Adaptive Mesh Refinement using Subdivision of Unstructured Elements for Conservation Laws[1]

Daniel B. Vollmer

1st September 2003

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

[1]Submitted to the Department of Mathematics, University of Reading, in partial fulfilment of the requirements for the Degree of Master of Science.

Abstract

An adaptive method based on recursive subdivision of unstructured elements for the solution of conservation laws is presented. The refinement of cells is based on regular subdivision into four children as indicated by a gradient-detector and is carried out "Just-In-Time" before the actual computation on that element takes place. In addition to spatial refinement, temporal refinement is carried out in conjunction with "lock-step" time-stepping to guarantee the availability of the proper states at the correct times across the mesh on all scales. The approach uses standard slope-limited finite volume methods of MUSCL-Hancock type with slight modifications to cater for different levels of subdivision in adjacent elements. We present some background to AMR and the finite volume framework, the algorithm itself and conclude with numerical examples of linear and non-linear scalar conservation laws in two dimensions.

Contents

List of Figures

Chapter 1

Introduction

This paper describes an adaptive finite volume scheme for computing solutions to hyperbolic conservation laws on unstructured (triangular) meshes in two dimensions. Solutions to this class of problems have a number of features which distinguish them from other types of partial differential equations (PDEs) — for example elliptic problems. Hyperbolic PDEs often feature shocks and other forms of non-stationary discontinuities in some parts of their domain, whereas other regions are very smooth and easily discretised. This led to the development of adaptive methods for these problems that try to concentrate the computational effort where it is needed most.

One particularly successful approach to this has been developed by Berger and Oliger [4] and works on rectangular cartesian grids. Nevertheless, the importance of unstructured meshes has soared as they are easier to align to boundaries or other features of the problem. But very little work has been done on adaptive methods on unstructured meshes. For example, Ahmadi *et al.* [1] have used an adaptive method for solving the Euler equations, but they only solve for steady-state problems and their "mesh adaption" is closer to a partial remeshing than adaptive refinement. Similarly, Berzins *et al.* [5] have developed a system that incorporates adaptive refinement; they apply "static rezoning" (remeshing in disguise) if their error tolerance is exceeded. Barth and Larson [2] mention an adaptive procedure in passing, but give very little detail as their work is focused on error estimation.

The algorithm closest to the one described herein is most likely the one used by Lou *et al.* [13] for their PYRAMID-package; it is essentially an extension of Berger and Oliger's to triangular / tetrahedral elements combined with efficient partitioning for parallelism. If one were to grade the above approaches to adaptivity from most coarse and static (e.g. complete remeshing every N time-steps) to most fine-grained and dynamic, most of them would be classified as rather static — with Berger and Oliger's [4] (and

consequentially Lou *et al.*'s [13]) somewhere in the middle.

By contrast, the method we describe leans very much towards the dynamic end of the imaginary scale and even modifies the mesh geometry during the computational phase. Thus, the distinction between adaption / refinement and traditional computation is no longer given — we refer to this refinement strategy as "Just-In-Time"[1]. Furthermore, the whole domain is (at least conceptually) covered by a single mesh instead of multiple overlaid meshes at differing resolutions.

One of the central factors which makes our method different from many others is that it is based on regular subdivision of elements instead of splits (for example as described by Rivara [14]). Splitting has the advantage of generating fewer elements than subdivision, but a split generally changes the aspect-ratio of the element and thus requires continually monitoring its anisotropy and taking proper counter-measures should it become too large.

The proposed scheme is general enough to work in conjunction with finite volume methods of almost any type. An implementation of the subdivision strategy described herein has been used to solve both linear and nonlinear problems with high accuracy and efficiency using a traditional slope-limited MUSCL-Hancock scheme [9].

[1]This term might be familiar to some readers from the methodology employed by many Java^{TM} virtual machines.

Chapter 2

Background

2.1 Adaptive Mesh Refinement

2.1.1 Philosophy

Recently, methods incorporating adaptive mesh refinement (AMR) have become recognised as a reliable and efficient means to compute numerical solutions to large-scale problems involving (non-linear) hyperbolic partial differential equations. The main idea behind AMR is to concentrate the computational effort where it is needed most — e.g. using a high resolution near "interesting" features of the solution and a lower resolution for its smooth regions.

This is somewhat analogous to using mesh generation for steady-state problems (most often in conjunction with finite elements). The better the mesh is adapted to the geometry and features of the problem, the more accurate the solution will be. Of course, for non-steady problems the updated solution will reflect a slightly different state and the original mesh might no longer be optimal. AMR remedies this by *adapting* the mesh to reflect the updated solution.

The use of adaptive methods obviously requires different parts of the computational domain to be represented at different resolutions, which in turn drives a need for more complicated data structures to represent them — usually some form of trees instead of simple arrays. These structures need to be dynamic enough to support frequent changes which occur at least every few time-steps.

2.1.2 Implementations

Most implementations of adaptive mesh refinement (e.g. [6] or [18]) are based on the very general approach developed by Berger and Oliger [4]. Their algorithm works on recursively embedded regular cartesian grids, although some restrictions apply as to how this embedding can be done; usually rotated grids with half the step-size are used.

The outline of their algorithm is as follows (see Figure 2.1 for an illustration):

1. Calculate the solution on the current grid.
2. Estimate the error for all points in the current grid and mark the ones that exceed a predefined threshold. Also check that the resolution of the current mesh is still needed.
3. Cluster the flagged points into new grids such that the unnecessarily refined area is kept to a minimum — but without creating too many small grids.
4. Up-sample the data from the current (coarser) grid onto the new grid(s).
5. Recursively apply the procedure to the newly created grids, reducing the time-step to conform to stability requirements.
6. Down-sample from the more accurate finer grid(s) onto the current (coarser) grid.
7. Repeat for the next time-step.

Steps 2–4 are generally only done every N time-steps. Typical values for N would be somewhere between 4 and 8, depending on the problem.

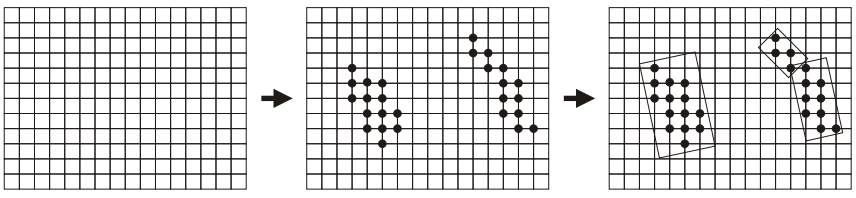

Figure 2.1: The three main steps of Berger and Oliger's [4] algorithm (from left to right): Compute, Flag Errors, Cluster.

2.1.3 Consequences

From the previous description of the algorithm, we can make a out a few problematic consequences of those above steps, which will be relevant in the later comparison of our approach.

2.1.3.1 "Buffer Zones"

Unless error-estimation and the subsequent refinement are carried out *every* time-step on *every* grid, the clustering algorithm needs to leave a "buffer zone" around every refined grid as some solution features — which are only properly resolved on the fine mesh — may move out of the refined region in between those time-steps. The optimal size of this zone depends on the problem and is difficult to estimate *a priori*.

If this "buffer zone" is small, only few points are unnecessarily refined — thus making the computation of the solution on the refined mesh more efficient — but the expensive regridding step has to be done more often to catch any features leaving the high-resolution mesh. If on the other hand the "buffer zone" is large, then the computation can become the bottleneck as too many points are refined speculatively. But that in turn makes it possible to do the regridding less often.

2.1.3.2 Clustering

The process of taking a point cloud and grouping these efficiently — according to some metric — into shapes is called clustering. The point cloud in question consists of the points on the current mesh that have been marked as needing refinement and the shapes are rotated rectangles. The aim is to calculate a set of bounding rectangles whose union encloses all the flagged points while keeping the covered area as small as possible.

Clustering is a provably hard problem, and no optimal algorithm exists. Most of the existing ones are either too slow to be used for this sort of problem or use (fallible) heuristics. Most implementations use a fixed set of rules for pattern recognition of the dominant cases and fall back to a more complicated approach if the previous results are unsatisfactory.

2.1.3.3 Overlaid Grids

In the regions of a grid where refined child-meshes exist, the solution is calculated effectively twice — once on the coarse and once on the fine grid, with the coarse solution later being overwritten with the down-sampled fine solution. Although one of these two computations is not done at the refined

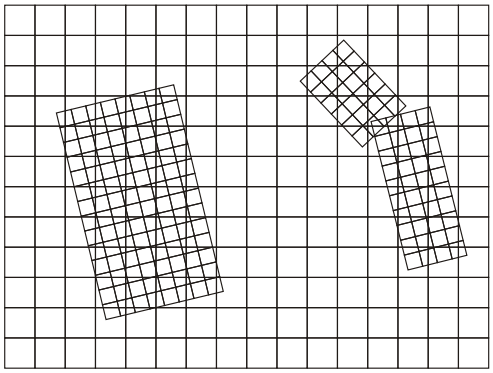

Figure 2.2: Finer meshes overlaid onto a coarse mesh. Correctly up- and down-sampling from one resolution to the other is rather difficult.

resolution it is still a considerable computational effort expended in vain. Because refined grids are allowed to overlap, worst case behaviour can result in major "overcomputation".

The up- and down-sampling process can also become quite difficult — in particular for the commonly used rotated meshes which allow for much more efficient clustering — as can be seen in Figure 2.2. Expensive bilinear or bicubic reconstructions may be necessary in order to achieve this.

Most of these points were already acknowledged by Berger and Oliger [4], but as they are direct consequences of the method, they are difficult or impossible to circumvent in general. In spite of these caveats, their general approach to AMR has been so successful that it has become the *de-facto* standard.

2.2 The Finite Volume Framework

Finite volume methods are widely used for computing numerical solutions of non-linear systems of conservation laws. Depending on one's viewpoint, they can be regarded as either finite difference schemes or subdomain collocation finite element methods. Furthermore, they can easily be defined on any type of mesh, but are most widely used on regular cartesian grids as this allows for an easy extension of 1D-schemes to higher dimensions via operator or dimensional splitting [11].

The following sections gives a brief overview of finite volume methods and is partly based on the material from [17]. For simplicity, u is treated as a scalar quantity but the key points of the discussion hold true for the case of u being a vector quantity as well. The biggest difference lies in how the solution to the Riemann-problem is computed (see for example van Leer [10] or Roe [15]).

2.2.1 Derivation

Consider the following two-dimensional homogeneous conservation law:

$$\frac{\partial}{\partial t}u(x,y,t) + \frac{\partial}{\partial x}f(u(x,y,t)) + \frac{\partial}{\partial y}g(u(x,y,t)) = 0. \tag{2.1}$$

If we now proceed to integrate Equation 2.1 over the discrete volume Ω with boundary $\partial\Omega$ we obtain

$$\int_\Omega \left(\frac{\partial}{\partial t}u + \frac{\partial}{\partial x}f(u) + \frac{\partial}{\partial y}g(u)\right) \mathrm{d}\Omega = 0,$$

which by application of the Divergence Theorem becomes

$$\frac{\partial}{\partial t}\int_\Omega u \,\mathrm{d}\Omega + \oint_{\partial\Omega} \vec{f} \cdot \mathrm{d}\vec{n} = 0,$$

where $\vec{f} = (f,g)^{\mathrm{T}}$ and $\vec{n}$ is the outward normal to $\partial\Omega$. This can be applied to a control volume Ω_j that is then discretised to give

$$\frac{\partial}{\partial t}\left(u_j\, V_{\Omega_j}\right) + \sum_{k=1}^{\substack{\text{edges}\\ \text{of } \partial\Omega_j}} \vec{f}_k^{*} \cdot \vec{n}_k = 0, \tag{2.2}$$

in which u_j is defined to be the average solution value within Ω_j, V_{Ω_j} denotes the area of the control volume Ω_j, $\vec{f}_k^{*}$ is a numerical flux function (across edge k) and $\vec{n}_k$ is again the outward normal to edge k (scaled by the length of the edge).

2.2.2 Control Volumes

Note that Equation 2.2 is independent of the number of edges of the control volume. This makes it possible for different control volumes to have a different number of edges. But for the resulting finite volume method to be conservative, they still have to satisfy a number of properties:

1.
$$\bigcup_{j=0}^{N} \Omega_j = \Omega$$
(i.e. the union of all control volumes covers the whole domain Ω).

2. Adjacent volumes *may* overlap as long as each internal boundary is common to an even number of control volumes (so that the corresponding internal fluxes cancel out).

3. Fluxes along a volume boundary have to be computed independently of the control volume in which they are considered.

As the control volumes need not coincide with the cells of the computational mesh, there are two distinct classes of finite volume methods. On the one hand there are the so called *cell-centred* schemes where the control volumes and mesh cells are one and the same and the u_js are thought of as representative of some point within the cell — usually the centroid, and on the other hand we have the *cell-vertex* schemes where they are not identical and the u_js are associated with mesh nodes.

2.2.3 Numerical Fluxes

For the further discussion, we take the control volumes to coincide with the mesh cells, resulting in a cell-centred scheme. Thus, $\vec{f}^*_k$ is effectively an approximation to the flux through edge k of $\partial\Omega_j$ and depends on the state (i.e. value of the solution variable(s)) on both sides of the edge and (possibly) on its position as well as time. This numerical flux is usually computed as the solution to a standard Riemann-Problem with the two states separated by a discontinuity.

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

$$\begin{aligned} \vec{f}^*_{\mathrm{AB}} \cdot \vec{n}_{\mathrm{AB}} &= -\vec{f}^*_{\mathrm{BA}} \cdot \vec{n}_{\mathrm{BA}} \\ \Leftrightarrow \qquad \vec{f}^*_{\mathrm{AB}} &= \vec{f}^*_{\mathrm{BA}}. \end{aligned} \tag{2.3}$$

If this so-called "telescoping" does not take place, spurious flux contributions will be generated within the domain Ω and thus the total amount of conserved quantity will be altered. Consequently, the only natural choice to evaluate

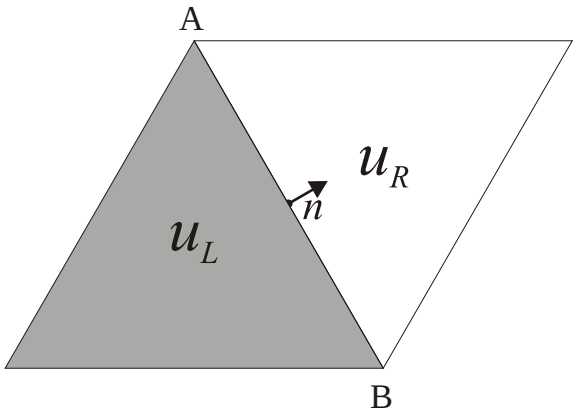

Figure 2.3: Conventions for the flux computation across edge AB (as seen from the shaded element).

the numerical flux is at the mid-point of each edge as that guarantees that Equation 2.3 holds true.

For example, the standard upwind flux — which is the one used by our implementation — across edge AB (as shown in Figure 2.3) takes the form

$$\vec{f}^{*}_{\mathrm{AB}} \cdot \vec{n}_{\mathrm{AB}} = \begin{cases} \vec{f}_{\mathrm{AB}}(u_L) \cdot \vec{n}_{\mathrm{AB}} & \text{if} \quad \vec{\lambda}_{\mathrm{AB}} \cdot \vec{n}_{\mathrm{AB}} \geq 0 \\ \vec{f}_{\mathrm{AB}}(u_R) \cdot \vec{n}_{\mathrm{AB}} & \text{otherwise} \end{cases} \tag{2.4}$$

for the locally frozen wave-speed

$$\vec{\lambda}_{\mathrm{AB}} = \begin{cases} \frac{\vec{f}(u_R) - \vec{f}(u_L)}{u_R - u_L} & \text{if } |u_R - u_L| > \epsilon \\ \frac{\partial \vec{f}}{\partial u} & \text{otherwise} \end{cases} . \tag{2.5}$$

2.2.4 Boundary Conditions and Source Terms

Another strength of finite volume methods is the ease with which boundary conditions can be incorporated. Periodic boundaries can be achieved by transporting the flux leaving a boundary back into the opposite side — for an unstructured mesh this is most easily achieved by modifying the connectivity information of the mesh. No-flux conditions can handled by ignoring those edges when summing the fluxes.

Source terms are slightly more complicated as they cannot simply be accounted for by adding them onto the right-hand side of the equation.

Nonetheless they can be dealt with in exactly the same manner as for non-adaptive methods, see for example van Leer [10] or LeVeque [12] for good discussions of the problem.

2.2.5 Higher-Order Accuracy

Much effort has been invested to achieve better than first-order accuracy with finite volume methods. The first hurdle is Godunov's Theorem, which states that non-oscillatory constant coefficient schemes can be at most first-order accurate. This has been overcome by the introduction of non-linear schemes such as MUSCL, ENO or Flux Corrected Transport (FCT).

One of the more popular ones is van Leer's MUSCL [9] approach, which stands for "Monotonic Upstream-Centered Scheme for Conservation Laws". It belongs to the class of Godunov-type methods, a class of non-oscillatory finite volume schemes that incorporate the (exact or approximate) solution to Riemann's initial-value problem (or a generalisation thereof). Instead of piecewise *constant* states within each cell, MUSCL uses piecewise *linear* ones (*cf.* Figure 2.4) which are carefully constructed from neighbouring states to both maintain conservation and not increase total variation (i.e. do not create over- or undershoots).

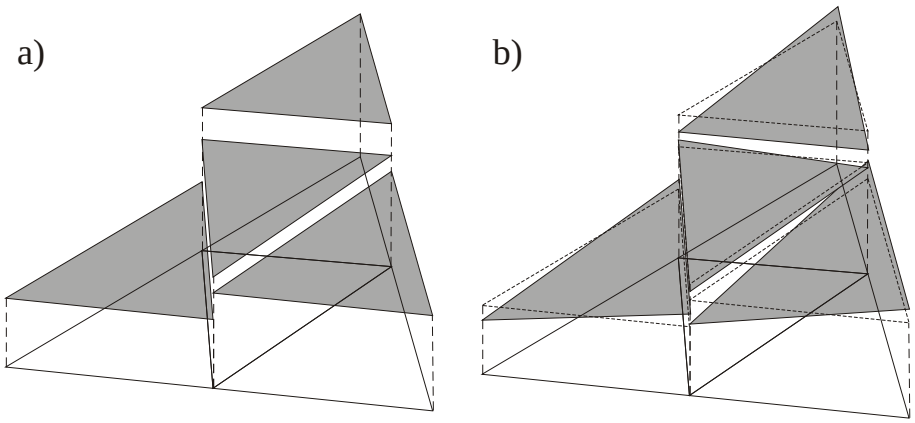

Figure 2.4: a) Constant reconstruction vs b) piecewise linear reconstruction.

This linear reconstruction u' still has to be conservative in the sense that

$$\frac{1}{V_{\Omega_j}} \iint_{\Omega_j} u' \,\mathrm{d}x \,\mathrm{d}y \;=\; u. \tag{2.6}$$

Equation 2.6 is satisfied if u' is of the form

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

for $\vec{r}$ being a vector from the centroid of Ω_j and a gradient operator $\vec{L}$. Most approaches follow Batten *et al.*'s [3] recommendation to construct a gradient plane through three nearby centroids A, B and C with normal vector

$$\vec{n} = (P_\mathrm{A} - P_\mathrm{B}) \times (P_\mathrm{C} - P_\mathrm{B}), \quad \text{with } P_i = \begin{bmatrix} x_i \\ y_i \\ u_i \end{bmatrix}$$

and the subsequent gradient operator

$$\vec{\nabla}(\triangle\mathrm{ABC}) = \begin{cases} \begin{bmatrix} -n_x \,/\, n_u \\ -n_y \,/\, n_u \end{bmatrix} & \text{if } n_u > \epsilon \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \text{otherwise} \end{cases} . \tag{2.7}$$

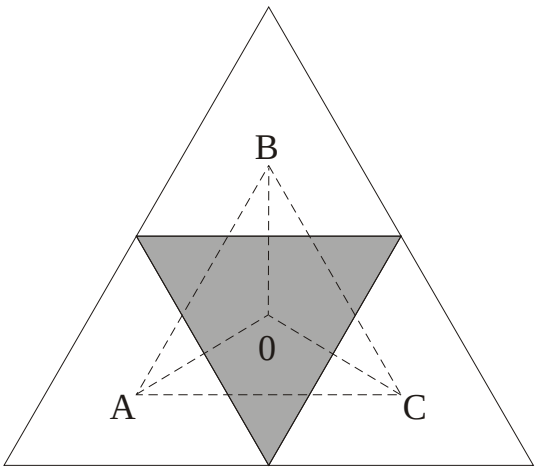

Figure 2.5: Naming Convention for the Limiting Procedure.

The gradient operator defined by Equation 2.7 is not yet limited and as such may exhibit non-physical over- or undershoots at the points where the operator is evaluated — usually at the midpoint of each edge (*cf.* Section

2.2.3). The limiting of the gradient operator therefore plays an important role as it directly influences the character and accuracy of the solution.

We present a short overview of the limiters used in Chapter 4. The labelling used relative to the "current" element 0 is shown in Figure 2.5; u_0 refers to the value in the current cell whereas u_k refers to the value of the element on the other side of edge k. Similarly, $\vec{r}_{0k}$ is the vector from the centroid of 0 to the midpoint of edge k.

2.2.5.1 Limited Central Difference (LCD)

The LCD-limiter is one of the earliest (and still most widely used) limiters in the context of MUSCL-schemes and is the most diffusive of the limiters presented here — but still an improvement over first order schemes. This limiter's advantages lie in its simplicity and speed.

1. Construct the unlimited gradient operator
$$\vec{L} = \vec{\nabla}(\triangle \mathrm{ABC}).$$
2. For each edge k calculate the scalar
$$\alpha^k = \begin{cases} \frac{\max(u_k,u_0)-u_0}{\vec{r}_{0k}\cdot\vec{L}} & \text{if } (u_0 + \vec{r}_{0k}\cdot\vec{L}) > \max(u_k,u_0) \\ \frac{\max(u_k,u_0)-u_0}{\vec{r}_{0k}\cdot\vec{L}} & \text{if } (u_0 + \vec{r}_{0k}\cdot\vec{L}) < \min(u_k,u_0) \\ 1 & \text{otherwise} \end{cases}.$$
3. Set
$$\vec{L}_{\mathrm{LCD}} = \left(\min_{\text{all } k} \alpha^k\right)\vec{L}.$$

2.2.5.2 Maximum Limited Gradient (MLG)

Batten *et al.* [3] introduced the Maximum Limited Gradient operator in 1996 and they have shown that it reduces to Roe's Superbee limiter in one dimension, which is the most compressive limiter that still lies within Sweby's second order TVD[1] region [16]. It is based on computing various gradient operators in an LCD fashion and then retaining the steepest one of them as follows.

1. Compute
$$\vec{L}^0 = \vec{L}_{\mathrm{LCD}}(\triangle \mathrm{ABC}), \qquad \vec{L}^1 = \vec{L}_{\mathrm{LCD}}(\triangle \mathrm{AB0}),$$
$$\vec{L}^2 = \vec{L}_{\mathrm{LCD}}(\triangle \mathrm{A0C}), \qquad \vec{L}^3 = \vec{L}_{\mathrm{LCD}}(\triangle \mathrm{0BC}).$$

[1] Total Variation Diminishing.

2. Set

$$\vec{L}_{\text{MLG}} = \vec{L}^i \quad \text{such that} \quad |\vec{L}^i| = \max_{0 \le k \le 3} |\vec{L}^k|.$$

From this formulation, it is easy to see that the MLG-limiter is slightly more than four times as expensive as the LCD-limiter.

2.2.5.3 Projected LCD (PLCD)

The most recent of the three limiters which have been applied in the context of this paper is Hubbard's [7] Projected LCD-limiter, which relies on the construction of the "Maximum Principle" (MP) region. This region is created from a set of inequalities — precisely one for each edge of the element — around the centroid of the cell. All points lying within this region satify the local maximum principle which in turn guarantees that no over- or undershoots can occur in the linear reconstruction.

1. Construct the unlimited gradient operator

$$\vec{L} = \vec{\nabla}(\triangle \text{ABC}).$$

2. If $\vec{L}$ does *not* need to be limited, set

$$\vec{L}_{\text{PLCD}} = \vec{L}.$$

3. Otherwise construct the MP region defined by

$$\begin{array}{rrcl} & \min(u_k, u_0) & \le \quad u_0 + \vec{r}_{0k} \cdot \vec{L} & \le \quad \max(u_k, u_0) \\ \Leftrightarrow & \min(u_k - u_0, 0) & \le \quad \vec{r}_{0k} \cdot \vec{L} & \le \quad \max(u_k - u_0, 0) \end{array}$$

   for each edge k, and

4. Project $\vec{L}$ onto the closest point of the MP region so that

$$\vec{L}_{\text{PLCD}} = \operatorname*{proj}_{\text{MP}}(\vec{L}).$$

Although the numerical construction of the MP region is not as expensive in terms of operation count, it is certainly more complicated than programming the MLG-limiter, which is nearly trivial if built onto a working implementation of the LCD-limiter.

Chapter 3

Using Subdivision for Refinement

The proposed algorithm is based on an ordinary unstructured low-resolution mesh which we will refer to as the "top-level" or "base" mesh. Such meshes are easily obtained by either traditional mesh generation or it can be a regularly constructed one (e.g. consisting of equilateral or right-angle triangles). The elements in this base mesh will be labelled as belonging to "level 0" (indicating no subdivision). They are then subdivided as necessary subject to the constraint that neighbouring triangles can differ by at most *one* level of subdivision.

3.1 Subdivision Strategy

The subdivision method employed is shown in Figure 3.1a. New vertices are generated at the mid-points of the edges and used in conjunction with the original vertices to create three of the the four new triangles. The fourth one is situated in the centre and uses all three new vertices. This subdivision scheme equi-distributes the original area so that each "child"-element has $\frac{1}{4}$ the area of its parent. An advantage of this subdivision is its preservation of the anisotropy (width to height ratio) of the original triangle. This is important as it makes measures such as Lou *et al.*'s "Mesh Quality Control" [13] unnecessary — as long as the base mesh is well-formed to begin with, for example a Delaunay triangulation.

Nevertheless, subdivisions generate slightly more elements than would have been generated by edge splits [14] which create *two* new triangles for a single refined edge (Figure 3.1b) in contrast to *four* by subdivision. This is offset by the fact that a simple edge split only refines a single edge whereas the

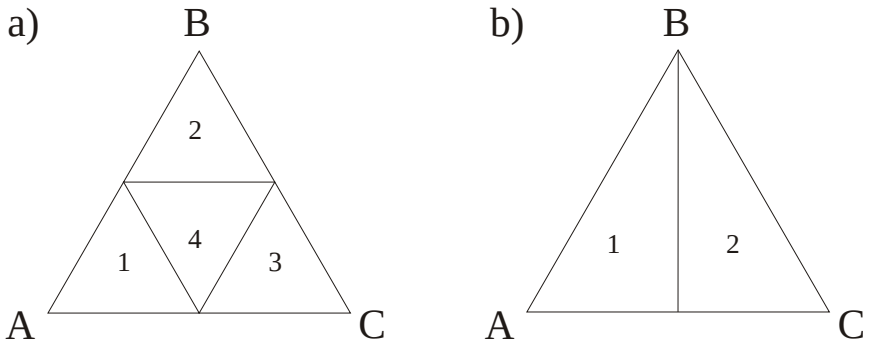

Figure 3.1: a) Subdivision of an element vs a b) split of the edge $\overline{\mathrm{CA}}$.

subdivision refines all three — which reduces the need for further refinement of that triangle.

3.2 Data Structures

The elements are kept in a constrained quad-tree data-structure with as many roots as there are elements in the top-level mesh. A quad-tree is a tree where each node in the tree has up to four children. The constraint for this application is that each node of the tree either has four children or it is a leaf (i.e. it does not have any). These four children are the four elements of the subdivision as described in Section 3.1. A quad-tree — due to its highly structured nature — is much more efficient than a fully dynamic mesh (which generally requires hash-tables or other such mechanisms to locate neighbours, vertices or edges) in terms of both run-time memory usage and speed of access.

Although many entries in the tree cover the same area — for example a triangle and its four children (or their 16 children) — only one set of them is actively used for computations at any one time. That *active* level is always the most refined one and therefore all computations take place with leafs from the tree to give the highest possible accuracy.

Each triangle also contains information about who its neighbours are and at which edge they meet. This is necessary for computing the fluxes through its edges as well as an eventual subdivision or derefinement. It also contains caches for the gradient operators and the half time-steps (*cf.* Equation 3.2) so that their relatively expensive computation only occurs as often as necessary (i.e. once each time-step).

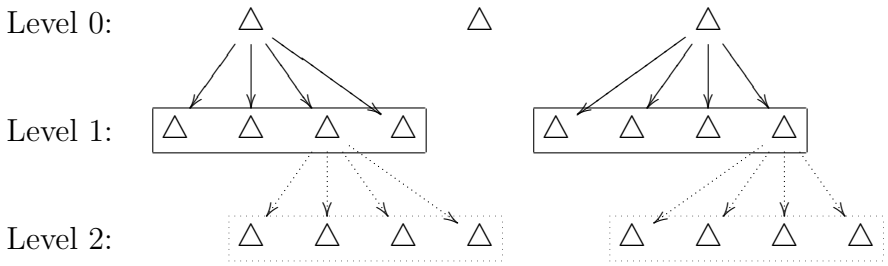

Figure 3.2: Illustration of a (rather sparse) constrained quad-tree.

3.3 Accuracy and Consequences

As can be seen from Equation 2.2 in the discussion of finite volume methods, the fluxes play a very important role for the accurate discretisation of a conservation law. For solving any non-trivial equation, these fluxes depend on the solution variable(s) (e.g. Burger's equation) and / or location (e.g. rotational problems). As these fluxes are always calculated across an edge, the shorter it is the more accurate the flux across it will be — due to the fact that smaller cells give higher accuracy of solution variables and that smaller edges have a higher spatial resolution so that the conserved quantities can be more accurately redistributed to the adjacent cells.

The decision whether an element is to be subdivided or not is therefore closely related to the magnitude of the numerical fluxes through its edges. These fluxes in turn depend largely on the states on either side of the edge. As these fluxes are often only approximate (but conservative) solutions of the Riemann-problem (e.g. [15]) — especially in the case of systems of equations — we have decided to rely on the states as subdivision criterion, i.e. refine if $|u_L - u_R| > \epsilon_r$ (where u_L, u_R are the states on either side of an edge and ϵ_r is a given threshold for refinement) for *any* edge of the element. This approach is also known as a *gradient detector*.

Unfortunately, any useful form of adaptive mesh refinement also needs to concern itself with the *derefinement* of regions, particularly for hyperbolic problems where shocks necessitating refinement can travel through the whole region of interest leaving a massive amount of refined elements in their wake which in turn slows the computation to a crawl. Fortunately, the above choice of subdivision criterion allows for a very closely related test for the derefinement of elements: Derefine if $|u_L - u_R| < \epsilon_d$ (where u_L, u_R are the states on either side of an edge and ϵ_d is a given threshold for derefinement)

for *all* edges of the element.

This serves as a reminder that refinement is a *necessary* step of the algorithm (to maintain accuracy) whereas derefinement is an *optional* step. This becomes an important consideration when dealing with implicit subdivisions in Section 3.4.4.

3.4 The Algorithm

This section describes the steps carried out for an adaptive computation using our subdivision method. This process is governed by three user-defined parameters (in addition to the other problem-dependent data such as the mesh, the top-level time-step ΔT, the particular fluxes used — which in turn describe the equation to be solved, and so on):

1. ϵ_r — The sensitivity of the gradient detector to refinement.
2. ϵ_d — The insensitivity of the gradient detector for derefinement.
3. N — The maximum level of subdivision carried out on the mesh.

Because we do not want to unnecessarily refine and derefine the same elements over and over again, it is a requirement that ϵ_r is *strictly less* than ϵ_d. As a rough guideline $\epsilon_d \approx 2\epsilon_r$.

3.4.1 Set-Up

These tasks are only executed once before the computational loop and as such are not *quite* as sensitive to optimisation as some of the other steps that are carried out thousands of times for each time-step.

3.4.1.1 Top-Level Mesh

The first requirement is a top-level mesh on which the subsequent adaptive mesh refinement can be carried out. This base mesh is never modified (although it "acquires" many children) and its vertices never move. This mesh can be produced using a traditional mesh generation package to take advantage of the geometry of the problem / boundaries or it can be a regular triangle mesh. If no connectivity information (i.e. which triangles are neighbours and at which edges do they meet) is available, it will also be generated at this point from the list of vertices and triangle indices.

The elements of the top-level mesh should also have approximately the same size because all elements within the same level of refinement use the

same time-step. In spite of all adaptivity, the base mesh also has to provide enough initial resolution so that that the gradient detector will be able to properly estimate where the mesh — due to the initial conditions — needs subdivision.

3.4.1.2 Initial Conditions

The next step is to populate this mesh with initial conditions. These are given as a function of x and y in the form $u(x, y, 0)$. As the values within each cell are area / volume averages, they cannot simply be sampled at the centroid of the cell. To compute these volume averages analytically, one would have calculate the area under an arbitrary triangular element placed on top of the initial conditions. This is far from impossible but very much dependent on the shape and storage of the initial data.

To facilitate conditions of a more general nature, we have adopted a stochastic approach which samples the initial conditions at a fixed number of (pseudo-)random points within the triangle and then averages these samples to give an estimate of the proper average. The points in the triangle are generated in barycentric coordinates to guarantee a normal distribution as presented by Turk [19] while the pseudo-random number sequences are computed with Knuth's method [8] for floating-point numbers.

This approach could be extended to use a variable number of samples depending on the distribution of the initial data, but as this process is only carried out once during the set-up stage, the subsequent savings are minimal.

3.4.1.3 Initial Mesh Adaption

The process of initial mesh adaption "teaches" the top-level mesh about the shape (or more accurately the gradients) of the initial data. It uses exactly the same gradient detector and subdivision strategy as normally employed during "regular" time-stepping. This ensures that the main computation code is presented with a valid and representative mesh to start with. The major difference though is that upon subdivision of a triangle its children are initialised with newly sampled initial conditions — thus representing them with higher accuracy as shown in Figure 3.3.

The gradient detector is applied to each cell of the base mesh and — if ϵ_r is exceeded — subdivides the cell in question to yield four new triangles. These are then given the corrected initial conditions (as the cell average is now over a smaller area / volume). Then the process is recursively applied to the new triangles until either the maximum level of subdivision N has been reached or until the all gradients are smaller than ϵ_r.

During the subdivision process the proper nesting has to be enforced so that neighbours in the computational mesh will only differ by *at most* one level of subdivision. This necessitates what we call implicit subdivisions (see Section 3.4.4) which come from nesting rather than computational accuracy requirements (i.e. the gradient detector).

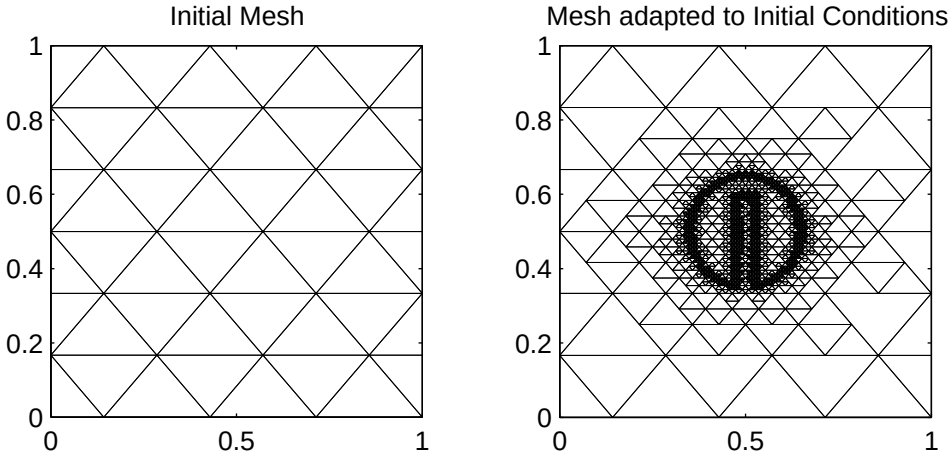

Figure 3.3: A low-resolution base mesh and the same mesh adapted to the initial condition of the rotating slotted cylinder problem with 5 levels of subdivision.

3.4.2 Advancing Time

The order of operations is inherently more important for any algorithm that operates on multiple scales than it is for their uniform brethren. This section deals with this problem and others that need to be overcome in order to make progress from one time-step to the next.

3.4.2.1 Time Stepping

The time-derivative has been left undiscretised in Equation 2.2 and can be discretised using any of the standard approaches — for example Euler's method. The MUSCL-scheme with linear reconstruction gives second-order accuracy in space so that it is only natural to look for second-order time accuracy as well. One such second-order method is the following TVD Runge-

Kutta time-stepping defined by

$$\begin{aligned}
\overline{u}_0 &= u_0^n - \frac{\Delta t}{V_{\Omega_0}} \sum_{k=1}^{\substack{\text{edges} \\ \text{of } \partial\Omega_0}} \vec{f}_k^*(u_0^n + \vec{r}_{0k} \cdot \vec{L}_0^n, u_k^n + \vec{r}_{k0} \cdot \vec{L}_k^n) \cdot \vec{n}_k \\
u_0^{n+1} &= \frac{u_0^n}{2} + \frac{\overline{u}_0}{2} - \frac{\Delta t}{2V_{\Omega_0}} \sum_{k=1}^{\substack{\text{edges} \\ \text{of } \partial\Omega_0}} \vec{f}_k^*(\overline{u}_0 + \vec{r}_{0k} \cdot \overline{\vec{L}}_0, \overline{u}_k + \vec{r}_{k0} \cdot \overline{\vec{L}}_k) \cdot \vec{n}_k,
\end{aligned} \tag{3.1}$$

which is rather expensive as it computes the numerical fluxes — and thus the solution to the Riemann-problem — twice.

An approximation to Equation 3.1 has been formulated by Hancock and is described by van Leer in [10]. It is of the form

$$\begin{aligned}
u_0^{n+1/2} &= u_0^n - \frac{\Delta t}{2V_{\Omega_0}} \sum_{k=1}^{\substack{\text{edges} \\ \text{of } \partial\Omega_0}} \vec{f}(u_0^n + \vec{r}_{0k} \cdot \vec{L}_0^n) \cdot \vec{n}_k \\
u_0^{n+1} &= u_0^n - \frac{\Delta t}{V_{\Omega_0}} \sum_{k=1}^{\substack{\text{edges} \\ \text{of } \partial\Omega_0}} \vec{f}_k^*(u_0^{n+1/2} + \vec{r}_{0k} \cdot \vec{L}_0^n, u_k^{n+1/2} + \vec{r}_{k0} \cdot \vec{L}_k^n) \cdot \vec{n}_k
\end{aligned} \tag{3.2}$$

and has the major advantage of solving only a single Riemann-problem as well as constructing half as many gradient operators. It can be viewed as extrapolating the cell-interface values half a time-step and then using those as the arguments to the flux-computation. On the other hand, the scheme is now only approximately TVD and may thus allow small-scale under- or overshoots to appear in the solution.

3.4.2.2 Stability and Temporal Refinement

Batten *et al.* provide a CFL-like stability criterion for any of the limiters satisfying Equation 2.6 on triangular grids which restricts the time-step within each cell to

$$\Delta t \leq \frac{V}{3 \max_k |\vec{\lambda} \cdot \vec{n}_k|} \tag{3.3}$$

for $\vec{\lambda}$ as defined in Equation 2.5.

This means that smaller cells require a smaller time-step than larger ones. Therefore the solution in smaller cells needs to be computed more often as more steps are needed. But the solution procedure needs information about the states in the neighbouring elements which may or may not have a different

resolution — and thus might be updated at a different frequency than the current cell.

To alleviate these complications, all top-level triangles are assumed to be of similar size. If this is the case, then the subdivision of any of these base elements will produce elements of roughly the same area. By induction this is true for elements at *any* level of refinement. Therefore all elements which belong to the same level of subdivision are treated as a unit and use the same time-step (*cf.* Figure 3.4). Conceptually, all cells of a subdivision level are updated in parallel and are therefore prone to the usual parallelisation / partitioning strategies (e.g. [13]).

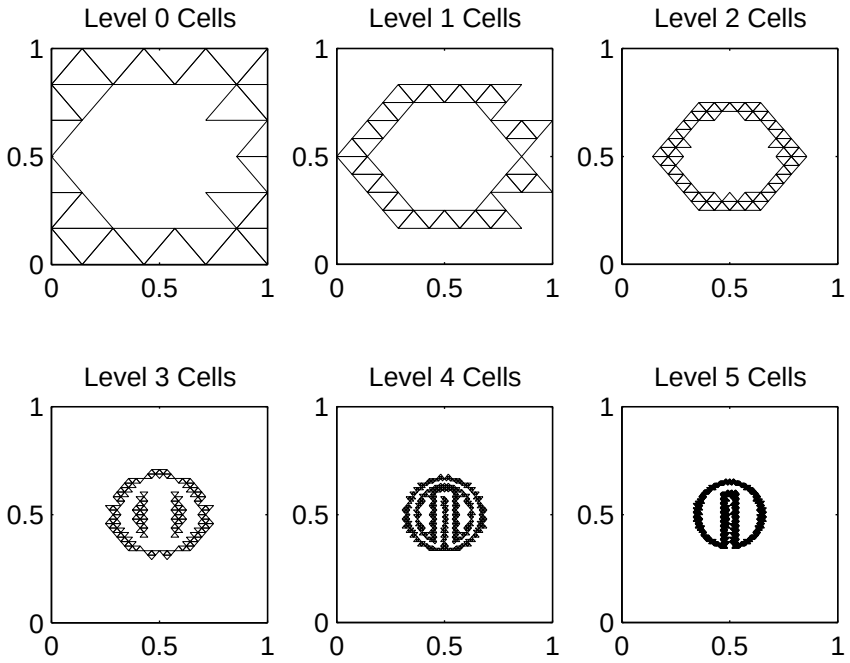

Figure 3.4: The 5 different levels of subdivision from Figure 3.3. Note that their union covers the region exactly once (i.e. no overdraw) and that each level's elements need not be contiguous (e.g. levels 3 & 4).

Each subdivided cell has $\frac{1}{4}$ the area of its parent and all the cells' edges are (approximately) $\frac{1}{2}$ the original length. Substituting these into Equation 3.3 at some level M gives

$$\Delta t(M+1) \leq \frac{\frac{1}{4}V}{3\max_k |\vec{\lambda} \cdot \left(\frac{1}{2}\vec{n}_k\right)|} = \frac{1}{2}\left(\frac{V}{3\max_k |\vec{\lambda} \cdot \vec{n}_k|}\right) = \frac{1}{2}\Delta t(M).$$

Thus we need to (at least) halve the time-step for each additional level of refinement, i.e.

$$\Delta t(M) \;=\; 2^{-M}\Delta T,$$

where ΔT is the time-step used on the top-level mesh.

The formulation of Equation 3.2 implies that for the update of a particular cell one needs to access the states in the neighbouring cells. These neighbouring cells — due to our nesting requirement — may be of a higher (by one level), lower (again by one level) or of the same resolution. The time-stepping of each level therefore has to be done with great care so that the neighbouring elements from an adjacent level are also evolved to the proper point in time. We refer to this as "lock-step" time-stepping and its pyramid-structure is depicted in Figure 3.5.

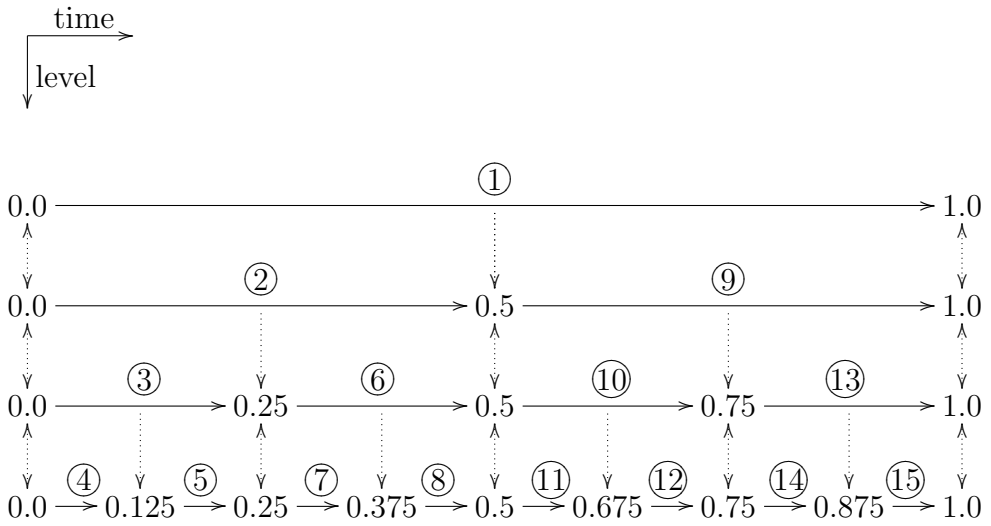

Figure 3.5: Lock-step time-stepping procedure with 3 levels of refinement. Dotted arrows indicate which values are referenced from other levels. Circled labels show the order of computations. Note that coarser levels are always computed *before* their finer counterparts.

3.4.3 Solution Update

The update-step uses the aforementioned MUSCL-Hancock finite volume scheme to compute the next state for all triangles of a particular level of

subdivision. We keep a list for each level of refinement that contains the triangles of that level of subdivision. This allows us to efficiently iterate over all the elements in a level without having to traverse the whole quad-tree data structure. These lists are incrementally updated to reflect promotions across levels as more and more elements are subdivided. The only time the lists are reconstructed from scratch is after a derefinement operation has been completed.

The only obstacle to the direct application of Equation 3.2 is the possibly different resolution of the neighbours used for the numerical fluxes. The three possibilities are depicted schematically in Figure 3.6.

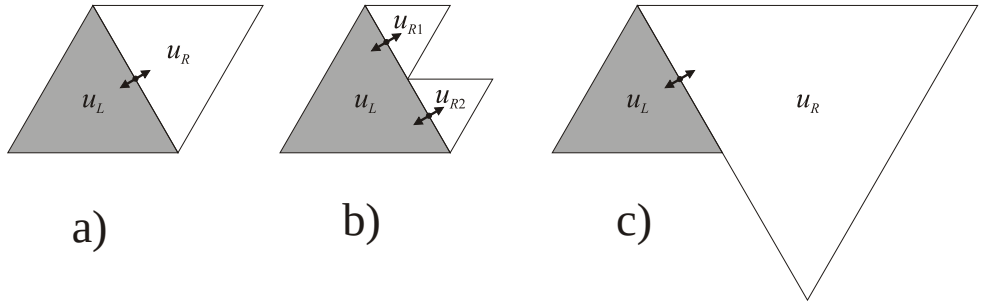

Figure 3.6: Flux Computation for a) a neighbour of the same resolution, b) neighbours of higher resolution and c) a neighbour of lower resolution. No other cases can occur because of the nesting requirement which does not allow the level of subdivision of neighbours to differ by more than one.

3.4.3.1 Edge Fluxes

a) This is the only trivial case as it is identical to the application of a finite volume scheme on a regular unstructured mesh. Therefore one can directly compute the flux through edge k of element 0 as

$$\vec{f}_k^*(u_0^{n+1/2} + \vec{r}_{0k} \cdot \vec{L}_0^n, u_k^{n+1/2} + \vec{r}_{k0} \cdot \vec{L}_k^n) \cdot \vec{n}_k. \tag{3.4}$$

b) The second case is treated as if edge k were two edges, k_1 and k_2, with half the original length each. In Figure 3.6b, k_1 would separate states u_L and u_{R1} whereas k_2 would separate u_L and u_{R2}. This of course makes changes to the limiting procedure necessary as the gradient operator L_0 is not evaluated at r_{0k} (the midpoint of edge k) anymore but rather

at r_{0k_1} and r_{0k_2}. The numerical flux can then be written as

$$\begin{aligned} &\vec{f}^*_{k_1}(u_0^{n+1/2} + \vec{r}_{0k_1} \cdot \vec{L}_0^n, u_k^{n+1/2} + \vec{r}_{k_10} \cdot \vec{L}_{k_1}^n) \cdot \vec{n}_{k_1} \\ &+ \vec{f}^*_{k_2}(u_0^{n+1/2} + \vec{r}_{0k_2} \cdot \vec{L}_0^n, u_k^{n+1/2} + \vec{r}_{k_20} \cdot \vec{L}_{k_2}^n) \cdot \vec{n}_{k_2}. \end{aligned} \tag{3.5}$$

Because these fluxes are written in terms of the "new" normal vectors n_{k_1} and n_{k_2}, no averaging has to be done because the normal vectors are scaled by the length of the respective interface.

c) This situation is in some sense the inverse of case b) and as such, special care has to be taken while constructing and limiting the gradient operator L_k because it is not evaluated at the midpoint of the *whole* edge of the coarse neighbour — although it still is the midpoint of edge k of the current element.

Another difficulty arises with cells whose coarser neighbours are evolved too far ahead because of their coarser time-steps (e.g. steps 5, 6, 8, 9, 12, 13 and 15 in Figure 3.5). In this situation, we approximate the coarse state at the current time by averaging its previous and its next state. The next state is guaranteed to have been updated already as coarser levels are always computed before the finer ones.

Consequently, the flux is identical to Equation 3.4 if the coarse cell is at the correct time; otherwise it is

$$\begin{aligned} &\vec{f}^*_k(u_0^{n+1/2} + \vec{r}_{0k} \cdot \vec{L}_0^n, \tilde{u}_k) \cdot \vec{n}_k \\ \text{for } \tilde{u}_k \;&=\; \frac{1}{2}\left(\left[u_k^{n+1/2} + \vec{r}_{k0} \cdot \vec{L}_k^n\right] + \left[u_k^{n-1/2} + \vec{r}_{k0} \cdot \vec{L}_k^{n-1}\right]\right). \end{aligned} \tag{3.6}$$

The gradient detector — controlled by ϵ_r — is applied to the reconstructed states (i.e. the arguments to f_k^*) at each edge to determine whether further refinement is needed — as long as the maximum refinement level N has not been reached. This process can naturally be done while the reconstruction is carried out, so that the expensive Riemann-problem solutions for the flux-computations need only be found if the cell is not to be refined. If the detector decides that a subdivision is necessary, then the element is subdivided, the level-lists are updated (i.e. the triangle is removed from the list of the current level and its children are added to the list of the next-finer level's elements) and the computation proceeds with the next element in the current level.

We refer to this subdivision strategy as "Just-In-Time" for the obvious reason that it takes place just before the additional accuracy is deemed to be necessary — in contrast to more traditional approaches which create (high

resolution) "buffer zones" to cope with events until the next remeshing step takes place.

3.4.3.2 Changes to Gradient Operators

As alluded to in the previous section, the gradient operators need to be modified so that the points where the fluxes are computed — which are not restricted to the midpoint of the edges anymore — are properly limited. These changes are two-fold. On the one hand, it is necessary to decide how the unlimited gradient operator is constructed, given that there may be more than three neighbouring states. And on the other hand, we need to make sure that these are properly limited at the points where they are evaluated.

The only difficulty in the construction of the unlimited gradient operator $\vec{\nabla}(\triangle \mathrm{ABC})$ occurs when any of the neighbours have a higher resolution than the current triangle. If that is the case, we currently average all four further-refined triangles belonging to the neighbour to give a representative value at the *current* resolution. Other options, such as only averaging the two finer triangles along edge k, are also possible.

Changing the limiting procedures for the LCD- and the PLCD-limiters is straight-forward as these can easily be defined on polygons with any number of edges. A triangle with a single subdivided edge is simply treated as having four edges on which the limiting is carried out.

The MLG-limiter constructs $\binom{4}{3} = 4$ LCD-limiters for a traditional triangle. For each refined edge this increases until the (unlikely but not impossible) worst-case behaviour of $\binom{7}{3} = 35$ is reached. As that many LCD computations are clearly undesirable, we have resorted to the same averaging procedure for finer neighbours as used in the unlimited gradient computation.

3.4.4 Refinement / Subdivision

Subdivisions are necessary for two distinct reasons: accuracy and nesting. We call subdivisions that arise due to nesting requirements *implicit* subdivisions whereas accuracy-based ones will simply be referred to as *"normal"* subdivisions. In spite of the different reasons for their execution, both of them are carried out in entirely the same fashion.

A subdivision of either class is then achieved in the following way:

1. Check whether any neighbours are of lower resolution and force them to subdivide if this is the case. These implicit subdivisions can recursively cause further (again implicit) subdivisions of *their* neighbours and so on. Nonetheless, this process is guaranteed to terminate.

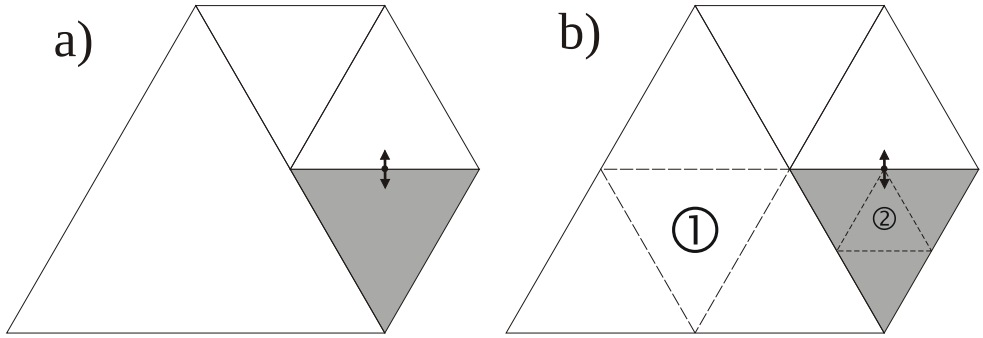

Figure 3.7: a) An accuracy based subdivision becomes necessary for the current (shaded) triangle at the indicated edge and b) makes an implicit subdivision (1) necessary which has to be completed before the original refinement (2) can take place.

2. Allocate space for the children and generate their vertices.

3. Compute the states in the children. This is done by evaluating the usual gradient operator of the parent at the children's centroids. This ensures proper conservation and retains more information than simply duplicating the constant parent state to all children. As the gradient operator is not necessarily correctly limited at the children's centroids this may cause under- or overshoots to appear. But these violations of the maximum principle are unlikely in practice as the children's centroids are rather close to the parent's centroid and therefore only incur small changes from the constant state.

4. Lastly, compute (in the case of the newly generated children) or update (for all neighbours of the parent) the connectivity information so that all neighbour-pointers now refer to the more finely resolved children instead of their parent.

After this process is completed, the parent effectively becomes dormant as no more computations are carried out on it; it is only needed in the hierarchy to provide information about its children.

It is essential that Step 1 takes place first so that all the implicit subdivisions take place before the real ones. If, for example, the normal subdivision labelled (2) in Figure 3.7 were to be executed first, then the mesh would end up in an invalid state because the refinement levels of two adjacent triangles would differ by *two*.

3.4.5 Derefinement

The derefinement process undoes all the refinement that is no longer necessary — for example after a shock front has moved past. If one were to refine only, there would be a large amount of highly refined triangles in otherwise bland regions. These small triangles are also the most computationally expensive ones because they have a strong restriction on their time-step for stability as given by Equation 3.3.

In a similar fashion to the traditional approaches to AMR, we need to find the right frequency of derefinement that balances its cost against the gain in computational efficiency because of the reduced number of elements. Seemingly this is no better than the semi-arbitrary frequency of remeshing used in those algorithms. On second look, it *is* an improvement because it is a much cheaper process than a remeshing operation. For the results in Chapter 4, derefinement has been carried out once after each top-level time-step.

For consistency, the gradient detector used for derefinement is the same as the one used for refinement — with two small changes. Firstly, it uses a different, larger threshold (ϵ_d instead of ϵ_r) to prevent elements from flipping back and forth incessantly between refinement and derefinement and secondly, it measures the gradients on the *piecewise constant* states and not the reconstructed *linear* ones. If an element is to be derefined — which is used as a synonym for collapsing the four children[1] of a triangle back to itself — then it and its neighbours need to have states that lie very close together. If that is the case, the linear reconstruction across the triangles involved is nearly flat anyway, so that using it will not be of any advantage; it will even be a disadvantage because of the extra computational cost involved.

If an element has determined that its children are no longer needed, they *cannot* automatically be collapsed. This is true because of elements that act as a "bridge" between different levels of refinement; in other words they are needed to satisfy the nesting requirement. Thus the children can only be removed if their neighbours are members of the same (or a lower) level of subdivision as the parent will be joined to them later. In situations where this criterion is not satisfied, the offending triangles can sometimes be derefined themselves. One can therefore draw a parallel between the subdivision of an element — which *forces* neighbours to execute an implicit subdivision in order to fulfil the nesting rules — and its derefinement: The derefinement *asks* its neighbours (if necessary) to collapse their children to be able to remove its own.

[1] These children of course have to be childless.

Chapter 4

Results

To verify the validity of our approach, the proposed adaptive scheme has been implemented in the `C++` programming language, whose object-oriented approach seems to be favoured by many implementations of AMR (for example Hornung and Trangenstein [6]) over the more traditional `FORTRAN`. The code used to obtain these results allows for arbitrary unstructured meshes with no-flux or periodic boundary conditions. The problem specification and the associated fluxes are kept separate from the algorithm in order to facilitate its use for different types of computations.

The computation of errors in the L_1 norm against known solutions is somewhat problematic on unstructured meshes. The approach we have adopted is to interpolate from the irregular AMR mesh to a regularly spaced grid and compute the norm from there. This introduces an additional interpolation error to those estimates; but as the size of the regular mesh increases, the interpolation error tends to 0. On a regular grid with N points we thus compute the L_1 error as

$$\frac{1}{N}\sum_{i,j}|u_{i,j} - u_{\text{exact}}(x_i, y_j)|.$$

4.1 Linear Advection

Problems of this type are rarely solved in practice as the analytic solution is known for most advection profiles. Nevertheless, they are a valuable and well-understood tool used for testing and comparing different numerical schemes.

4.1.1 Estimating Order of Accuracy

A similar set-up to Batten *et al.* [3] has been used to estimate the order of accuracy of our scheme. The estimate is computed by solving

$$u_t + u_x + u_y \;=\; 0$$

(i.e. linear advection with wave-speed $(1,1)^{\mathrm{T}}$) with the initial data

$$u(x,y,0) \;=\; \sin(2\pi x)\sin(2\pi y).$$

on a mesh consisting of right-angle triangles on the region $[0,1]\times[0,1]$ with different levels of refinement and then comparing the respective L_1-errors.

The given problem is not well-suited to adaptive computation *at all*. Most — if not all — of the region is refined because of the smoothly changing nature of the initial data. Furthermore, if the thresholds are chosen inappropriately, many triangles will incessantly refine and derefine as their gradients fluctuate around the thresholds. Therefore an adaptive method might spent much time on its "adaptivity" without any computational benefit, making the fixed mesh computation a faster alternative.

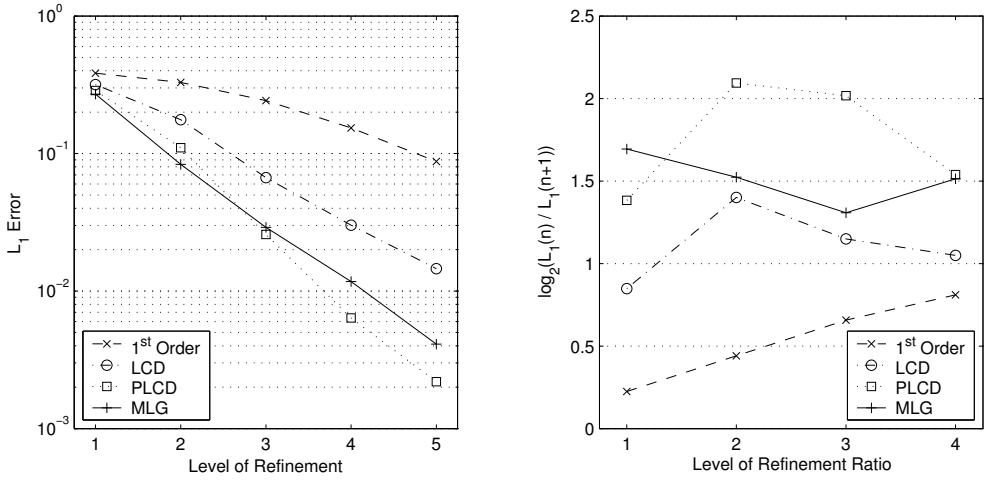

Figure 4.1: The L_1 error and the resulting estimate for the order of accuracy for different types of limiters.

It is interesting to note that the PLCD-limiter actually becomes more accurate than the MLG-limiter for high levels of subdivision; this could be attributed to the "over-compression" of the MLG-limiter and was also observed on an equilateral triangle mesh. One would not expect "real" (in contrast to theoretical) second order accuracy in normal applications, though.

The contour plots in Figure 4.2 and 4.3 support the L_1-results. Both the MLG- and the LCD-limited solutions exhibit grid based distortion, although it is far less pronounced with the MLG-limiter. The PLCD-limiter does very well — the peaks have nearly the same magnitude as the highly compressive MLG-scheme and no distortion is evident. This matches the observations made by Batten *et al.* [3] and Hubbard [7].

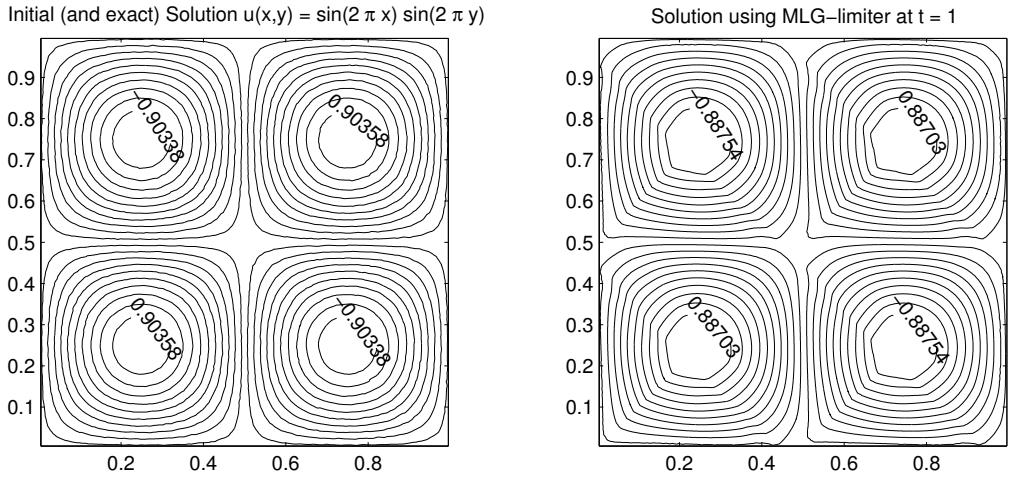

Figure 4.2: Contours of the exact (left) double sine wave function and the MLG-solution. The MLG-solution shows a directional bias which leads to a subtle distortion in the lower-left quadrants vs the upper-right ones.

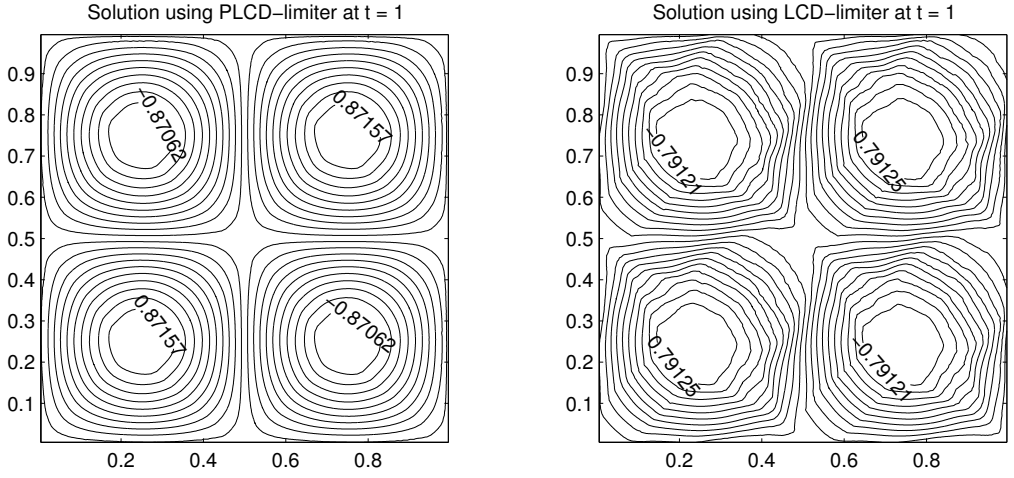

Figure 4.3: Contours of the PLCD-solution (left) and the LCD-solution. Note the smooth profile of the PLCD-limiter in contrast to the heavily distorted LCD-solution.

4.1.2 Rotating Slotted Cylinder

Zalesak [21] first presented the slotted cylinder in 1979 and it is regarded as one of the hardest benchmarks for advection schemes. The version used here is scaled and shifted but otherwise identical. The problem is to solve

$$u_t + \left(y - \frac{1}{2}\right) u_x - \left(x - \frac{1}{2}\right) u_y = 0 \tag{4.1}$$

with initial condition

$$u(x, y, 0) = \begin{cases} 1 & \text{if } r > 0.15 \vee (|x - 0.5| \leq 0.025 \wedge y - 0.5 \leq 0.1) \\ 0 & \text{otherwise} \end{cases}$$

for $r = \sqrt{(x - 0.5)^2 + (y - 0.5)^2}$. Equation 4.1 describes a clock-wise rotation of the whole region about the point $(0.5, 0.5)^{\mathrm{T}}$. A whole revolution of the region is completed at $t = \pi$ so that the solution is periodic with period π. This also means that the exact solution at $t = \pi$ is equal to the initial data.

The problem is solved in the region $[0, 1] \times [0, 1]$ with no-flux boundary conditions on a top-level mesh with 48 equilateral triangles as seen in Figure 3.3. The time-step used is $\Delta t = 0.05$ which upon initial observation is outside the stability-region defined by Equation 3.3. This is no cause for concern though as the advection-vector can safely be set to $\vec{0}$ for $r \geq 0.3$ where $u(x, y, t)$ is identically zero. This extends the stable region to $\Delta t \leq 0.063927$. For the results in Figure 4.4 a refinement threshold of $\epsilon_r = 0.025$ and a derefinement limit of $\epsilon_d = 0.05$ were used in conjunction with five levels of refinement.

4.1.2.1 Quality of Solution

As one can see from the L_1-errors in Figure 4.4 the limiters play a crucial role: All of them[1] limit correctly across the adaptive mesh; no oscillations are present in the solution, which would certainly not be the case with more traditional second order methods used on regular grids such as Lax-Wendroff, Warming & Beam or — to a lesser extent — Fromm. If the limiting is removed for any of the linear reconstructions then the solution grows unboundedly within a *single* top-level time-step due to the sharp discontinuities in the initial data.

[1] Except for the first-order scheme which does not need any limiting due the fact that no reconstruction / extrapolation is being done.

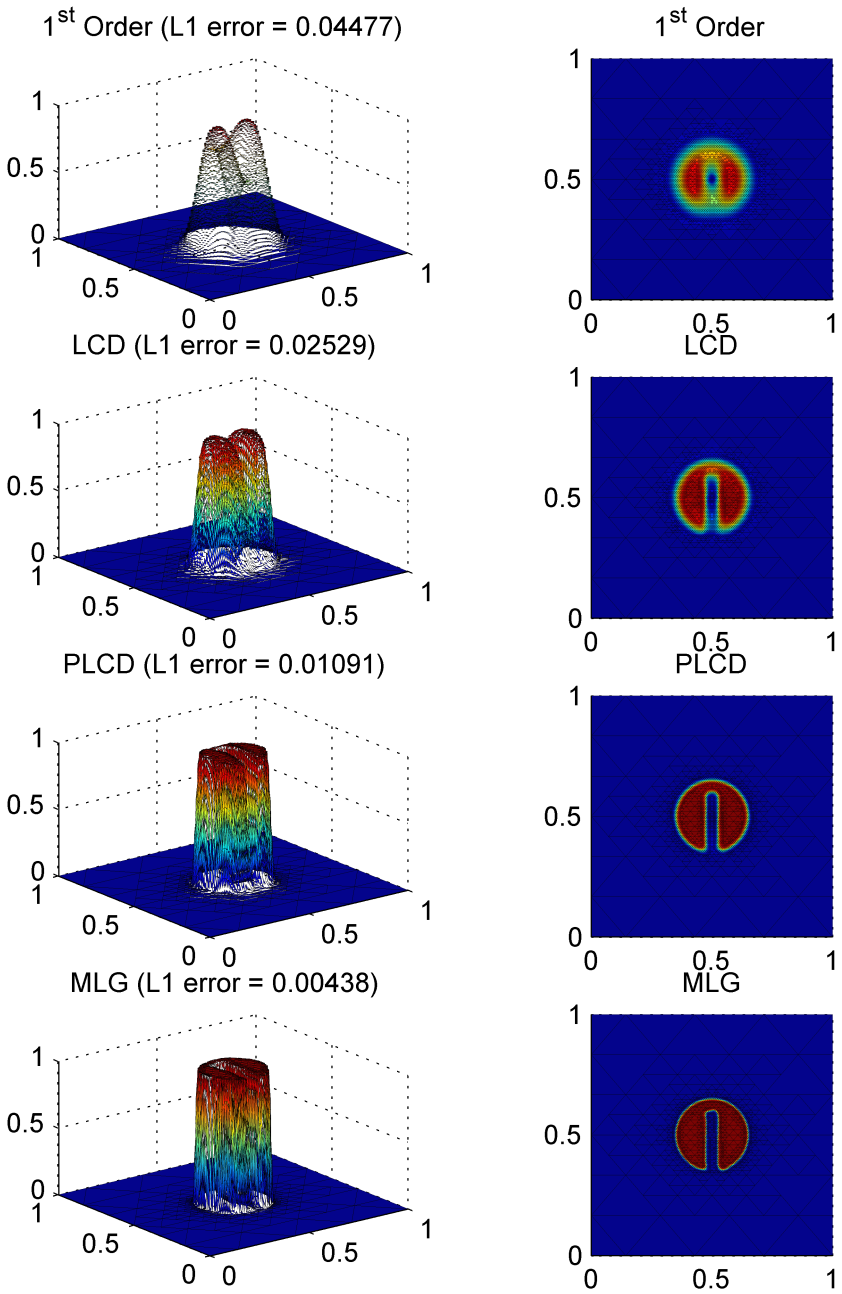

Figure 4.4: Graphs of the "Rotating Slotted Cylinder" problem at $t = \pi$ for different limiters ($\Delta t = 0.05$) with 5 levels of refinement.

From the first-order scheme to the MLG-operator each successive limiter *halves* the error in this example, resulting in a ten times more accurate solution with the MLG limiter than the first-order scheme. This stresses the importance of a good reconstruction- / limiting-procedure over "raw" resolution. Nevertheless, the MLG-limiter seems to introduce a very slight distortion to the top-right of the slot and to the bottom-right "tip". This minuscule distortion is also evident in higher-resolution computations with further subdivisions.

The Projected LCD-slope is slightly more diffusive than the Maximum Limited Gradient — the edges of the "discontinuity" are now spread out over four of the most-refined triangles in contrast to three on the MLG-limited computation. But the projected limiter is less susceptible to grid based distortion as the solution looks perfectly symmetric.

The widely used Limited Central Difference-approach is even more diffusive but it manages preserves most of the original shape — although the top of the cylinder is now smoothly rounded instead of flat and the slot is beginning to close (or at least rise).

The solution produced by the first-order scheme is nearly unrecognisable. It is not immediately apparent which side of the cylinder had the slot embedded in it and which had not. Many triangles are refined due to the overly diffusive nature of the method.

4.1.2.2 Impact of Limiters

But accuracy is not the only reason for which it is important to choose a good reconstruction- / limiting-scheme. The compressiveness of the limiter directly influences the cost of the adaptive computation. If the scheme used is too dispersive, the discontinuities get spread out over a larger area which then needs more refined triangles.

This is very well illustrated by Figure 4.5 which depicts the cost of each top-level time-step (similar to the measure defined in Section 4.2.1 but only for a *single* top-level time-step instead of the whole computation). The cost for the Maximum Limited Gradient-limiter stays nearly constant during the whole computation, which means that the information in the solution is well retained and that the derefinement removes about as many "old" elements as new ones are being refined.

The PLCD-solution is not quite as efficient as it causes slightly more elements to be subdivided at a an additional cost of about 28% for each time-step near the end of the computation. The number of elements only seems to increase at a very low rate however.

Both the first-order upwind update and the LCD-scheme are not a very

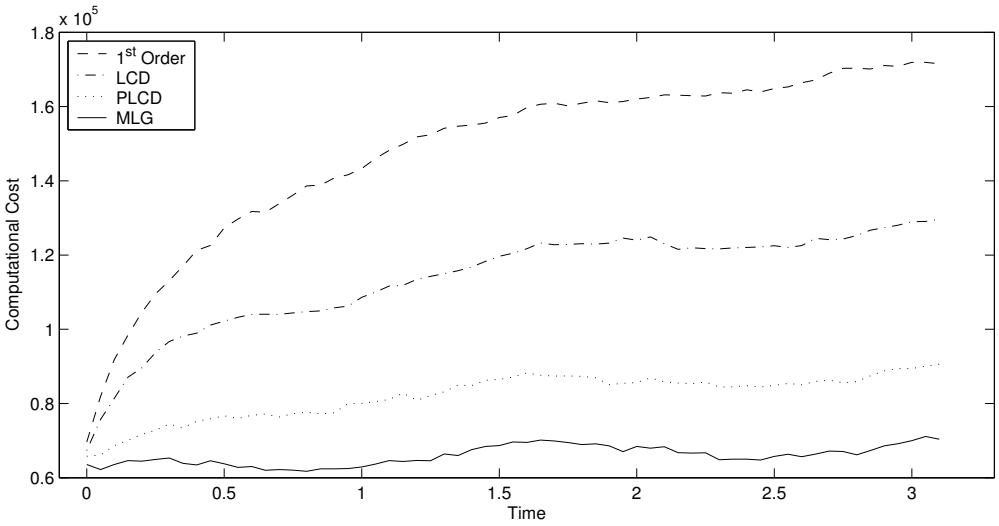

Figure 4.5: The workload / cost per top-level time-step during the 5-level solution of the rotating slotted cylinder.

good choice for adaptive computations in general and problems that contain discontinuities in particular. The use of the first-order method makes the whole computation (i.e. the area under the respective graph) more than twice as expensive as with the most compressive limiter. Nonetheless, they are an important foundation on which to formulate more accurate methods — the MLG-operator for example uses the steepest of four different LCD-gradients.

Taken together, this implies that — particularly in the context of adaptive methods — a more expensive but less diffusive limiter may well offset the higher cost of its construction during a whole computation. If one takes the cost of the computation of the Limited Central Difference as a base-line, then the MLG-operator is slightly more than four times as expensive to construct while the PLCD-limiter needs about two to three times as much. We have found it essential to cache limited gradient-operators for each triangle to avoid their recomputation as they are needed at least four times — once for the computation in the current triangle and then once for each neighbour's computation.

4.1.2.3 Cost of Refinement

One major advantage of adaptive mesh refinement is that it allows for either much larger simulations to be run than would have been possible with regular

meshes or solve existing problems with higher accuracy. It is an important feature of good adaptive methods that they are not only more efficient with computational resources but also with memory storage.

To show the trends that adaptive computing exhibits while refining solutions further and further, the methodology from Section 4.1.1 is applied to the slotted cylinder problem on the same equilateral triangle mesh with 48 base-triangles as used in Section 4.1.2 and repeatedly solved for different levels of refinement. This time however, the cost is investigated as well as the accuracy.

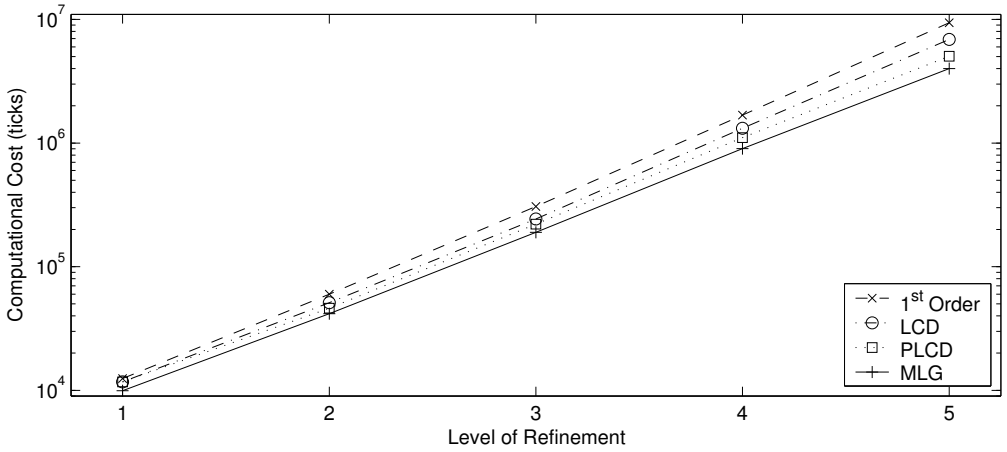

Figure 4.6: The cost of each additional level of refinement for different limiters on the rotating slotted cylinder.

The adaptive scheme needs between 4.2 and 5.6 times more ticks to halve the mesh-spacing (i.e. increasing the maximum refinement by a level). Unsurprisingly, the Maximum Limited Gradient fares best with an average increase of about 4.5 per level and the first-order scheme at the bottom end of the scale (figuratively speaking) needs more resources with an approximate increase of 5.25. The other two limiters are spread in-between in the by now familiar order: The Projected LCD scheme becomes 4.6 times as expensive for each refinement — only a 0.1 difference to the MLG growth-rate — and the LCD-limiter consumes 4.9 times as many ticks.

The equivalent of increasing the level of refinement by one for our method on a non-adaptive mesh would be to halve the mesh-spacing of its elements. This in turn would result in *four* times as many elements at *half* the time-step, which would make the finer computation *eight* times as expensive in terms of tick-count.

Level	1st Order	LCD	PLCD	MLG
1	0.07699	0.06809	0.04460	0.03611
2	0.07021	0.05314	0.03695	0.02902
3	0.06534	0.04739	0.02673	0.01717
4	0.05564	0.03548	0.01699	0.00875
5	0.04477	0.02529	0.01091	0.00438

Table 4.1: L_1 errors for different levels of refinement and limiters on the "Rotating Slotted Cylinder".

The convergence-rates implied by the L_1-errors given in Table 4.1 look rather worrying compared to the results from Section 4.1.1. The first-order method barely manages to achieve $O(0.3)$ on the highest refinements while the MLG-operator approaches $O(1)$ very closely. The LCD-limiter would deserve the title of a not quite "half-order" method for this particular application and the Projected LCD-gradient does a bit better at $O(0.65)$ for the finer meshes but is still quite a bit worse than the real first-order. For this problem the wider choice of gradient operators available to the Maximum Limited Gradient seems to be more useful than the projection of a single one onto the MP region.

It may come as a relief that the theoretically attainable maximum order of accuracy for this type of problem is $O(1)$. This is due to the fact that the initial data consists only of constant states separated by discontinuities. Any higher order scheme relies on smoothness in the solution for its increased accuracy and reverts to *at most* first-order near discontinuities.

Once more, the "rift" between the more recent and compressive limiters (MLG and PLCD) and the older schemes on which they are based (LCD and first-order upwind) becomes apparent.

4.2 Comparison to a Fixed Mesh

To construct a reasonably fair comparison to a fixed mesh computation, we have defined a cost function for the rotating slotted cylinder problem with the MLG-limiter. The unit in which this cost is given is the computation carried out for a single time-step on a single element of the mesh — which we have called a "tick". Therefore refined cells use more ticks than top-level ones as they need more time-steps to advance the same amount of time.

First, an AMR computation is carried out which keeps track of its cost. After that, we construct an equally expensive fixed mesh and compute the

solution on there. The adaptive computation has to be used as the "baseline-cost" to compare against because it is hard to predict the cost *a priori* due to the variable number of cells.

4.2.1 Estimating Cost

The cost of the AMR computation is defined to be

$$C_{\text{AMR}} = \left\lceil \frac{T}{\Delta t} \right\rceil \cdot \sum_{\text{all tris}} 2^M$$

where T is the final time and M represents the level of subdivision of each triangle. For this particular example, $T = \pi$ and $\Delta t = 0.063927$ so that $C_{\text{AMR}} = 3332216$ ticks with five levels of subdivision.

We now seek to construct a fixed mesh — consisting of equilateral triangles — that incurs a similar cost to compare the respective results. The cost on the fixed mesh for a particular number of elements N_e is

$$C_{\text{FXD}} = \left\lceil \frac{T}{\Delta t} \right\rceil \cdot N_e. \tag{4.2}$$

But Δt depends on N_e for stability reasons. From before we know that

$$\Delta t = \frac{V_\triangle}{3 \max |\vec{n} \cdot \vec{\lambda}|} = \frac{V_\Omega}{N_e 3 \max |\vec{n} \cdot \vec{\lambda}|} \tag{4.3}$$

where V_Ω is the area of the whole computational region Ω. Now it is necessary to estimate

$$\begin{aligned} \max |\vec{n} \cdot \vec{\lambda}| &\leq \max(|n_x||\lambda_x| + |n_y||\lambda_y|) \\ &\leq \max(|n_x|, |n_y|)(\max |\lambda_x| + \max |\lambda_y|). \end{aligned} \tag{4.4}$$

The length of any triangle edge in the mesh, say a, can be used to approximate $\max(|n_x|, |n_y|)$. Then it possible to calculate a from N_e via

$$V_\triangle = a^2 \sin\gamma \quad \Leftrightarrow a = \sqrt{V_\triangle \frac{2}{\sqrt{3}}} = \sqrt{\frac{2V_\Omega}{\sqrt{3}N_e}}. \tag{4.5}$$

Finally, combining Equations 4.2–4.5 yields the cost-estimate

$$C_{\text{FXD}} \leq \frac{1}{V_\Omega} \left(3TN_e^2 \sqrt{\frac{2V_\Omega}{\sqrt{3}N_e}} (\max |\lambda_x| + \max |\lambda_y|) \right). \tag{4.6}$$

4.2.2 Resulting Errors

Using Equation 4.6 to estimate an equal cost fixed mesh gives $N_e \approx 6700$. This number of elements has a cost of 4909548 ticks due to the various approximations used. The actual number of elements has consequently been fixed at $N_e = 5376$ for which $C_{\text{FXD}} = 3404863$ ticks with a time-step of $\Delta t = 0.00496$.

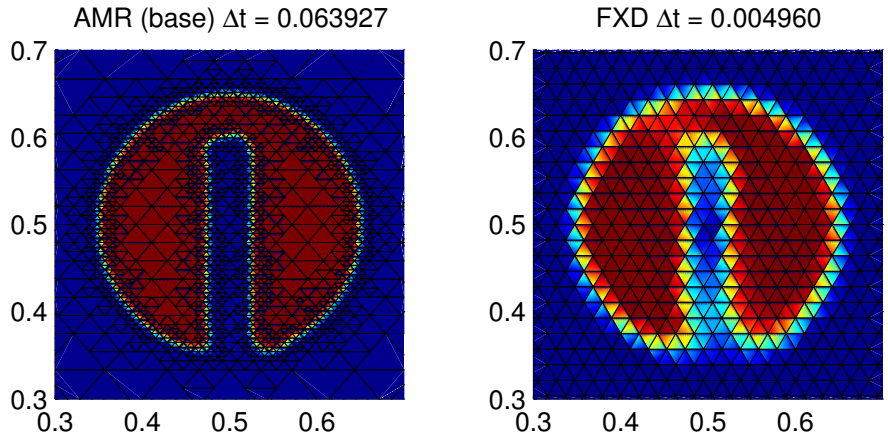

Figure 4.7: Magnification of the results of the AMR and fixed mesh computations at $t = \pi$ for approximately equal cost.

The adaptive computation has an L_1 error of 0.0043 whereas the fixed mesh results in an L_1 error of 0.0158. This means that the AMR method is nearly four times more accurate (according to the L_1 norm) for this particular problem and for a similar amount of computational resources consumed. The comparison is not entirely on equal grounds though as the cost of subdivisions and derefinement are ignored for the AMR scheme. These are not particularly large but hard to quantify in terms of the above-mentioned cost-metric and have thus been omitted.

4.3 Nonlinear Problems

The solution of *nonlinear* hyperbolic partial differential equations is probably one of the largest application areas for finite volume methods.

4.3.1 Burger's Equation

The numerical solution of Burger's equation is considerably more difficult than the previous advection problems. Suddenly, one is confronted with shocks appearing and disappearing, smooth initial data turning into discontinuities and other difficulties. But it is also these features that make the presented adaptive method worthwhile.

The inviscid Burger's equation in two dimensions is defined as

$$u_t + uu_x + uu_y = 0 \tag{4.7}$$

which implies that

$$f(u) = g(u) = \frac{1}{2}u^2$$

when Equation 4.7 is written as

$$u_t + f(u)_x + g(u)_y = 0$$

to match the definition of a conservation law in Equation 2.1.

The method itself stays exactly the same compared to earlier applications — the only thing that needs to be adapted is the numerical fluxes in the problem description. These need to be defined carefully to ensure conservation and thus correct shock speeds.

Equation 4.7 is solved on the region $[0,1] \times [0,1]$ with $\Delta t = 0.05$ on the previously used equilateral triangle mesh with five levels of subdivision and the initial data

$$u(x,y,0) = \begin{cases} \frac{1}{2} & \text{if } x < \frac{1}{2} \text{ and } y < \frac{1}{2} \\ -\frac{1}{2} & \text{if } x > \frac{1}{2} \text{ and } y > \frac{1}{2} \\ \frac{1}{4} & \text{otherwise} \end{cases}. \tag{4.8}$$

This is the same initial data as used by Berger and Oliger [4], although we apply different — namely periodic — boundary conditions. Also, the properly limited schemes implemented do not need numerical work-arounds such as adding artificial viscosity to dampen oscillations that were employed by them.

Due to the different boundary conditions used, no analytic solution to compare against was available. Much progress has been made in recent years in the classification and analytic solution of two-dimensional Riemann-problems (for example Wagner's paper [20]), but work remains to be done until those solutions are easier to construct. In this particular example it

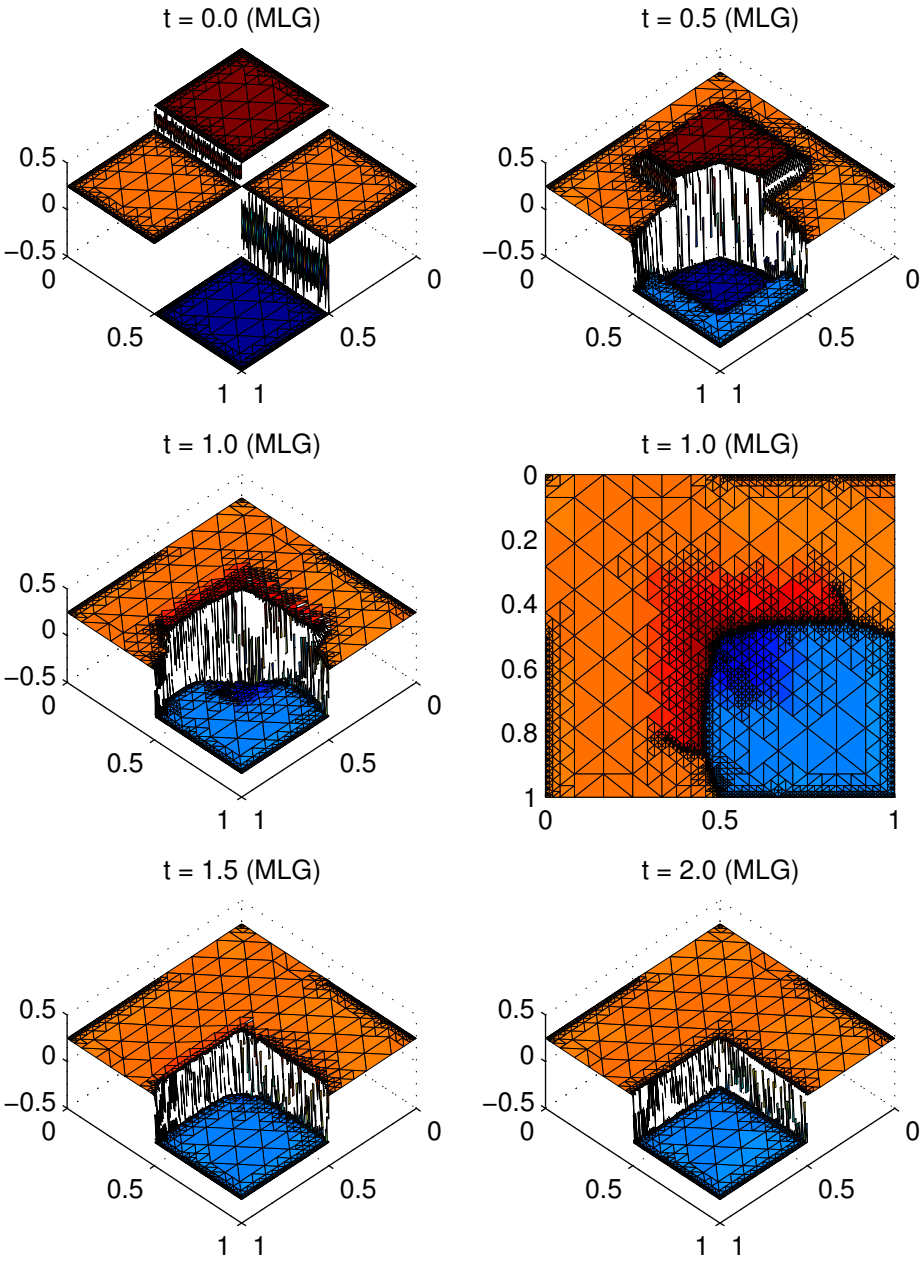

Figure 4.8: Numerical solution to Burger's equation with the MLG-limiter. Note the shocks that are later overtaken by the respective expansion-waves.

was possible to verify that the general shape of the solution is correct. The shocks and expansion-waves also seem to move with the correct speeds from previous experience in the one-dimensional case.

As seen in Figure 4.8, the adaptive scheme with the MLG-operator gives very good resolution of shock-fronts and chooses appropriate resolutions for features at all levels: constant states use very little computational resources as one would expect and expansion-wave gradients are properly resolved without being overly fine. The other limiters (not shown) do surprisingly well considering the difficult nature of the problem, although the expansion fans seem to be slightly elongated when using the first-order scheme and (to a lesser degree) the LCD-limiter. The stronger dispersion in those limiters is the most likely cause for that.

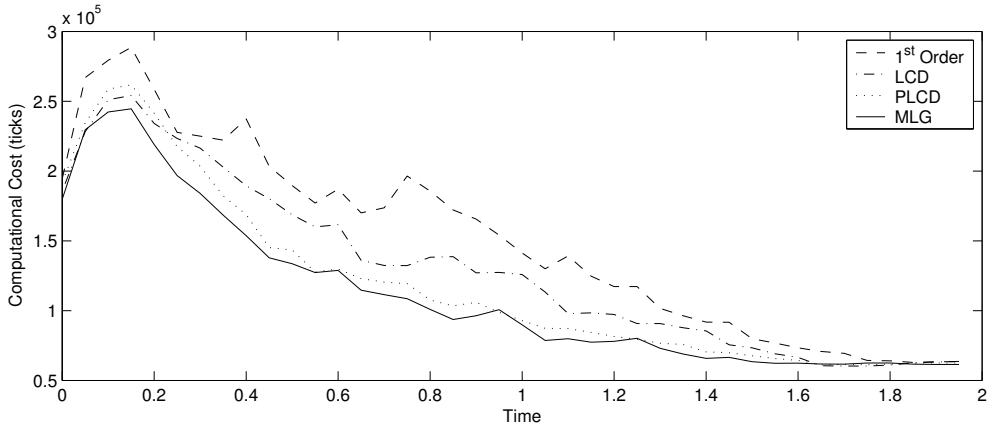

Figure 4.9: The workload / cost per top-level time-step during the 5-level solution of Burger's equation. The whole computation took less than 30 seconds on the author's machine for the MLG-case.

The "adaptive" cost advantage shows up in a slightly different form. The initial data given in Equation 4.8 has *four* plateaus separated by discontinuities. The final solution at $t = 2$ only contains *two* distinct states and therefore contains much fewer highly refined edges where those states meet.

From the computing cost displayed in Figure 4.9 it is immediately apparent that the computation speeds up drastically towards the end. The time needed per time-step near the end of the computation is reduced to nearly a quarter of the time used during its early phase. This results in a noticeable shortening of the computation-time after the 50%-mark. The different limiters have a comparable cost to before; although there is not much separating the MLG-operator from the PLCD-gradient this time around.

In regard to the solution of this problem, Berger and Oliger [4] have remarked that *"this problem is a hard test for mesh refinement because such a large fraction of the region is refined."* Due the ability of our subdivision-based refinement to allow for locally higher resolution without affecting the rest of the mesh (other than through implicit subdivisions of course), a weakness of traditional adaptive mesh refinement has been turned into an advantage.

Chapter 5

Conclusions

An adaptive algorithm for the efficient solution of hyperbolic conservation laws on unstructured meshes has been introduced. The proposed scheme is formulated in terms of standard finite volume methods on triangular elements, although some extensions have been made to allow for inter-level updates with those methods which involve polygons with more than three edges. The simple quad-tree data-structure is essential to the performance of the algorithm as it facilitates the atomic operations such as subdivision into four children, derefinement and neighbour queries on which the method relies. It also restricts neighbouring triangles to have a refinement-difference of at most one level which ensures that there are no abrupt changes in resolution. Another advantage of using subdivision of elements is that no monitoring or adjustment of individual element's anisotropy is necessary if the original top-level mesh is well-formed (e.g. a Delaunay triangulation). Furthermore, the introduction of "Just-In-Time" refinement makes it possible to defer any decisions about the resolution at which computations are to be done until they are actually about to happen. This is achieved by very fine-grained adaptivity: The "adaptiveness" of the scheme is not something that is done every now and then between traditional computations — it is an integral part of the algorithm.

The mesh refinement method was implemented in two dimensions with a variety of [9] slope-limiters and has been successfully used to solve both traditional linear problems as well as non-linear ones. These numerical experiments have shown the superior efficiency of the scheme compared to computations on fixed meshes. They have also demonstrated the necessity of high-quality finite volume methods of at least second order accuracy for efficient computations; the standard second-order Limited Central Difference scheme is already too diffusive to make highly localised refinement feasible and makes the computations more expensive than necessary, although

the diffusive behaviour of *any* scheme is lessened by the use of our method compared to fixed mesh computations. We have found Batten *et al.*'s [3] Maximum Limited Gradient to be very good at resolving discontinuities or shock fronts — although it is not immune to mesh distortion — and Hubbard's [7] Projected LCD-operator excels on continuous problems and shows no dependency on the underlying mesh geometry.

5.1 Further Work

There are quite a few distinct areas which may prompt further research. Foremost would be the extension to non-linear systems of equations — for example using Roe's approximate Riemann-solver [15] in conjunction with the proper averages. There are already quite a few reformulations of "popular" PDEs into conservation form for genuinely higher-dimensional finite volume methods on unstructured grids, [7] for example shows how they can be applied to the Shallow Water equations. This should be fairly straightforward to do as our adaptive method does rely on largely unmodified finite volume schemes.

The extension of the scheme to three dimensions is another interesting subject, although we do not foresee any new topological difficulties in the process. The main changes would be to replace triangles with tetrahedra and to modify the subdivision and data-structures accordingly.

It may very well be worthwhile to replace our rather crude threshold-based gradient detector with something more sophisticated as the linear reconstruction of the second-order schemes allows us to compute accurate solutions for linear data *without* the need for much refinement. The work of Barth and Larson [2] on error estimates for finite volume methods may be relevant in that regard.

The behaviour of different gradient-operators for particular types of data prompts the question whether it may be appropriate to use different operators at different levels of refinement or for exceeding different thresholds in the gradient-detector. One could for example apply the PLCD-limiter on the children of a triangle if the refinement was a "close call" (i.e. relatively smooth states) and use the MLG-operator if the region was determined to contain discontinuities — while still maintaining proper conservation.

Appendix A

Acknowledgments

Many thanks to my supervisor, Dr. P. K. Sweby, for many helpful suggestions and hints as well as general guidance towards the topic of this paper.

Prof. M. J. Baines has also been very generous in answering many of my questions.

I am also grateful to the EPSRC for providing partial financial support.

Last but not least, I wish to thank my parents for putting up with me for the past 24 years and making all this possible.

I would also like to thank the following for keeping me (remotely) sane during all this time:

- *Björk*, *Elton John*, *Lamb*, *Last Days of April*, *Mazzy Star*, *The Natural*, *The Prodigy* and *Yoko Kanno*.
- The whole $\log_2$-crew for being there.
- My iPod for making the walks to campus that much more interesting.
- The Reading Anime Society — and in particular Chris — for making Mondays something to look forward to.

Appendix B

Bibliography

[1] M. Ahmadi and W. S. Ghaly: "A Finite Volume Method for the Two-Dimensional Euler Equations with Solution Adaptation on Unstructured Meshes", *6th ASME International Congress on Fluid Dynamics and Propulsion, Egypt* (1996)

[2] T. J. Barth and M. J. Larson: "*A Posteriori* Error Estimates for Higher Order Godunov Finite Volume Methods on Unstructured Meshes", *NASA Technical Report* **NAS-02-001** (2002)

[3] P. Batten, C. Lambert and D. M. Causon: "Positively Conservative High-Resolution Convection Schemes for Unstructured Elements", *International Journal for Numerical Methods in Engineering* **39** (1996)

[4] M. J. Berger and J. Oliger: "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations", *Journal of Computational Physics* **53** (1984)

[5] M. Berzins, R. Fairlie, S. V. Pennington, J. M. Ware and L. E. Scales: "SPRINT2D: Adaptive Software for PDEs", *ACM Transactions on Mathematical Software* **24-iv** (1998)

[6] R. D. Hornung and J. A. Trangenstein: "Adaptive Mesh Refinement and Multilevel Iteration for Flow in Porous Media", *Journal of Computational Physics* **136-ii** (1997)

[7] M. E. Hubbard: "Multidimensional Slope Limiters for MUSCL-Type Finite Volume Schemes on Unstructured Grids", *Numerical Analysis Report* **8/98**, *Department of Mathematics, University of Reading* (1998)

[8] D. E. Knuth: "Seminumerical Algorithms", *The Art of Computer Programming (3rd Edition), Volume* **2** (1997)

[9] B. van Leer: "Towards the Ultimate Conservative Difference Scheme V. A Second-Order Sequel to Godunov's Method", *Journal of Computational Physics* **32** (1979)

[10] B. van Leer: "On the Relation between the Upwind-Differencing Schemes of Godunov, Engquist-Osher and Roe", *SIAM Journal of Scientific and Statistical Computing* **5** (1984)

[11] R. J. LeVeque: "Numerical Methods for Conservation Laws", *Lectures in Mathematics – ETH Zürich* (1990)

[12] R. J. LeVeque: "Finite Volume Methods for Hyperbolic Problems", *Cambridge Texts in Applied Mathematics* (2002)

[13] J. Z. Lou, C. D. Norton and T. A. Cwik: "A Robust and Scalable Software Library for Parallel Adaptive Refinement on Unstructured Meshes", *NASA HPCCP Computational Aerosciences Workshop* (1999)

[14] M. C. Rivara: "Selective Refinement/Derefinement Algorithms for Sequences of Nested Triangulations", *International Journal of Numerical Methods in Engineering* **28** (1989)

[15] P. L. Roe: "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes", *Journal of Computational Physics* **43-ii** (1981)

[16] P. K. Sweby: "High Resolution Schemes using Flux Limiters for Hyperbolic Conservation Laws", *SIAM Journal of Numerical Analysis* **21-v** (1984)

[17] P. K. Sweby: "Numerical Techniques for Conservation Laws", *Lecture Notes, University of Reading* (2003)

[18] J. A. Trangenstein and Zhuoxin Bi: "Multi-Scale Iterative Techniques and Adaptive Mesh Refinement for Flow in Porous Media", `http://www.math.duke.edu/~johnt/multigrid.ps` (Preprint 2002)

[19] G. Turk: "Generating Random Points in Triangles", *Graphic Gems* **I** (1990)

[20] D. H. Wagner: "The Riemann Problem in Two Space Dimensions for a Single Conservation Law", *SIAM Journal of Mathematical Analysis* **14-iii** (1983)

[21] S. T. Zalesak: "Fully Multidimensional Flux-Corrected Transport Algorithms for Fluids", *Journal of Computational Physics* **31** (1979)