraction, it represents a mixture of the

two fluids and implicitly defines the interface location. A schematic representation of volume fractions on a discrete mesh is shown in Fig. 1.9.

Computational requirements for this method are much lower than for marker-type schemes while offering the same functionality, making it attractive as a general purpose method. The indicator function is stored as a single value per cell and only one extra transport equation is solved to propagate the volume fraction. The similarity of the indicator function with the other transport equations in a two-fluid system allows for an unhindered incorporation of VOF into existing flow simulation codes. Another advantage is that VOF is readily extendable to use arbitrary three-dimensional grids whilst retaining its computational efficiency.

The main drawback of the volume-of-fluid method is that the indicator function only prescribes the transport of the volume fraction and not the actual shape of the interface. While additional steps can be taken to maintain a sharp interface and to reconstruct that shape from the volume fraction field, the interface definition is still intrinsically linked to the resolution of the grid. By increasing the fineness of the grid, the shape of the interfacial front is better resolved and more accurately transported.

Three techniques can be implemented to obtain better definitions of the convected interface. These are geometric techniques, higher-order VOF techniques and higher-order combined formulations.

- Geometric techniques, described by Aulisa *et al.* (2003), Greaves (2004) and Malik (2004) amongst others, calculate the advection of the volume fraction based on a geometric analysis of the interface and cell. Using approximate areas and planar line segments, the propagation of the interface is calculated. The technique is restricted to rectangular mesh cells and not easily extended to three dimensions.

- Higher-order VOF techniques are based on flux approximations that calculate the amount of fluid entering a target cell by evaluating the amount of fluid contained in its source cell and the local interface orientation. This information is used to construct a stencil of pseudo-cells that enables the use of higher-order discretization schemes for the indicator function. Ubbink (1997) extended VOF to unstructured meshes and developed a convection technique that offer bounded and conservative advection of the volume fraction and momentum. The convection scheme exhibits a linear convergence rate with mesh refinement on uniform meshes (Ubbink & Issa, 1999).

- Higher-order combined schemes use a more accurate, but computationally much more expensive, combination of the continuous Euler formulations for the background flow field and discrete Lagrangian grids for the second phase. The development of such higher resolution VOF schemes are presented by Shahbazi *et al.* (2003); Yamaguchi & Takushima (2004).

Mesh refinement can theoretically be used in conjunction with any of the above techniques. The improvement of grid resolution leads to more accurate interface definitions, but at a cost of using more computational points in the domain. For interface tracking, it is undesirable to refine the complete domain when the region of interest (the interface) is distributed over a relatively small number of cells.

To reduce the unneeded computational effort, a way is sought to apply refinement only to the interfacial region. The next section describes some previous attempts for combining the volume-of-fluid method and non-uniform mesh refinement.

Adaptive grids and interface modelling

The localized nature of the interfacial region in two-fluid systems is an ideal target for adaptive meshing. The refinement criterion simply specifies that all cells containing a fraction of both fluids (i.e. the interface) must be refined. Surrounding cells are refined based on the 1-irregularity concept: the refinement level of neighbouring cells may not be more than one level apart. The combination of the two disciplines is still an emerging field, with most published work coming from the maritime simulation field. Azcueta *et al.* (1999) and Ferziger & Peric (2002) present results on adaptive block-structured grids for the flow around ship hulls. Yiu *et al.* (1996) describes the implementation of an octree structure for adaptive grids, subsequenly applied to surface tracking (Greaves, 2001) and scalar convection using VOF with prescribed flow fields on structured grids(Greaves, 2004). Malik (2004) illustrates adaptive meshing for scalar convection using the *Piecewise Linear Interface Construction* (PLIC) method on structured grids.

1.6 Literature status quo

The state of the union between adaptive refinement and two-phase flow modelling can be summarized as follows:

- Object-oriented simulation toolkits are widely available for finite element methods. To the best of the author's knowledge, there are only two toolkits available that exclusively use a finite volume approach. This is despite the fact that finite volume methods offer a much cleaner separation between the mathematical formulations and numerical implementation — an ideal situation from a toolkit design perspective.

- The FVM-based toolkits do not offer dynamic, adaptive mesh refinement as part of their included mesh tools. Providing a simulation toolkit with multiphysics capabilities is already a daunting task — one that is further complicated by adding adaptive meshing to the mix. The arbitrary field definitions allowed by toolkits lead to increased computational overheads when grids are

adapted: additional object layers are required to translate grid information between the new grid and legacy code.

- Hierarchical grid refinement has been shown to be the most efficient way to obtain localized refinement of a higher resolution than the surrounding mesh. The existing designs for grid hierarchies are unfortunately limited to structured grids, which precludes their direct use on arbitrary grids.

- Grid adaptation for transient two-phase flow problems requires a rigorous treatment of the variable transfer between new and old grids. Existing methods of variable transfer are limited to two-dimensional geometric methods on structured grids or, for three dimensions on unstructured grids, require a trade-off between interface shape preservation and the conservation of properties.

- Adaptive refinement is not yet widely combined with two-phase flow models, even though it offers significant advantages by focusing the computational resources at the interface where it is needed most. Studies done using refinement and volume tracking methods are scarce and based on block structured grids. The majority of results are for scalar convection on regular structured grids using prescribed flow fields and where the fluid motion is not actually calculated.

- No published results were found for adaptive refinement applied to two-phase models on arbitrary grids where the fluid motion is solved as part of the model.

- The only approach suitable for the efficient solution of two-fluid models on unstructured grids is the VOF method. The volume-of-fluid method can also be readily extended to use an adaptive strategy without restricting the type of grid used.

From the literature, the following shortcomings in the union can be identified and posed as the requirements for an adaptive toolkit:

- The simulation toolkit must have dynamically adaptive grid capabilities.

- The need exists for a refinement strategy and grid design that works in a hierarchical manner on both structured and unstructured grids.

- The toolkit must treat all equations in a consistent, generic manner to enable refinement to be applied to arbitrary problem definitions.

- Methods are needed to transfer variables from old grids to the new refined grids in a conservative manner. For transient solutions, temporal and spatial accuracy is of crucial importance for the indicator function as well as other flow properties. Flow solutions require a conservative mapping of flux variables in addition to the normal cell-based field variables transferred between grids.

The next section presents the solution strategy adopted by this study to address the identified shortcomings in adaptive two-phase flow simulations.

1.7 Contribution of this study

The contributions proposed by this study to address the shortcomings in two-fluid simulations can be outlined by the following points:

- A simulation framework was co-developed as part of this study to enable the creation of two-fluid models based on the VOF method that solves the indicator function and general fluid flow problems.
- A new grid design is introduced that provides hierarchical, unstructured refinement capabilities for grids. The refinement can be done in an anisotropic manner based on the local orientation of cells in the domain.
- The grid object is used by the simulation framework to provide adaptive refinement for any models using the toolkit. The models can use an arbitrary definition of fields and problem variables and employ structured or unstructured grids. The refinement of the grid and transfer of variables are handled by the grid object without embroiling the user in unnecessary programming operations.
- The adaptive toolkit is used to obtain solutions for two-fluid models with local refinement in the interface region on structured and unstructured grids.
- Methods are presented that use a second-order accurate transfer of generic variables and enforce the volume-based conservation of extrapolated properties.
- For fluid flow models a reconstruction strategy on refined grids is presented that use the volume-conserved components of the momentum equations to reconstruct conservative flux distributions and consistent pressure fields.

The following outline of the chapters gives an overview of the structure presented in the rest of this thesis.

1.8 Outline of thesis

Chapter 2: Modelling two-fluid systems The mathematical formulations used in two-phase flow modelling are described and the equations are derived in a form suitable for use by the framework and the refinement strategy. Next, the numerical discretization of the equations are given with the solution algorithms for two-fluid systems completing the chapter.

Chapter 3: Adaptive grid generation Arbitrees are introduced as the hierarchical, unstructured grid objects developed in this study. The main concepts behind the design and implementation are discussed, along with algorithms that illustrate how an arbitree object is created.

Chapter 4: Adaptive grid generation for two-fluid systems This chapter describes the use of adaptive refinement in two-fluid systems, the refinement criteria and strategies for the conservative transfer of field variables between grids. Algorithms are given that describe the use of adaptive strategies in two-fluid systems.

Chapter 5: An object-oriented toolkit A short overview of the use of the simulation framework is given. As an example the implementation of an adaptive two-fluid model is described by giving code snippets with commentary documenting the use of the code along the way.

Chapter 6: Test cases The test cases illustrate the convergence behaviour of the refined models and give an indication of the computational savings that are obtained through the use of locally refined grids. The advection of different geometric shapes for oblique, rotational and shear flow is illustrated and as a real world problem the simulation of the collapse of a water column is presented.

Chapter 7: Conclusion and recommendation The last chapter provides a summary of the study and the wisdom gained throughout the process. No single study can hope to address all the issues related to a subject, and recommendations are made for further work to investigate the questions that were not addressed in this study.

Chapter 2

Modelling two-fluid systems

2.1 Introduction

In the first chapter an overview of the object-oriented design process was given. An overview of literature on adaptive refinement was presented and different methods for modelling two-phase fluids were introduced. From the literature it was deduced that the *volume of fluid* method is the most suitable approach to use with the adaptive object-oriented framework developed in this study.

This chapter describes the underlying theory and numerical model of two-phase flow in more detail and serves as foundation for design decisions in the implementation phase. The mathematical equations are rewritten in suitable forms that can be translated into equation objects that are later used in the framework to assemble a complete simulation.

The two immiscible fluids are modelled using a continuum approach, as described by Jasak & Weller (1995), Ubbink (1997), Weller *et al.* (1998), Muzaferija & Peric (1998) and Tezduyar & Aliabadi (2000) amongst others. In a continuum field, there are no sharp discontinuities in the representative equations, making the equations differentiable over the whole domain. As an approximation, both fluids are represented by a single continuous fluid, but with a jump in fluid properties at the interface. If the properties, instead of a step change, are changed over a small finite distance, the conditions of continuity and differentiability still holds (Brackbill *et al.*, 1992). The different fluids are represented by the values of an indicator function and the interface is implicitly defined where the indicator varies between the limiting values of zero and one.

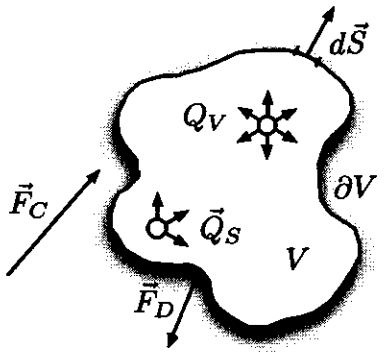


Figure 2.1: The general form of the conservation law.

2.2 Mathematical model

Conservation principles

Fluid flow is governed by the principles of mass, momentum and energy conservation. These conservation laws are independent of the type and properties of the fluid - fluid properties are prescribed by closure equations describing their relationship to each other. Complete descriptions of how the general equations are derived can be found in any fluid mechanics textbook, including Versteeg & Malalasekera (1995) and Ferziger & Peric (2002) and only a short summary is presented of how the two-phase fluid equations are derived from the general equations.

From Versteeg & Malalasekera (1995:25) and Ubbink (1997:42) the general transport equation for the quantity ϕ in Fig. 2.1 can be written in differential form as:

$$\frac{\partial \phi}{\partial t} + \nabla \cdot \vec{F}_C = \nabla \cdot \vec{F}_D + Q_V + \nabla \cdot \vec{Q}_S \tag{2.1}$$

where t is time, $\vec{F}_C = \phi \vec{U}$ is the net convective flux due to fluid motion at velocity $\vec{U}$, $\vec{F}_D$ is the net diffusive flux, Q_V is the volume source due to internal body forces and $\vec{Q}_S$ is the source on the boundary.

The finite volume method takes it name from the next step, which is to integrate Eqn. (2.1) over the finite volume of V to obtain the conservative form of the general transport equation:

$$\frac{\partial}{\partial t}\int_V \phi dV + \oint_{\partial V} \vec{F}_C \cdot d\vec{S} = \oint_{\partial V} \vec{F}_D \cdot d\vec{S} + \int_V Q_V dV + \oint_{\partial V} \vec{Q}_S \cdot d\vec{S} \tag{2.2}$$

where $d\vec{S}$ is the local surface vector and ∂V is the boundary of volume V.

By assuming continuous fluxes and surface terms, Gauss's theorem may be applied to Eqn. (2.2) to obtain for an arbitrary volume:

$$\frac{\partial}{\partial t}\int_V \phi dV + \int_V \nabla \cdot \vec{F}_C dV = \int_V \nabla \cdot \vec{F}_D dV + \int_V Q_V dV + \int_V \nabla \cdot \vec{Q}_S dV \qquad (2.3)$$

Using the general forms in Eqn. (2.1) or Eqn. (2.3), the transport equation can subsequently be recast for different scalar and vector quantities. By substituting ϕ with the appropriate properties, the governing equations for the fluid-flow system are derived in the following sections. For readibility, the derived equations are given in the differential form.

Governing equations

Momentum equation

Momentum transport is obtained by substituting ϕ with $\rho\vec{U}$, the momentum per unit volume. It is assumed that when the fluid is at rest, no momentum diffusion can take place, so that $\vec{F}_D = 0$. The surface forces that act on the control volume are determined by the stress tensor for Newtonian fluids and internal forces due to surface tension. The only body force acting on the fluid in the control volume is gravity, given by $\rho\vec{g}$ where $\vec{g}$ is the gravitational acceleration. This study focuses on laminar flow, which means that no turbulent sources are present in the equations.

A Newtonian fluid in local thermodynamic equilibrium is subject to the stress tensor $\vec{T}$:

$$\vec{T} = -\left(P + \frac{2}{3}\mu\nabla\cdot\vec{U}\right)\vec{I} + \mu\left(\nabla\otimes\vec{U} + \left(\nabla\otimes\vec{U}\right)^T\right) \qquad (2.4)$$

where P is the pressure, μ is the dynamic viscosity and $\vec{I}$ is the unit tensor.

Surface tension is a tensile force that acts tangentially to the interface and tries to keep the fluid molecules of a specific fluid together. It is balanced out by a pressure jump across the interface that depends on the surface tension coefficient and the curvature of the interface. According to Koshizuka *et al.* (1995) and Ubbink (1997:163), surface tension effects may be neglected for the class of test cases analyzed in this study, based on the large scale of the physical geometry. When modelling smaller physical problems, i.e. rising bubble flow, surface tension must be accounted for through the surface source term. The most common method is the *Continuum Surface Force* (CSF) model presented by Brackbill *et al.* (1992). High-resolution schemes, such as the one presented by Lörstad & Fuchs (2004), are also available which feature an improved VOF method, specifically designed for a wide range of bubble diameters.

After the above assumptions are applied to Eqn. (2.1) and all momentum sources are considered, the momentum balance is given by:

$$\frac{\partial\rho\vec{U}}{\partial t} + \nabla\cdot(\rho\vec{U}\otimes\vec{U} - \vec{T}) = \rho\vec{g} \qquad (2.5)$$

For incompressible fluids, the continuity equation derived in Eqn. (2.16) is exploited to simplify the stress tensor, giving the final form:

$$\frac{\partial \rho \vec{U}}{\partial t} + \nabla \cdot (\rho \vec{U} \otimes \vec{U}) - \nabla \cdot (\mu \nabla \otimes \vec{U}) = -\nabla P + \rho \vec{g} + (\nabla \otimes \vec{U}) \cdot (\nabla \mu) \tag{2.6}$$

Continuity equation

Conservation of mass in a control volume means that the mass of an incompressible fluid inside an enclosed control volume cannot change, except when there is flow across the boundaries.

If the density per unit volume ρ, is substituted for ϕ in the general equation, the continuity equation that describes the mass conservation in the absence of sources is obtained:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \vec{U} = 0 \tag{2.7}$$

Indicator and closure equations

The indicator function denotes the normalized volume-based percentage of a certain fluid in a control volume. With the volume fraction method, α is defined as the fraction of the volume in a cell that fluid 1 of a two fluid system occupies.

$$\alpha_P = \frac{\text{Volume of fluid 1}}{\text{Total volume of the control volume}} \tag{2.8}$$

The initial definition of the indicator function is as step function with a sharp interface between the two fluids, but that formulation negates the continuum approach that is used to solve the indicator equation. By introducing a small and finite interface region δ, the continuum approach is again valid by allowing the fluid properties to vary between the fluids in a continuous and differentiable manner. For two-fluids, the indicator function α defined at a location $\vec{x}$ at time t is defined as:

$$\alpha(\vec{x}, t) = \begin{cases} 1 & \text{if the point } (\vec{x}, t) \text{ is inside fluid 1} \\ 0 & \text{if the point } (\vec{x}, t) \text{ is inside fluid 2} \\ 0 < \alpha_\delta < 1 & \text{if the point } (\vec{x}, t) \text{ is inside } \delta \end{cases} \tag{2.9}$$

The initial conditions are defined by specifying $\alpha(\vec{x}, 0)$.

By introducing the interface region, it is tacitly implied that the fluid in the interface region is a mixture of both fluids. The density and dynamic viscosity of the mixture can then be determined through the use of the indicator value, giving the closure equations:

$$\rho = \alpha \rho_1 + (1 - \alpha) \rho_2 \tag{2.10}$$

and

$$\mu = \alpha\mu_1 + (1-\alpha)\mu_2 \tag{2.11}$$

where ρ and μ are the mixture density and viscosity, and the subscripts denote the different fluids.

For incompressible fluids, the fluid mixture propagates at an average velocity through the interface control volume (i.e. the two fluids have the same velocity in a mixture cell). This means the continuity equation in Eqn. (2.7) must hold for both fluids in a control volume.

For fluid one, the continuity is given by:

$$\frac{\partial \alpha\rho_1}{\partial t} + \nabla \cdot \alpha\rho_1\vec{U} = 0 \tag{2.12}$$

and for fluid two:

$$\frac{\partial (1-\alpha)\rho_2}{\partial t} + \nabla \cdot (1-\alpha)\rho_2\vec{U} = 0 \tag{2.13}$$

The indicator function for incompressible flow is obtained by dropping the density from Eqn. (2.12):

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot \alpha\vec{U} = 0 \tag{2.14}$$

By expanding Eqn. (2.13) we obtain:

$$\frac{\partial \rho_2}{\partial t} - \frac{\partial \alpha\rho_2}{\partial t} + \nabla \cdot \rho_2\vec{U} - \nabla \cdot \alpha\rho_2\vec{U} = 0 \tag{2.15}$$

The density is dropped and Eqn. (2.14) is substituted into Eqn. (2.15) to obtain the continuity equation for incompressible, two-phase flow:

$$\nabla \cdot \vec{U} = 0 \tag{2.16}$$

Final forms of the equations

The equations representing the two-phase fluid system are summarized in this section. Since analytical solutions for the system can only be found under very restricted circumstances, the equations must be solved simultaneously by using an iterative numerical solver. The final equations that form the system are: the continuity equation in Eqn. (2.16), the momentum equation of Eqn. (2.6), the indicator function in Eqn. (2.14) and the closure equations for density and dynamic viscosity given by Eqn. (2.10) and Eqn. (2.11).

Boundary conditions

All numerical problems are defined by their equations, the initial conditions and boundary conditions. It is critical to understand and correctly specify the boundary conditions of all variables. Vastly different solutions can be obtained for the same set of equations, with the only difference being the type and value of boundaries used in the model.

In a mathematical sense, there are only two different types of boundaries: the so-called Dirichlet condition (fixed value) or Von Neuman condition (fixed gradient). Selecting the appropriate boundary type directly from these two is a daunting task given the mixed nature of the numerical systems of the typical CFD problem. Instead, it makes more sense from a modelling viewpoint to describe the conditions based on physical arguments and then define the behaviour of different variables on the various boundary types. Descriptions of boundary conditions and their implementations can be found in Versteeg & Malalasekera (1995) and Ferziger & Peric (2002) and are summarized here:

Inlet boundary The distribution of all variables need to be specified at the inlet boundary. For transient problems, this will be the initial field at zero time-step. By definition, the velocity distribution at the inlet must be known. The pressure distribution is usually unknown and may be found by interpolation from inside the domain. The position of the interface (i.e. the indicator function) must also be known beforehand.

Outlet boundary For best results, outlet boundaries are usually placed away from regions of interest in the flow and flow may only leave the domain. The aim is to place the boundary at such a location that gradients normal to the boundary are as small as possible to enable zero Von Neuman conditions to be applied. In flows with high Reynolds numbers there is a weak upstream propagation of errors so that such conditions may be applied. There is no extrapolation scheme that can guarantee mass conservation at the outlet (Ferziger & Peric, 2002:206) and it is necessary to do an explicit correction on the outlet velocity so that the outlet mass flux equals the inlet flux.

Pressure or open boundary Pressure boundaries can be used in locations where the pressure is known, but other quantities are not. Typically this might be the top of open containers, models of external flow and internal flows with multiple outlets. The pressure value is fixed and other quantities are extrapolated. At the pressure boundary the same velocity profile correction done at the outlet needs to be done, except that flow may also enter the domain and needs to be taken into account.

Wall boundary The so-called *zero slip* condition is applied to the velocity on a wall: this means that the fluid takes the velocity of the wall, which is zero for all direction components. The indicator function may be approximated

by a zero-gradient boundary (Brackbill *et al.*, 1992). When surface tension is a significant model variable, an implementation of wall adhesion is needed to correctly calculate the wetting angle of the fluid, as described by Ubbink (1997:54). Pressure at the wall is unknown and may be extrapolated from the interior of the domain using the pressure gradient in the fluid next to the wall. It is important that the approximation of the pressure is consistent with the treatment of the velocity at the wall to ensure momentum conservation.

Symmetry plane When flow is symmetrical, the solution domain can be halved along the symmetry plane and the solution in the *missing* halve is mirrored by the boundary. This implies that the velocity component normal to the plane is zero (no flow is allowed through the boundary) and the gradient of the tangential component is zero. Other properties such as scalars, pressure and the indicator function should be treated as zero-gradient boundaries.

Initial fields should be specified such that they are consistent with each other. The pressure field at the beginning is not relevant for the first iteration of the solution process, and may be specified as a zero field. If a pressure field is specified, it should conform with the initial velocity distribution, otherwise the iteration process takes additional calculations to initiate the correct pressure field.

Closure

The mathematical equations needed to simulate a two-fluid system were described and an overview of the boundary conditions for the domain was given. The following section describes how the equations are discretized as part of a numerical method that enables their simultaneous solution. The chapter closes with the algorithms used to determine the convective coefficients and the calculation cycle for a two-fluid problem.

2.3 Numerical model

Introduction

The mathematical formulations derived in the previous section constitute the conservation principles for a system of two-fluid equations. To enable the solution of the differential forms (Eqn. (2.1)) of the equations over the domain, the equations must be rewritten in the volume-based conservation form of Eqn. (2.3). The physical domain is then divided into a set of non-overlapping discrete control-volumes and the conservative equations are evaluated for each control volume in turn. By integrating the equations over all the control volumes, a conservative solution for the complete domain is automatically obtained. This manner of volume division and integration is known as the finite volume method and is widely used in research and commercial CFD codes. It has been well documented in textbooks on the subject from amongst others Versteeg & Malalasekera (1995), Anderson (1995) and Ferziger & Peric (2002).

Spatial formulation

The control volumes may be defined in two distinct ways: 1) staggered control volumes and 2) co-located volumes. Staggered control volumes, described by Patankar (1980) and Versteeg & Malalasekera (1995), implies that all variables (pressure, temperature etc.) are defined at the centres of the volumes but the velocity components are defined at the centres of volumes constructed around the faces surrounding any individual control volume. When using staggered control volumes, an uncoupling of pressure and velocity during the solution stages is prevented (Versteeg & Malalasekera, 1995:135). In the momentum equation (Eqn. (2.3)) the effect of pressure is only represented by the gradient of the pressure in the source term. That means that highly irregular pressure distributions may have the same gradient across a cell volume as the more realistic (and correct) pressure field, but the momentum equation cannot distinguish between the two, and the solution could converge to the wrong pressure field. The advantage of staggered volumes is that this "checker-boarding" of pressure is prevented, but at the higher computing cost of having to store a larger amount of grid information. Staggered volumes further add to the complexity when using unstructured grids—especially when volumes are defined by arbitrary numbers of faces. Under these circumstances, the definition of the additional staggered volumes around three-dimensional faces become very involved and computationally expensive.

The co-located approach in Ferziger & Peric (2002) define all the variables, including velocity, at the centres of the original control volumes. Resource-wise, this is a much more efficient approach, as only one set of grid data needs to be stored. An extension to unstructured, three-dimensional grids is straightforward and natural using this paradigm. Fig. 2.2 depicts the definition of an unstructured control volume with an arbitrary number of faces and is relevant in all following discussions.

The general notation and form for the equations follow that used by Jasak (1996) and Ubbink (1997).

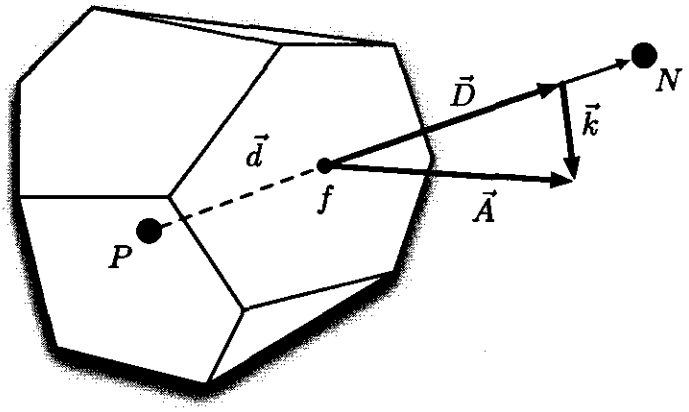


Figure 2.2: Polyhedron representation of an arbitrary control volume

The point P is the main computational node and is defined at the centre of the volume. The neighbouring point is denoted by N and the directional vector $\vec{d} = \vec{PN}$ connects points P and N. The face area vector $\vec{A}$ is defined as normal to the plane of the face, and points outwards from the control volume. The face is defined as non-orthogonal if $\vec{A}$ and $\vec{d}$ are not collinear; the vectors $\vec{D}$ and $\vec{k}$ are subsequently introduced to account for the orthogonal and non-orthogonal components of vectorized quantities on the cell faces respectively. Jasak (1996:85) experimented with three different methods of determining the relation between $\vec{D}$ and $\vec{k}$. The *over-relaxed* approach was found to give the best results, as the non-orthogonal contribution to P and N increases with an increase in the non-orthogonality of the face.

The over-relaxed approach defines $\vec{D}$ and $\vec{k}$ as:

$$\vec{D} = \frac{\vec{d}}{\vec{d} \cdot \vec{A}} |\vec{A}|^2 \tag{2.17}$$

with $\vec{k}$ obtained by subtracting the two area vectors:

$$\vec{k} = \vec{A} - \vec{D} \tag{2.18}$$

The value of a scalar (or component of a vector) at the point on the face is generally obtained through linear interpolation between the values of owner and neighbour cells:

$$\vec{\phi}_f = \mathcal{L}_P \vec{\phi}_P + (1 - \mathcal{L}_P)\vec{\phi}_N \tag{2.19}$$

where $\mathcal{L}_P$ is an interpolation factor obtained by calculating the ratio of the distance from the neighbour cell centre to the face centre $f\vec{N}$, to the distance between cell centres $\vec{PN}$:

$$\mathcal{L}_P = \frac{|f\vec{N}|}{|P\vec{N}|} \tag{2.20}$$

The spatial formulation described in this section will be used in the derivation of the discrete momentum, pressure and indicator equations. The next section presents some commonly-used mathematical expressions that form the basis for the discrete form of the equations and the software components described in Chapter 5.

General mathematical operators

Upon examination of the final equations in §2.2, it is noted that there are some general mathematical operators that appear in all equations. It is useful for the numerical derivation and software design to gather these operators together and present their respective discrete forms.

The general form of Gauss' theorem (Stewart, 1998:979) is used to obtain the surface integral forms for the different volume-integrated operations:

Gradient of a scalar field

$$\int_V \nabla\phi dV = \oint_{\partial V} d\vec{S}\phi \tag{2.21}$$

Divergence of a vector field

$$\int_V \nabla \cdot \vec{\phi} dV = \oint_{\partial V} d\vec{S} \cdot \vec{\phi} \tag{2.22}$$

Curl of a vector field

$$\int_V \nabla \otimes \vec{\phi} dV = \oint_{\partial V} d\vec{S} \otimes \vec{\phi} \tag{2.23}$$

where ϕ is an arbitrary scalar variable and $\vec{\phi}$ an arbitrary vector function. The surface vector $d\vec{S}$ and volume surface ∂V is defined in Fig. 2.1.

The volume surface ∂V of any control volume/cell is comprised of a number of faces with discrete areas. The final step in transforming the continuum equations into a discrete form is to transform the continuous surface integrals given in Eqn. (2.21) – Eqn. (2.23) to discrete face-based integrals. By way of illustration, for Eqn. (2.21) it follows that:

$$\int_V \nabla\phi dV = \oint_{\partial V} d\vec{S}\phi = \sum_{f=1}^{n} \left(\int_f d\vec{S}\phi \right) \approx \sum_{f=1}^{n} \vec{A}_f \phi_f \tag{2.24}$$

where f is the centre of the face, n is the number of faces of a control volume and $\vec{A}_f$ the face area vector (see Fig. 2.2).

With the construction of the numerical grid, faces and control volumes are uniquely defined (no two faces are allowed to be defined by the same set of vertices).

Control volumes are constructed first and then faces are defined that establish the connectivity between different cells in the grid. Connectivity is defined by a single face linking only two cells together, one on each side of the plane of the face. The face construction and allocation are done on a per cell basis; the cell that first defines a face is termed the *owner* or *inside* cell of the face. Any following cell that shares the face is termed the *neighbour* or *outside* cell.

The connectivity defined in this way elegantly allows face-based calculations, such as $\sum_{f=1}^{n} \vec{A}_f \phi_f$ from Eqn. (2.24), to be expressed as sums over owner and neighbour faces:

$$\sum_{f=1}^{n} \vec{A}_f \phi_f = \underbrace{\sum_{f=1}^{n} \vec{A}_f \phi_f}_{owners} - \underbrace{\sum_{f=1}^{n} \vec{A}_f \phi_f}_{neighbours} \tag{2.25}$$

where n either indicates the number of faces for a single cell, or the total number of faces defined in the domain.

The operations defined in Eqn. (2.22) and Eqn. (2.23) are expressed as face-based summations in a similar way as shown in Eqn. (2.24), to obtain the divergence:

$$\int_V \nabla \cdot \vec{\phi} dV = \sum_{f=1}^{n} \vec{A}_f \cdot \vec{\phi}_f \tag{2.26}$$

and curl operations:

$$\int_V \nabla \otimes \vec{\phi} dV = \sum_{f=1}^{n} \vec{A}_f \otimes \vec{\phi}_f \tag{2.27}$$

Second-order accurate predictions of the gradient, divergence and curl for a flow property over a cell with a volume V_P can be calculated by using Eqn. (2.24), Eqn. (2.26) and Eqn. (2.27).

$$(\nabla \phi)_P = \frac{1}{V_P} \sum_{f=1}^{n} \vec{A}_f \phi_f \tag{2.28}$$

$$(\nabla \cdot \vec{\phi})_P = \frac{1}{V_P} \sum_{f=1}^{n} \vec{A}_f \cdot \vec{\phi}_f \tag{2.29}$$

$$(\nabla \otimes \vec{\phi})_P = \frac{1}{V_P} \sum_{f=1}^{n} \vec{A}_f \otimes \vec{\phi}_f \tag{2.30}$$

These face-based operations form the foundation for the discretization of the transport equations described in the following sections.

Momentum equation

The differential form of the momentum equation, Eqn. (2.6) is integrated over the control volume and time step δt to obtain the finite volume form:

$$\int_t^{t+\delta t}\left(\int_V \frac{\partial \rho \vec{U}}{\partial t} dV\right) dt + \int_t^{t+\delta t}\left(\int_V \nabla \cdot (\rho \vec{U} \otimes \vec{U}) dV\right) dt - \int_t^{t+\delta t}\left(\int_V \nabla \cdot (\mu \nabla \otimes \vec{U}) dV\right) dt = -\int_t^{t+\delta t}\left(\int_V \nabla P dV\right) dt + \int_t^{t+\delta t}\left(\int_V \rho \vec{g} dV\right) dt + \int_t^{t+\delta t}\left(\int_V (\nabla \otimes \vec{U}) \cdot (\nabla \mu) dV\right) dt \tag{2.31}$$

Since the velocity is a vector quantity, the momentum equation prescribes the conservation of each of the component quantities. Eqn. (2.31) can be broken up into the three directional components and solved for each Cartesian component in turn during the simulation cycle.

The volumetric flux F_f, appears in all the equations of the system, and affects the conservative transport of the different quantities. The flux at the face can be defined as:

$$F_f = \vec{A}_f \cdot \vec{U}_f \tag{2.32}$$

The flux definition from Eqn. (2.32) and the face operations described in §2.3 are substituted in the momentum equation to get the semi-discretized form:

$$\int_t^{t+\delta t}\left(\left(\frac{\partial \rho \vec{U}}{\partial t}\right)_P V_P + \sum_{f=1}^{n} \rho_f F_f \vec{U}_f - \sum_{f=1}^{n} \mu_f \vec{A}_f \cdot (\nabla \otimes \vec{U}_f)\right) dt = \int_t^{t+\delta t}\left(\vec{S}_{\vec{U}_P} V_P\right) dt \tag{2.33}$$

where the source term $\vec{S}_{\vec{U}_P}$ includes contributions from the pressure gradient, gravitation, viscous effects and other external forces:

$$\vec{S}_{\vec{U}_P} = -(\nabla P)_P + \vec{g}\rho_P + (\nabla \otimes \vec{U})_P \cdot (\nabla \mu)_P \tag{2.34}$$

More detail on the convection and diffusion terms are given in the following section. The gradient operations in the pressure and viscous terms are calculated with Eqn. (2.28) to Eqn. (2.30) while the gravitational source term is calculated as body source for the volume.

Convective term

The convective term is discretized in conservative form as:

$$\int_V \nabla \cdot (\rho \vec{U} \otimes \vec{U}) dV = \sum_{f=1}^{n} \rho_f F_f \vec{U}_f \tag{2.35}$$

using the volumetric flux defined in Eqn. (2.32).

The product $\rho_f F_f$ denotes the conservative mass flux through a face and is valid for flow of constant or varying density. By already assuming incompressible flow to derive the indicator function (Eqn. (2.14)), the indicator value at the face is calculated to ensure conservative volumetric fluxes. To obtain mass conservation with the volumetric fluxes, the density at the face needs to be consistent with the calculated face value of the indicator function. In §2.3, a convection scheme is described that was developed by Ubbink (1997) to specifically ensure conservative fluxes in interfacial flows.

The density on the face is calculated as;

$$\rho_f = \alpha_f^* \rho_1 + (1 - \alpha_f^*) \rho_2 \tag{2.36}$$

where α_f^* is the indicator value that guarantees conservative mass fluxes at the faces.

The face values of the velocity are approximated by using the well known upwind differencing scheme:

$$\vec{U}_f = \begin{cases} \vec{U}_P & \text{if } F_f \geq 0 \\ \vec{U}_N & \text{if } F_f < 0 \end{cases} \tag{2.37}$$

that guarantees boundedness for face velocities.

Diffusion term

The diffusion term is written as:

$$\int_V \nabla \cdot (\mu \nabla \otimes \vec{U}) dV = \sum_{f=1}^{n} \mu_f \vec{A}_f \cdot (\nabla \otimes \vec{U}_f) \tag{2.38}$$

In a manner analogous to the calculation of the face density, the face viscosity is obtained by:

$$\mu_f = \alpha_f^* \mu_1 + (1 - \alpha_f^*) \mu_2 \tag{2.39}$$

To reformulate the diffusion term in Eqn. (2.38) in terms of cell values, the term $(\nabla \otimes \vec{U}_f)$ can be interpolated to the face using Eqn. (2.19):

$$(\nabla \otimes \vec{U}_f) = \mathcal{L}_P (\nabla \otimes \vec{U})_P + (1 - \mathcal{L}_P)(\nabla \otimes \vec{U})_N \tag{2.40}$$

The standard curl operator defined in Eqn. (2.23), is used to express the gradient over the cell:

$$(\nabla \otimes \vec{U})_P = \frac{1}{V_P} \sum_{f=1}^{n} \vec{A}_f \otimes \vec{U}_f \tag{2.41}$$

This approach results in a computational molecule that involves not only the two cells straddling the face and their nearest neighbours, but through the gradient calculation skew and far neighbours too. It becomes difficult to treat all of the extra neighbours in an implicit manner, so that it is a better decision to split the gradient calculation in two parts. The gradient in the r.h.s. of Eqn. (2.38) can be manipulated by using the non-orthogonal definitions in Eqn. (2.17) and Eqn. (2.18).

For an orthogonal face, $\vec{A}_f = \vec{D}_f$ as seen from Fig. 2.2 when $\vec{k}$ is zero. A second-order accurate approximation for the gradient at the face can be obtained by using the cells bracketing the face, which allows the orthogonal component of the gradient to be defined through:

$$\vec{A}_f \cdot (\nabla \otimes \vec{U})_f = \vec{D}_f \cdot (\nabla \otimes \vec{U})_f = |\vec{D}_f| \frac{\vec{U}_N - \vec{U}_P}{|\vec{d}_f|} \tag{2.42}$$

When the face is non-orthogonal, the gradient must be adjusted with the non-orthogonal contribution. The over-relaxed formulation from Jasak (1996) in Eqn. (2.18) provides $\vec{A} = \vec{D} + \vec{k}$, where $\vec{k}$ is the non-orthogonal correction vector.

The final form of the diffusion gradient is then given by:

$$\vec{A}_f \cdot (\nabla \otimes \vec{U})_f = \underbrace{\vec{D}_f \cdot (\nabla \otimes \vec{U})_f}_{\text{orthogonal}} + \underbrace{\vec{k}_f \cdot (\nabla \otimes \vec{U})_f}_{\text{non-orthogonal}} \tag{2.43}$$

In practice, the orthogonal contribution is then calculated by Eqn. (2.42) and treated implicitly as part of the solution. The non-orthogonal contribution is calculated by Eqn. (2.40) and treated explicitly as part of the source term. The wider band of the computational molecule contains extra neighbours and is lagged in time. In cases of severe non-orthogonality this may impact negatively on convergence behaviour and extra iterations over the explicit terms will be required.

Temporal discretization

The temporal discretization of the momentum equations follows the standard *Crank-Nicholson* (CN) scheme as presented by Versteeg & Malalasekera (1995:168). When equations are discretized in time, an assumption about the temporal variation of the properties needs to be made. It is further important that the same scheme is used for the temporal discretization of both the momentum and indicator equations (presented later) to ensure the consistent definition of these properties.

Fully implicit temporal differencing schemes introduces numerical diffusion in the flow direction, while fully explicit schemes introduce diffusion tangential to the flow direction. It was proven by Ubbink (1997:125) during the development of his

convection scheme that the CN temporal scheme introduces the least amount of numerical diffusion because it offers a second-order accurate half-weighting between fully implicit and explicit schemes.

After applying the temporal scheme, the final form of the momentum equations for a non-orthogonal mesh reduces to:

$$\frac{(\rho\vec{U})_P^{t+\delta t}}{\delta t} + \frac{1}{V_P}\sum_{f=1}^{n}\rho_f F_f \vec{U}_f^{t+\delta t} - \frac{1}{V_P}\sum_{f=1}^{n}\mu_f \vec{D}_f \cdot (\nabla \otimes \vec{U})_f^{t+\delta t} = \vec{S}_{\vec{U}_P} - (\nabla P)_P \quad (2.44)$$

where the source term ($\vec{S}_{\vec{U}_P}$) is defined by:

$$\vec{S}_{\vec{U}_P} = \frac{(\rho\vec{U})_P^t}{\delta t} + \vec{g}\rho_P + (\nabla \otimes \vec{U})_P^t \cdot (\nabla\mu)_P + \frac{1}{V_P}\sum_{f=1}^{n}\mu_f \vec{k}_f \cdot (\nabla \otimes \vec{U})_f^t \quad (2.45)$$

As mentioned in §2.3, the contribution of non-orthogonal terms and the gradient of the viscosity is lagged in time, and if needed extra iterations over the explicit source terms must be done as part of the solution process.

The face velocities that appear in Eqn. (2.44), can be re-written in terms of a cell and its nearest neighbours. This casts the equation in a form that is suitable for solution with a numerical solver:

$$a_P \vec{U}_P^{t+\delta t} = \sum_{nb=1}^{n} a_{nb}\vec{U}_{nb}^{t+\delta t} + \vec{S}_{\vec{U}_P} - (\nabla P)_P \quad (2.46)$$

where a_P is the diagonal coefficient of the velocity in cell P, nb indicates a neighbour and n is the number of neighbours for the cell.

For the derivation of the pressure correction equation in the next section, it is useful to define the following form of the discretized momentum equation:

$$\vec{U}^{t+\delta t} = \frac{\vec{H}(\vec{U})_P}{a_P} - \frac{1}{a_P}(\nabla P)_P \quad (2.47)$$

where $\vec{H}(\vec{U})_P$ represents the transport part of Eqn. (2.47)

$$\vec{H}(\vec{U})_P = \sum_{nb=1}^{n} a_{nb}\vec{U}_{nb}^{t+\delta t} + \vec{S}_{\vec{U}_P} \quad (2.48)$$

With the momentum equation re-written in a suitable face-based form, in the next section it is combined with the continuity equation to obtain an equation for the pressure correction.

Continuity equation

In the momentum equations, the effect of pressure is accounted for through the pressure gradient, but no explicit equation is available for the pressure. To solve this problem, the continuity equation from Eqn. (2.7), can be utilized to obtain an equation for the pressure.

The manner in which the continuity equation is used to determine the pressure solution, forms the basis of various pressure-velocity coupling schemes. The first scheme was the venerable SIMPLE[1]-algorithm for steady flows. It was introduced by Patankar (1980) and is essentially a guess-and-correct procedure to obtain the pressure for systems using the Navier-Stokes equations. The continuity equation is discretized to obtain a pressure correction, which in turn is used to adjust the velocity distribution until continuity is satisfied. Other variants of the SIMPLE-algorithm, like SIMPLE-R[2] and SIMPLE-C[3] offer more accurate treatments of the discarded velocity correction factors of SIMPLE. For transient solutions, the PISO[4] algorithm was developed which uses the continuity equation to derive an equation which can be used for direct solution of the pressure.

In this section, only the basic SIMPLE method is described, detail on the implementation of the other variants can be found in Versteeg & Malalasekera (1995), Ubbink (1997) and Ferziger & Peric (2002).

Interpolation of face velocities

The integral form of the continuity equation for incompressible flow, Eqn. (2.16):

$$\oint_V \nabla \cdot \vec{U} dV = 0 \tag{2.49}$$

can be reformulated in a face-based way by using Eqn. (2.29):

$$\sum_{f=1}^{n} \vec{A}_f \cdot \vec{U}_f = 0 \tag{2.50}$$

To avoid the decoupling of pressure and velocity during the simulation, special care must be taken in a co-located variable environment (Versteeg & Malalasekera, 1995:137). The face interpolation proposed by Rhie & Chow (1983) is used to obtain the correct velocity value on the face.

The first step is to remove the contribution of the pressure gradient from the velocity in the cell:

$$\vec{U}_P^{\dagger} = \vec{U}_P + (\nabla P)_P \tag{2.51}$$

[1]Semi-Implicit Pressure Linked Equations
[2]SIMPLE -Revised
[3]SIMPLE-Consistent
[4]Pressure Implicit with Splitting of Operators

The pseudo velocity $\vec{U}^{\dagger}$ is then interpolated using linear interpolation to obtain the value on the face:

$$\vec{U}_f^{\dagger} = \mathcal{L}_P \vec{U}_P^{\dagger} + (1 - \mathcal{L}_P)\vec{U}_N^{\dagger} \tag{2.52}$$

Finally, the pressure gradient calculated with the two cells straddling the face is added to the pseudo velocity to obtain the interpolated face velocity.

$$\vec{U}_f = \vec{U}_f^{\dagger} - \frac{P_N - P_P}{|\vec{d}_f|} \tag{2.53}$$

Pressure correction equation

To initiate the solution cycle, an initial pressure distribution P^* is guessed. This is used in Eqn. (2.47) to solve for the velocity $\vec{U}^*$:

$$\vec{U}^* = \frac{\vec{H}(\vec{U}^*)_P}{a_P} - \frac{1}{a_P}(\nabla P^*)_P \tag{2.54}$$

The pressure correction can be defined as P' and is the difference between the correct pressure field P and the guessed pressure field P^*:

$$P = P^* + P' \tag{2.55}$$

For the velocity, a similar correction can be defined with U':

$$\vec{U} = \vec{U}^* + \vec{U}' \tag{2.56}$$

The guessed velocity field is subtracted from the correct velocity field:

$$(\vec{U} - \vec{U}^*) = \frac{\vec{H}(\vec{U} - \vec{U}^*)_P}{a_P} - \frac{1}{a_P}(\nabla(P - P^*))_P \tag{2.57}$$

The equations for the pressure and velocity corrections can be used to obtain:

$$\vec{U}' = \frac{\vec{H}(\vec{U}')_P}{a_P} - \frac{1}{a_P}(\nabla P')_P \tag{2.58}$$

The SIMPLE algorithm makes an assumption that the contribution of the neighbour velocity corrections, defined by $\frac{\vec{H}(\vec{U}')_P}{a_P}$ may be neglected to obtain the final form of the velocity correction:

$$\vec{U}' = -\frac{1}{a_P}(\nabla P')_P \tag{2.59}$$

Back-substitution of the velocity correction in Eqn. (2.56) gives:

$$\vec{U} = \vec{U}^* - \frac{1}{a_P}(\nabla P')_P \tag{2.60}$$

The velocity is substituted in the continuity equation (Eqn. (2.50)):

$$\sum_{f=1}^{n} \vec{A}_f \cdot (\vec{U}_f^* - \frac{1}{a_P}(\nabla P')_f) = 0 \tag{2.61}$$

and the terms are separated to obtain:

$$\sum_{f=1}^{n} \vec{A}_f \cdot \vec{U}_f^* = \frac{1}{a_P} \sum_{f=1}^{n} \vec{A}_f \cdot (\nabla P')_f \tag{2.62}$$

The term on the r.h.s. of Eqn. (2.62) is treated in the same way as the diffusion term of the momentum equation (Eqn. (2.43)) by splitting it into orthogonal and non-orthogonal contributions:

$$\vec{A}_f \cdot (\nabla P')_f = \underbrace{\vec{D}_f \cdot (\nabla P')_f}_{orthogonal} + \underbrace{\vec{k}_f \cdot (\nabla P')_f}_{non-orthogonal} \tag{2.63}$$

The orthogonal contribution is given by:

$$\vec{D}_f \cdot (\nabla P')_f = |\vec{D}_f| \frac{P'_N - P'_P}{|\vec{d}_f|} \tag{2.64}$$

and the non-orthogonal contribution is:

$$\vec{k}_f \cdot (\nabla P')_f = \vec{k}_f \cdot \left(\mathcal{L}_P \left(\frac{1}{V_P} \sum_{f=1}^{n} \vec{A}_f P'_f \right)_P + (1 - \mathcal{L}_P) \left(\frac{1}{V_N} \sum_{f=1}^{n} \vec{A}_f P'_f \right)_N \right) \tag{2.65}$$

The non-orthogonal contribution is explicitly added to the source term of the pressure correction equation:

$$\sum_{f=1}^{n} \left(\frac{1}{a_P} \right)_f \vec{D}_f \cdot (\nabla P')_f = \vec{S}_P \tag{2.66}$$

where the source term for cell P contains the flux imbalance and the non-orthogonal contribution:

$$\vec{S}_P = \sum_{f=1}^{n} \vec{A}_f \cdot \vec{U}_f^* - \sum_{f=1}^{n} \left(\frac{1}{a_P} \right)_f \vec{k}_f \cdot (\nabla P')_f \tag{2.67}$$

The equation form for the numerical solver is given by:

$$a_P P'_P = \sum_{nb=1}^{n} a_{nb} P'_{nb} + S_P \tag{2.68}$$

The volume fluxes are corrected to obtain the conservative flux for the new time step by using the interpolation scheme of Eqn. (2.51)–Eqn. (2.53):

$$F_f = F_f^* - \vec{A}_f \cdot \left(\frac{1}{a_P}\right)_f (\nabla P')_f \tag{2.69}$$

The conservative volumetric flux from Eqn. (2.69) is used in the next section to calculate the new distribution of the indicator field.

Indicator function

The finite volume discretization for α is obtained by integrating the differential form of the indicator function, Eqn. (2.14) over the control volume and time step:

$$\int_t^{t+\delta t} \left(\int_V \frac{\partial \alpha}{\partial t} dV \right) dt + \int_t^{t+\delta t} \left(\int_V \nabla \cdot \alpha \vec{U} dV \right) dt = 0 \tag{2.70}$$

For a constant-volume cell, with centre P and volume V_P, the volume integral of the temporal term in Eqn. (2.70) can be reduced to:

$$\int_t^{t+\delta t} \left(\int_V \frac{\partial \alpha}{\partial t} dV \right) dt = \int_t^{t+\delta t} \frac{\partial \alpha}{\partial t} V_P dt = (\alpha_P^{t+\delta t} - \alpha_P^t) V_P \tag{2.71}$$

By using Gauss' theorem, Eqn. (2.22), the convective term in the indicator equation can be rewritten in a face-based form:

$$\int_t^{t+\delta t} \left(\int_V \nabla \cdot \alpha \vec{U} dV \right) dt = \int_t^{t+\delta t} \left(\sum_{f=1}^{n} \alpha_f F_f \right) \tag{2.72}$$

As discussed in §2.3, the Crank-Nicholson temporal scheme is used for the momentum and indicator functions. This scheme introduces a half-weighting factor between old and new time steps, so that the convective term in Eqn. (2.72) may be discretized in time as:

$$\int_t^{t+\delta t} \left(\sum_{f=1}^{n} \alpha_f F_f \right) = \sum_{f=1}^{n} \frac{1}{2} \left(\alpha_f^t + \alpha_f^{t+\delta t} \right) F_f \delta t \tag{2.73}$$

Combining the temporally discretized terms of the indicator function gives:

$$(\alpha_P^{t+\delta t} - \alpha_P^t) V_P + \sum_{f=1}^{n} \frac{1}{2} \left(\alpha_f^t + \alpha_f^{t+\delta t} \right) F_f \delta t = 0 \tag{2.74}$$

By separating the α-values at old and new time steps, Eqn. (2.74) can be rearranged as:

$$\alpha_P^{t+\delta t}\frac{V_P}{\delta t}+\sum_{f=1}^{n}\frac{1}{2}\alpha_f^{t+\delta t}F_f=S_{\alpha_P} \tag{2.75}$$

with the temporal source for α defined as:

$$S_{\alpha_P}=\alpha_P^t\frac{V_P}{\delta t}-\sum_{f=1}^{n}\frac{1}{2}\alpha_f^t F_f \tag{2.76}$$

The equation for α at the new time step uses values for α at the centre of control volume and at the faces. In the next section, the CICSAM[5] compressive differencing scheme developed by Ubbink (1997) is described. Comparitive studies by Ubbink (1997) and recently Greaves (2004) indicated that CICSAM offers superior interface transport compared to volume tracking schemes and even geometry-based schemes. CICSAM guarantees a bounded solution while maintaining the sharpness of defined interfaces. The scheme supplies a weighting factor β which is used to express the α face value in terms of a cell and its nearest neighbours:

$$\alpha_f=\beta\alpha_P+(1-\beta)\alpha_N \tag{2.77}$$

By using Eqn. (2.77) to reformulate Eqn. (2.75), the indicator equation is written in the required form for the solver:

$$a_P\alpha_P^{t+\delta t}=\sum_{nb=1}^{n}a_{nb}\alpha_{nb}^{t+\delta t}+S_{\alpha_P} \tag{2.78}$$

The derivation of CICSAM is summarized in the following section— the CICSAM weighting factors calculated there is used in the solution of the discretized indicator function, Eqn. (2.75).

CICSAM Volume-of-Fluid

The scheme derived by Ubbink (1997) is a compressive scheme and based on the *Normalized Variable Diagram* (NVD) of Leonard (1979). CICSAM combines the *Convection Boundedness Criteria* (CBC) with the *Ultimate Quickest* (UQ) (Leonard, 1991) differencing scheme through a weighting factor determined by the orientation of the interface to the direction of motion.

The terminology of the donor (D), acceptor (A) and upwind (U) cells is illustrated in Fig. 2.3. When the flux direction is considered, the donor cell is the cell from which fluid is flowing into the acceptor cell. The upwind cell is a pseudo cell, located on the vector connecting the donor and the acceptor cells, at the same distance from the donor cell as the acceptor, but in the opposite direction. The scheme is directionally based on the flux through any face and used throughout the description in this section.

[5]Compressive Interface Capturing Scheme for Arbitrary Meshes

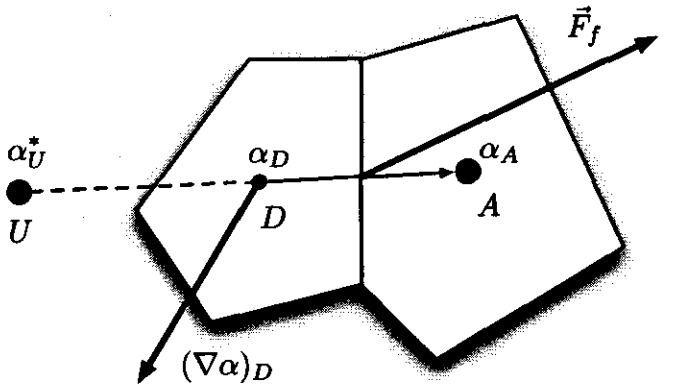


Figure 2.3: The donor-acceptor cell formulation and the predicted upwind value.

Throughout the derivation and application of CICSAM, the α values are rigorously bound between the global limits of one and zero, to ensure local boundedness.
The CBC value $\tilde{\alpha}_{fCBC}$, for a face is determined by:

$$\tilde{\alpha}_{fCBC} = \begin{cases} min\left\{1, \frac{\tilde{\alpha}_D}{c_D}\right\} & \text{for } 0 \leq \tilde{\alpha}_D \leq 1 \\ \tilde{\alpha}_D & \text{for } \tilde{\alpha}_D < 0 \text{ and } \tilde{\alpha}_D > 1 \end{cases} \tag{2.79}$$

and the UQ value $\tilde{\alpha}_{fUQ}$, is given by:

$$\tilde{\alpha}_{fUQ} = \begin{cases} min\left\{\frac{8c_D\tilde{\alpha}_D+(1-c_D)(6\tilde{\alpha}_D+3)}{8}, \tilde{\alpha}_{fCBC}\right\} & \text{for } 0 \leq \tilde{\alpha}_D \leq 1 \\ \tilde{\alpha}_D & \text{for } \tilde{\alpha}_D < 0 \text{ and } \tilde{\alpha}_D > 1 \end{cases} \tag{2.80}$$

The upwind value is calculated by using the gradient of α in the donor cell and bound between the physical limits:

$$\alpha_U = min\{max\{(\alpha_A - 2(\nabla\alpha)_D, 0\}, 1\} \tag{2.81}$$

The normalized value $\tilde{\alpha}_D$, for the donor cell is given by:

$$\tilde{\alpha}_D = \frac{\alpha_D - \alpha_U}{\alpha_A - \alpha_U} \tag{2.82}$$

The courant number (c_D) used in the calculation of the UQ and CBC values is found by summing the fluxes leaving each control volume. This gives an indication of the amount of fluid available in a cell, so that more fluid is not removed from the cell in a time step than is available in the volume:

$$c_D = \sum_{f=1}^{n} max\left\{\frac{-F_f\delta t}{V_D}, 0\right\} \tag{2.83}$$

To calculate the weighting factor γ_f, the orientation of the interface and the direction of motion is taken into account:

$$\gamma_f = min\left\{k_\gamma \frac{\cos(2\theta_f)+1}{2}, 1\right\} \tag{2.84}$$

where the recommended value of k_γ is 1.0, and the angle of the interface θ_f is determined by using the gradient of α in the donor cell:

$$\theta_f = \arccos\left|\frac{(\nabla\alpha)_D \cdot \vec{d}_f}{|(\nabla\alpha)_D||\vec{d}_f|}\right| \tag{2.85}$$

The normalized face value $\tilde{\alpha}_f$, for the CICSAM scheme is:

$$\tilde{\alpha}_f = \gamma_f \tilde{\alpha}_{fCBC} + (1-\gamma_f)\tilde{\alpha}_{fUQ} \tag{2.86}$$

The CICSAM weighting factor β_f, is then calculated:

$$\beta_f = \frac{\tilde{\alpha}_f - \tilde{\alpha}_D}{1-\tilde{\alpha}_D} \tag{2.87}$$

After the indicator equation (Eqn. (2.76)) has been solved, the Crank-Nicholson weighting for the old and new time steps of α at the face (α_f^*) is used for the approximation of the face densities (Eqn. (2.10)) and the new mass flux for the momentum equations.

The new face value for α^* is obtained from:

$$\alpha_f^* = (1-\beta_f)\frac{\alpha_D^t + \alpha_D^{t+\delta t}}{2} + \beta_f \frac{\alpha_A^t + \alpha_A^{t+\delta t}}{2} \tag{2.88}$$

Ubbink (1997) also developed a corrector step in the event of non-physical values for α. The level of unboundedness is determined and the β_f correction factors is adjusted to balance the unboundedness and the new indicator values are determined.

This concludes the discretization of the equations for a two-fluid system. The next section presents the algorithm for solving the velocity and pressure, and then the combined algorithm to solve for a two-fluid system.

2.4 Solution algorithms

The momentum and pressure equations are solved using a segregated approach, where momentum and pressure is solved sequentially through a process of iteration. A transient version of the SIMPLE algorithm is implemented in this study, and follows the steps put forward by Versteeg & Malalasekera (1995).

Pressure-velocity coupling with SIMPLE

The transient SIMPLE algorithm is summarized as follows:

Step 1: Initialization At the initial time step, distributions for the velocity and pressure are determined based on the physical conditions and problem geometry.

Step 2: Momentum prediction The guessed pressure field P^* is used to solve the momentum equations in Eqn. (2.54). The new velocity field $\vec{U}^*$ is not conservative.

Step 3: Pressure correction equation The equation for the pressure correction (Eqn. (2.68)) is assembled by using the new velocities to determine the volume flux in-balance. In cases of severe mesh non-orthogonality, explicit iterations over the non-orthogonal terms in the pressure correction equation can be done here.

Step 4: Correction step Using the pressure correction (P') calculated in the previous step, correctors for the volume flux are calculated with Eqn. (2.69) and the pressure and velocity are explicitly corrected with Eqn. (2.55) and Eqn. (2.56).

Step 5: Test for convergence If the required convergence level has been reached, advance the time step to the next level and start the process from step two again. If convergence has not been reached, repeat from step two, but without advancing the time step.

Solving the combined two-fluid system

The solution of the combined two-fluid system can be summarized with the following steps from Ubbink (1997):

Step 1: Initialization At the initial time step, distributions for the velocity, pressure and volume fraction are determined based on the physical conditions and problem definition.

Step 2: New time step Calculate the Courant number and adjust the time step if necessary.

Step 3: Indicator function Using the old time level's volumetric fluxes, solve the indicator function, Eqn. (2.75) and calculate the new α^* with Eqn. (2.88)

Step 4: Fluid properties Use the new α^* values to determine the fluid properties with the closure equations in Eqn. (2.10) and Eqn. (2.11). This gives new density and viscosity fields and updated face density values.

Step 5: Flow solution Using the new fluid properties, continue with the SIMPLE algorithm until convergence for the time step has been reached.

Step 6: Final time step If the final time step has not been reached, the process is started from step two again.

2.5 Closure

In this chapter the theoretical definitions and numerical discretizations of the governing equations for two-fluid systems were given. An indicator function, based on volume fractions, was defined to determine the distribution of the fluids and associated properties. The equations were derived for unstructured meshes and non-orthogonality is treated explicitly as source terms in the equations. The chapter closed with a description of a compressive convection scheme for the indicator function that maintains the interface shape in a conservative manner. The algorithms for the determination of the correction factor and the solution of the combined two-fluid system were presented. In the following chapters, an adaptive mesh refinement strategy is developed that will ultimately be combined with the two-fluid system.

Chapter 3

Adaptive grid generation

3.1 Introduction

In this chapter, hierarchical grid generation techniques are examined. The popular quadtree method enables subdivided grid elements to be stored in a hierarchical structure. By using recursive functions, the tree can be traversed to find neighbours and establish connectivity between elements. The use of quadtrees has been shown by Greaves (2004) and Malik (2004) to be effective in extending the advantages of adaptive mesh refinement to the modelling of interfacial flows. (In all further discussions, what applies to quadtrees in two dimensions apply to octrees in three dimensions, only with added complexity in the algorithms).

A closer examination of the practical aspects of quadtrees revealed that the method is hampered by a strong dependence on a block structured mesh topology. Quadtrees depend on the fact that every sub element is logically divided in the same manner, e.g. a cell is subdivided into four cells, and those cells again subdivide into four cells. A subcell cannot be subdivided into for example, eight cells and still exist in the quadtree structure. The establishment of neighbour connectivity is another very important task in general mesh processing. When a quadtree is created, the indices used to label its elements are used to calculate connectivity matrices. When the mesh is unstructured, there are no indices or labelling information with which to establish those connections. Quadtrees are thus dependent on the logical structure of the mesh for connectivity and not the geometrical structure in physical space.

To provide two-fluid models on unstructured grids with a workable mesh refinement strategy, the issue of connectivity between refined cells on unstructured grids must be adressed in an *efficient* manner. In Chapter 2 different refinement strategies and options were discussed, but it was found that the researcher who requires hierarchical refinement on unstructured meshes is left wanting. The provision of automatic adaptation requires either 1) unstructured tetrahedral meshes that are regenerated at the required fineness, or 2) block structured Cartesian meshes.

Recent two-fluid simulation strategies use face-based discretization schemes (e.g. Ubbink (1997) and Ferziger & Peric (2002)) that were specifically developed for use

on arbitrary meshes. Suitable grid designs should support this face-based approach and be able to provide the grid information in the format of a list of unique faces connected to strictly two adjacent cells.

The solution to this problem is a grid structure that has three important features: 1) hierarchical structures for effective creation, storage and refinement of elements, 2) connectivity that is established using geometric methods instead of logical methods, 3) the grid information can be supplied in an abstraction that is suitable for face-based discretization strategies.

The following sections describe the basic principles of quadtrees for comparison purposes. A new arbitrary design is then introduced that provides a grid with the hierarchical structure required and supports face-based methods.

3.2 Quadtrees

The suitability of the quadtree data structure for unstructured mesh refinement is evaluated by looking at the creation of quadtrees, discussing the topology and how grid operations are performed and identifying advantages and shortcomings in the concept.

Quadtree generation

The origin of quadtrees is in image processing (Foley *et al.*, 1990). It is then fitting to discuss the structure by referring to a typical problem encountered in computer graphics: how to represent a geometrical shape (and its outline) as efficiently and accurately as possible in a data structure. Fig. 3.1 shows a part that is represented by a uniform spatial enumeration method, and the same part when it is represented by a quadtree structure. The connection between the graphics field and simulation is of course that a simulation problem is also defined on a numerical grid, where the active cells in the numerical problem are analogous to the "filled" cells of the graphics object. In both instances, the occurence of active cells needs to be minimized.

The uniform structure seems at first to be "good enough". By virtually dividing the part into several uniform elements, the part is stored by marking an element as "on" or "off". The part is represented with reasonable accuracy, and by increasing the resolution of the grid, the accuracy is improved. The inefficiency of uniform structures is however soon realized when more accurate representations are required. The resolution is increased globally, even in solid regions, with a corresponding increase in storage requirements and access times.

A more effective approach is using a system of quadtrees. Fig. 3.2 depicts the part, with an illustration of its quadtree. The part is first divided into coarse elements, and every element that contains a piece of the part is flagged as "filled" (or "empty" if there is no containment). The filled elements are then recursively examined and in turn subdivided, until a predetermined level of subdivision is reached.

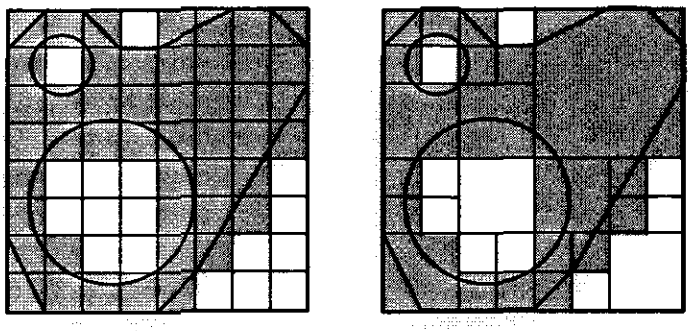


Figure 3.1: An object defined with (a) spatial-occupancy enumeration (b) a quadtree. (Foley *et al.*, 1990)

After an element is subdivided, the "filled" status of the children elements are determined, and if the region is homogeneous, no more refinement is needed. If there are mixed filled and empty children, the element is flagged as "partial" and the recursion process is continued for the filled children.

Traversal and neighbour finding

The most important grid operation that must be done in a quadtree is the process of finding neighbours for the cell. A short summary of the process is given here; detailed implementation details can be found in Foley *et al.* (1990), Yiu *et al.* (1996) and Malik (2004).

Traversing a quadtree is dependent on the numbering system for the constituent elements of the tree. The numbering scheme can be seen in Fig. 3.2 and is repeated exactly for all subsequent refined children of an element. The search for a neighbour of an original element consists of two operations. The first is to traverse the tree upwards to find the common ancestor for the original and neighbour element; the second is to traverse downwards in the neighbour's branch to find the correct neighbour. The system of finding the correct neighbour is based on the logical elimination of approach directions between the two elements, until the only possibility is the direction of the neighbour. For example, a north neighbour cannot be approached by any other neighbours than a south one, thus it can be deduced that the south element's neighbour is the first north element found enroute the downwards path. A typical path between two neighbouring elements are shown in Fig. 3.3. To find the neighbour on e.g. the north side of element A, the tree is traversed upwards to the root element and one level down to find the neighbour B. (The common ancestor is the first element not reached from a child on the element's desired direction side. In this case, the root element).

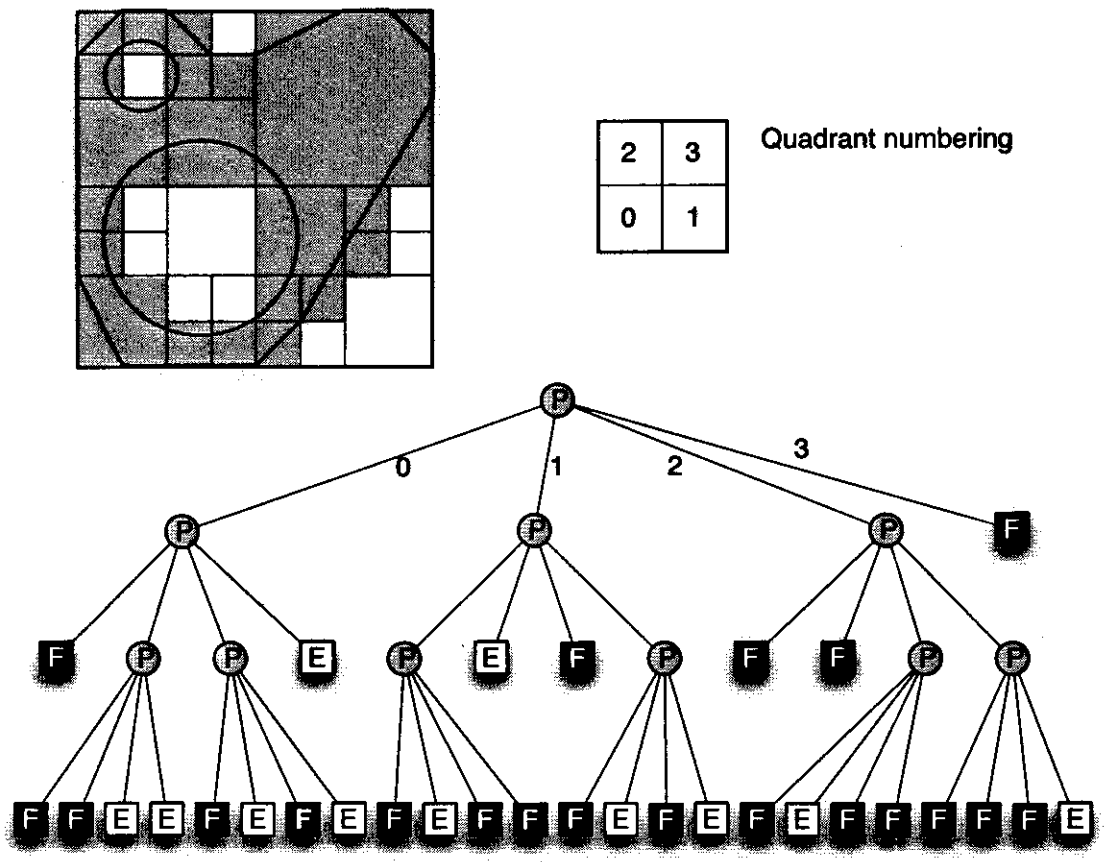


Figure 3.2: An object represented using a quadtree and the quadrant numbering scheme. For the quadtree data structure, use F = full, P = partially full, E = empty. (Foley *et al.*, 1990)

Advantages and shortcomings

The advantages of quadtrees are twofold: 1) it is an efficient data structure that can be used to store a *virtually* compressed version of an object and 2) algorithms for access and tree traversal are uncomplicated because of the uniformly repeating structure of the grid. The main limit on performance is the number of levels in the tree and the computational effectiveness of traversal algorithms. Certain computational operations are very effective because parent elements can be examined first and based upon that, the decision can be made to examine children elements.

A major shortcoming (or arguably a design feature) of quadtrees is the dependance on the logical order of elements to determine connectivity. If an element is logically numbered as being on the north side of an edge as element zero, connectivity is established using that definition. (Even if in reality the element is not situated there anymore). This approach is then naturally only valid when all elements are

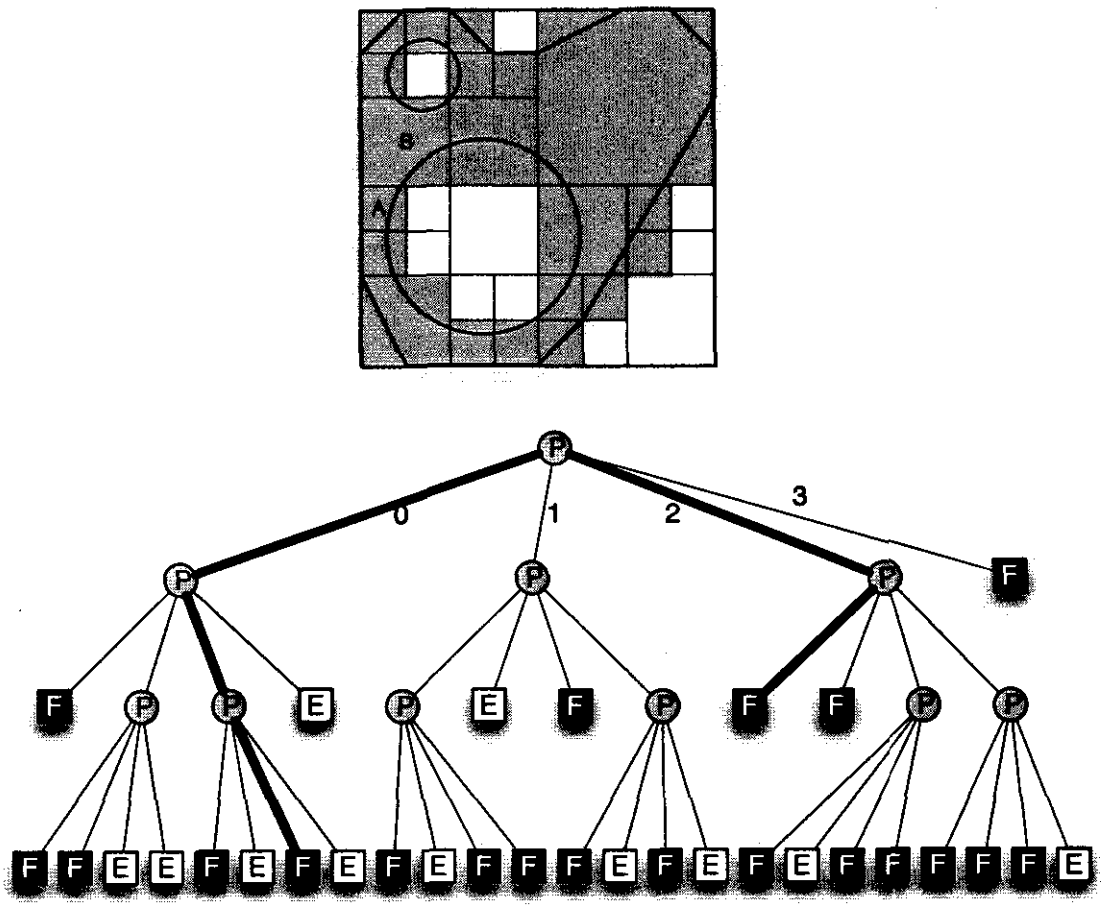


Figure 3.3: Finding the neighbour of a quadtree element. (Foley *et al.*, 1990)

created and numbered in the *same* logical sequence.

Where the quadtree structure breaks down is when an attempt is made to use quadtrees for unstructured meshes. In unstructured formulations, no previous connectivity information exists except the faces that connect cells. There are no logical numbering sequence, and indeed, elements may not even have sequential numbers, not to mention any structure which can be labeled as north, south or panel 0, panel 1, etc. That means that existing recursive strategies for quadtrees and their numbering sequences are unusable for arbitrary grids.

In the next section, an alternative structure to the logical design of quadtrees is presented. This structure has the same hierarchical advantages, but connectivity is based on the geometrical relations between elements.

3.3 Arbitrees

Introduction

A new hierarchical structure for unstructured meshes is introduced in this section. It is called "arbitrees" and lexicologically derives from *arbitrary*, meaning "based on or derived from random choice" and "tree", meaning "a diagram with a structure of branching connecting lines (Oxford, 1995). The most important difference between quadtrees/octrees and arbitrees is the arbitrary nature of the hierarchy stored in an arbitree.

The key element of the new design is the recursive grid cell shown in Fig. 3.4. In Fig. 3.4(a) the structure of a regular quadtree is shown where the child cells are stored in a linked list in the parent cell. Compare this structure with the arbitree in Fig. 3.4(b), where children are stored in the grid, rather than in the parent itself. The grid cell manages all the typical housekeeping duties like creating children cells and defining faces. The logical outlay of the recursive grid cell object is shown in Fig. 3.5. Recursive grid objects can store any number of children cells and may also be anisotropically defined.

Terminology

Arbitrees represent full three-dimensional entities. In this study, unstructured hexahedrons were used as primitive shape, but the methodology could be extended to tetrahedrons as well. For clarity, all the figures referred to during the discussions are presented as two dimensional illustrations. In the discussions, some references are made to software design concepts. The design of the grid object depends on object-oriented features in the software language and makes full use of data encapsulation and abstraction. New cells for example, are created and stored in the refined grid object, which makes memory allocations much more dynamic and more granular. There are no large, globally structured data storage arrays for cells, faces and vertices.

The following terminology applies to the arbitree structure, and is used to clarify discussions in the following sections:

Root grid The root grid is the base level in the hierarchy and represents the coarsest version of the domain discretization. Cells in the base level may be refined to a higher level, but they may not be unrefined past the base level. The base level should be fine enough to capture the boundaries even as the cells are refined. This is further discussed in §3.3. The merging of cells located at the base level into larger cells is not supported, because it is seldom needed in this context. The ability to define an unstructured base grid enables the model to be designed in such a way that inactive regions already have larger base cells. The root grid further contains grid management features and data that are only needed at the base level. Additional data structures representing the

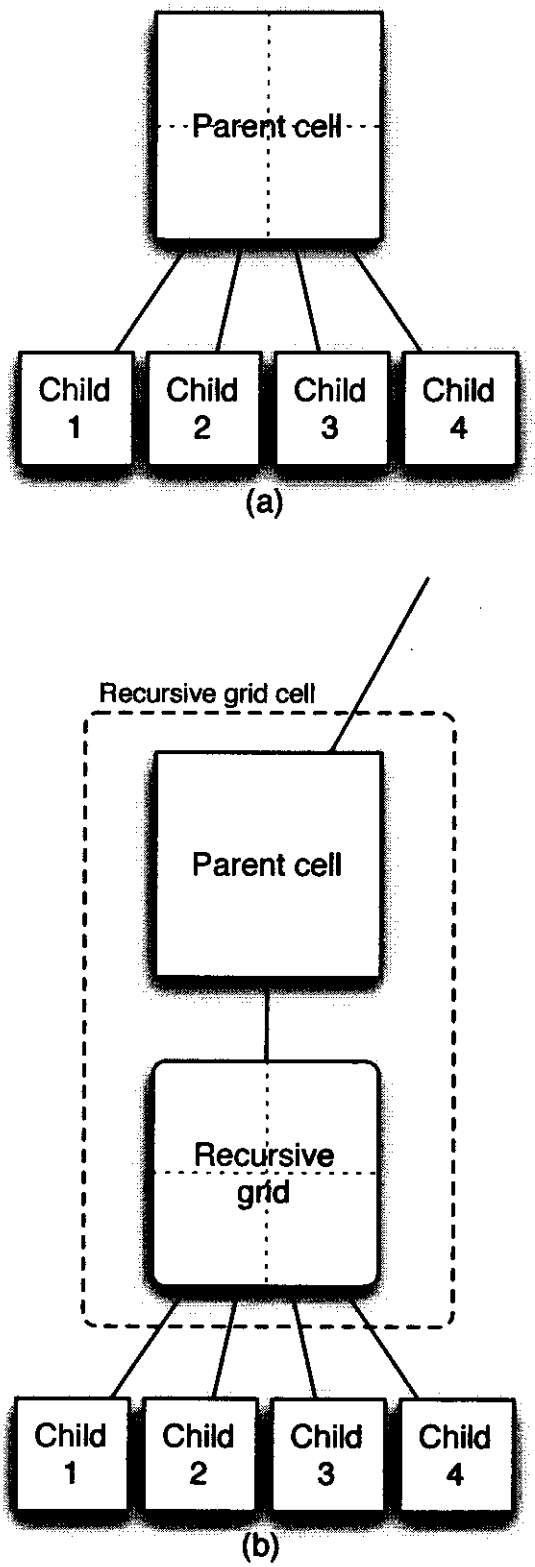


Figure 3.4: The concept of a recursive grid cell. a) Typical quadtree b) Recursive grid cell for the same element structure.

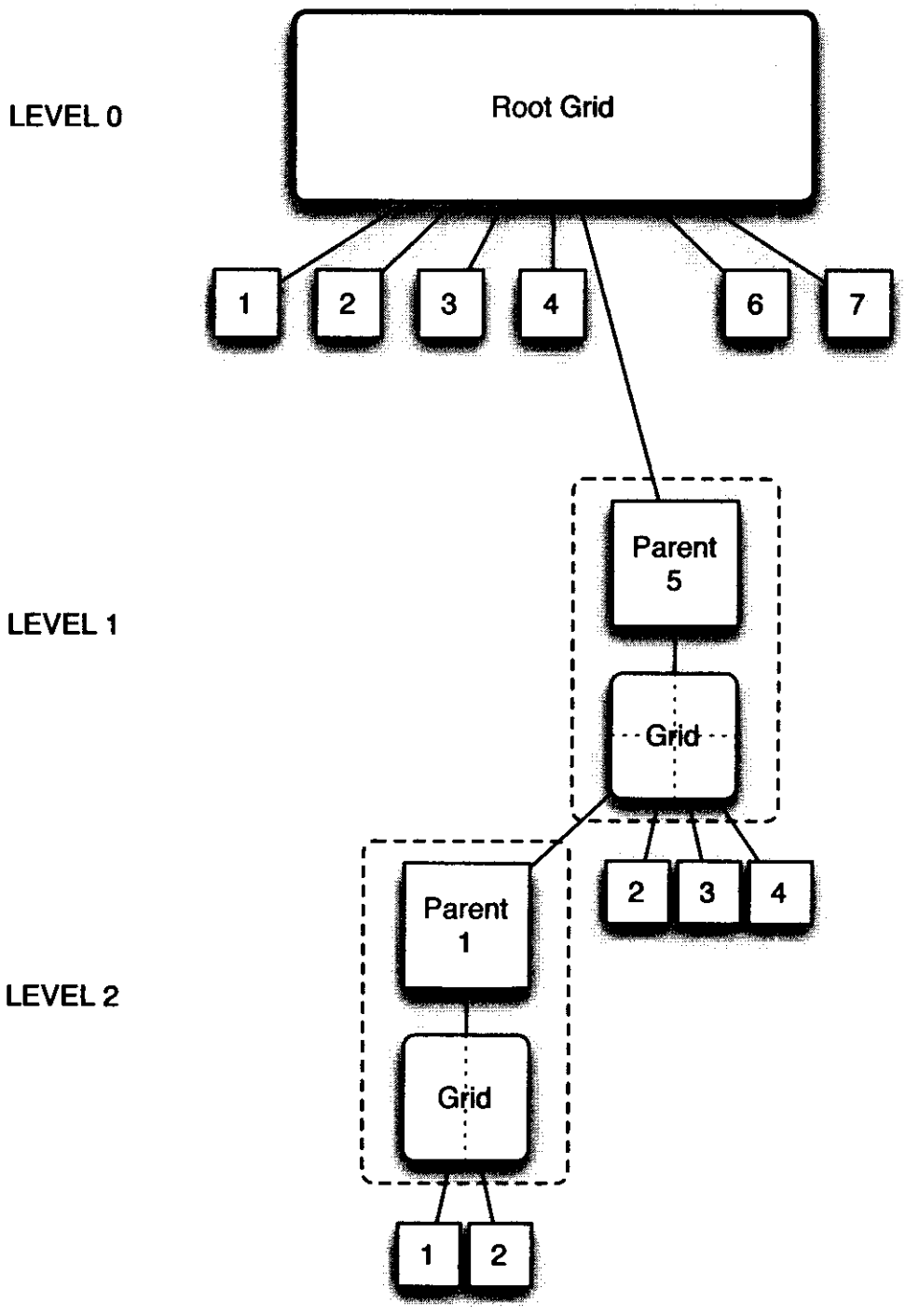


Figure 3.5: The hierarchical view of an arbitree.

logical structure of the grid are found here: the global cell front, global refined list, active inner faces and active outer faces.

Regular grid object The grid object forms the base object for the root grid and is used as the recursive object in cells. It contains the inner faces, outer faces, cells and vertices. A pointer to the parent cell is provided, linking the grid to its parent in the lower level of the tree. The grid also contains local axes directions, to enable anisotropic refinement of cells as described by Jasak (1996).

Vertex Vertices or node points define the locations of cell corners in three-dimensional Cartesian space. To create new cells, the vertices are created first at the required positions and the cells are then constructed from these defined corners. Vertices are stored locally in the parent grid.

Recursive grid cell A cell is defined by its corner vertices and describes their topology. A recursive grid cell is defined as a cell containing a grid object. Cells are stored locally in the parent grid.

Parent cell When a cell is refined, that cell is the parent cell for the recursive grid object it now contains.

Parent grid When a grid is created, that grid is the parent grid for all the cells, faces and vertices defined in the grid.

Child cell, face or vertex The objects stored in a parent grid is said to be the child objects of their parent grid's parent cell.

Inner faces Inner faces define the connectivity between cells in the same parent grid. It is constructed in a manner defined by the cell topology. Only unique faces are allowed, in the sense that between two cells, there may be only one face. Inner faces are stored locally in the parent grid.

Outer faces Outer faces define the boundaries of their parent grid. For a root grid, this will be the external boundaries of the problem definition. For cells at a higher level of refinement, the outer faces are used to determine connectivity between neighbouring cells but with different parent grids. An outer face only have one cell on its inside.

Global cell front The global cell front is a list of pointers to all cells that are not refined (i.e. do not contain an initialized grid object). It represents the cells that are available as computational nodes and is used to assemble the matrices needed for the numerical solver.

Global refined list The global refined list is a list of pointers to all cells that are refined. By way of mutual exclusion, these cells are not in the global refined list. The list provides efficient access to cells that may be tested for unrefinement.

Active inner faces The *active inner faces* list stores pointers to the connecting faces between cells in the global cell front. It represents the faces that are available as computational nodes. It is analogous to the face list of indices used in face-based numerical methods for inner face calculations (Chapter 2).

Active outer faces The *active outer faces* list stores pointers to the faces that are connected to external (domain defined) boundary faces. It is analogous to the face list of indices used in face-based methods for boundary face calculations (Chapter 2).

Arbitree generation

Before describing the steps of arbitree grid generation, three common operations that occur during grid generation is explained. These are 1) the linear interpolation between vertices to obtain a new vertex, 2) the method used to determine internal connectivity between structured cells in the same parent grid and 3) the method used for unstructured connectivity between refined cells from different parent grids and/or levels.

Vertex interpolation

Vertices are interpolated using linear distance weighting in three dimensions, as shown in Fig. 3.6. The formula for the vertex distance of $\vec{V}_v$ for any n vertices, so that $v \leq n$ and $\vec{V}_v$ lies between the endpoints $\vec{V}_1$ and $\vec{V}_2$, is given by:

$$\vec{\delta}_V = \frac{1}{n+1}(\vec{d}_{21}) \tag{3.1}$$

for the incremental distance $\vec{\delta}_V$
and

$$\vec{V}_v = \vec{V}_1 + v\vec{\delta}_V \tag{3.2}$$

where v is the vertex number and $v \leq n$.

Note: When constructing the recursive grid, the corner vertices obtained from the parent cell can be treated in two ways. The first is to use the vertices directly as part of the new grid's list of vertex references and the second is to create new vertices and copy the locations. It is a subtle design decision with far reaching implications.

When the *same* vertices are used in the new grid, vertex-based connectivity for refined cells can be determined. This is mostly useful when vertex values of variables are calculated as averages from surrounding cells[1]. The usefulness becomes a disadvantage for the algorithm used to determine inner cell connectivity. When

[1]Vertex-based values make life easier when constructing contour plots of variable values (Hearn & Baker, 1994). Without vertex-based values, contours and other post-processing tasks are done on a cell-by-cell basis using reconstructed gradients (Mavriplis, 2003)

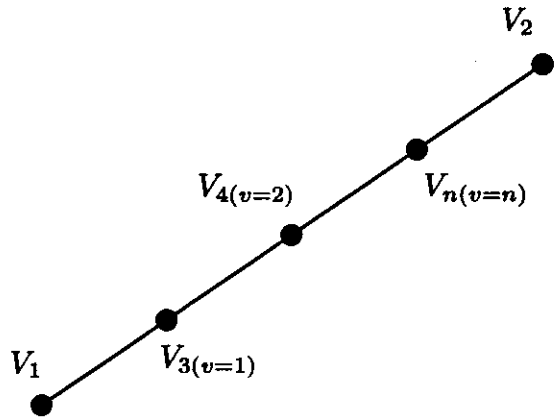


Figure 3.6: Vertices are linearly interpolated between two endpoints.

cells are refined—especially with more than one level—the refined faces are then added to vertices stored at a few levels lower than the current cell's level. To unrefine a cell, the recursive grid must be deleted, including its faces, which implies that the faces must be removed from the lists stored by lower level vertices. This slows down unrefinement steps because of the extra manipulation of rather extended lists. Additional faces referenced by a vertex in this way also increase the time taken by the connectivity search for new inner cells. The stored faces cannot be used to calculate the inter-level refined connectivity, because with hanging nodes there are no connections to the higher levels.

When only the locations of the parent cell's corners are used, the advantage is that the face lists stored on the vertices only store references to faces in the same parent grid as the cells and vertices. For the purpose of the inner-cell connectivity algorithm, only local cells can be neighbours, so the reduced lists are a great speed improvement. When a grid is deleted, the faces can be deleted too without adjusting the reference lists stored by lower level vertices. The disadvantages are that vertex-based connectivity is lost and the ability to cheaply calculate vertex-based average values. *caveat emptor*

Boundary vertex interpolation

Vertices that define boundary faces are treated in the same manner as other vertices. New boundary vertices are constructed with linear interpolation between endpoint vertices. In the case of curved boundaries, this can lead to the situation of Fig. 3.7(a) where the interpolated vertex is not located on the boundary. This is a consequence of approximating curved surfaces with flat faces, and can be alleviated by using spline definitions for the boundary, or by increasing the resolution of cells on the boundary, shown in Fig. 3.7(b). In this study, the second approach of more refined boundary cells are used.

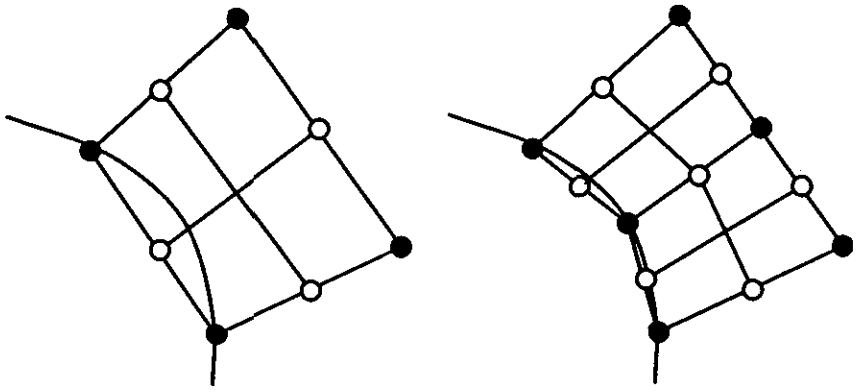


Figure 3.7: Interpolating boundary vertices. a) New vertex lies outside boundary, rather use b) with more refined base cells for a better approximation.

Structured face connectivity

To obtain the connecting faces between cells in the same parent grid, the structured nature of the grid can be exploited. Grids created by a refinement operation are constructed with a local sequence of numbers, depicted in Fig. 3.8. Cell definition for hexagonal cells follows the right hand rule, where vertices are numbered counter-clockwise in the lower level, before advancing a level and repeating the sequence. The cell faces are constructed with the same right-hand rule, where numbering are done counter-clockwise so that the area vector of the face points outwards. To search for matching inner faces, an algorithm developed by Du Toit (1998) is used. Face numbers are rotated in sequence so that the smallest number is first in the sequence. Note that this rotation does not influence the direction of the area vector. The face pointer is then stored in a list of cached faces by the smallest index. As new faces are added to the grid, their numbers are rotated, and the face is added to the smallest number for the face, and so on. If there are already faces stored at that number, a check is performed to determine if an existing face uses the same numbers. If no matches for the face are found, it is a unique face and may be added to the inner face list. If a match is found, the face is passed over and the algorithm continues.

Unstructured face connectivity

When evaluating the unstructured connectivity illustrated in Fig. 3.9(a), a more general approach than comparing vertex numbers is needed. A comprehensive test of polygon intersection between the two faces would indicate if there is a common area between them. Computationally, this is an expensive test to execute for the thousands of faces that could possibly be connected to each other. The need for

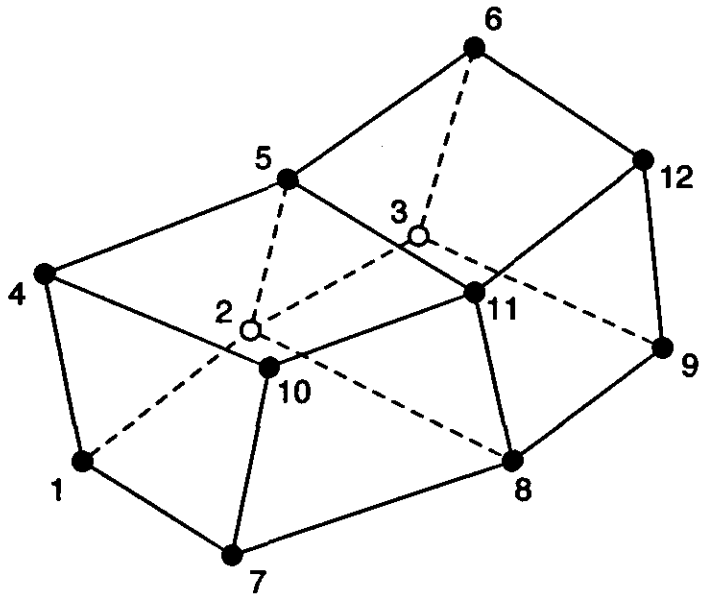


Figure 3.8: The numbering sequence for structured cells. First cell is: 1 2 5 4 7 8 11 10, second cell: 2 3 6 5 8 9 12 11. The inner face, numbered from the first side is: 2 5 11 8 and numbered from the second side is: 2 8 11 5

a more effective algorithm is further stressed when using an adaptive mesh, where connectivity is recalculated dynamically as part of the solution process.

The problem can be simplified by examining the orientation and size of the faces that will be tested for connectivity in a hexagonal arbitree grid. Because of the grid construction, faces that might be possible connective faces, are always located within each other (i.e. a smaller refined face connecting to a larger face at a lower level in the grid) and connection takes place over complete face areas. Larger faces are completely covered by smaller faces and faces are co-planar with each other and not twisted or warped. When these features are taken into consideration, it becomes clear that it is sufficient to only test if the centre of the smaller face is situated inside the polygon described by the bigger face.

The method proposed by Bourke (1987) in Fig. 3.9(b) offers an efficient solution to the classical point-in-polygon test problem. It is valid for three dimensional calculations, and works as follows: for any point situated inside a polygon, the sum of angles, $\sum \theta$ between the test point and every pair of edge points is determined. The sum will only be 2π if the point is on the polygon plane and on the interior. The angle sum tends to zero if the point lies outside the polygon and away from the plane. A numerical tolerance can be included in the algorithm to handle slightly warped faces: faces that might not be exactly coplanar but the numerical error introduced is negligibly small. Typically this should only be used to eliminate errors due to machine inaccuracies. If there are significantly warped faces in the grid, the mesh

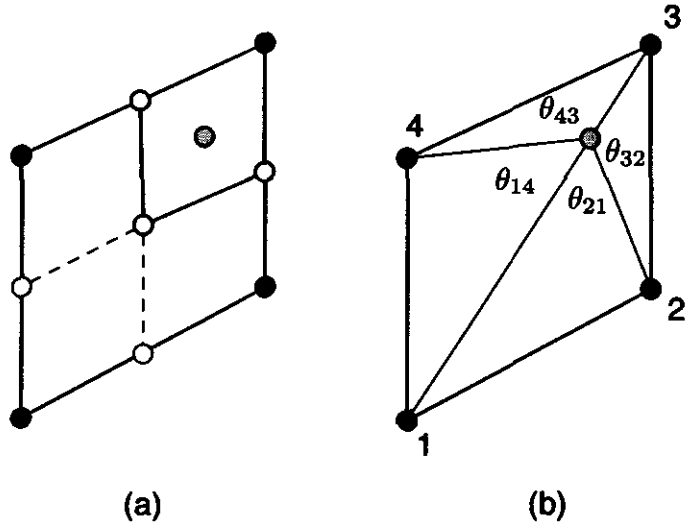


Figure 3.9: Unstructured connectivity between a refined face and parent face. a) The actual connectivity b) Connectivity approximated as point-in-polygon test. If the point is co-planar and on the interior, $\sum\theta = 2\pi$

should be properly repaired before attempting simulations.

To create an arbitree grid

The steps to create a refined cell with an arbitree structure are now described:

Step 1: Identify the cell

The cells that need refinement are identified by the refinement algorithm. The process of evaluating and marking cells for subdivision is discussed in the following chapter. For now, it can be assumed that the cells have already been chosen for refinement.

Step 2: Initialize recursive grid cell

The next step in Fig. 3.10 is to initialize the parent cell's recursive grid pointer. The grid object is instantiated and enough memory is allocated to store the grid elements (cells, faces, vertices). The memory size and cell distribution of the new grid is prescribed by the refinement scheme and is not limited to a fixed $2 \times 2 \times 2$ scheme like octrees. Combinations like $1 \times 2 \times 1$ or $1 \times 4 \times 2$ can also be used for the number of cells in the local grid directions[2].

[2]All of the test cases presented in this study used a quadtree-like scheme of $2 \times 2 \times 1$. In all cases, the local grid directions coincide with the global Cartesian axes. The subdivision applied to each cell was then $2\times$ in the x-direction, $2\times$ in the y-direction and $1\times$ in the z-direction.

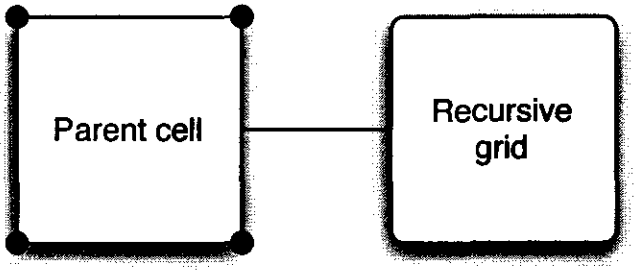


Figure 3.10: Initializing the recursive grid pointer of a cell.

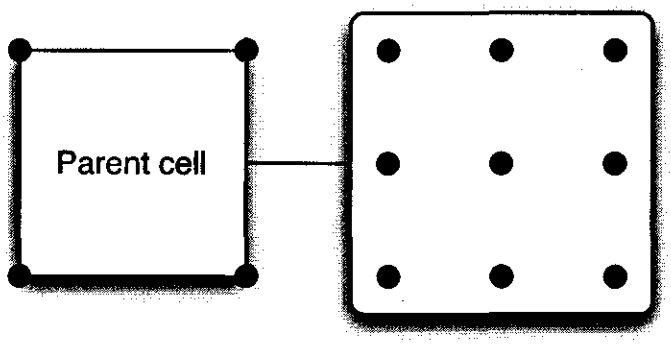


Figure 3.11: The cell vertices are interpolated for the new grid.

Step 3: Interpolate vertices

With memory allocated to the grid, construction of vertices can commence. Corner vertices for the grid are created with the coordinates supplied by the parent cell's vertices. The necessary vertices for cell creation is linearly interpolated between the corner vertices, see Fig. 3.11. Every vertex is unique, so that there are no double vertices in the parent grid.

Step 4: Cell creation

In Fig. 3.12 the cells are created to fill the mesh of vertices that was defined in the previous step. The linear interpolation and unique definition of the vertices ensure that the cells are created without overlapping volumes.

Step 5: Face definition and inner-cell connnectivity

The new cells are each processed in turn to define the faces. As the cells are processed, the faces of every cell is compared with the other cells (§3.3) to determine connectivity. If a face is found to be a "double" face that is already assigned to a

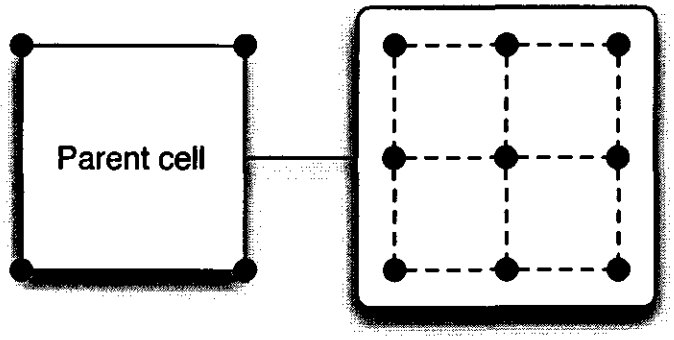


Figure 3.12: Cells are filled in from the vertices.

previous cell, the face is passed over and processing continues. Unique lists of inner and outer faces are obtained as shown in Fig. 3.13.

Step 6: Connectivity between grid and parent cell

The final step in Fig. 3.14 is to determine the connectivity between the parent cell and the new grid. This involves a comparison of the grid's outer face list with the cell's faces and finding matching pairs. Pointers to the matching refined faces are then stored by the cell faces.

By assigning grid faces to the corresponding cell face at the creation stage, the complexity of searching for refined neighbours is reduced. The inter-level neighbour search is discussed in the next section.

Traversal and neighbour finding

In this section, it is assumed that the proper root grid has already been created. The grid shown in Fig. 3.15 is used as an example grid for the discussion below.

Step 1: Create global cell front and refined cells list

Traversal starts in the cell list of the root grid and both lists are cleared. All the cells are examined in turn; when a refined cell is found, a pointer is added to the refined cells list. If the cell is not refined then it is added to the global cell front. The cells that are marked in Fig. 3.16 represent the two lists after the update.

Step 2: Update active inner faces

Traversal starts in the inner face list of the root grid and the active inner face list is cleared. All the inner faces are examined in turn. Shown in Fig. 3.17, a face can have one of three configurations: a) it is the only face between two unrefined cells,

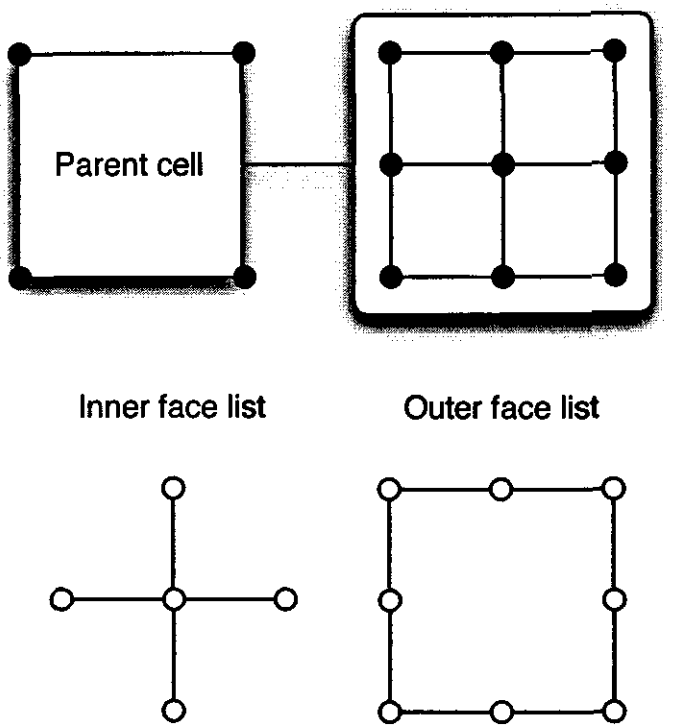


Figure 3.13: The faces are defined and tested for connectivity to obtain inner and outer face lists.

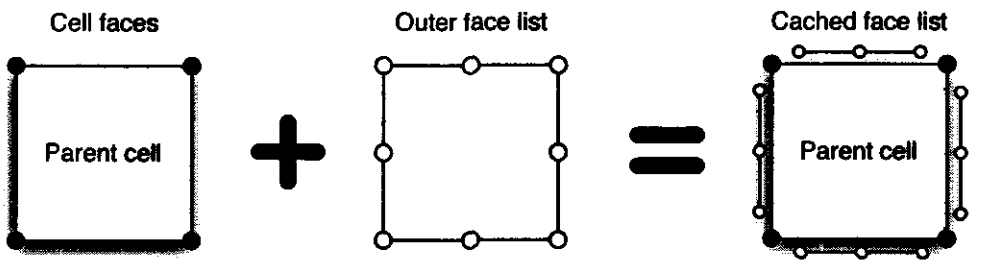


Figure 3.14: The outer faces of the grid are cached on the connecting cell faces.

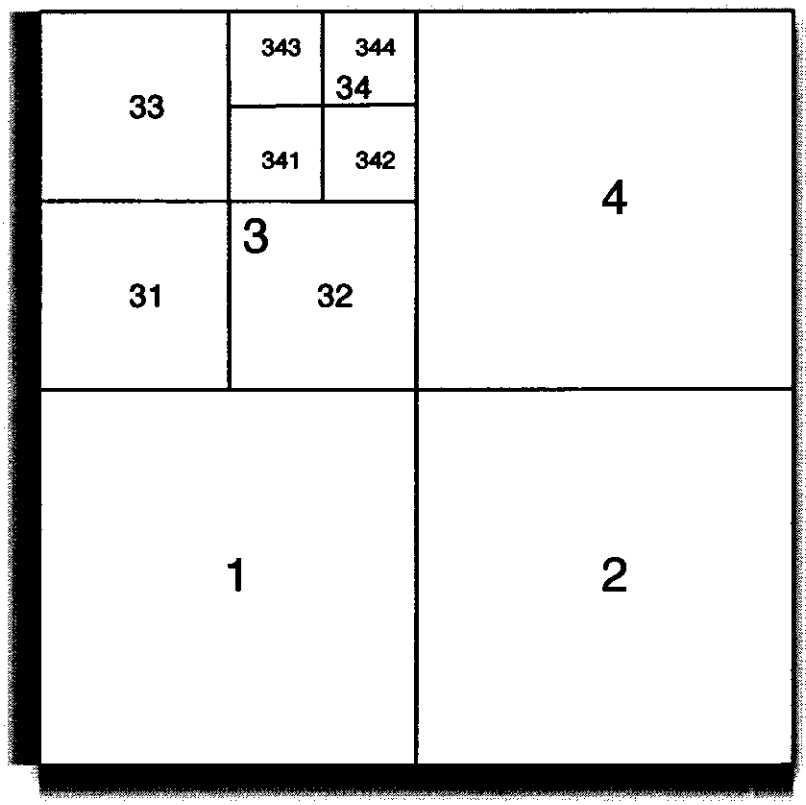


Figure 3.15: The root grid with two levels of refined cells. Local cell numbering is shown, where the parent cell number is added as a digit in front of the child numbers.

b) one of the neighbour cells is refined and c) both the neighbour cells are refined. The next action depends on the state of the neighbour cells:

a) No refined neighbours No further action is needed, the face pointer is directly added to the inner face list.

b) One neighbour is refined Examine the cached faces of the refined cell's recursive grid (from Step 6 above). If a cached face is not refined, add it to the inner face list. If the cached face is refined, recursively repeat Step 2 until no refined faces are encountered. For every recursion level, the inner face list of the recursive grid of the next refined neighbour is used as traversal list. Fig. 3.18 shows the connections between faces via the stored pointers where the marked faces indicate those that are eventually added to the list.

c) Both neighbours are refined The same basic procedure as in (b) is followed, but it is repeated for each cell. Instead of storing faces directly in the inner

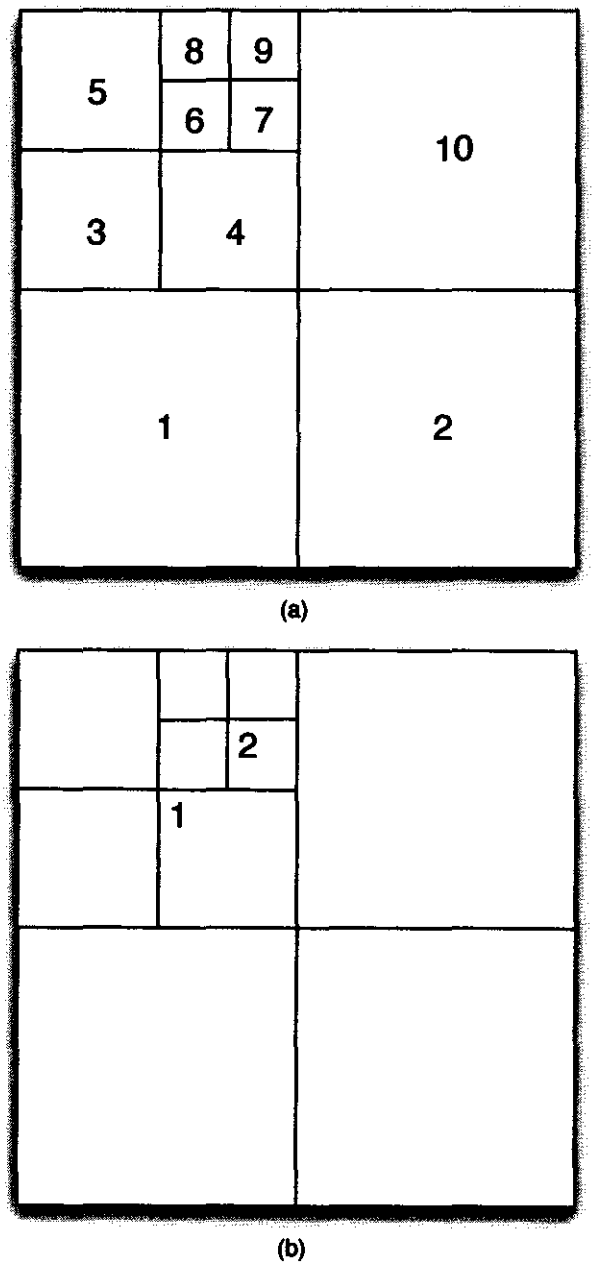


Figure 3.16: Updating the cell lists. a) Global cell front b) Global refined cells list.

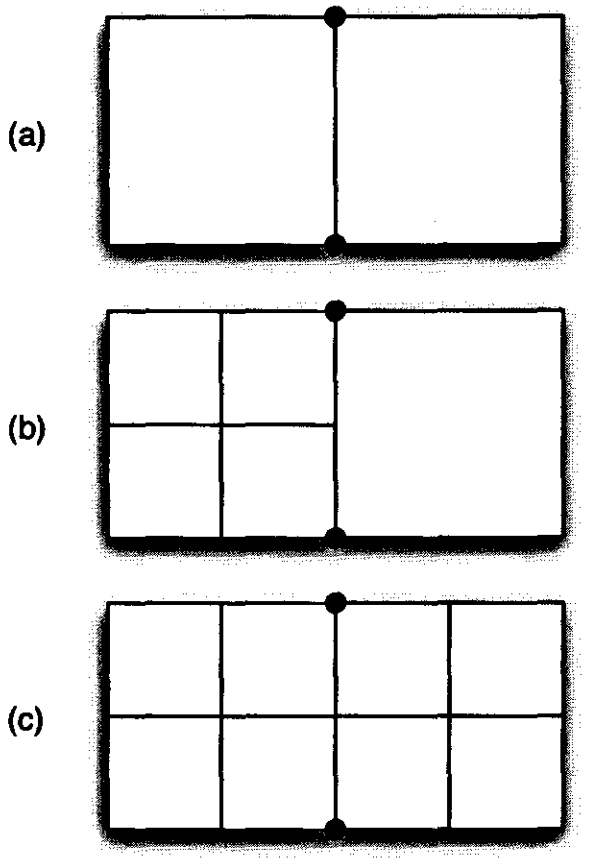


Figure 3.17: Different connections for an inner face. a) no refined neighbours b) one refined neighbour c) both neighbours are refined.

face list, the connecting faces from each side are stored in temporary lists. These lists are compared with each other for physical connectivity and the smallest connecting faces are then stored in the inner face list.

In all cases, when two faces are compared and the areas are found to be of equal size, the face on the side of the inside cell is preferred.

Step 3: Update active outer faces

Traversal starts in the outer face list of the root grid and the active outer face list is cleared. The same strategy as for Step 2(b) is followed, except that the faces are stored in the outer face list. While examining the boundaries, the opportunity is seized to copy the boundary groups from the root boundaries to the connecting faces, see Fig. 3.19.

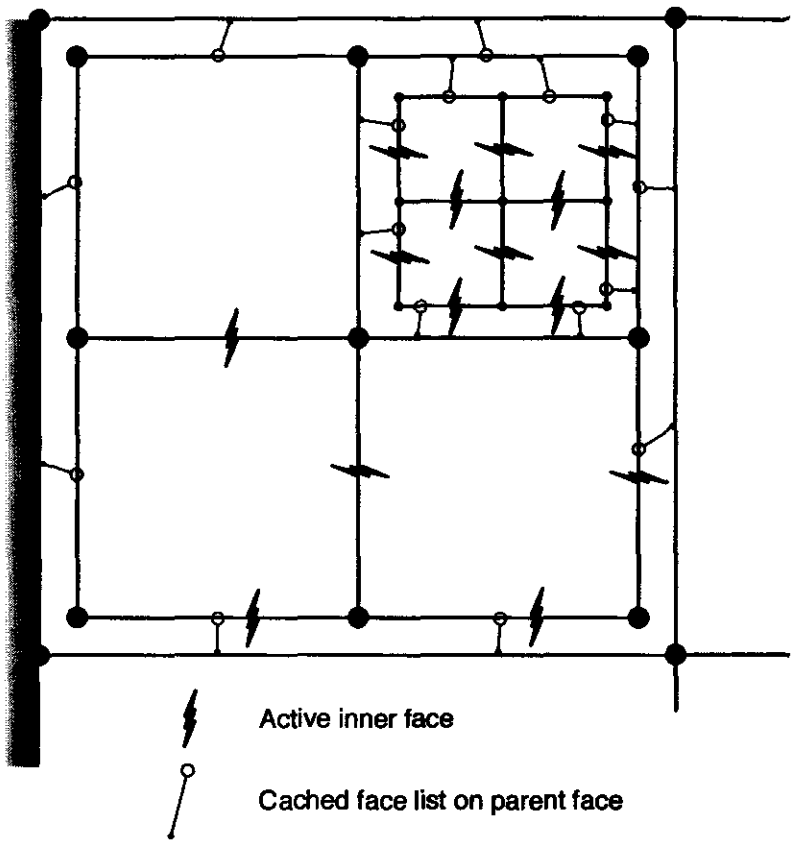


Figure 3.18: Updating the active inner face list from the cached face lists on parent cell faces.

Step 4: Regularization

After the global cell front and active face lists have been updated and new connectivity established, the relative levels of neighbouring cells are examined. This is done by traversing the active inner face list and where neighbour levels differ by more than one level, the cell with the lowest level is marked for refinement. Cells can only be marked once and just for one level of refinement per sweep. The process of modifying adjacent cell levels, as shown in Fig. 3.20 is called "regularization" and is applied to obtain smooth gradings between different mesh sizes.

When all the cells have been compared, the marked cells are refined and **Steps 1–4** are repeated. For every level of regularization applied to the grid, the cell connectivity changes, so that the whole neighbour finding algorithm is repeated once per level up to the maximum level specified for the grid.

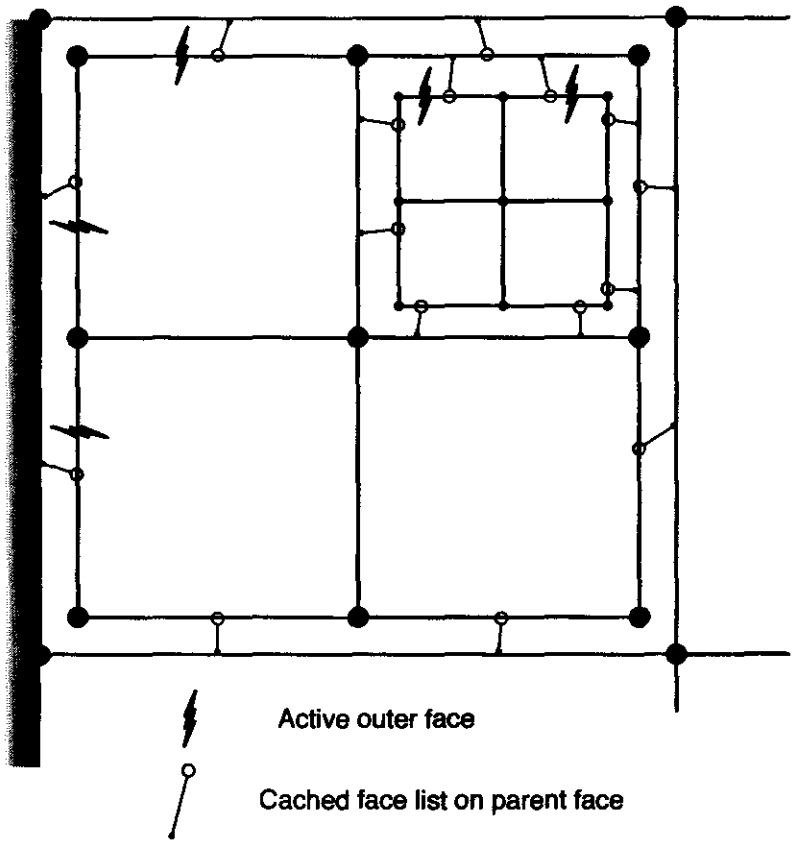


Figure 3.19: The active outer face list updated from the cached face lists on parent cell faces.

3.4 Arbitree generation algorithms

In this chapter, the concept of unstructured recursive grids was introduced as arbitrees. The process of creating the grid structure is summarized in this section and the chapter ends with some closing remarks on the practicality of the design.

Arbitree generation

This procedure is executed for all instances where refinement is required.

Step 1: Identify the cell that must be subdivided.

Step 2: Initialize the grid pointer and allocate memory for the grid object.

Step 3: Create a vertex mesh by interpolating the vertices between the parent cell's corner vertices.

Step 4: Create cells with non-overlapping volumes by filling the vertex mesh.

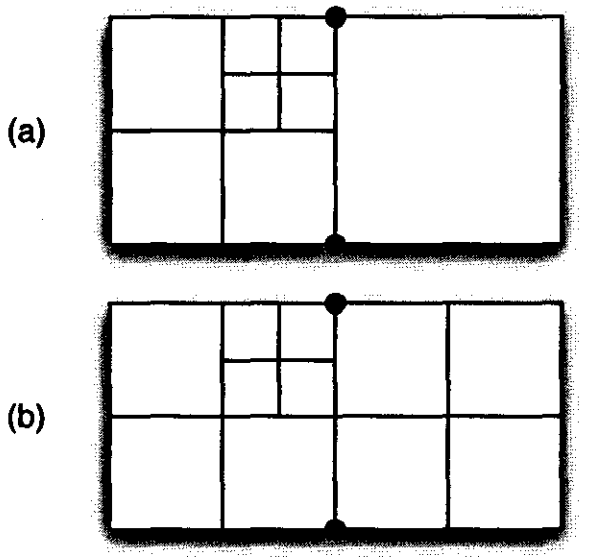


Figure 3.20: Adjusting the mesh grading through regularization. a) Before regularization, neighbouring cells differ by two levels of refinement b) After regularization, neighbours are one refinement level apart.

Step 5: Create faces and determine inner and outer faces for the grid.

Step 6: Establish connectivity between the grid's outer faces and the parent cell and store the matching face pointers in cache lists.

Traversal and neighbour finding

This procedure is executed to update global connectivity. It must be applied whenever there is a change in the mesh composition that influences connectivity.

Step 1: Determine the global cell front list and refined cells list.

Step 2: Update the active inner face list.

Step 3: Update the active outer face list.

Step 4: Regularize the grid to smoothen the mesh grading and mark additional cells that must be refined.

Step 5: Apply refinement using the procedure summarized in §3.4.

Step 6: Repeat from Step 1, until no more cells are marked in Step 4.

3.5 Closure

The concept of arbitrees presented in this chapter enables unstructured cells to be refined using a hierarchical approach. The advantages are that cells may be subdivided in levels of increasing fineness and be unrefined back to the base grid level where the researcher controls the maximum level required for sufficient spatial accuracy. Arbitrees do have a more complicated internal structure than quadtrees, but this is necessitated by the increased complexity of unstructured compared to structured meshes. The use of geometric properties to determine neighbours is potentially slower than an index-based search algorithm, but with the implementation of various cached lists, performance tests have shown the negative speed impact to be satisfactorily reduced. The following chapter describes adaptive refinement applied to two-phase flow models and how the complete simulation cycle is assembled.

Chapter 4

Adaptive grid generation for two-fluid systems

4.1 Introduction

The previous chapters discussed the mathematical formulations used for two-fluid systems and the procedures used to refine selected cells in a grid. The purpose of this chapter is to present a methodology for combining both concepts, so that two-fluid systems can be modelled on dynamically adapting grids.

Adaptive refinement brings with it some extra considerations. Previously, when the grid generation procedures were developed it was assumed to be known which cells are to be refined. That process of selection is now addressed. In the first two sections, the criteria are described that determine if a cell should be refined for added accuracy, or coarsened to reduce resource requirements. The steps needed to transfer the solution from a grid at the old time step to the new, adapted grid at the next time step are discussed, and the chapter closes with the combined algorithms for adaptive two-fluid systems.

Multiple refinement criteriums may be applied to the same problem. Compliant cells are marked as needing refinement, and after all the criteria are evaluated, the cells are refined and regularized.

4.2 Refinement criteria

The refinement criterium used in this study, defines that cells are marked for refinement whenever the cell contains a part of the interface. This refinement approach was used with success by Greaves (2004) in modelling different scalar convection configurations for two-phase flows.

The criterium is based on the interface definition for the volume fraction function in Eqn. (2.9) and plainly states that for any cell P, when:

$$0 < \alpha_P < 1 \tag{4.1}$$

the cell must be refined.

To prevent spurious cell refinements triggered by inaccuracies in the solution, the criterium is slightly modified with a numerical limiter ϵ:

$$\epsilon < \alpha_P < (1 - \epsilon) \tag{4.2}$$

where $\epsilon \approx 1e^{-2}$ to prevent unneeded refinement.

Other criteria could be used in conjunction with the interface definition, such as error estimations (Jasak & Gosman, 2000a) or curvature and reconstruction correctness (Malik, 2004). In this study, an examination of the interfacial front calculated in test cases, revealed that the highest levels of activity for other variables also occurred along the moving front. Implicitly, by tracking the interfacial front, the refined region also captures high error regions for other flow properties. With the chaotic nature of the breaking interface in the test cases, a criterium based on curvature was not considered to be able to make a significant impact in reducing the number of refined cells. The reconstruction correctness is formulated for structured grids and is not known to be directly applicable to unstructured grids.

An example of the application of Eqn. (4.2) is illustrated in Fig. 4.1. During the time step δt, the interface was convected from within a front of refined cells, Fig. 4.1(a), into a previously unrefined region, shown in Fig. 4.1(b). The cells containing the new volume factions, indicate the position of the interface at time $t+\delta t$ and are marked for refinement in Fig. 4.1(c). The refinement must be applied in such a way as to conserve mass and the interface definition. A method to interpolate the values to achieve the best approximation is presented in §4.4.

4.3 Coarsening criteria

When evaluating the parent cells, the logical opposite of the refinement criterium is applied. The volume-weighted average value of ϕ for any a parent cell P (*without any refined children*) is determined by:

$$\phi_{Pavg} = \frac{1}{V_P} \sum_{c=1}^{n} \phi_c V_c \tag{4.3}$$

where V_P is the volume of the parent cell, c indicates the child cell containing fraction α_c with volume V_c and n indicates the number of children cells. The scalar property ϕ represents α and other flow properties.

The average value for α of the parent cell is then evaluated against the coarsening criterium:

$$\alpha_{Pavg} \leq \epsilon \text{ or } \alpha_{Pavg} \geq (1 - \epsilon) \tag{4.4}$$

An example of the coarsening test can be seen in Fig. 4.1(d) and the new grid after all allowable cells were unrefined in Fig. 4.1(e). During the time step δt the interface is convected further away from a previously refined region. When the

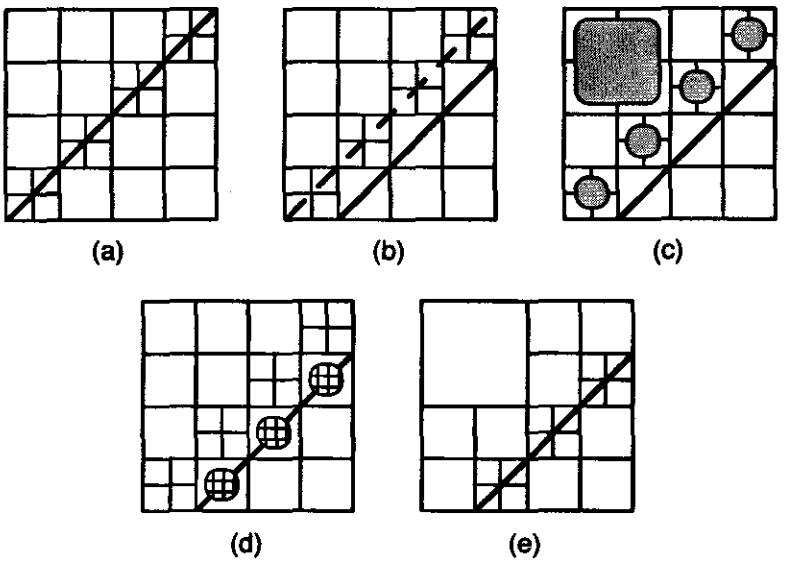


Figure 4.1: Steps for dynamic grid adaptation. a) Inter-facial front at old time step t, and b) at the new time step $t + \delta t$. c) Cells marked for refinement. d) Cells marked for possible unrefinement. e) Adapted grid at $t + \delta t$.

parent cells are evaluated, the interface at the time step at $t + \delta t$ is far enough removed so that some of the cells do not contain enough of the volume fraction anymore. These cells are marked as being allowed to unrefine.

When a parent cell is coarsened, the only additional step is to delete the recursive grid contained by the cell. As part of the coarsening algorithm, the average cell value for the parent was calculated and this calculation need not be repeated.

4.4 Transfer of solution

When a cell containing part of the interface is refined, the value of the parent cell can be transferred to the children cells using one of two methods: the first is to inject the parent cell value as an average value into the children cells and the second is to use a method of gradient-based extrapolation.

The first method is self-explanatory, only the extrapolation method will be described in more detail in this section. The value contained in the parent cell is extrapolated to obtain the distribution of values in the children cells. The extrapolation as illustrated in Fig. 4.2 is done by using the gradient of the field in the parent cell (Ferziger & Peric, 2002; Jasak & Gosman, 2000b):

$$\phi_c = \phi_P + \nabla\phi \cdot (\vec{r}_c - \vec{r}_P) \tag{4.5}$$

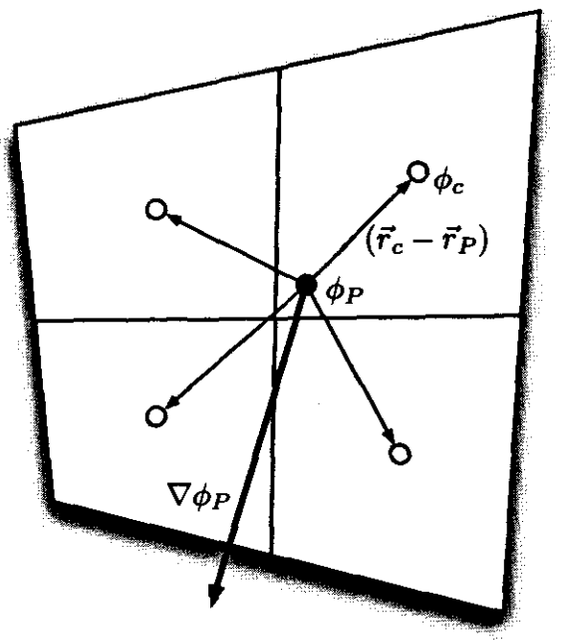


Figure 4.2: Transfer of variables from coarse grid to refined grid.

where the vector distance $\vec{r}_c - \vec{r}_P$, is the distance between the parent cell centre and the centre of the child cell.

In regions where a sharp gradient in a property is present, the extrapolation described in Eqn. (4.5) may lead to unbounded values. This problem is similar to the unboundedness experienced when upwind values are estimated using gradient calculations in the CICSAM scheme. The solution is to bind the extrapolated child cell values to obtain ϕ_c^*, but rather with the local minimum and maximum values and not the global limits as used in Eqn. (2.81). If the global limits are used, it may lead to unrealistic interpolated values that locally distort the interface.

The bounded values may not have volume-weighted conservation of cell properties anymore and a correction factor is calculated by comparing the quantity of ϕ in the parent with the total amount contained in the children:

$$\zeta = \frac{\phi_P V_P}{\sum_{c=1}^{n} \phi_c^* V_c} \tag{4.6}$$

The correction factor ζ is used to modify the interpolated values ϕ_c^* in an iterative process until a bounded and volume-weighted value conserving distribution is obtained for the children cells.

As noted by Jasak (1996:238), the transfer of the volume flux for two-fluid simulations needs to be handled with even more care. Since not all the inner faces of the new grid have equivalent faces in the previous grid, it is not possible to transfer

face-based quantities directly. The volume flux calculated during the previous time step satisfied continuity on the previous set of inner faces. The volume flux on the new faces is guessed by using Eqn. (2.51)–Eqn. (2.53) to interpolate face velocities on the new faces. The flux distribution is then adapted by solving for the pressure correction (Eqn. (2.66)) and velocity corrections (Eqn. (2.59)) of SIMPLE. By transferring the cell-based $\frac{1}{a_P}$ part of the velocity equation to the new grid the pressure correction equation can be re-assembled. The flux imbalance resulting from the transfer of the volume fluxes to the new grid forms the source term and the pressure correction is calculated.

For the two-fluid system, the additional corrector step is applied to the velocity and pressure to obtain proper flux conservation before calculating the solution for the new time step. The location in the algorithm is indicated in the description of the simulation cycle in the next section.

4.5 Refinement algorithm

The refinement algorithm depends on the two lists (the global cell front and the global refined cells list) assembled during the creation of the arbitree structure in §3.4. These lists are used to restrict the evaluation of the refinement criteria to only those cells that might be possible candidates for either operation.

The steps to evaluate a new solution at time $t + \delta t$ and apply the refinement and coarsening techniques are summarized as follows:

Step 1: Test for coarsening Evaluate entries in the global refined cells list against the coarsening criteria in Eqn. (4.4) and mark as preliminary candidates for coarsening.

Step 2: Test for refinement Evaluate entries in the global cell front list against the refinement criteria from Eqn. (4.2) and mark for refinement.

Step 3: Apply refinement Use the refinement procedure defined in §3.4 to refine the cells marked in Step 2 and update the connectivity lists. When a cell is refined, the parent cell value is transfered to the children cells with Eqn. (4.5).

Step 4: Regularize Regularization is applied as part of the mesh generation procedure in Step 3. If a cell was marked in Step 1 for possible coarsening, the regularization step may remove the privilege if the cell must remain refined as part of the regularized front.

Step 5: Apply coarsening The final step is to process the cells that are still marked as candidates for coarsening with Eqn. (4.3).

Step 6: Final connectivity After the cells have been deleted, the final connectivity list for the grid can be compiled as described in §3.4.

The steps described in this section dynamically adapt the grid to the solution obtained at the new time step. The sequence of operations is specifically chosen to keep mesh operations to a minimum by not refining and coarsening cells unnecessarily.

An alternative sequence could be to immediately coarsen all the cells outside of the refinement front at the beginning of the sequence. This reduces the complexity of the testing procedure for the regularization step. The drawback to this approach is that a significant number cells are deleted that would possibly be required for the regularized cell front. In adaptive two-fluid simulations, the inter-facial front *by design* propagates at a speed of less than a cell volume per iteration, so that large numbers of cells in the refined and regularized front retain their refined status during the course of a few iteration cycles. Changes in refined status are gradual and occur on the fringes of the front.

4.6 Adaptive two-fluid algorithm

The algorithm for adaptive two-fluid simulations is a combination of the procedures developed in the previous chapters to generate the grid, solve the flow equations for a two-fluid model, adapt the grid and transfer the solution from the old grid to the new grid. It can be summarized as follows:

Step 1: Create root grid The physical domain of the problem is discretized in space to obtain the numerical grid. This grid is the root grid for the remainder of the simulation.

Step 2: Initialize problem variables For the first iteration, the problem is initialized with the specified flow field and volume fraction distribution. For subsequent iterations, the fields from the previous iteration form the initial values for the current iteration.

Step 3: Reconstruct volume flux The volume flux is reconstructed from the cell-based values for the velocity and by applying the correction factors from §4.4.

Step 4: Solve two-fluid equations The algorithm for the solution of a two-fluid system is executed (§2.4) to obtain the new volume fraction and flow field distributions.

Step 5: Apply refinement and transfer values The refinement procedure of §3.4 is applied to obtain the new grid. Field values are transfered using the methods described in §4.4.

Step 6: Increment time step The time step is incremented and the iteration cycle repeats from Step 2 until the final time step is reached.

4.7 Closure

The adaptive refinement of two-fluid systems was described in this chapter. The use of a suitable refinement strategy was discussed along with the methods used to transfer values from the old grid to the newly refined grid at the next time step. Finally, the combined solution algorithm was given that enables two-fluid simulations to be combined with adaptive mesh refinement. In the next chapter, the development is described of the object-oriented toolkit that was used for the simulations. The final chapter presents test cases illustrating the solution of a combined two-phase flow model.

Chapter 5

An object-oriented toolkit

5.1 Introduction

The project designed in this study is introduced as Xtree/FTK. Xtree denotes the arbitree-based data management philosophy of the grid object and FTK is an acronym for *Flow Toolkit*, the object-oriented framework that handles the numerical side. The project composition is introduced first, with the discussion of the components and an example of their use presented later in the chapter.

The synergy between the different components of the project can be described by referring to Fig. 5.1. The relationship between components can be understood as one of the components interacting directly with each other through their individual interfaces, but still under control of the main project application that controls the simulation process. More informative design layouts of the objects are given in the following sections, where the data stored by the objects, their internal functionality and interfaces to other objects are explained.

A well established *design pattern* in object-oriented software design is the modular *model-view-controller*(MVC) paradigm, formalized in Gamma *et al.* (1995). The diagram in Fig. 5.2 illustrates the concept, which is based around a model (the object that does the work *and* contains the data), different views of the model data (typically the *graphical user interface* (GUI) of the application) and the controller object that links the model and views. The MVC concept is however not limited to model and GUI designs: it can also be implemented at lower levels in the code. Applied in this case, it strives to separate the software instructions for a problem (the numerical "model") as much as possible from its high-level mathematical appearance to the user (the "view" of the problem physics), to improve code development, code re-use and maintenance.

The layout of Xtree/FTK depicted in Fig. 5.1 contains five main objects. Each object is in a certain MVC relationship with the other objects and so have certain responsibilities:

FTK The framework responsible for mathematical objects and operators, low-level

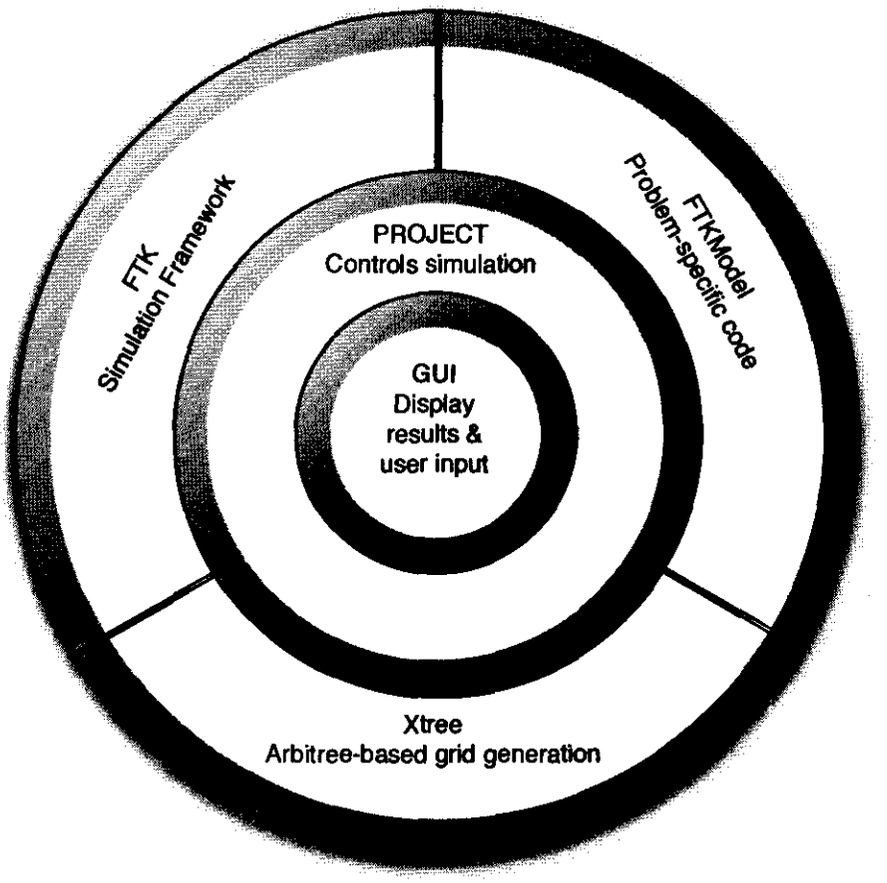


Figure 5.1: The project composition for Xtree/FTK and application code.

data handling and iterative solution of equations.

Xtree The grid object that handles geometric operations and stores grid data. Grid refinement and transfer of variables are automatically handled on request from the other components and new grid information is communicated through the interface objects.

FTKModel The problem-specific solution algorithm is encapsulated in a component. The operators provided by FTK is assembled into equations and the correct sequence of operations is encoded. The assembled problem is then passed back to FTK for solution.

Project The controller component manages the data flow between FTK, Xtree and FTKModel and controls the user interface.

GUI Transfers input from the user to the controller and displays data in the correct format determined by the controller. The GUI does not store any data from

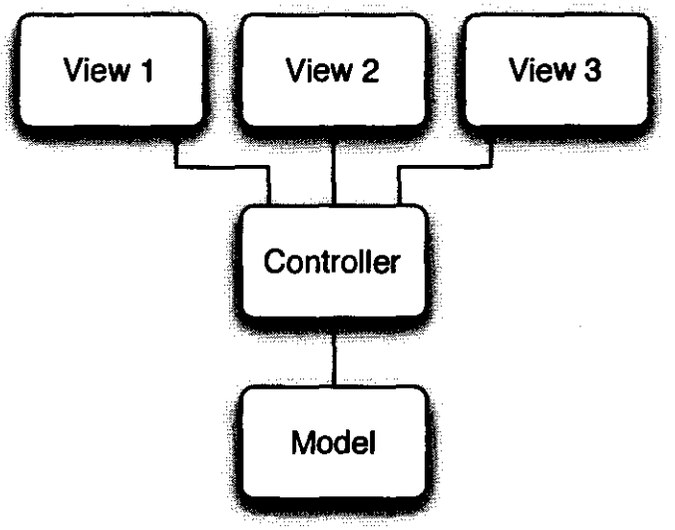


Figure 5.2: The model-view-controller design pattern.

the model; different views (or representations) of the model data can thus be displayed in the GUI without affecting the original data.

5.2 The generic finite-volume solver

Multi-layered design

The software described in this chapter was developed with a face-based, multi-layered design, as illustrated in Figure 5.3 in the layout diagram. The first layer contains the model description, the second layer is the mathematical operators used to assemble the model, the third level is the container objects representing the grid geometry, cell/volume and surface data and the fourth level contains the basic data storage elements such as vectors. Depending on the user's research subject, work is usually done in the first layer on the numerical algorithms and in the second layer on new operators or interpolation schemes. Development in the third and fourth layers is more programmatic in nature and not usually needed by researchers.

The object-oriented design features of operator overloading[1] and inheritance enables standard operators to be extended beyond the built-in C++ variable types to

[1]In any programming language there are certain built-in data types, for instance integers (or ints) and double precision numbers (or doubles). The standard versions of the mathematical operators are written in such as way as to support the built-in datatypes. Through operator overloading the developer can supply an additional definition for an operator so that it accepts user-defined types too. When the standard addition operator "+" is used with built-in types: A=B+C where A, B, C are i.e. integers; operator overloading would allow A=B+C to instead be used between vector fields defined by the user.

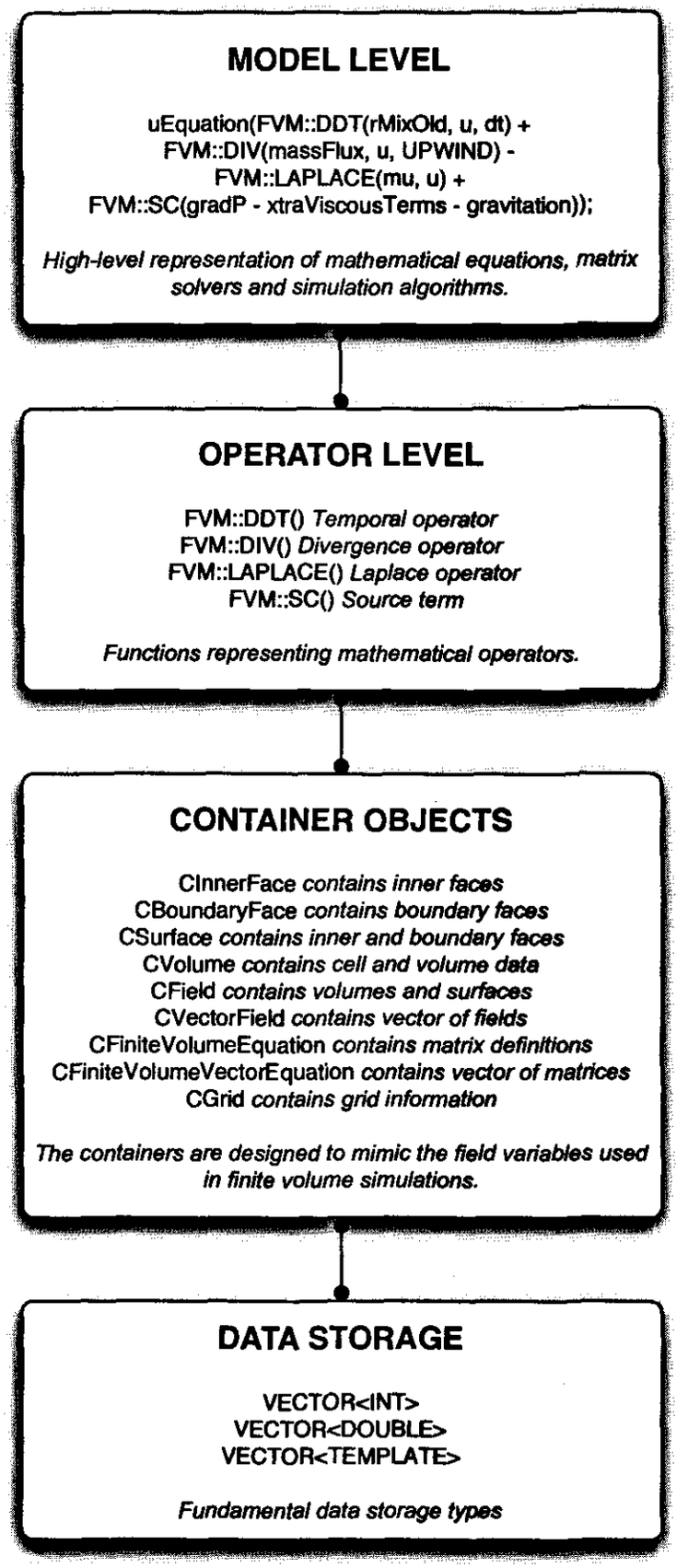


Figure 5.3: The different layers of the object-oriented simulation software.

accept user data types. This enables a set of user data types to be defined (shown in Fig. 5.3) that mimic the field variables encountered in fluid dynamics and a set of operators that accept these fields as target variables. The operators used to write the code for the simulation objects have a quasi-mathematical appearance, which greatly simplifies the development of a simulation model. Pioneering work was done by Weller *et al.* (1998) in developing a toolkit based on the finite volume method called FOAM (Field Object and Manipulation). FOAM introduced the numerous advantages of object-oriented techniques that enable the efficient *abstraction* of the simulation physics away from its numerical implementation.

Programming with equations

The FTK toolkit co-developed in this study employs similar techniques as those used by Weller *et al.* (1998) to isolate physics modelling and numerical detail from each other.

The complete toolkit contains several operators and field manipulation functions. By way of example a single equation is treated in this section and the description follows the multi-layered design of the toolkit during the discussion of the components.

To illustrate the code abstraction, the standard, steady state Navier-Stokes momentum equation is shown, firstly written in tensor notation and then as code implementation.

The momentum equation from Eqn. (2.44) is repeated as:

$$\frac{(\rho\vec{U})_P^{t+\delta t}}{\delta t} + \frac{1}{V_P}\sum_{f=1}^{n}\rho_f F_f \vec{U}_f^{t+\delta t} - \frac{1}{V_P}\sum_{f=1}^{n}\mu_f \vec{D}_f \cdot (\nabla \otimes \vec{U})_f^{t+\delta t} = \vec{S}_{\vec{U}_P} - (\nabla P)_P$$

with the source term ($\vec{S}_{\vec{U}_P}$) defined by:

$$\vec{S}_{\vec{U}_P} = \frac{(\rho\vec{U})_P^t}{\delta t} + \vec{g}\rho_P + (\nabla \otimes \vec{U})_P^t \cdot (\nabla\mu)_P + \frac{1}{V_P}\sum_{f=1}^{n}\mu_f \vec{k}_f \cdot (\nabla \otimes \vec{U})_f^t$$

Model level

By using the toolkit operators, the equation can be cast into problem-specific code:

```
CFiniteVolumeVectorEquation uEquation
(FVM::DDT(rMixOld, u, dt) +
 FVM::DIV(massFlux, u, UPWIND) -
 FVM::LAPLACE(muMix, u) +
 FVM::SC(gradP - xtraViscousTerms - gravitation);
```

The cross-derivative contributions of the viscous terms to the velocity source are calculated by a separate function and added to the source before solving the system.

The operators require different properties defined from a selection of scalar and vector fields as arguments. The container objects transparently manage the construction of coefficient matrices and have direct access to, and control over, the numerical solvers alleviating the tedium of these tasks for the user. In the next section, operators are described first and then an overview of the properties is given.

Operator level

Operator function design and implementation closely follows the derivation of the face-based equation system presented in Chapter 2. The operator functionality is directly derived from the face-based Gauss formulations given by Eqn. (2.24), Eqn. (2.26) and Eqn. (2.27). The implicit orthogonal coefficients are calculated and stored in the coefficient matrix as required for the solution of the equations. In the process, the non-orthogonal terms are explicitly added to the equation source. Key features of the operators are discussed by describing the components of the momentum equation:

`DDT` The time derivative operator takes as arguments the density, velocity (the dependent variable in this case) and current time step. The previous time step derivative is explicitly added to the source term of the equation.

`DIV` The divergence operator requires the surface-stored conservative mass flux, the dependent variable and the specified choice of convective scheme.

`LAPLACE` The laplace operator is designed to provide a more specialized version of $\nabla(\nabla)$ and applies a more optimal discretization scheme than would be used when calling a nested `GRAD(GRAD(muMix, u))`.

`SC` The source term operator requires volume-stored fields that are added to the equation source term.

The symbols represent various quantities and flow variable fields, which are described in more detail:

`rMix` The volume-stored scalar field of density. Since this is a two-phase flow momentum equation, the mixture density (Eqn. (2.10)) must be used.

`u` Volume-stored velocity field containing three vector components.

`dt` The current time step. Time steps may be defined as fixed or adjustable. In the case of an adjustable time step, the maximum allowable Courant number is used to adjust the time step.

`massFlux` The surface-stored conservative mass flux calculated with a suitable pressure-velocity coupling algorithm (§2.4) on the faces during the previous time step.

`muMix` The volume-stored scalar field of viscosity. The mixture viscosity is used, calculated with Eqn. (2.11).

`gradP` The volume-stored vector field describing the pressure gradient.

`gravitation` The volume-stored vector field containing the gravity source term.

The *volume-stored* and *cell-stored* types of containers are customized for the different grid locations of computational points and are described in the next section.

Container storage

Values of field variables are required as computational points on different geometrical locations in a grid. Suitable container definitions store the values in data structures with logical abstractions that aid code development. These container abstractions are analogous to the components of the grid structure described in Chapter 3, §3.3. The different containers are:

Inner faces The container for values required on inner faces in the grid. Values in this structure are obtained from face objects contained in the active inner faces list.

Boundary faces Values required on the boundary faces are stored in this container. The structure is populated with values from the active outer faces list.

Surfaces The container constructed through the composition of inner and boundary faces. The typical use is to store the flux values for a domain.

Volumes This container stores volume-based values: those values that are defined on the cell centre locations and represent volume-averaged values for the cells. The corresponding grid object is the global cell front list.

Fields The field containers are composed of surfaces and volumes and store a complete representation of values for all three components (cells, inner and outer faces) of a grid object.

Vector fields By combining a field container for every vector direction, the vector fields offer a convenient storage solution for the vectorized components of a variable.

Equation fields Equation fields store the coefficients and source terms for the numerical solver and have access to the solver objects and matrix elements.

Vector equation fields The equation fields can be vectorized in a similar manner as done with the construction of vector fields and represent the matrix definitions for a system containing vector fields.

Grid The grid object contains and manages all the grid-related information.

Data storage

All the container types derive from the basic data storage types. The basic structure of data storage is the *vector template.* The template nature of the vector allows it to store an array of any user defined data type, ranging from integers and doubles to other types such as face objects. All the container objects from the previous section contain a vector representation of the data under its care and the size of the vector is determined by the size of the specific list.

Coding an adaptive two-fluid simulation

The operators and data types described previously can be combined to assemble an algorithm for the simulation of a two-fluid system. The simulation procedure consists of four main steps: 1) the generation of the root grid, 2) problem definition and initialization of the variable fields, 3) execution of the main simulation routine and 4) visualization and further post-processing of results. In this section, the focus falls on the second and third steps. For brevity, a condensed version of the C++ code for the algorithm is presented, with *emphasized* user comments where applicable.

Initialization

The field variables used in the simulation are defined and initialized with appropriate values. When a specific shape is used as the initial state for the volume fraction (e.g. a circle or square) the shape is defined by marking those cells as 100% filled whose centres fall inside the geometric boundaries of the shape. To obtain an improved initial resolution, the grid is refined based on the gradient of the volume fraction, and the cell centres are tested again for compliance with the original shape. This process is repeated once for every level of refinement allowed in the grid, so that the original shape of the interface is captured at the highest possible resolution of the grid.

The most important steps of the initialization procedure are given by:

```
(iterator->iterCurrent != iterator->iterInit)
{
  u.V1.Field.Initialize(0.0);
  u.V2.Field.Initialize(0.0);
  u.V3.Field.Initialize(0.0);
  p.Field.Initialize(0.0);
  pp.Field.Initialize(0.0);
```

The `iterator` *object controls the iteration process and keeps count of the iterations and elapsed time of the simulation. The velocity* `u`*, pressure* `p` *and pressure correction fields* `pp` *are initialized to zero. If an initial velocity distribution is required, it may be prescribed in this step, instead of the zero velocity of this example.*

```
a.Field.Initialize(0.0);
InitiateA(a, DAMBREAK);
gradA = FVC::Grad(a.Field);
```

The volume fraction `a` *is initialized with the required shape determined by the specifier, in this case* `DAMBREAK` *for the problem of a collapsing water column. The gradient of the volume fraction is calculated and stored in* `gradA` *to be used in the initial refinement iterations of the grid.*

```
#include "density_interpolation.inc"
rFace = CFaceMethod::Interpolate(rMix, distWeighting);
uFace = CFaceMethod::Interpolate(u, distWeighting);
massFlux = FVC::Dot(rFace*uFace, faceAreaVector);
volumeFlux = massFlux/rFace;
```

The volume fraction distribution is used to determine the density `rMix` *of the mixture. The mixture density and velocity are interpolated to the faces to obtain* `rFace` *and* `uFace`. *The product of the face values for density and velocity with the face area vector* `faceAreaVector` *determines the mass and volume fluxes for the system,* `massFlux` *and* `volumeFlux`.

```
  pGrid->ApplyRefinement(gradA, maxLevel, cellFill);
}
```

The final step in the initialization procedure is to refine the grid based on the gradient of the volume fraction field, `gradA`. *The maximum allowable level of refinement* `maxLevel` *and cell distribution for refined cells* `cellFill` *determine the outcome of the refinement step.*

Two-fluid algorithm

The algorithm for a two-fluid system was discussed in §2.4. This section presents a condensed version of the algorithm with *emphasized* user comments in the form of C++ code, as it would be implemented as part of the Xtree/FTK framework introduced in §5.1. The solver method employed for the volume fraction, momentum and pressure equations in the main algorithm is the well-known *Bi-Conjugate-Gradient-Method* (BiCGM) for sparse matrix systems.

```
while(iterator->iterCurrent != iterator->iterLast)
{
  #include "simple_reconstruct.inc"
```

On adaptive grids with fluid solutions, the first step is to execute the reconstruction of the pressure and velocity fields to regain the conservative volume flux distribution of the previous time step.

```
iterator++;
dt = iterator->ReturnTimeStep(rMix, volumeFlux);
iterator->CalcCourantField(rOne, volumeFlux, &courantFieldNew);

u.FieldOld = u.Field;
a.FieldOld = a.Field;
rMixOld = rMix;
```

The iterator is advanced by one iteration and the new time step `dt` *for the next cycle is calculated. The time step calculation is based on the maximum Courant number allowed for the problem and the current density and volume flux distributions. With the new time step, the Courant field* `courantFieldNew` *for the current iteration can be calculated. The field values of the previous time step are stored in the* `FieldOld` *containers of the velocity, volume fraction and mixture density.*

```
CFiniteVolumeEquation aEquation
(FVM::DDT(rOne, a, dt) +
 FVM::DIV(volumeFlux, a, courantFieldNew, CICSAM_PRED, &Betaf));
aEquation.Solve();

#include "CICSAM_Corrector.inc"
```

The equation for the volume fraction (Eqn. (2.75)) is assembled using the conservative volume flux from the previous iteration `volumeFlux` *and the current Courantfield* `courantFieldNew`*. The convection scheme for the first sweep is selected as the predictor step of the CICSAM scheme (§2.3) by defining* `CICSAM_PRED` *and the new face values for the volume fraction are returned in the* `Betaf` *container. When the equation has been assembled, the BiCGM solver is triggered by calling* `aEquation.Solve()` *to obtain the matrix solution. The new values for the volume fraction are tested for boundedness and if necessary, modified in the corrector step before progressing to property interpolation and subsequent solution of the momentum equation.*

```
#include "density_interpolation.inc"
massFlux = rFace*volumeFlux;
```

The density and viscosity is interpolated, based on the values of the new volume fraction, to obtain the face properties. The product of the face density and volumetric flux gives the mass flux for the calculation of the momentum equation.

```
while(iterator->iterSweep != iterator->iterLastSweep)
{
  gradP = FVC::Grad(p.Field);
  #include "xtraViscousTerms.inc"
  real gravY = -9.81;
  gravitation.V2 = rMix.Volumes*gravY*cellVolumes;
```

The explicit source terms for the momentum equation are obtained in this step. The gradient of the pressure field `gradP` *is calculated from the pressure solution* `p` *at the previous time step. The viscous stress terms* `xtraViscousTerms` *and gravity source term* `gravitation` *are calculated and prepared as volumetric source terms.*

```
CFiniteVolumeVectorEquation uEquation
(FVM::DDT(rMixOld, u, dt) +
 FVM::DIV(massFlux, u, courantFieldNew, UPWIND) -
 FVM::LAPLACE(muMix, u) +
 FVM::SC(gradP.VolumeVector - xtraViscousTerms - gravitation));
uEquation.Solve();
```

The assembly of the momentum equation has been discussed in a previous section. Suffice to say here, is that this is where the equation fits into the complete solution algorithm. The solution of the equation is obtained by calling `uEquation.Solve()` *which solves the momentum system for every vector coordinate in turn and assigns the new field values to the dependent variable, in this case the vector form of* `u`.

```
#include "simple.inc"
massFlux = rFace*volumeFlux;

p.UpdateBoundaries();
u.UpdateBoundaries();

#include "PrintResiduals.inc"
} // return on sweeps
```

The pressure-velocity coupling is done with the SIMPLE algorithm described in §2.3. The conservative volume flux calculated with SIMPLE is combined with the interpolated density to obtain the conservative mass flux. The pressure and velocity boundary values are updated and the residual levels for the iteration is output to the active console for inspection by the user.

```
  pGrid->ApplyRefinement(a, maxLevel, CellFill);
} // return on iterations
```

After the volume fraction and pressure/velocity distributions have been solved, the grid is refined by evaluating the volume fraction field. The refinement step includes the transfer of values to the new grid, as described in §4.4. When the pressure and velocity are transfered to the refined grid, the volumetric and mass fluxes are not necessarily conservative. The flux conservation is restored by the reconstruction step executed when the algorithm is repeated for the next time step.

5.3 Closure

The design and usage of an object-oriented toolkit for the finite volume method was described in this chapter. The relationships between different components of the toolkit were illustrated and discussed by referring to examples from the implementation of a typical two-fluid simulation. The ease with which a complete adaptive simulation can be assembled was evident. By encapsulating common operators into function objects and creating data containers that mimic field variables, the different components of a simulation cycle can be defined using a quasi mathematical programming style. This abstraction of the physical representation of a problem offers a significant practical benefit for the researcher, while the numerical implementation of calculation routines are more maintainable from a programmer's point of view.

Results from test cases are presented in the next chapter. The test cases illustrate the use of adaptive refinement for two-fluid simulations and were implemented with the Xtree/FTK framework.

Chapter 6

Test cases

6.1 Introduction

The implementation of an adaptive two-fluid simulation methodology was described in the previous chapters. This chapter presents a selection of test cases which are used to validate the methodology. The performance of the adaptive system is evaluated on uniform and arbitrary grids for different levels of refinement, with test cases using prescribed motion and cases where the motion of the different fluids is calculated as part of the simulation process. The test cases correspond to the standard academic cases investigated by Greaves (2004) on adaptive grids and Rudman (1997) and Ubbink & Issa (1999) on stationary grids.

The first set of test cases comprises models of theoretical case studies where analytical solutions exist that can be used for comparison. These cases demonstrate the advection of basic geometric shapes under the influence of a prescribed velocity field, where the flow field remains constant and only the shapes are transported.

The second set of the test cases contains models of real-world problems. Results from the adaptive two-fluid simulations are compared with results from the experimental models. While exact measurements of the interface structure are not available, general comparisons with photographic visualization data can be made. Limited data concerning the height of the collapsing water column and speed of the interface advancement is available and used for the verification of the adaptive model.

The purpose of the test cases is to illustrate that adaptive refinement can successfully be used in conjunction with two-fluid models. The cases show that scalar advection can be done on refined grids with an accuracy in property conservation and shape preservation comparable to uniform meshes containing a much larger number of cells. The real world cases show that adaptive refinement can also be used to solve practical problems with sufficient accuracy and with a significant saving in computational resources.

The test cases presented in this chapter are the following:

Prescribed advection test cases

- Hollow, square shape in a constant, oblique flow field
- Hollow, rotated square shape in a constant, oblique flow field
- Hollow, circlular shape in a constant, oblique flow field
- Slotted circular shape in a rotating flow field
- Circular shape in a shear flow field

Two-fluid simulations

- Collapse of a liquid column with a return wave
- Collapse of a liquid column across an obstacle

In this chapter only selected results from the test cases are shown. On the accompanying CD-ROM, more explanatory animations of the various simulations can be found. The animations illustrate the evolvement of the volume fraction and the corresponding adaptive refinement of the mesh.

6.2 Prescribed advection test cases

Numerical grids

The test cases for prescribed advection use several different numerical grids. The grids are created in structured and unstructured versions for the same physical domain, but at coarser resolutions than those employed by Rudman (1997) and Ubbink & Issa (1999) for the same domain discretization. Through the use of different refinement levels for cells in the base (or genesis) grid, the final *effective* resolutions correspond to the uniform mesh resolutions used by the other researchers.

The different grids are:

- 25×25 uniformly spaced cells for $x, y \in [0, \pi]$. At two refinement levels, it is equivalent to a 100×100 uniform grid.
- 25×25 uniformly spaced cells for $x, y \in [-2, 2]$. At three refinement levels, it corresponds to a 200×200 uniform grid.
- an unstructured mesh containing 632 cells for $x, y \in [0, \pi]$ that corresponds to the 25×25 uniform grid.
- a similar unstructured mesh for the domain $x, y \in [-2, 2]$.

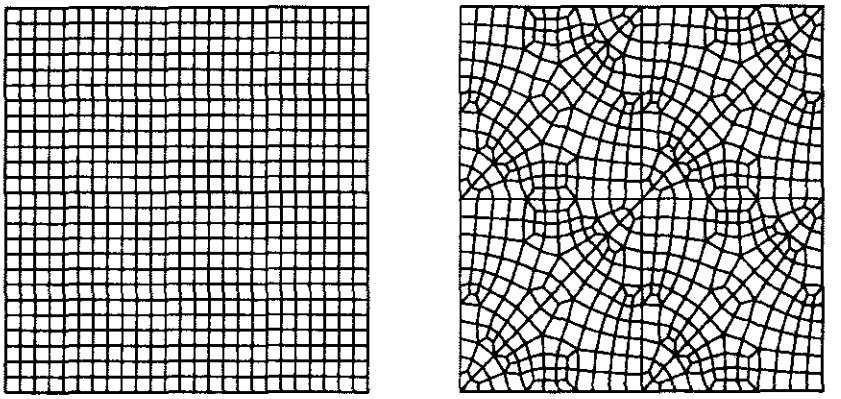


Figure 6.1: Different grids used for test cases. a) Structured grid consisting of 25×25 uniform cells. b) Unstructured grid with 632 cells.

The uniform mesh is shown in Fig. 6.1(a) and the unstructured version is illustrated in Fig. 6.1(b). The size distribution of the cells in the genesis grid also affect the relative size distribution of refined cells, as the evolving solution cause different cells with different base volumes to subdivide into refined cells sharing the original volume.

The time step in all cases is chosen to maintain a maximum mesh Courant number of 0.25. Runs on unstructured meshes are completed at the same fixed time step, but with a higher resultant Courant number on those meshes.

Field initialization

The initial field distribution for the volume fraction is calculated by determining if the location of a cell centre lies within the region bounded by the prescribed shape. If the cell lies within the region, the volume fraction in the cell is assigned the value of one (completely filled). Cells that lie outside the shape's region is deemed completely empty. The "all-or-nothing" approach to assign the initial field has an effect on the accuracy with which a shape is represented on the grid; this is reflected in the applicable error norms that reduce with an increase in mesh resolution.

To represent a geometric shape on a base grid with refined cells requires an additional initialization step over all cells for every level up to the maximum refinement level. Cells are successively refined and tested against the shape criteria until the final shape represented with the refined cells is obtained.

Error estimation

Solution errors on adaptive grids

To obtain an assessment of the accuracy of the adaptive two-fluid method, solution errors are calculated for the test cases. The solution errors from Rudman (1997) and Ubbink & Issa (1999) are repeated here for comparison purposes. Numerical methods all contain errors to some degree, depending on approximations made in the mathematical formulations. In two-fluid models, these errors usually manifest as a lack of conservation of flow properties and a non-realistic distortion of the interface. For the adaptive method, there are two contributing factors that influence the solution errors and which must be kept in mind when making comparisons between sets of errors.

The first is the manner in which the fields are initialized and how the analytical solution is applied to the grid. A shortcoming of the initialization method described in the previous section and used in this study is that cells are not assigned partial volume fractions during initialization and error calculations. This leads to a discrepancy between the errors calculated in this study and the errors presented for methods using area-mapping techniques.

The second factor is the non-orthogonality error introduced by the adaptive meshing. When a cell is refined, the direction vector between two neighbour cells changes from the orthogonal vector in Fig. 6.2(a) to the two separate non-orthogonal vectors in Fig. 6.2(b) connecting a larger cell to two smaller cells. In the CICSAM convection scheme, used by the indicator function, the orientation of the interface relative to the flow direction θ_β is used to determine the relative weighting between a compressive or more diffusive behaviour of the convective scheme. The convection scheme strives to maintain the sharpness of the interface normal to the flow direction and to preserve the gradient of the interface tangential to the flow direction (Ubbink, 1997:103).

When an interface is advected through a cell, the blending factor for a certain face is calculated based on the flow conditions and this factor determines the flux of volume fraction through the face. When the interface is advected across a refined face, the direction vectors $\vec{d}_1$ and $\vec{d}_2$ are in different directions and results in the two smaller faces having unequal blending factors ($\theta_{\beta 1} \neq \theta_{\beta 2}$) that are inconsistent with the factor the larger parent face would have. One face for example may try to compress the interface while the other face applies some diffusion; the end result is that the interface gets numerically distorted while passing through refined faces.

The combination of the two phenomena generates errors that are difficult to quantify, but are manifested in the higher error values of solutions on the adaptive grid and visible distortion of the interface not always present on uniform grids. Interestingly, the error values for the adaptive grids are still in the same order of magnitude as errors on the uniform grids, with the advantage that the solutions were obtained using much fewer cells in total.

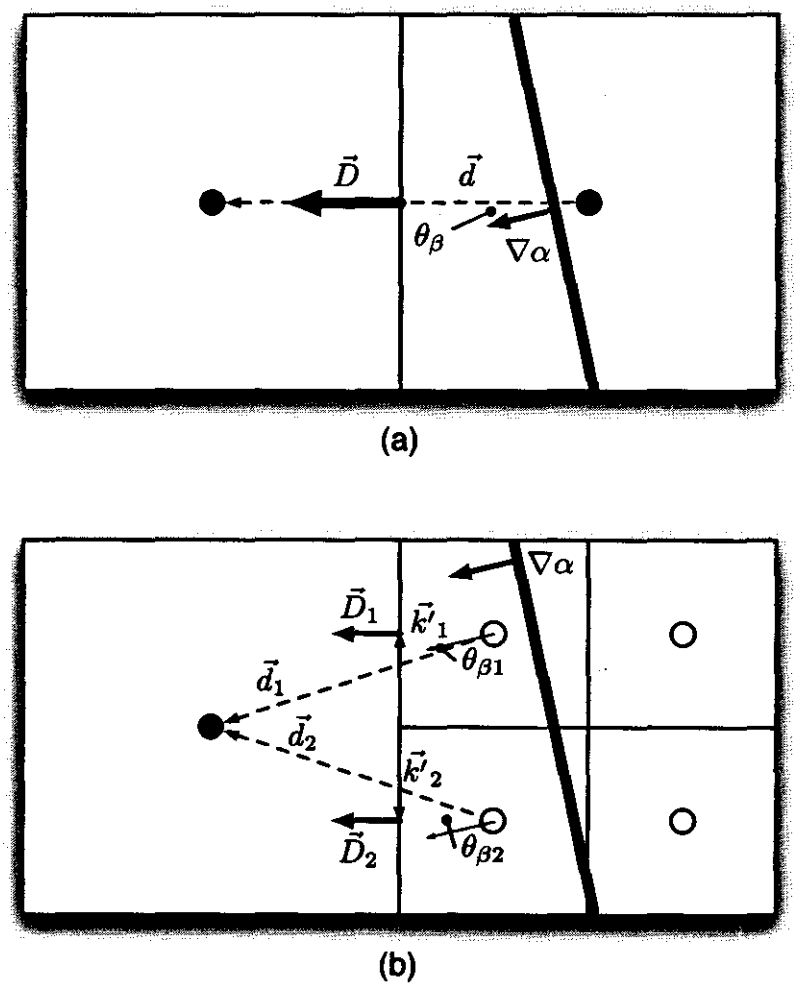


Figure 6.2: Grid non-orthogonality error. a) No error for regular cells, b) Errors are introduced after one cell is refined. The symbols: $\nabla\alpha$ is the gradient of the fraction and normal on the interface, $\vec{D}$ is the face vector, $\vec{k}$ is the error made because $\vec{d}$ that connect the cells is not intersecting the face centre and θ_β indicates the angle between the interface and flow direction.

Different solution errors

Three solution errors are calculated for the test cases with prescribed advection. The first error gives an indication of how well the method preserves the advected shape, the second error indicates how well the total amount of volume fraction is conserved during a simulation run and the third is an indication of how well the initialization method approximates the analytical form of the prescribed shape.

The solution errors E are calculated with the following equations:

Shape error:

$$E = \frac{\sum_i^{\text{all cells}} |\alpha_i^n V_i - \alpha_i^a V_i|}{\sum_i^{\text{all cells}} \alpha_i^o V_i} \tag{6.1}$$

where α^n is the solution at time step n, Vol is the cell volume, α^a is the analytical solution and α^o is the initial condition specified for the volume fraction.

Volume-fraction error:

$$E = \frac{\sum_i^{\text{all cells}} |\alpha_i^o V_i| - \sum_i^{\text{all cells}} |\alpha_i^n V_i|}{\sum_i^{\text{all cells}} \alpha_i^o V_i} \tag{6.2}$$

Approximation error:

$$E = \frac{|\alpha^a V_a| - \sum_i^{\text{all cells}} |\alpha_i^o V_i|}{|\alpha^a V_a|} \tag{6.3}$$

where $\|\alpha^a Vol_a\|$ is the amount of volume fraction contained in the analytical definition for the initial shape.

These definitions make provision for variable cell volumes, as required for calculations using the non-uniform cells in the unstructured mesh.

Convergence

The convergence of the adaptive method is compared with the convergence of errors obtained with uniform grids. The domain is specified as $x, y \in [-2, 2]$ with a rotational velocity field. The shape used as test model is a circle with radius 1.6 and located at [0, 0]. Adaptive grids are refined in such a manner that every level of refinement correspond to a specific uniform mesh resolution. The grid resolutions tested are the 10×10 base grid, the 20×20 base grid (level 1 refinement), 40×40 grid (level 2) and the 80×80 grid corresponding to a level 3 refinement. The solution errors are given in Table 6.1 together with the order of error reduction. It can be seen that there is a first order reduction in the error as the mesh resolution is increased. By keeping in mind that the interface is always represented across at least three cells in any mesh density, the first order improvement is the best obtainable with mesh refinement.

Table 6.1: Solution errors and order of convergence for the rotation of a circle on grids with multiple refinement levels compared to uniform grids.

		Solution Errors			
Mesh	**Level**	Shape	Order	Conservation	Analytical approx.
10×10	0	1.17e-01	-	7.9e-02	3.45e-02
10×10	1	5.95e-02	1.63	2.58e-04	3.45e-02
10×10	2	3.27e-02	1.57	0.0	9.64e-03
10×10	3	2.33e-02	1.33	0.0	3.42e-03
20×20	0	4.91e-02	-	0.0	3.45e-02
40×40	0	3.09e-02	1.03	0.0	9.64e-03
80×80	0	1.86e-02	1.48	0.0	3.42e-03

The purpose of adaptive refinement is to achieve comparable levels of accuracy on grids with fewer cells than uniform grids. The solution errors presented in Table 6.1 show that comparable accuracy can indeed be achieved on adaptive grids. In Table 6.2 the savings in the number of cells and relative solution times for the convergence test are shown. Although the relative savings in computational resources are determined by a large number of factors, (e.g. problem type, levels of refinement, size of refinement front, efficiency of algorithms etc.), the results for this controlled test already show the ample benefits of adaptive refinement. It can be seen that for small grids, the uniform grid outperforms the adaptive grid, due to its lower algorithm overheads (no refinement procedures) but the advantage is quickly lost and the performance gap between high-resolution uniform and adaptive grids grow with every level of refinement added.

Constant, oblique flow field

The first set of tests are based on the advection of a geometric shape by a prescribed flow field. For these tests the 25×25 structured mesh and its equivalent unstructured mesh are used on $x, y \in [-2, 2]$ with a constant, oblique flow field of $\vec{U} = (2, 1)$. The initial position of the shapes is at [-1.2, -1.2] and the final theoretical position is at [1.3, 0.05].

Three different shapes are advected across the domain:

- a hollow, square shape aligned with the main coordinate axes
- a hollow, square shape rotated at 26.57° to the main x-coordinate axis
- a hollow, circular shape

Table 6.2: The benefits of adaptive refinement demonstrated by the convergence test.

Mesh	Level	Number of cells	Normalized time
10×10	0	100	1
10×10	1	208	2.1
10×10	2	484	6.9
10×10	3	1108	11.8
20×20	0	400	2.6
40×40	0	1600	11.5
80×80	0	6400	28.8

The outer side length of the square is 0.8 and the inner length is 0.4. The outer diameter for the circle is 0.8 and the inner diameter is 0.4. The same flow conditions and geometric shapes are used on both the structured and unstructured grids.

Detail of the solutions are illustrated in Fig. 6.3 for the regular square, Fig. 6.4 for the rotated square and in Fig. 6.5 for the circle. The correlation with errors from Rudman (1997) and Ubbink & Issa (1999) is given in Table 6.3. The results obtained in this study with the structured, adapted grid are indicated by Xtree-S and the results obtained with the unstructured, adapted grid are denoted by Xtree-U.

A comparison of the errors reveal that the adaptive grids using the CICSAM scheme are not as accurate as the stationary CICSAM grids, but errors are still of the same magnitude as the other methods with the bonus of added savings in computational cost obtained by using locally refined grids. An interesting feature of the errors for the different shapes is that the errors lie close to one another, suggesting that the non-orthogonal error introduced by the refinement is the largest contributor to the error. If the error caused by inaccurate shape mapping was more significant, the regular square would have shown a much lower solution error since the shape boundaries lie on cell boundaries and require no partial mapping of cell values.

Rotating flow field

In this test, a circular shape with a slot is exposed to a rotating flow field and advected for one revolution. The same grids as for the oblique velocity test are used for the domain $x, y \in [-2, 2]$. The centre of rotation for the flow field is (0, 0) and the centre of the circle is defined at (0.0, 0.75). The diameter of the circle is 1.0, the width of the slot is 0.12 and the depth of the slot is 0.6. Time steps are fixed so that one revolution is equal to 2524 time steps, which corresponds to the time step used by Rudman (1997) and Ubbink & Issa (1999).

Table 6.3: Shape errors for shapes in a oblique flow field.

	Square	Rotated square	Circle
SLIC†	1.32e-01	1.08e-01	9.18e-02
Hirt-Nicols†	6.86e-03	1.60e-01	1.9e-01
FCT-VOF†	1.63e-08	8.15e-02	3.99e-02
Youngs-VOF†	2.58e-02	3.16e-02	2.98e-02
CICSAM-S⋆	2.50e-02	4.00e-02	4.33e-02
CICSAM-U⋆	3.97e-02	4.00e-02	2.84e-02
Xtree-S	7.88e-02	9.86e-02	8.51e-02
Xtree-U	8.21e-02	9.87e-02	8.79e-02

† Results from Rudman (1997)
⋆ Results from Ubbink & Issa (1999)

The results of the test is shown in Fig. 6.6. After one revolution, the circular shape is preserved, but definition was lost on the corners on the inside of the slot and those on the outer edge. Solution errors are presented in Table 6.4 and exhibit the same relative tendencies between different methods as the tests for oblique flow.

Shear flow field

A more rigorous test of an interface tracking scheme is shape preservation in conditions of shear flow. The specification for a shear flow field given by Rudman (1997) is used to facilitate direct comparison of errors:

$$\vec{U} = (sin(x)cos(y), -cos(x)sin(y), 0) \tag{6.4}$$

with $x, y \in [0, \pi]$. For this test the 25×25 base grid and accompanying unstructured grids are used, with two levels of refinement to correspond with the 100×100 grid used by Rudman (1997) and Ubbink & Issa (1999). The initial shape is a circle with radius 0.2π and centre $(0.5\pi, 0.2(1+\pi))$. The test shape is exposed to the flow field for a certain number of time steps, after which the flow direction is reversed. A perfect convection scheme would return the volume fraction field to its initial state before advection was started.

Two sets of results are presented. The first set continues for 1000 iterations, after which the flow is reversed for another 1000 iterations. The second set continues for 2000 iterations and then reverses the flow for the remaining 2000 iterations. Predictions for the interface shapes are shown in Fig. 6.7 for 1000 steps forwards and 1000 steps backwards and in Fig. 6.8 for 2000 steps forwards and 2000 steps

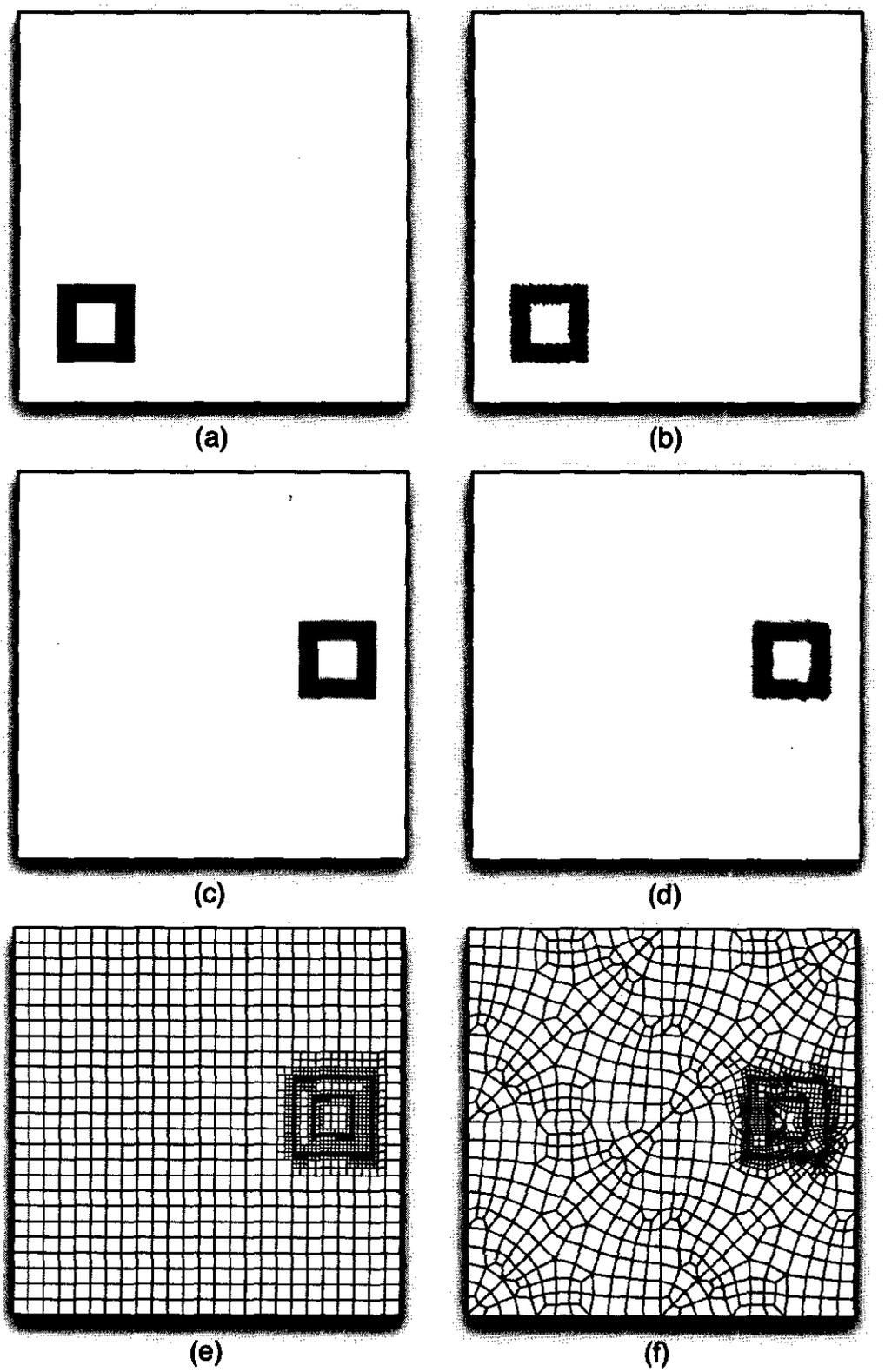


Figure 6.3: Advection of hollow square in an oblique flow field. a) Structured and b) unstructured initial shapes. c) Structured and d) unstructured final shapes. e) Structured and f) unstructured refined grids.

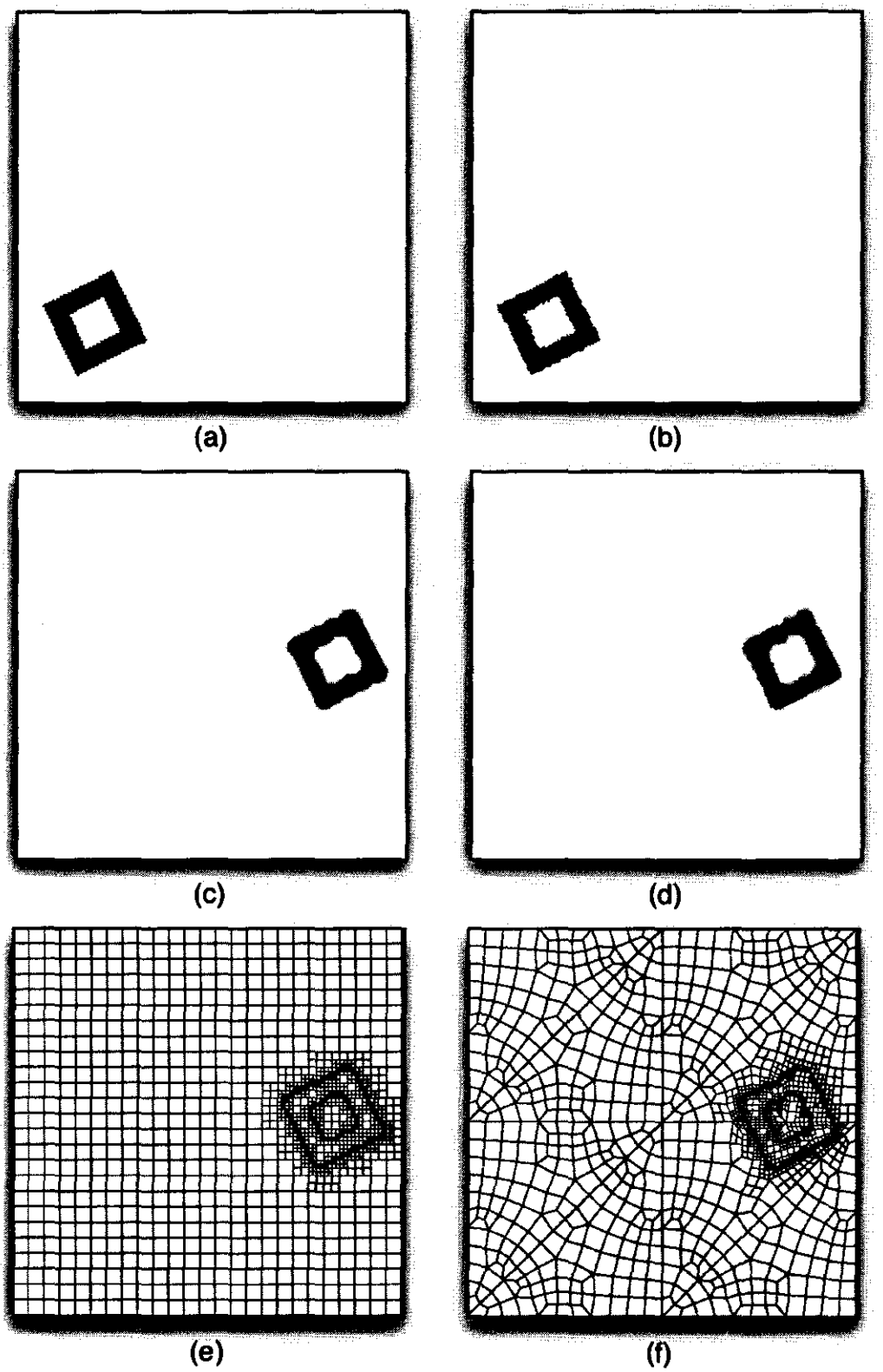


Figure 6.4: Advection of rotated, hollow square in an oblique flow field. a) Structured and b) unstructured initial shapes. c) Structured and d) unstructured final shapes. e) Structured and f) unstructured refined grids.

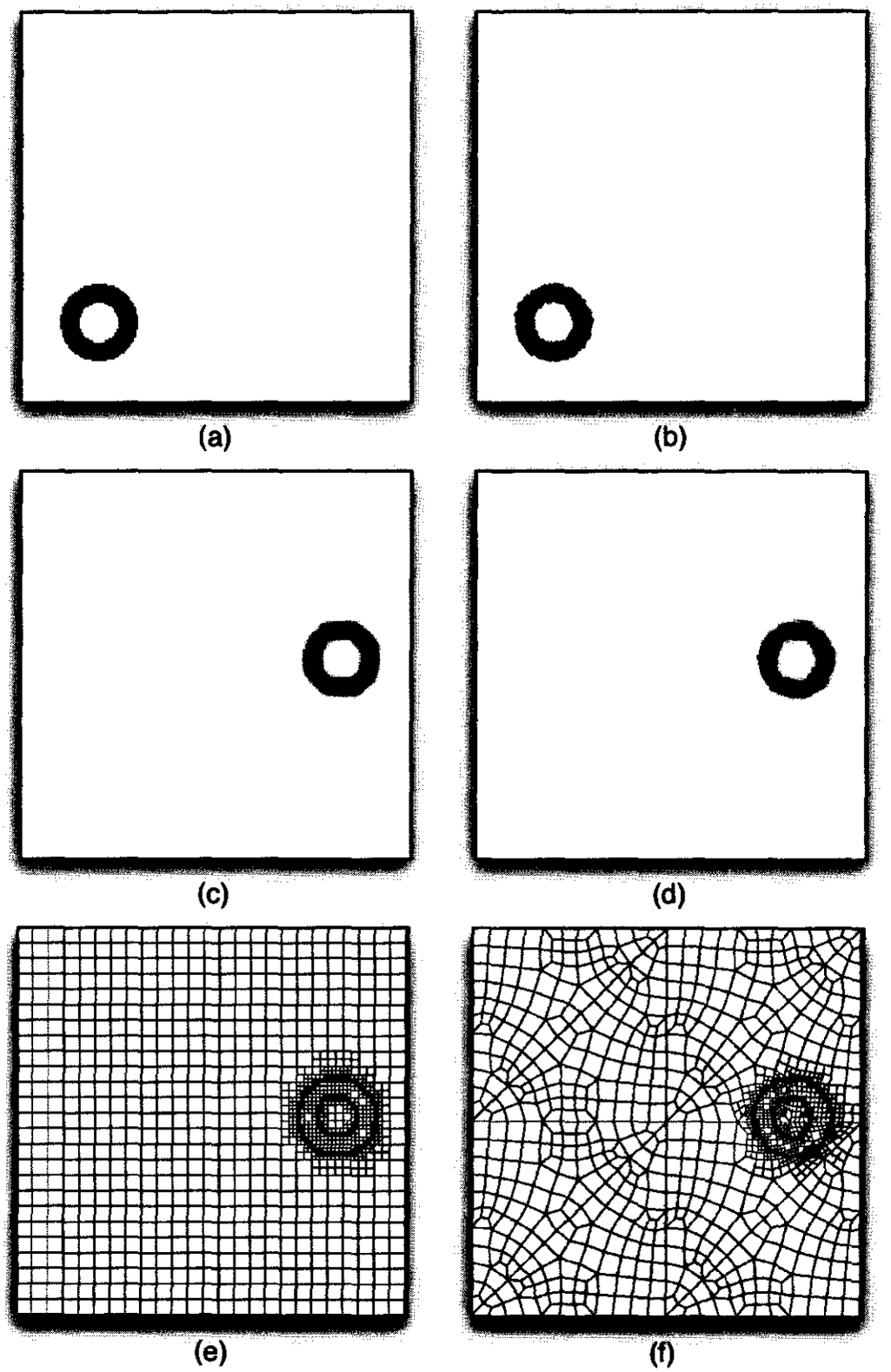


Figure 6.5: Advection of hollow circle in an oblique flow field. a) Structured and b) unstructured initial shapes. c) Structured and d) unstructured final shapes. e) Structured and f) unstructured refined grids.

Table 6.4: Shape errors for a slotted circle in rotating flow.

	Slotted circle shape
SLIC†	8.38e-02
Hirt-Nicols†	9.62e-02
FCT-VOF†	3.29e-02
Youngs-VOF†	1.09e-02
CICSAM-S⋆	1.62e-02
CICSAM-U⋆	2.02e-02
Xtree-S	3.30e-02
Xtree-U	4.73e-02

† Results from Rudman (1997)
⋆ Results from Ubbink & Issa (1999)

backwards. The relative errors for the different interface tracking schemes are tabulated in Table 6.5 and again the adaptive grids offer acceptable errors for interface recovery at the end of the specified number of iterations.

For the second set, the interface tracking scheme is tested to its limits. At $N = 2000$ iterations, the interface is stretched very thin and in places two parts of the interface go through a single cell. On unstructured grids this occurs even more readily because some cells have larger volumes than the average cell volume in the mesh. Adaptive unstructured grids can improve the situation because a larger cell is automatically refined when it contains the interface. With enough refinement levels, the occurrence of multiple interface sections in the same cell can largely be avoided. Despite these limitations in the interface tracking methodology, the shear flow results are realistic and the original shape is recovered well enough to allow comparison with the initial conditions.

6.3 Two-fluid simulations

The combined adaptive two-fluid system is validated in this section with two test cases modelling real world problems. As basis, the test cases use the classical experiment of the collapse of a liquid column in a confined container. The first case considers the unobstructed collapse of the column, while the second case is modified by placing an obstruction in the way of the collapsing wave front. The collapse of a liquid column is widely used as validation case for interface tracking flow, as shown by Martin & Moyce (1952), Hirt & Nichols (1981), Koshizuka *et al.* (1995)

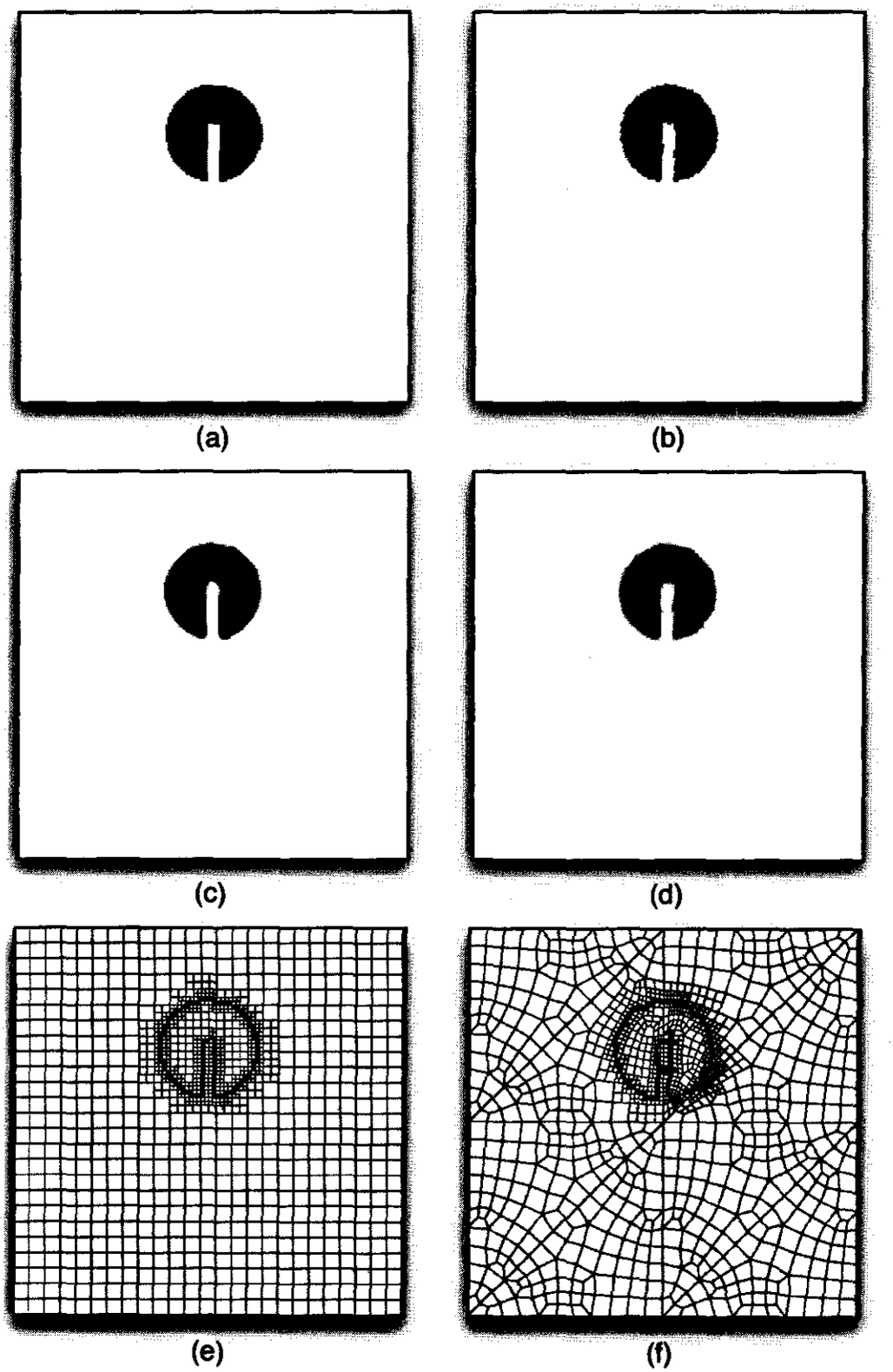


Figure 6.6: Slotted circle in rotating flow field. a) Structured and b) unstructured initial shapes. c) Structured and d) unstructured shapes after one revolution. e) Structured and f) unstructured refined grids.

Table 6.5: Shape errors for a circle in shear flow after 2000 iterations.

	Circle shape
SLIC†	9.02e-02
Hirt-Nicols†	1.09e-01
FCT-VOF†	1.44e-01
Youngs-VOF†	3.85e-02
CICSAM-S*	5.67e-02
CICSAM-U*	4.17e-02
Xtree-S	3.44e-02
Xtree-U	4.67e-02

† Results from Rudman (1997)
⋆ Results from Ubbink & Issa (1999)

and Ubbink (1997) amongst others. The studies of Martin & Moyce (1952) and Koshizuka *et al.* (1995) provide photographic records of the interface evolution and limited measurements of the initial reduction in height and speed of advancement of the wave front.

Collapse of liquid column

The collapse of an unobstructed liquid column is compared with the measurements taken by Martin & Moyce (1952) and the photographs supplied by Koshizuka *et al.* (1995). Fig. 6.9 shows the physical domain that correlates with the experimental setup used by Koshizuka *et al.* (1995). The tank is made from glass and constructed to restrict flow phenomena to two dimensions in the plane of the tank when viewed from the side. The base length of the tank is $0.584m$ and open on top with an undefined height $> 0.292m$ for the sides. The base length of the liquid column is $0.146m$ with a height of $0.292m$. The column is initially supported on the side by a vertical plate which is rapidly drawn upwards at $t = 0.0s$. The time taken to remove the plate is $\approx 0.05s$ and the initial shear effects on the column interface is absorbed so rapidly by the collapsing column that its effects may be neglected in a numerical model.

When the vertical plate is removed, the interface movement of the column is initially dominated by viscous effects. Gravity causes the column to collapse past the point where viscous and surface tension effects play a role and drives the flow to rush across the bottom of the tank. The wave front continues upwards along the

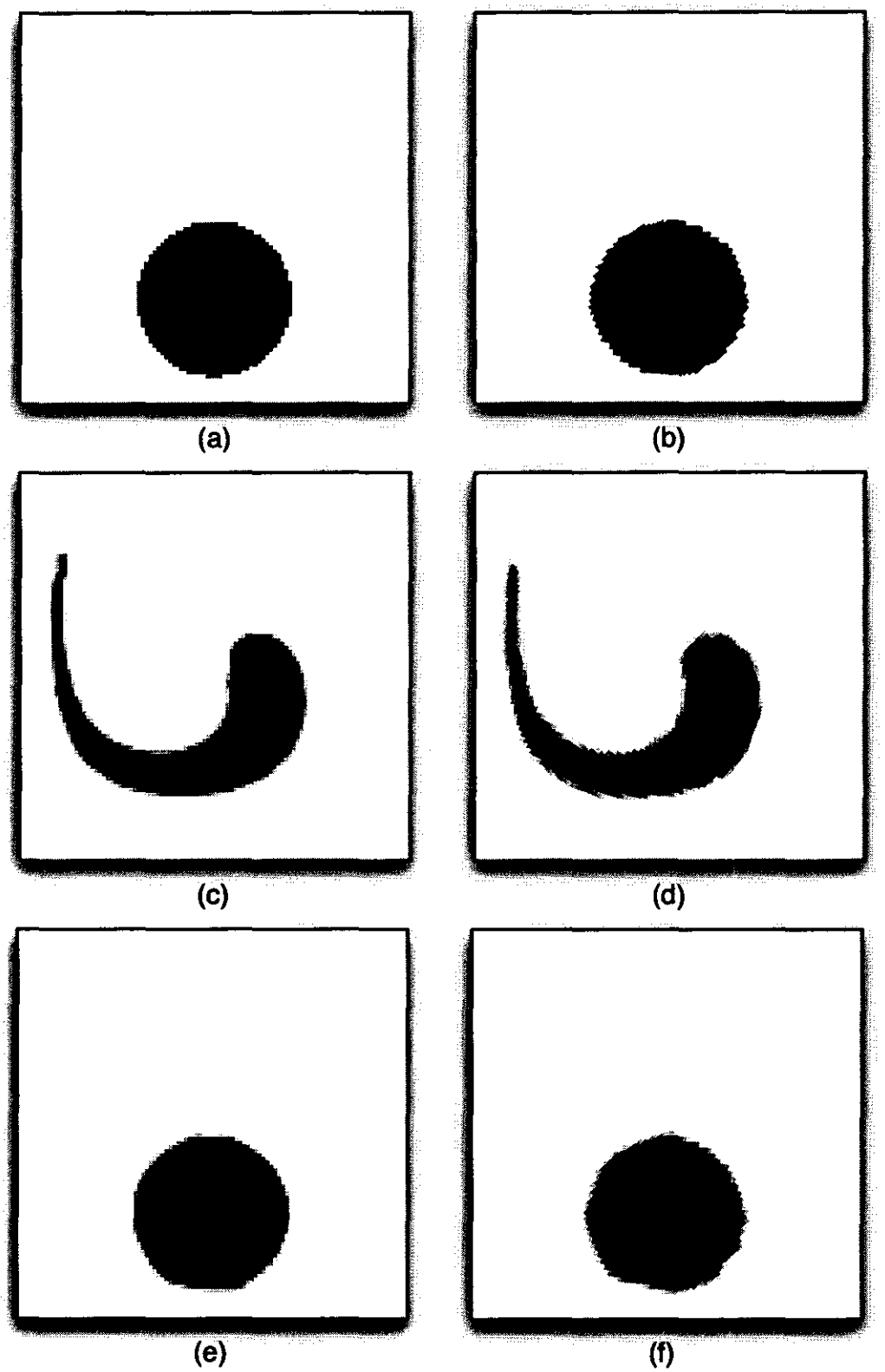


Figure 6.7: Circle in shear flow field. a) Structured and b) unstructured initial shapes. c) Structured and d) unstructured shapes after 1000 steps forwards. e) Structured and f) unstructured shapes after 1000 steps forwards and 1000 backwards.

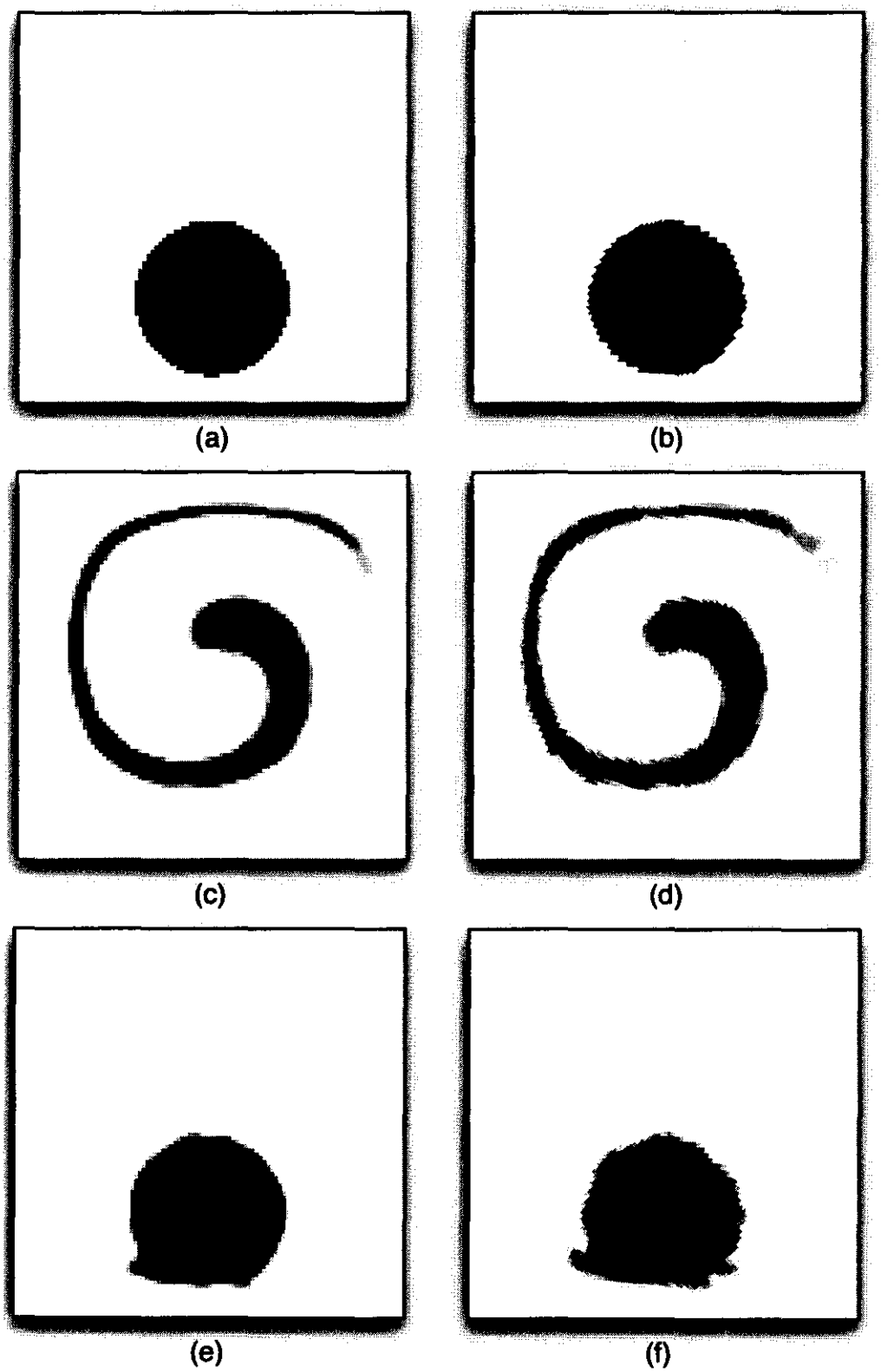


Figure 6.8: Circle in shear flow field. a) Structured and b) unstructured initial shapes. c) Structured and d) unstructured shapes after 2000 steps forwards. e) Structured and f) unstructured shapes after 2000 steps forwards and 2000 backwards.

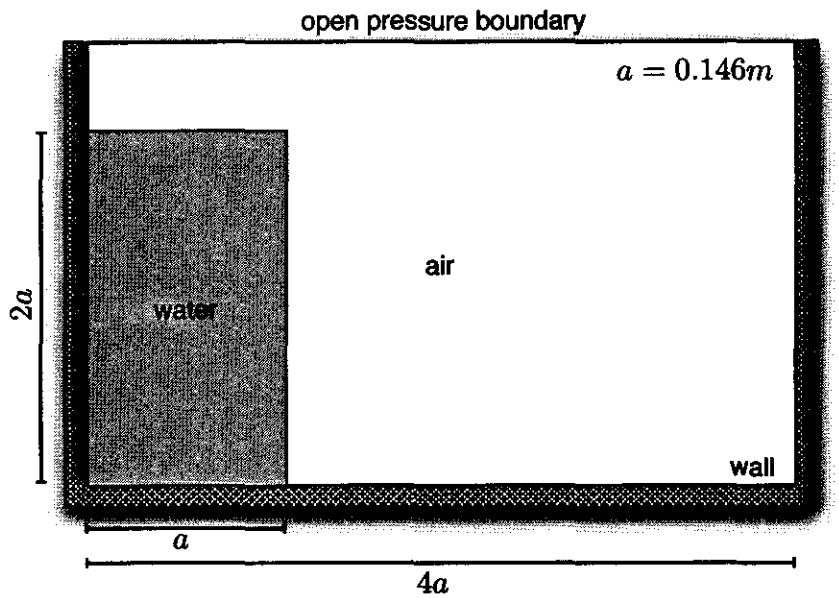


Figure 6.9: The geometry of the model for the collapsing liquid column.

opposite wall, rolls back and eventually comes to rest along the bottom of the tank with the lowest possible level of energy.

The numerical simulation models the open tank by specifying no-slip boundaries on the walls and bottom of the tank and a pressure boundary fixed at atmospheric pressure along the top. The liquid phase is water with density of $1000kg/m^3$ and the gas phase is air with density $1kg/m^3$. Water and air are allowed to leave the domain through the pressure boundary, although only air is allowed to re-enter the domain. This allows the simulation of the effect of water splashing out of the tank at $t = 0.4s$. In the simulation, only two-dimensional effects are modelled and to this end, slip conditions (i.e. symmetry planes) have been applied to the sides of the tank. Due to the physical scale and flow conditions of the problem, surface tension effects may be neglected in the numerical model.

The photographs from Koshizuka *et al.* (1995) are compared to results obtained using adaptive refinement on both structured and unstructured grids. The base grid for the structured mesh is shown in Fig. 6.10(a) and the unstructured version in Fig. 6.10(c). Results for the following time steps are presented in figures that combine the relevant photograph with the predicted volume fraction field on the structured and unstructured grids. The maximum number of cells defined by the adaptive structured grid is 1725 cells, and for the unstructured grid it is 1668. The figures are Fig. 6.11: $t = 0.0s$, $0.1s$, Fig. 6.12: $t = 0.2s$, $0.3s$ and Fig. 6.13: $t = 0.4s$, $0.5s$.

An examination of the results reveals the following:

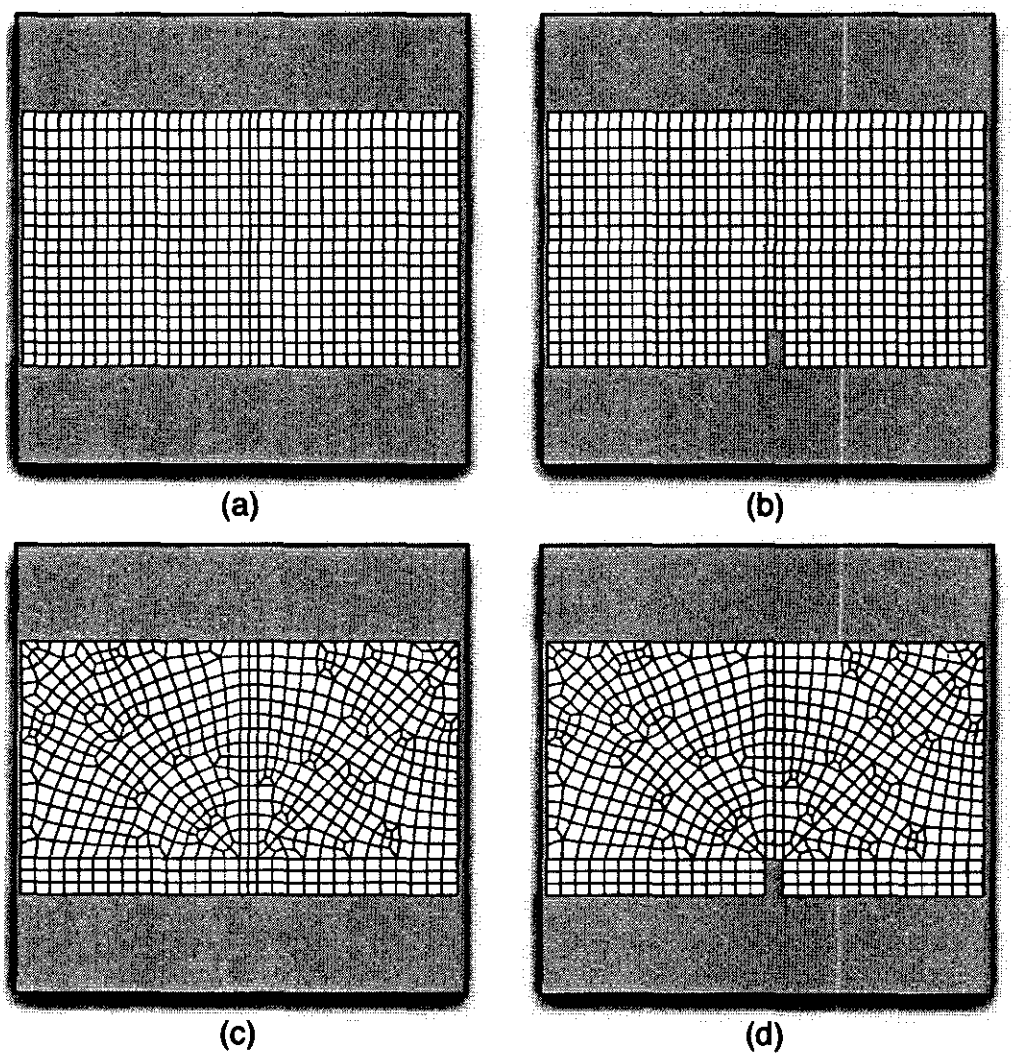


Figure 6.10: Base grids for the collapsing liquid column test case. a) Structured, 648 cells, no obstacle. b) Structured, 642 cells, with obstacle. c) Unstructured, 684 cells, no obstacle. d) Unstructured, 678 cells, with obstacle.

Time $t = 0.1s$: The rapid removal of the side wall induces shear flow effects which are visible in the photograph as slight waves in the side of the column. The removal of the side is not implemented in the current numerical model, explaining why the model only shows the foot of the column moving outwards and otherwise a smooth interface.

Time $t = 0.2s$: The shear flow effects in the interface is absorbed and approximately three-quarters of the base is covered with water.

Time $t = 0.3s$: The water hits the opposite corner and starts moving upwards. The interface is now a smooth slope from left to right before turning at the wall.

Time $t = 0.4s$: The water continues to move upwards along the opposite wall and is starting to leave the tank at the top.

Time $t = 0.5s$: The momentum of the water has reached a balance between upwards movement and gravitation downwards and the water is starting to fold down back over itself. Some droplet formation can be seen in the photographs on the face of the folding water column. These droplets are not modelled as separate entities with the current fluid model and are contained in the water column against the right-hand side, making it somewhat thicker than the experimental front.

Measurements of the change in height of the column b and of the advancement of the leading edge Z during a collapse were published by Martin & Moyce (1952). In Fig. 6.14 the non-dimensional height of the experiment is compared with results obtained on the structured and unstructured grids. The calculation of the height for both grids are the same and in good agreement with the measured height. The non-dimensional position of the leading edge is compared with the numerical predictions in Fig. 6.15. The predicted edge appears to move faster than the measured position; this can be attributed to the difficulty in measuring the exact position of the thin film preceding the actual edge and the numerical limitation on modelling thin film flow. The tendency was also reported by Koshizuka *et al.* (1995) and Ubbink (1997) that the movement of the leading edge increases in speed as the grid resolution is refined. The same difficulty of determining the exact position of the edge was experienced by Martin & Moyce (1952) during the experiment, leading to two different sets of measurement data being supplied.

Collapse of liquid column with obstacle

A more challenging version of the collapsing liquid column is the situation where an obstacle is placed in front of the advancing wave front. Koshizuka *et al.* (1995) also investigated this test case in experiments and photographs of the evolving wave front are available for comparison purposes. The experimental setup is shown in Fig. 6.16. The physical layout and dimensions of the tank are the same as for the

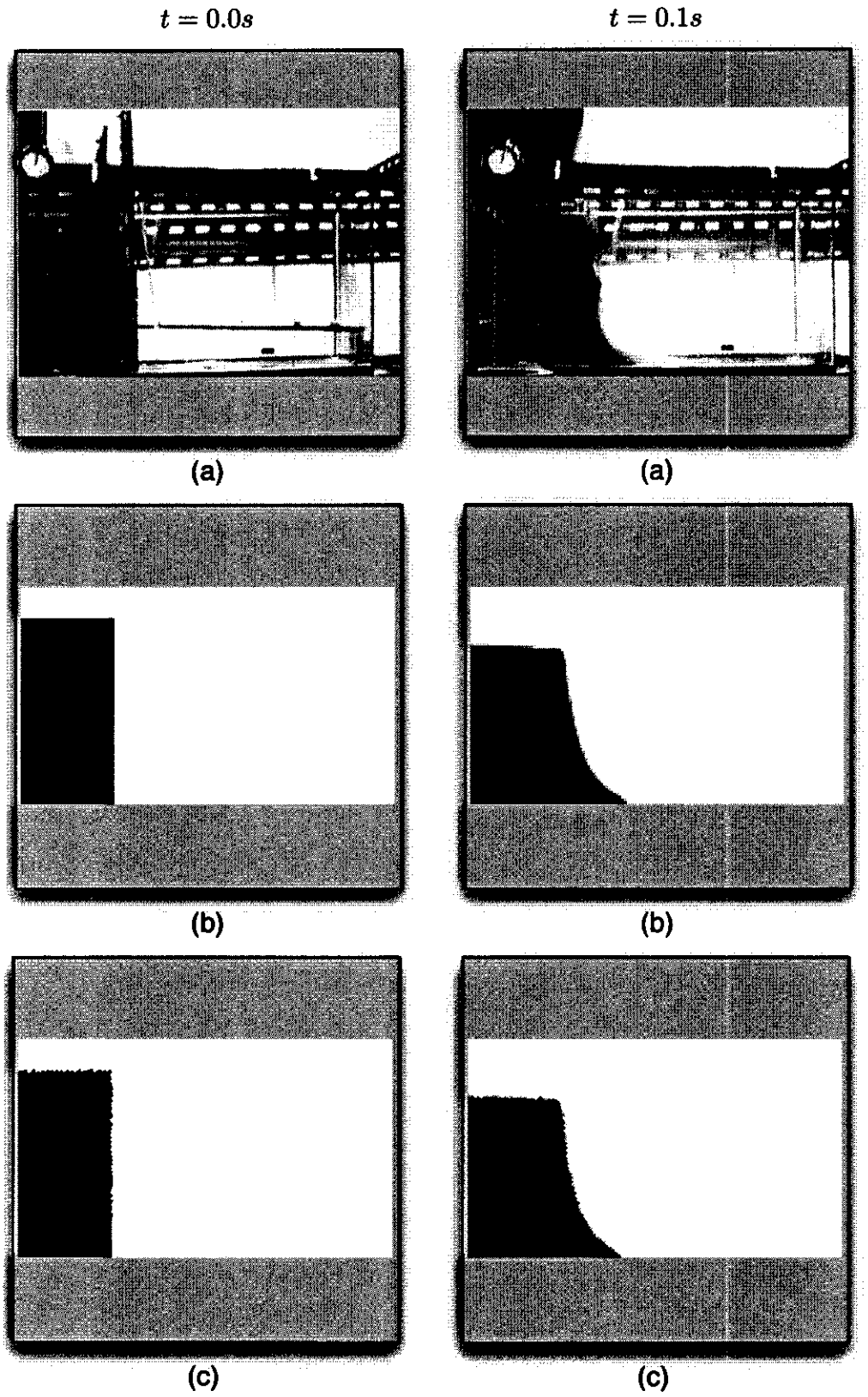


Figure 6.11: Collapsing liquid column at $t = 0.0s$ and $t = 0.1s$ a) Photographs from Koshizuka *et al.* (1995), b) structured and c) unstructured solutions

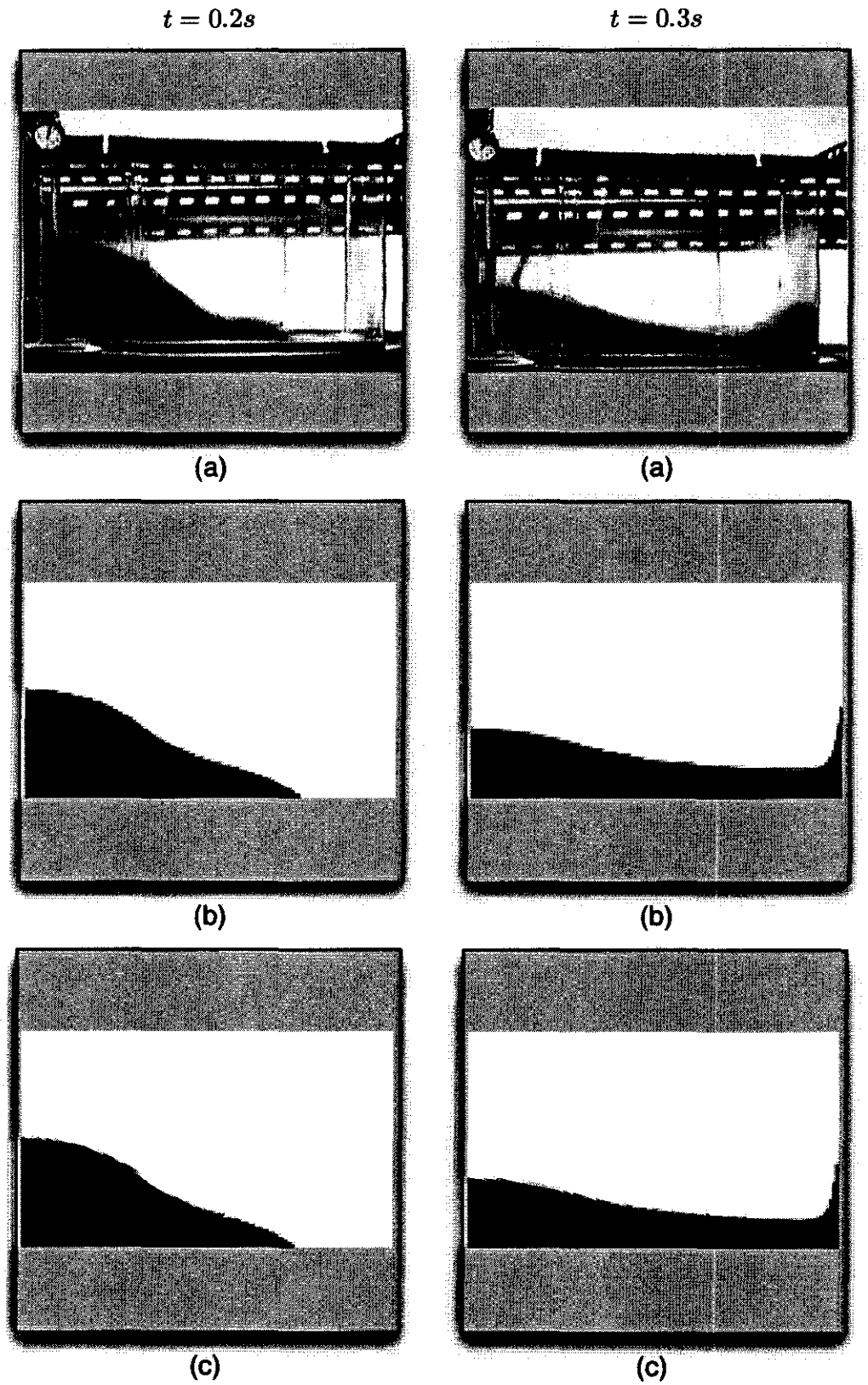


Figure 6.12: Collapsing liquid column at $t = 0.2s$ and $t = 0.3s$ a) Photographs from Koshizuka *et al.* (1995), b) structured and c) unstructured solutions

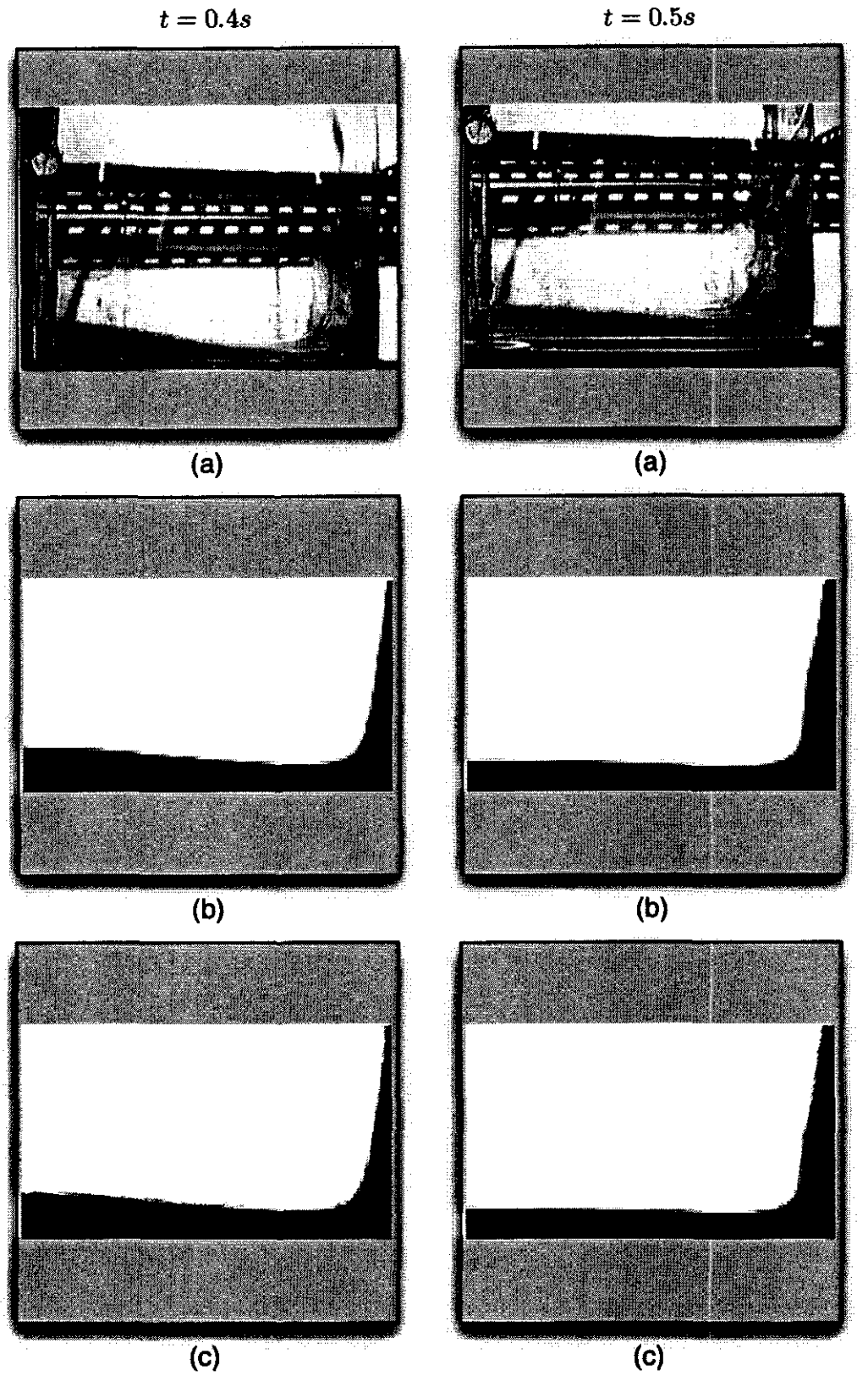


Figure 6.13: Collapsing liquid column at $t = 0.4s$ and $t = 0.5s$ a) Photographs from Koshizuka *et al.* (1995), b) structured and c) unstructured solutions

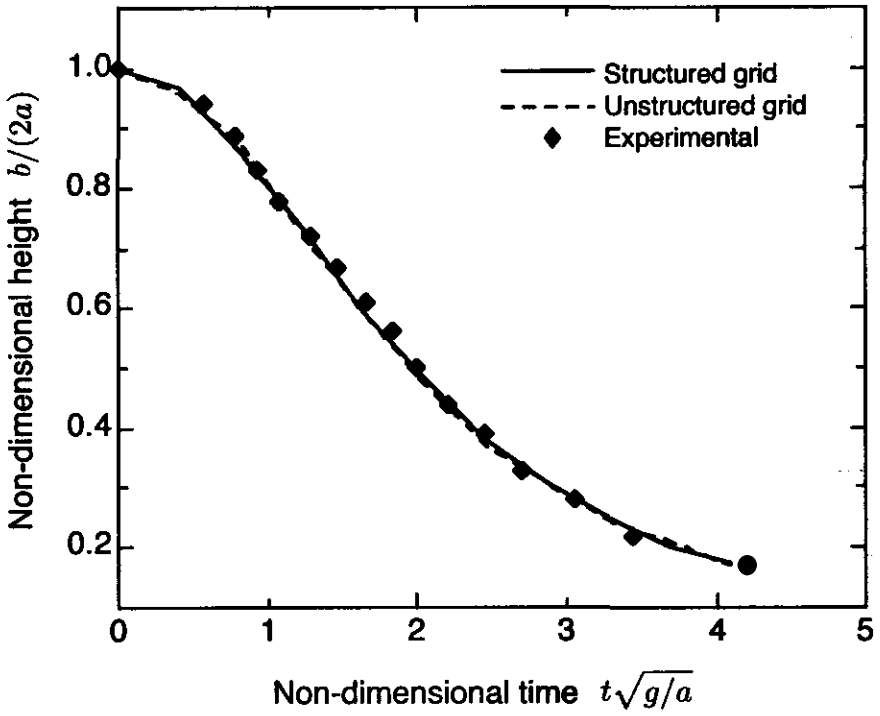


Figure 6.14: The height of the collapsing liquid column versus time.

previous test, with only the addition of the obstacle at a distance of $2a$ from the side wall. The obstacle is $0.024m$ wide and $0.048m$ high.

For the numerical model, the previous configuration of an open pressure boundary as the top of the tank and no-slip wall boundaries is used with the only change being the inclusion of the obstacle on the bottom of the tank. The same flow conditions and initial volume fraction distribution are tested on the structured and unstructured grids shown in Fig. 6.10(b) and Fig. 6.10(d).

The results are shown per time step for each of the three configurations: the experimental shape versus the interface captured on a structured grid and the prediction of the flow on the unstructured grid. The maximum number of cells defined by the adaptive structured grid is 2109 cells, and for the unstructured grid it is 1914. For the same time step, a comparison between the cell usage of the non-obstacle and obstacle cases reveal that the model with the obstacle requires $\pm 20\%$ less cells. This is related to the average size of cells in the mesh, where larger cells result in fewer refined cells.

The figures are Fig. 6.17: $t = 0.0s$, $0.1s$, Fig. 6.18: $t = 0.2s$, $0.3s$ and Fig. 6.19: $t = 0.4s$, $0.5s$.

The sequence of events that take place when the plate is lifted, is best described by referring to the photographs taken by Koshizuka *et al.* (1995) at the time steps

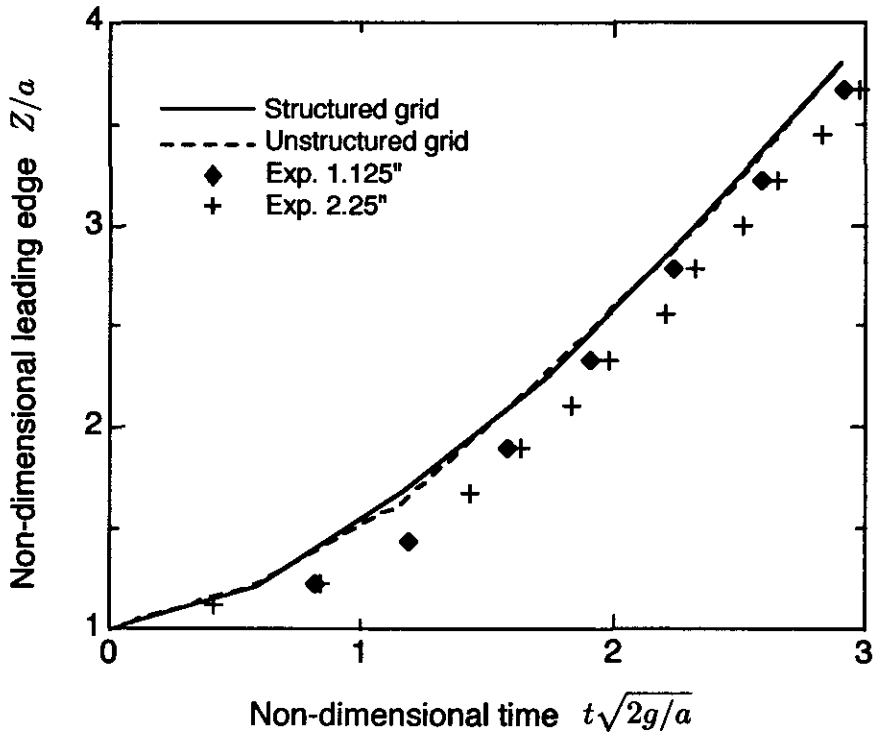


Figure 6.15: The position of the front of the collapsing liquid column versus time.

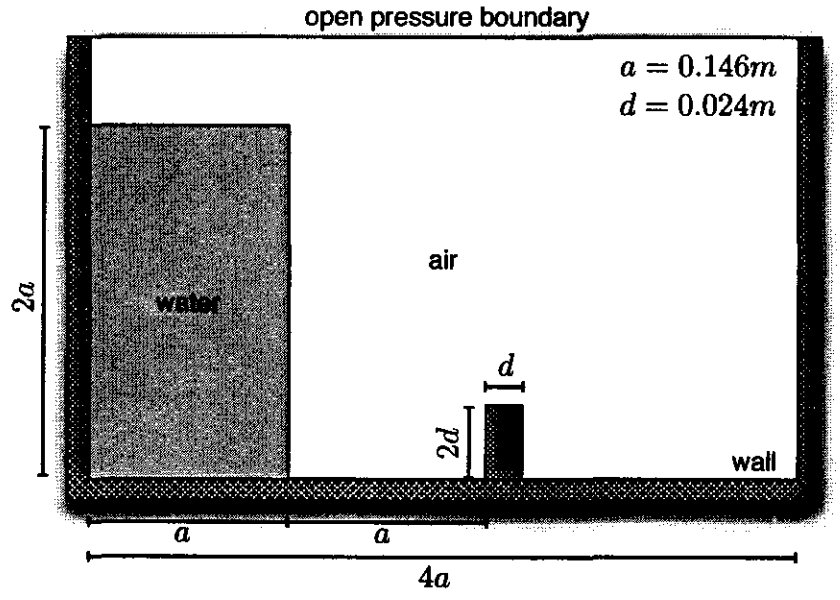


Figure 6.16: The model for the collapsing liquid column with an obstacle.

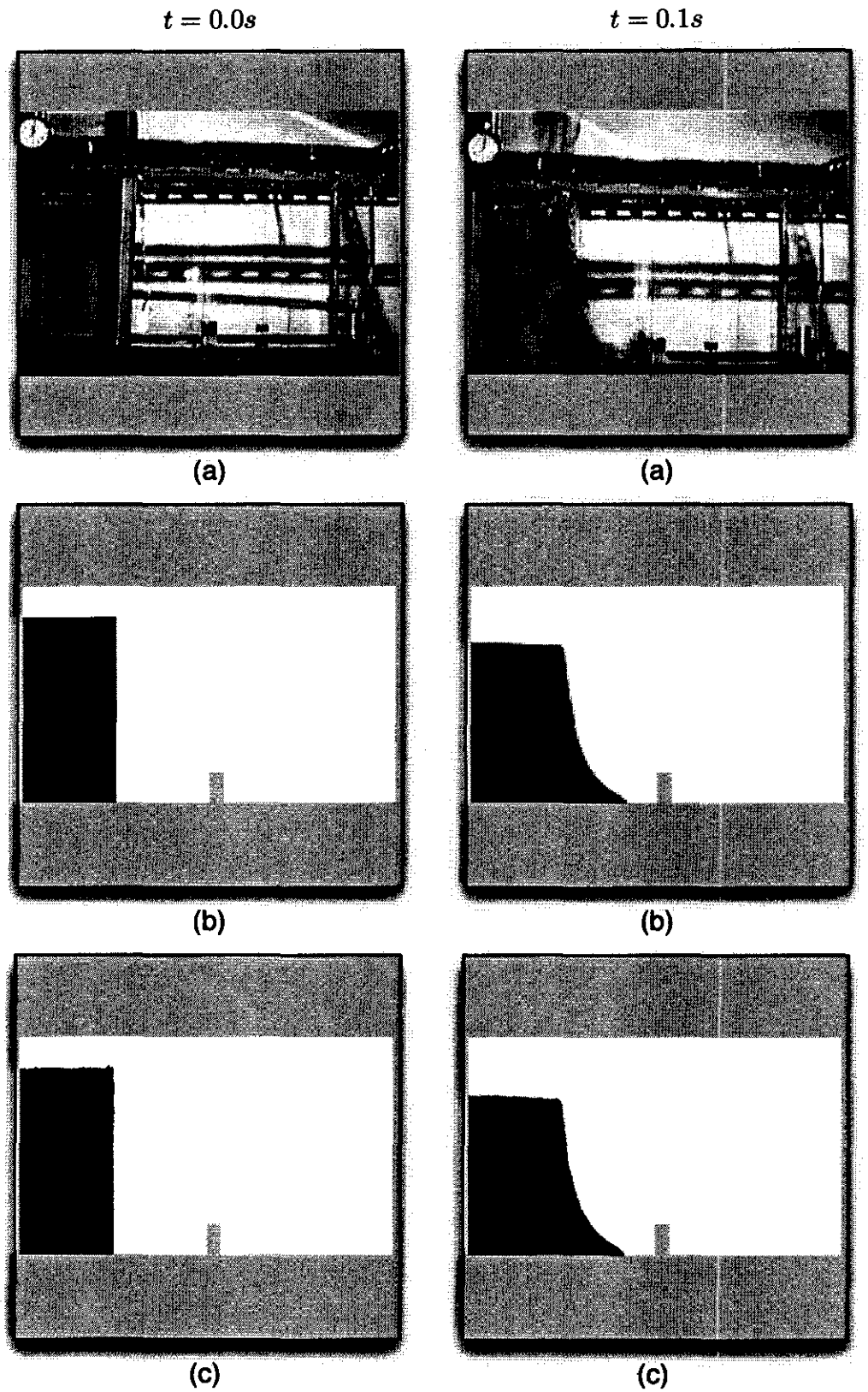


Figure 6.17: Collapsing column with obstacle at $t = 0.0s$ and $t = 0.1s$ a) Photographs from Koshizuka *et al.* (1995), b) structured and c) unstructured solutions

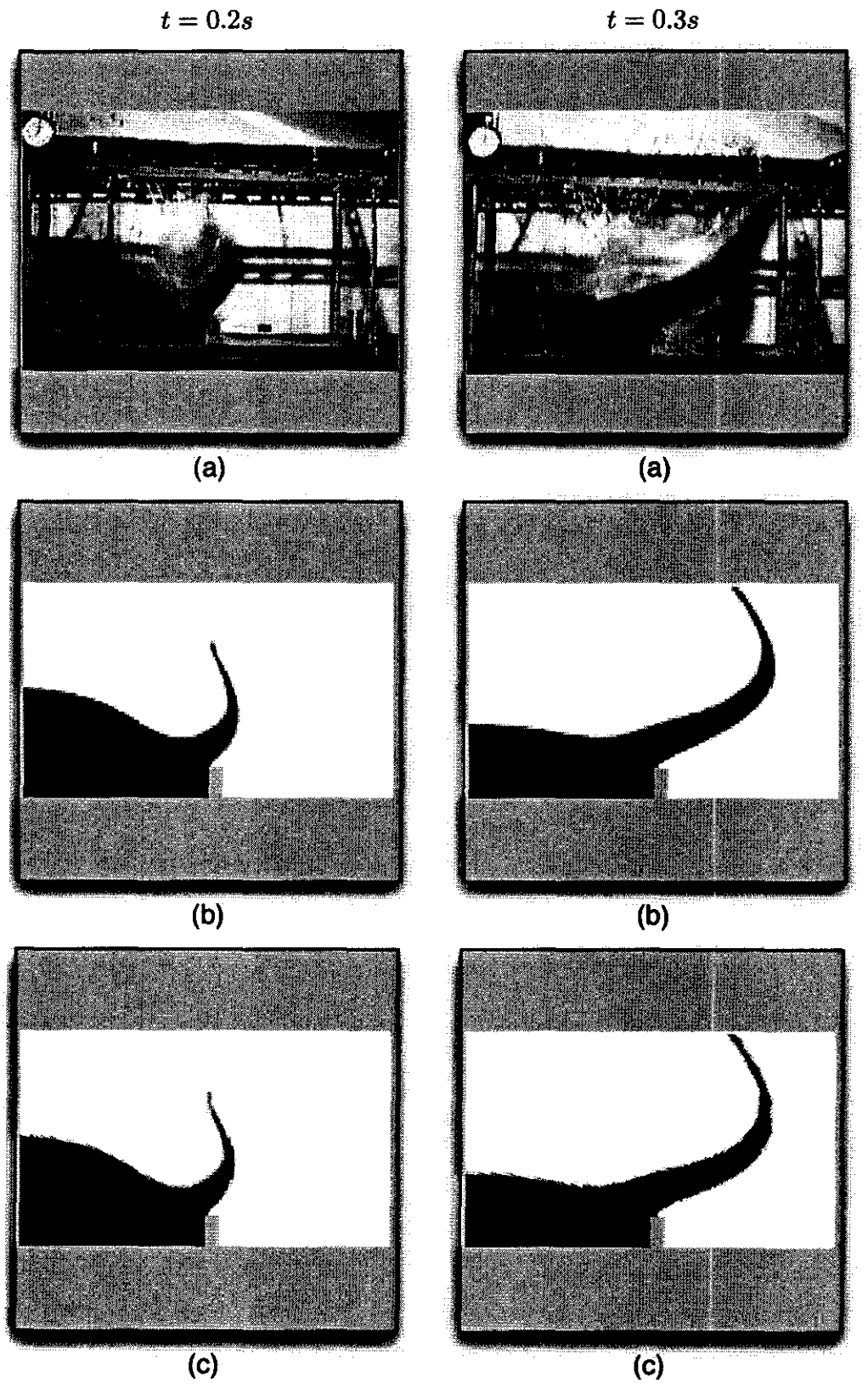


Figure 6.18: Collapsing column with obstacle at $t = 0.2s$ and $t = 0.3s$ a) Photographs from Koshizuka *et al.* (1995), b) structured and c) unstructured solutions

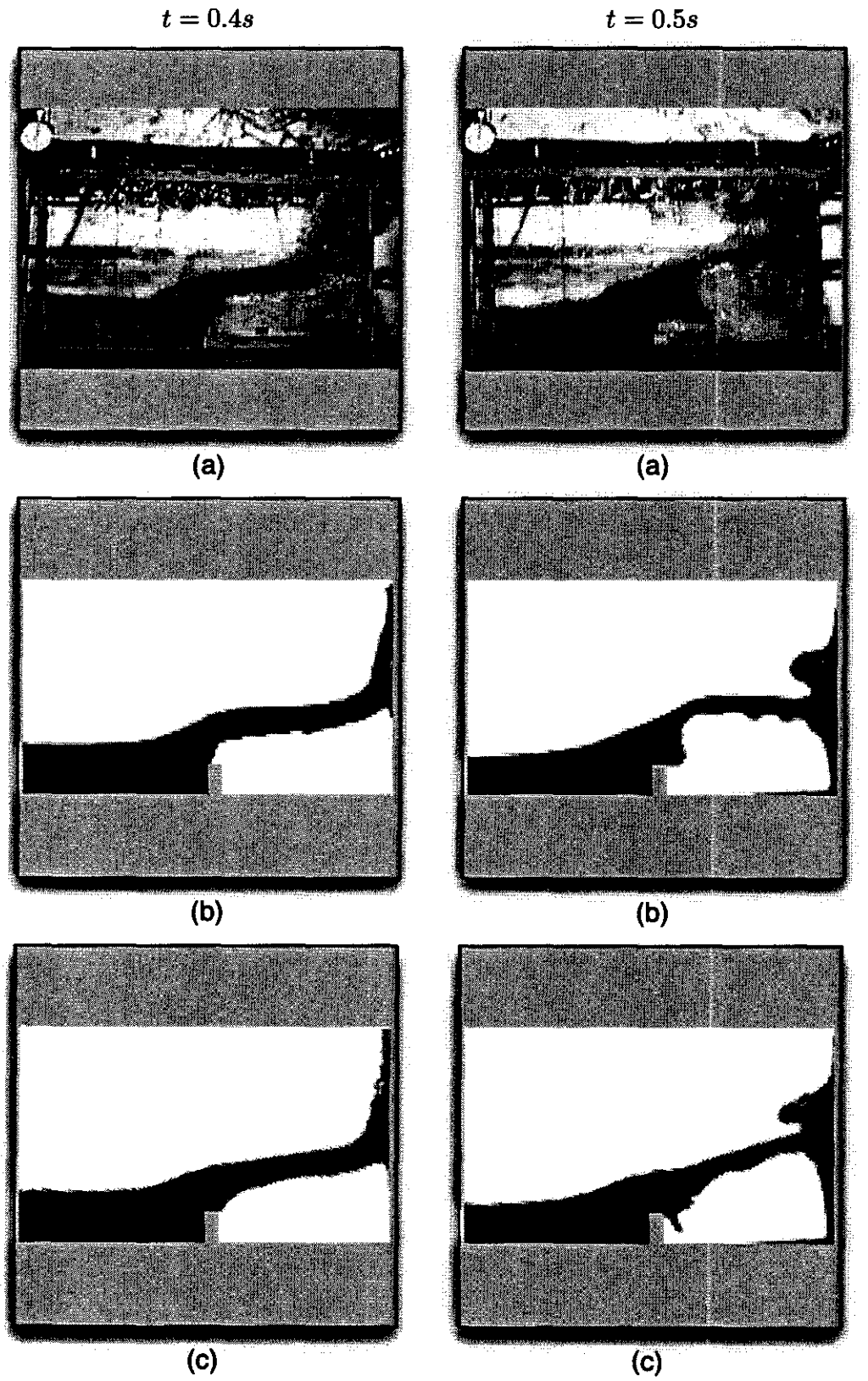


Figure 6.19: Collapsing column with obstacle at $t = 0.4s$ and $t = 0.5s$ a) Photographs from Koshizuka *et al.* (1995), b) structured and c) unstructured solutions

indicated:

Time $t = 0.1s$: The plate has just been lifted, the column is starting to collapse and the foot of the column is moving outwards towards the obstacle.

Time $t = 0.2s$: The wave front has hit the obstacle and shoots into the air over the obstacle. While moving upwards the small tongue of water stays attached to the upper-left corner of the obstacle.

Time $t = 0.3s$: The continuing collapse of the initial column has caused the tongue of water to increase in size and it is now halfway across the domain, stretching from the obstacle to the opposite wall.

Time $t = 0.4s$: The wave crashes into the wall, trapping a large air bubble underneath the tongue. In the experiment, high levels of dispersed flow are created, which are treated as part of the continuum phase by the current numerical model. Despite this limitation, the interface predicted by the numerical model gives a good representation of the bulk interface of the two-phase flow.

Time $t = 0.5s$: A secondary tongue is formed that splashes across the obstacle. The water is starting to run down the opposite wall and the air bubble in the corner is starting to burst through the sheet of water above it.

These combined two-fluid problems serve to illustrate the practicality and advantages of local mesh refinement in solving real world problems.

6.4 Closure

In this chapter the adaptive meshing methodology was evaluated when applied in conjunction with two-fluid models. The concept of local refinement was tested on structured and unstructured grids using the arbitree grid structure and simulation toolkit described earlier. Convergence tests illustrated that adaptive grids can reduce grid errors to the same level of magnitude as that obtained by refining uniform grids, but with a large saving in computational resources—savings that are compounded whenever grid resolutions need to be improved.

Modelling tests were first done by using a prescribed flow field and only solving for the indicator equation and refining the grid where needed. Three different cases were investigated to test the advection of the volume fraction in a oblique flow field. Mesh adaptation was also investigated for cases with rotating flow and reversed shear flow. Numerical errors for the shape preservation compared favourably with published errors on uniform grids.

Two real world cases were subsequently tested to assess the adaptive methodology when used in a complete two-fluid simulation. The first case was the collapse of a liquid column with the second case a variation on the first, where an obstacle is

placed in front of the leading edge of the flow. Successful comparisons could be made between photographs of the flow pattern and the column behaviour predicted by the numerical model. Limited experimental measurements of the height and speed of the column were also used to validate the numerical results.

One of the challenges encountered with all cases was the non-orthogonal error introduced when cells in a mesh are locally refined. In effect this means that a grid that starts out as a well behaved uniform grid, transforms into a locally unstructured, non-uniform grid when refinement is applied. The non-orthogonal error can cause numerical distortion of the interface, especially when the interface is tangential to the flow direction.

The test cases illustrated that the adaptive methodology can successfully be used together with two-fluid models to obtain solutions with the same order of numerical error as uniform grids, but with fewer cells. It was further shown that the dynamic nature of the mesh adaptation successfully follows the evolution of the interface during a simulation to apply refinement only where it is needed.

In the next chapter, the main findings of the study are summarized and suggestions for future research are made, based on shortcomings identified in the adaptive methodology and interesting challenges faced during the course of the study.

Chapter 7

Conclusion and recommendation

The previous chapters introduced an adaptive grid methodology for the finite volume method, in particular to aid the simulation of two-fluid systems. Adaptive refinement using a hierarchy of refined cells on a base grid was developed for uniform and arbitrary meshes. The adaptive grid was linked to a simulation toolkit using arbitrary field and problem definitions and was successfully used to model a selection of well known two-fluid test cases.

The main findings of the study are presented in the following section, followed by recommendations for future research and the final conclusion.

7.1 Summary

In the first chapter, an overview of the state of the union between simulation toolkits, adaptive mesh refinement and two-fluid simulation was given. Different design paradigms for toolkits were evaluated, mesh refinement techniques were investigated and the current state-of-the-art in two-fluid modelling techniques were examined. These evaluations were done with the aim of establishing a framework combining mesh refinement and two-fluid simulations on arbitrary meshes in a natural and computationally efficient manner. It is desirable to combine local mesh adaptation with interface modelling, because the highly localized nature of the interface creates the ideal situation where cells can be selected for local refinement. Local refinement offers a significant reduction in computational overheads and provides first order mesh-based error reduction where it is needed most.

The most appropriate toolkit paradigm to use as foundation was identified as one where simulation models are constructed with the arbitrary combination of user-defined fields and algorithms through field-based operators. The volume-of-fluid method was chosen as the two-fluid component because of the similarity between the VOF formulation and the other fluid equations in the system. The VOF method offers additional advantages such as the robust handling of rupturing interfaces and computational efficiency. For mesh adaptation, a hierarchical strategy was adopted

that recursively refines cells only in the regions of interest without creating a broad, unwanted area of refined cells.

After the most suitable technology components were identified, the process started to combine it all into a single framework. The mathematical formulations for two-fluid models were derived in such a manner to support the face-based operator design of the toolkit. The use of mesh refinement with two-fluid models required the development of appropriate techniques to transfer variables between grids in a conservative manner. The values are transfered between grids by using the local field gradient and enforcing the volume-weighted conservation of properties when cells are subdivided into refined cells. For fluid flow problems, an additional reconstruction step is done to obtain conservative fluxes for the next iteration.

Hierarchical mesh adaptation was identified as the most efficient strategy for local refinement. Previous methods of hierarchical refinement were designed for uniform, structured meshes where the local numbering structure is used to determine connectivity and could not directly be included in a method for unstructured meshes.

To overcome this problem, a new recursive grid structure called *arbitrees* was developed in this study. The new design allows the anisotropic subdivision of cells and through the use of geometry-based connectivity detection can transparently be used on uniform and arbitrary meshes.

To validate the practicality of the framework design, containing the field-based operators and arbitree grid structures, several two-fluid test cases were investigated. These cases covered a wide spectrum, ranging from simple advection tests on a unifom flow field, to models containing rotating and reversed shear flow. The final test cases proved the success with which the new mesh adaptation strategy can be applied to two-fluid simulations of real world problems.

7.2 Recommendation

The main challenge for any refinement strategy is to transfer the conserved properties from one grid to another grid while ensuring volume-based conservation of properties in refined cells. The strategy employed in this study was to use a second-order accurate extrapolation scheme based on the local gradient and to enforce the conservation afterwards. To ensure conservation of a generic property, the extrapolated values in refined cells are iteratively corrected with an explicit correction factor based on the ratio between the amount in the parent cell and the average amount in the refined cells.

The explicit correction process works well most of the time, except when only one cell of e.g. four or eight children must be modified. The relative correction that can be applied to a single cell is small compared to the fixed contribution of the other cells and can increase the required number of explicit iterations by an order of magnitude.

The calculation of conservative fluxes on the faces of refined cells is another challenge on its own. When a cell is refined, there is a definite relationship between a parent cell and its children. The new cell faces on the other hand, do not necessarily have corresponding faces in the parent cell, making the direct transfer of conservative fluxes impossible. The current method used transferred the cell-based components of the pressure equation to the new grid and from there calculated new conservative volume fluxes. A more elegant reconstruction method that only operates in the refined cell and does not modify the rest of the domain is required.

Further work could be done to obtain an implicit formulation for the variable field defined on the cells and faces of children of a refined cell. An efficient method is one that enforces property conservation within a few iterations. Reconstruction schemes based on least-squares-fit methods take the whole domain into account when values and fluxes are reconstructed and does not guarantee local conservation of properties. When the numerical model contains localized phenomena like an interface, one does not want domain-wide effects to contaminate the local position of the interface. It is envisioned that a modified form of the least-squares-fit method could however be used to calculate the new conservative distribution of values and fluxes in a refined cell.

Mesh errors caused by cell refinement is another area where further research is warranted. The subdivision of a cell introduces non-orthogonal errors when the connecting vectors between neighbouring cells do not pass through the centres of cell faces anymore. To improve this situation would probably involve modifications to the discretization schemes, especially the scheme used for the convection of the volume fraction. The CICSAM scheme used in the current methodology determines the weighting factors based on the angle between the interface and the flow direction, and results on the adapted grids suggest that the non-orthogonality errors influence the CICSAM weighting factors. This phenomenon manifests itself in a distortion of the interface that occurs when adapted grids are used, but the distortion is absent when the corresponding uniform grid is used for the same test case.

When hierarchical refinement is applied to the cells in an unstructured grid, the resolution and size composition of the base grid have a profound effect on the quality of the refined grid. An inspection of the typical unstructured grid used in this study reveals that cell volumes fall between being 50% smaller to being 50% larger than the average volume. For smaller cells it could thus be beneficial to allow lower levels of refinement only, while larger cells may benefit by allowing higher levels of refinement. This would lead to a more homogeneous size distribution in a grid. The refinement could also be controlled based on the volume-weighted error content of a cell. The maximum allowable level of refinement would then change throughout the grid based on a cell's relative volume and error contribution. Future implementations of the refinement algorithm could be modified to allow different maximum refinement levels for cells.

The current study largely ignored the effects of surface tension on the interface—an assumption afforded by virtue of the large physical scale of the test cases. Ad-

aptive refinement could be used with great success in models containing surface tension, where the increased mesh resolution of the interface region would make pressure balances on the interface more accurate and help eliminate the problem of erroneous parasite currents on the interface.

7.3 Conclusion

This study set out to prove that local, adaptive mesh refinement can successfully be used by two-fluid simulations on unstructured meshes. The adaptive methodology developed in this study offers arbitrees, a concept for recursive grid designs that enable any type of three-dimensional grid to be refined in a hierarchical manner. It allows for the efficient solution of transient two-fluid systems with sharp interfaces by concentrating the computational cells around the interface and adapting the mesh as the solution evolves. The adaptive methodology however, reaches beyond two-fluid models, as any numerical formulation applied on volume-based domains will also benefit from the use of such a technique. It is the hope of the author that this study forms a stepping stone on the road towards efficient multi-fluid simulations by offering improved accuracy and saving computational resources.

Bibliography

ALCRUDO, F. 1999. A state of the art review on mathematical modelling of flood propagation. *Technical report.* Universidad de Zaragoza, Spain.

ANDERSON, J. D. 1995. *Computational Fluid Dynamics - The basics with applications.* McGraw-Hill.

ANON 1995a. *Pipes and Pipelines - Principles and Practice.* K. Myles and Associates.

ANON 1995b. *Pumps - Principles and Practice.* K. Myles and Associates.

APPLE 2002. *The Objective-C Programming Language.* Apple Computer, Inc.

AULISA, E., MANSERVISI, S., SCARDOVELLI, R. & ZALESKIB, S. 2003. A geometrical area-preserving volume-of-fluid advection method. *Journal of Computational Physics*, **192**: 355–364.

AZCUETA, R., HADZIC, I., MUZAFERIJA, S. & PERIC, M. 1999. Computation of flows with free surfaces. (***In:*** MARNET-CFD. MARNET-CFD First Workshop - Barcelona 1999.)

BAKER, T. J. 1997. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, **25**: 243–273.

BALSARA, D. S. & NORTON, C. D. 2001. Highly parallel structured adaptive mesh refinement using parallel language-based approaches. *Parallel Computing*, **27**: 37–70.

BARRY, W. J., JONES, M. T. & PLASSMANN, P. E. 1998. Parallel adaptive mesh refinement techniques for plasticity problems. *Advances in Engineering Software*, **29**(31): 217–225.

BOHN, S. & THORNTON, E. 1994. Environment reconstruction for robot navigation. (***In:*** SPIE. Proceedings of the International Society for Optical Engineering)

BOIVIN, C. & OLLIVIER-GOOCH, C. 2004. A toolkit for numerical simulation of PDEs. II. Solving generic multiphysics problems. *Computer Methods in Applied Mechanics and Engineering*, **193**: 3891–3918.

BOUCHARD, P., BAY, F. & CHASTEL, Y. 2003. Numerical modelling of crack propagation: automatic remeshing and comparison of different criteria. *Computer Methods in Applied Mechanics and Engineering*, **192**: 3887–3908.

BOURKE, P. 1987. Determining if a point lies on the interior of a polygon. *Technical report.* Swinburne Centre for Astrophysics and Supercomputing.

BRACKBILL, J., KOTHE, D. & ZEMACH, C. 1992. A continuum method for modeling surface tension. *Journal of Computational Physics*, **100**: 335–354.

BURCHARD, H. & BECKERS, J.-M. 2004. Non-uniform adaptive vertical grids in one-dimensional numerical ocean models. *Ocean Modelling*, **6**: 51–81.

CAREY, G. F., ANDERSON, M., CARNES, B. & KIRK, B. 2004. Some aspects of adaptive grid technology related to boundary and interior layers. *Journal of computational and applied mathematics*, **166**: 55–86.

CASTAÑOS, J. G. & SAVAGE, J. E. 1999a. Parallel refinement of unstructured meshes. (***In:*** IASTED. Proceedings of the IASTED Internetional Conference 1999. p. 1–11.)

CASTAÑOS, J. G. & SAVAGE, J. E. 1999b. PARED: a framework for the adaptive solution of PDEs. (***In:*** SIAM. Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computation.)

CLER, D. L. 2002. Tank and artillery cannon muzzle brakes - reducing gun recoil quietly. *Fluent News*, **11**(2): 10–11.

DEL-PINO, S. 2004. A hierarchical and view dependent visualization algorithm for tree based AMR data in 2D or 3D. (***In:*** ESPGV. Eurographics Symposium on Parallel Graphics and Visualization (2004).)

DENDY, E. D., PADIAL-COLLINS, N. T. & DER HEYDEN, W. 2002. A general-purpose finite-volume advection scheme for continuous and discontinuous fields on unstructured grids. *Journal of Computational Physics*, **180**: 559–583.

DU TOIT, C. 1998. Setting up the cell neighbour array and the list of external faces for a finite volume mesh. *R & D Journal*, **14**: 1–7.

ENRIGHT, D., FEDKIW, R., FERZIGER, J. & MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, **183**: 83–116.

FERZIGER, J. & PERIC, M. 2002. *Computational methods for fluid dynamics.* 3 ed. Springer-Verlag.

FILIPPONE, S., COLAJANNI, M. & PASCUCCI, D. 1999. An object-oriented environment for sparse parallel computation on adaptive grids. (*In:* IPPS/SPDP. Proceedings of the 13th International Parallel Processing Symposium.)

FOLEY, J. D., VAN DAM, A., FEINER, S. K. & HUGHES, J. F. 1990. *Computer Graphics - Principles and Practice.* 2 ed. Addison-Wesley.

FRAGA, E. S. 1998. Simulation of a fixed bed system using a geometrically based adaptive grid method. *Computers in Chemical Engineering*, **22**: 897–900.

FRIEDRICH, J. 1999. Object-oriented design and implementation of CFDLab: a computer-assisted learning tool for fluid dynamics using dual reciprocity boundary element methodology. *Computers and Geosciences*, **25**: 785–800.

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. 1995. *Design Patterns - Elements of reusable object-oriented software.* Addison-Wesley.

GINZBURG, I. & WITTUM, G. 2001. Two-phase flows on interface refined grids modeled with VOF, staggered finite volumes, and spline interpolants. *Journal of Computational Physics*, **166**: 302–335.

GLOBISCH, G. 1995. On an automatically parallel generation technique for tetrahedral meshes. *Parallel Computing*, **21**: 1979–1995.

GLOTH, O., HÄNEL, D., TRAN, L. & VILSMEIER, R. 2003. A front tracking method on unstructured grids. *Computers and Fluids*, **32**: 547–570.

GREAVES, D. 2001. Interface capturing with hierarchical grid refinement. (*In:* MARNET-CFD. MARNET-CFD Final Annual Workshop.)

GREAVES, D. 2004. A quadtree adaptive method for simulating fluid flows with moving interfaces. *Journal of Computational Physics*, **194**: 35–56.

HEARN, D. & BAKER, P. 1994. *Computer Graphics.* 2 ed. Prentice-Hall International.

HENTSCHEL, R. 1998. The creation of lift by sharp-edged delta wings. An analysis of a self-adaptive numerical simulation using the concept of vorticity content. *Aerospace Science and Technology*, **2**: 79–90.

HIRT, C. & NICHOLS, B. 1981. Volume of fluid (VOF) method for dynamics of free boundaries. *Journal of Computational Physics*, **39**: 201–225.

HOPPE, R. 2004. Adaptive multigrid and domain decomposition methods in the computation of electromagnetic fields. *Journal of Computational and Applied Mathematics*, **168**: 245–254.

HYMAN, J. M., LI, S., KNUPP, P. & SHASHKOV, M. 2000. An algorithm for aligning a quadrilateral grid with internal boundaries. *Journal of Computational Physics*, **163**: 133–149.

IDELSOHN, S. R., STORTI, M. A. & OÑATE, E. 2001. Lagrangian formulations to solve free surface incompressible inviscid fluid flows. *Computer Methods in Applied Mechanics and Engineering*, **191**: 583–593.

IVANENKO, S. A. & MURATOVA, G. V. 2000. Adaptive grid shallow water modeling. *Applied Numerical Mathematics*, **32**: 447–482.

JASAK, H. 1996. *Error analysis and estimation for the finite volume method with applications to fluid flows.* Imperial College of Science, Technology and Medicine, London. (Thesis - D.Phil.).

JASAK, H. & GOSMAN, A. 2000a. Automatic resolution control for the finite-volume method, Part 1: A-posteriori error estimates. *Numerical Heat Transfer*, **38**: 237–256.

JASAK, H. & GOSMAN, A. 2000b. Automatic resolution control for the finite-volume method, Part 2: Adaptive mesh refinement and coarsening. *Numerical Heat Transfer*, **38**: 257–271.

JASAK, H. & GOSMAN, A. 2000c. Automatic resolution control for the finite-volume method, Part 3: Turbulent flow applications. *Numerical Heat Transfer*, **38**: 273–290.

JASAK, H. & WELLER, H. 1995. Interface tracking capabilities of the inter-gamma differencing scheme. *Technical report.* Imperial College of Science, Technology and Medicine.

JESSEE, J. P., FIVELAND, W. A., HOWELL, L. H., COLELLA, P. & PEMBER, R. B. 1998. An adaptive mesh refinement algorithm for the radiative transport equation. *Journal of Computational Physics*, **139**: 380–398.

KALLINDERIS, Y. 1996. A 3-D finite-volume method for the Navier-Stokes equations with adaptive hybrid grids. *Applied Numerical Mathematics*, **20**: 387–406.

KHAWAJA, A., MINYARD, T. & KALLINDERIS, Y. 2000. Adaptive hybrid grid methods. *Computer Methods in Applied Mechanics and Engineering*, **189**: 1231–1245.

KHOKHLOV, A. 1998. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *Journal of Computational Physics*, **143**: 519–543.

KITAGAWA, A., MURAI, Y. & YAMAMOTO, F. 2001. Two-way coupling of Eulerian-Lagrangian model for dispersed multiphase flow using filtering functions. *International Journal of Multiphase Flow*, **27**: 2129–2153.

KOHN, S. R. 2001. Parallel software abstractions for structured adaptive mesh methods. *Journal of Parallel and Distributed Computing*, **61**: 713–736.

KOSHIZUKA, S., TAMAKO, H. & OKA, Y. 1995. A particle method for incompressible viscous flow with fluid fragmentation. *Computational Fluid Dynamics Journal*, **4**(1): 29–46.

KRISHNAMOORTHY, C., RAPHAEL, B. & MUKHERJEE, S. 1995. Meshing by successive superelement decomposition (MSD) - a new approach to quadrilateral mesh generation. *Finite Elements in Analysis and Design*, **20**: 1–37.

KWAK, D.-Y. & IM, Y.-T. 2003. Hexahedral mesh generation for remeshing in three-dimensional metal forming analyses. *Journal of Materials Processing Technology*, **138**: 531–537.

KWON, J. & JEONG, H. 1996. Solution-adaptive grid generation for compressible flow. *Computers and Fluids*, **25**(6): 551–560.

LEONARD, B. P. 1979. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, **19**: 59–98.

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.

LI, J. & CHEN, C. 2002. A simple efficient algorithm for interpolation between different grids in both 2D and 3D. *Mathematics and Computers in Simulation*, **58**: 125–132.

LIM, Y., LE LANN, J. & JOULIA, X. 2001. Moving mesh generation for tracking a shock or steep moving front. *Computers and Chemical Engineering*, **25**: 653–663.

LIU, J.-L., LIN, I.-J., SHIH, M.-Z., CHEN, R.-C. & HSIEH, M.-C. 1996. Object-oriented programming of adaptive finite element and finite volume methods. *Applied Numerical Mathematics*, **21**.

LI, Y., ZHANG, J. & FAN, L.-S. 1999. Numerical simulation of gas-liquid-solid fluidization systems using a combined CFD-VOF-DPM method: bubble wake behavior. *Chemical Engineering Science*, **54**: 5101–5107.

LÖRSTAD, D. & FUCHS, L. 2004. High-order surface tension VOF-model for 3D bubble flows with high density ratio. *Journal of Computational Physics*, **200**: 153–176.

MACNEICE, P., OLSON, K. M., MOBARRY, C., DE FAINCHTEIN, R. & PACKER, C. 2000. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, **126**: 330–354.

MALIK, M. 2004. *Volume tracking with adaptive refinement.* Department of Mechanical and Industrial Engineering, University of Toronto. (Thesis - M.Sc.).

MARGOLIN, L. & SHASHKOV, M. 2003. Second-order sign-preserving conservative interpolation (remapping) on general grids. *Journal of Computational Physics*, **184**: 266–298.

MARTIN, J. C. & MOYCE, W. J. 1952. An experimental study of the collapse of liquid columns on a rigid horizontal plane. *Philosophical Transactions of the Royal Society, London*, **A244**: 312–324.

MAVRIPLIS, D. 2003. Revisiting the least-squares procedure for gradient reconstruction on unstructured meshes. *Technical report.* National Institute of Aerospace, Hampton VA.

MENCINGER, J. 2004. Numerical simulation of melting in two-dimensional cavity using adaptive grid. *Journal of Computational Physics*, **198**: 243–264.

MUNTHE, O. & LANGTANGEN, H. P. 2000. Finite elements and object-oriented implementation techniques in computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, **190**: 865–888.

MUZAFERIJA, S. & PERIC, M. 1998. *Computation of free surface flows using interface-tracking and interface-capturing methods*, Computational Mechanics Publications, Southampton, chapter 2, 23 p.

NEEMAN, H. J. 1996. *Autonomous hierarchical adaptive mesh refinement for multiscale simulations.* University of Illinois at Urbana-Champaign. (Thesis - D.Phil.).

OLDEHOEFT, R. 2000. Taming complexity in high-performance computing. *Mathematics and Computers in Simulation*, **54**: 341–357.

OLLIVIER-GOOCH, C. 2003. A toolkit for numerical simulation of PDEs. I. Fundamentals of generic finite-volume simulation. *Computer Methods in Applied Mechanics and Engineering*, **192**: 1147–1175.

OXFORD 1995. *The Concise Oxford Dictionary.* 9 ed. Oxford University Press.

PANTALÉ, O., CAPERAA, S. & RAKOTOMALALA, R. 2004. Development of an object-oriented finite element program: application to metal-forming and impact simulations. *Journal of computational and applied mathematics*, **168**: 341–351.

PATANKAR, S. V. 1980. *Numerical Heat Transfer and Fluid Flow.* Hemisphere Publishing Corporation.

PEMBER, R. B. & BELL, J. B. 1995. An adaptive Cartesian grid method for unsteady compressible flow in irregular regions. *Journal of Computational Physics*, **120**: 278–304.

PHONGTHANAPANICH, S. & DECHAUMPHAI, P. 2004. Adaptive delaunay triangulation with object-oriented programming for crack propagation analysis. *Finite Elements in Analysis and Design*, **40**: 1753–1771.

PLAZA, A., PADRÓN, M. A. & CAREY, G. F. 2000. A 3D refinement/derefinement algorithm for solving evolution problems. *Applied Numerical Mathematics*, **32**: 401–418.

POPINET, S. 2003. GERRIS: A tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics*, **190**: 572–600.

RHIE, C. M. & CHOW, W. L. 1983. A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation. *AIAA Journal*, **21**: 1525–1532.

ROETTGER, S., SCHULZ, M., BARTELHEIMER, W. & ERTL, T. 2002. Flow visualization on hierarchical Cartesian grids, *Lecture Notes in Computational Science and Engineering (Proceedings of 3rd International FORTWIHR Conference on HPSEC)*, Vol. 21, Springer Verlag, p. 139–146.

RUDMAN, M. 1997. Volume-tracking methods for interfacial flow calculations. *International Journal of Numerical Methods in Fluids*, **24**: 671–691.

SELMAN, A., HINTON, E. & BICANIC, N. 1997. Adaptive mesh refinement for localised phenomena. *Computers and Structures*, **63**(3): 475–495.

SHAHBAZI, K., PARASCHIVOIU, M. & MOSTAGHIMI, J. 2003. Second order accurate volume tracking based on remapping for triangular meshes. *Journal of Computational Physics*, **188**: 100–122.

SOULI, M. & ZOLESIO, J. 2001. Arbitrary Lagrangian-Eulerian and free surface methods in fluid mechanics. *Computer Methods in Applied Mechanics and Engineering*, **191**: 451–466.

STEWART, J. 1998. *Multivariable Calculus - Concepts and Contexts.* Brooks/Cole Publishing Company.

STEWART, J. R. & EDWARDS, H. C. 2004. A framework approach for developing parallel adaptive multiphysics applications. *Finite Elements in Analysis and Design*, **40**: 1599–1617.

STOUT, Q. F., DE ZEEUW, D. L., GOMBOSI, T. I. & GROTH, C. P. T. 1997. Adaptive blocks: a high performance data structure. (***In:*** SC97. Scientific Computing SC97 Techinical Papers.)

STROUSTRUP, B. 1991. What is "object-oriented programming"? *Technical report.* AT & T Bell Laboratories.

STROUSTRUP, B. 1997. *The C++ Programming Language.* 3 ed. Addison-Wesley.

STROUSTRUP, B. 1999. *The handbook of object technology*, CRC Press LLC, chapter An overview of the C++ programming language, p. 1–23.

TEZDUYAR, T. & ALIABADI, S. 2000. EDICT for 3D computation of two-fluid interfaces. *Computer Methods in Applied Mechanics and Engineering*, **190**: 403–410.

TRYGGVASON, G., BUNNER, B., ESMAEELI, A., JURIC, D., AL-RAWAHI, N., TAUBER, W., HAN, J., NAS, S. & JAN, Y. J. 2001. A front tracking method for the computations of multiphase flow. *Journal of Computational Physics*, **169**: 708–759.

UBBINK, O. 1997. *Numerical prediction of two fluid systems with sharp interfaces.* Imperial College of Science, Technology and Medicine, London. (Thesis - D.Phil.).

UBBINK, O. & ISSA, R. 1999. A method for capturing sharp fluid interfaces on arbitrary meshes. *Journal of Computational Physics*, **153**: 26–50.

VAN DER PIJL, S., SEGAL, A. & VUIK, C. 2003. A mass-conserving level-set (MCLS) method for modeling of multi-phase flows. *Technical report.* Delft University of Technology.

VERSTEEG, H. & MALALASEKERA, W. 1995. *An introduction to computational fluid dynamics.* Addison-Wesley Longman Limited.

VIERENDEELS, J., MERCI, B. & DICK, E. 2004. A multigrid method for natural convective heat transfer with large temperature differences. *Journal of Computational and Applied Mathematics*, **168**: 509–517.

WANG, Z. 1998. A quadtree-based adaptive Cartesian/quad grid flow solver for Navier-Stokes equations. *Computers and Fluids*, **27**(4): 529–549.

WANG, Z., PRZEKWASB, A. & LIU, Y. 2002. A FV-TD electromagnetic solver using adaptive Cartesian grids. *Computer Physics Communications*, **148**: 17–29.

WEBER, G. H., OHLER, M., KREYLOS, O., SHALF, J. M., BETHEL, E. W., HAMANN, B. & SCHEUERMANN, G. 2003. Parallel cell projection rendering of adaptive mesh refinement data. (***In:*** IEEE. IEEE PVG 2003.)

WELLER, H., TABOR, G., JASAK, H. & FUREBY, C. 1998. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics*, **12**(6): 620–631.

WILLE, S. 1996. The prolonged adaptive multigrid method for finite element Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, **138**: 227–271.

YAMAGUCHI, H. & TAKUSHIMA, A. 2004. Simulating interfacial deformation by arbitrary Lagrangian-Eulerian approach. *Computer Methods in Applied Mechanics and Engineering*, **193**: 4439–4456.

YIU, K., GREAVES, D., CRUZ, S., SAALEHI, A. & BORTHWICK, A. 1996. Quadtree grid generation: information handling, boundary fitting and CFD applications. *Computers and Fluids*, **25**(8): 759–769.

ZHENG, Y. & LIOU, M.-S. 2003a. A novel approach of three-dimensional hybrid grid methodology: Part 1. Grid generation. *Computer Methods in Applied Mechanics and Engineering*, **192**: 4147–4171.

ZHENG, Y. & LIOU, M.-S. 2003b. A novel approach of three-dimensional hybrid grid methodology: Part 2. Flow solution. *Computer Methods in Applied Mechanics and Engineering*, **192**: 4173–4193.

ZIEGLER, U. 1998. NIRVANA: An adaptive mesh refinement code for gas dynamics and MHD. *Computer Physics Communications*, **109**: 111–134.

ZIEGLER, U. 1999. A three-dimensional Cartesian adaptive mesh code for compressible magnetohydrodynamics. *Computer Physics Communications*, **116**: 65–77.