Problem Solving Environments for the Numerical Solution of Partial Differential Equations

by

Paul R. Pratt

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds
School of Computer Studies
September 1995

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Abstract

The complexity and sophistication of numerical codes for the simulation of complex problems modelled by partial differential equations (PDEs) has increased greatly over the last decade. This makes it difficult for those without direct knowledge of the PDE software to employ it efficiently. Problem Solving Environments (PSEs) are seen as a way of making it possible to provide an easy-to-use layer surrounding the numerical software. The users can then concentrate on gaining an understanding of the physical problem through the results the code is providing. PSEs aim to aid novice and expert users in the problem specification process and to provide a natural way to solve the problem. They also decrease the time spent on the problem solving process.

This study is concerned with the construction of a PSE for the numerical solution of PDEs. This is one area where PSEs can be used to particularly good effect because the solution process is complicated and error prone. It will be shown how PSEs can remedy these issues by allowing the user to easily specify and solve the problem.

The construction of a prototype PSE is achieved through the utilisation and integration of existing scientific software tools and systems. An examination of the solution process of PDEs is used to identify the various components required in a PSE for such problems. The PSE makes use of an open design environment and incorporates the knowledge of the users and developers of the numerical code together with a set of generic software tools based on emerging standards. This combination of tools allows the PSE to automate the solution procedure for a number of PDE problems. Finally, the success of this approach to building PSEs is examined by reference to an engineering PDE problem.

Acknowledgements

I would like to thank my supervisors Dr. Martin Berzins and Dr. Laurence Scales (Shell Research Ltd) for advice and encouragement. I would also like to thank Dr. Ken Brodlie for helpful discussions concerning the publication of material from this thesis.

Thanks to my fellow researchers, in particular Justin Ware, David Hodgson, Philip Capon, Gary Stead and Jeremy Littler, for help, discussions and friendship. Finally I would like to thank Victoria who didn't know what she was letting herself in for.

This research was supported by EPSRC and Shell Research UK Ltd through a EPSRC CASE award.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in scientific computing have resulted in the ability to build powerful numerical software capable of solving a wide range of difficult problems. With this dramatic increase in the use of computers for the modelling and simulation of complex processes many scientists and engineers find that computing has become a predominant part of their work. This has resulted in an increase in the use of scientific computing as well as an increasing dependence upon it.

However, with the increase in the complexity and sophistication of numerical software it has become increasingly difficult for those without detailed knowledge of the software to employ it efficiently. This is especially true in the area of computational fluid dynamics. The specification process from the way a scientist or engineer would naturally specify the problems to a form required by the numerical code to solve the problem is often long and complicated. Even when the numerical code has been employed to good effect the understanding of the results given again presents a similar problem. This is obviously unsatisfactory. In an ideal world those with the greatest knowledge of the problem should be able to utilise the software as a powerful tool rather than getting tied up in the details of the software.

The aim should be to reduce the time spent on the specification of the problem and allow the focus to be on the results of the simulation, including visualisation aids to help the users to understand the result. In effect there should be an easy-to-use interface layer surrounding the software. This interface can then be utilised, to aid the solution process, by not only those with in depth knowledge of the problem but also by novice users. It can also be used as an aid to the builders of the software to help in the software

development process. This can allow developers to concentrate on one particular area of a wider solution process. Problem solving environments (PSEs) are seen as the way to provide this easy-to-use layer.

In this thesis Problem Solving Environments surrounding numerical software for the solution of two-dimensional partial differential equations will be discussed. The construction of PSEs in this area will be discussed. A prototype PSE surrounding a general purpose PDE solver for two-dimensional convection-dominated PDEs will be undertaken as a case study to highlight this. The way in which PSEs are built, the environment in which they are built and the tools used to build the PSE will all be examined. The use of an open environment including the users and developers of the numerical code for the development of the PSE is looked at also if this open environment is advantageous and provides the best way to focus the available resources.

1.1 Contents of Thesis

This thesis is organised in the following way. Chapter 2 looks at the development of PSEs in scientific computing. It highlights the differing views of PSEs from early aspirations of PSEs to more recent views about the structure and layout of a PSE. It looks in particular at PSEs in the area of partial differential equations (PDEs). From these varying viewpoints the aims and properties of a PSE can be determined. This chapter also looks at some of the current tools available in scientific computing and if they can be utilised to construct PSEs.

Chapter 3 considers the numerical solution of PDEs as a backdrop for the numerical code the PSE will surround. It looks at what is required to solve these problems and the current method used. It outlines what is needed by numerical software in this area and how the user is expected to provide this. Chapter 4 looks at the first of two generic software tools that are part of a prototype PSE surrounding the numerical code. The visual domain specification tool (VDS tool) is concerned with the specification of the numerical domain required in PDE problems. The Visual Problem Specification system (VPS system) described in Chapter 5 is a further tool that when combined with the VDS tool allows the user to complete the specification process. Both these chapters look at the construction of these tools and how they fit into the prototype PSE.

Chapter 6 looks at a different aspect of the problem solving process and discusses

interpolation methods designed to compliment the numerical solution process. The interpolant, as well as providing an aid to the visualisation of the solution, can also be used within the numerical process to recover solution values and derivative values.

Finally Chapter 7 aims to draw all this together to determine the success of the approach taken. It gives a critical evaluation of the PSE by looking at a difficult engineering problem. This is used to highlight the gaps between what can currently be achieved and what is ultimately required in the future for PSEs in this area.

Chapter 2

Problem Solving Environments

2.1 Introduction

The increase in scope of problems that complex numerical codes can solve has resulted in the emergence of computing as a large part of many scientist and engineers work. The gap between those that can successfully use these powerful solvers and those who would wish to use them has widened. What is required is the addition of an easy-to-use interface layer surrounding the numerical software, this can enhance the problem solving process and allow the focus to be on the results of the simulation. Problem Solving Environments (PSEs) aim to provide this layer.

As scientific software develops and general purpose PDE software to solve a broad class of problems emerges, any environment surrounding the software must be able to adapt. It should therefore be stressed that the aim here is to distinguish between a suitable PSE for a general purpose PDE solver and that of a *user friendly specific environment surrounding a particular numerical solver.* Whilst the latter can be seen as providing an easy to use layer it is the former that can provide a more generic environment. It is this property that makes PSEs desirable.

In this chapter the nature and layout of a PSE is investigated. The way PSEs have developed in the area of the numerical solution of partial differential equations will be considered. The evolution of two PSEs is discussed and others systems surrounding numerical solution packages are outlined. The use of symbolic algebra systems coupled with numerical software is also examined.

From this review, information about the requirements for a successful PSE may

be obtained. This information can then be applied to the construction of a PSE as a case study to investigate the requirements. Tools to construct this PSE are then outlined and the reasons why these can be considered suitable building blocks.

2.2 Layout of Problem Solving Environments

The form of a PSE is by no means clear except that the system tends to be heavily problem orientated. Moreover the nature of PSEs have changed over time. Early ideas expressed by Cryer [24] and Ford & Iles [34], define a PSE as a user interface connected to a knowledge base or expert system capable of driving numerical software. A workshop on future directions of PSEs in computer science [41] defines PSEs as capable of solving problems by communicating in the user's own terms. The report goes on to define the characteristics a PSE should possess, use of modern computing facilities, state-of-the-art solution methods, review of problem solving task, management of computer resources. The report states that a PSE should be all things to all people, supporting novice and expert users, although they admit PSEs that fulfill all these characteristics do not, as yet, exist.

Stetter, in a recent paper Tools for Scientific Computing [82], looks at the wide variety of scientific tools currently available and advocates; the way forward for PSEs is the utilisation and integration of such tools to form PSEs systems that are devoid of explicit programming. A PSE can therefore be viewed as a collection of tools that provide a bridge between the problem the user wishes to solve and a collection of scientific software. Such a view will fit the nature of a PSE outlined by Cryer, Ford & Iles and Stetter. Any expert system can be seen as another tool in the collection. The aim of a PSE is that it should provide an enhancement to the solution process that would not otherwise be present. Such an enhancement, for example, may be to increase the reliability of the solution, decrease the time or cost spent from specification to solution, or provide a more convenient means to use the numerical software. This need to enhance the way computers are used in scientific computing is also recognised by Peskin et. al. [68] when looking at user interfaces for scientific computing.

Stetter, in his paper, discusses PSEs and points to the Parallel ELLPACK (//ELLPACK) system [47], [49] as probably the most advanced problem solving environment for a class of mathematical problems. Stetter continues by adding that such

systems are not the norm in scientific computing, enormous expertise, experience and effort has been expended in the development of the //ELLPACK environment. The amount of effort and the monetary costs involved lead Stetter to point towards less specialised and less perfect scientific environments utilising the range of currently available scientific tools. One example of this perhaps is the work by Flaherty et. al. [65], [33], [3] which looks at constructing a collection of tools for the solution of PDEs. A similar approach of using a combination of software tools to form a PSE is the one taken in this thesis. A description of the system outlined by Flaherty et. al. the //ELLPACK system, and some of the other PSEs in scientific computing, especially the computational fluid dynamics area, will be given in the next section. The report on future directions for PSEs [41] also provides an overview of the current status of PSEs in scientific computing and for PDE-based systems.

2.3 Problem Solving Environments for Partial Differential Equations

The range of applications for PSEs for partial differential equations (PDEs) is wide. Many systems focus on the use of expert systems to select the appropriate algorithms for the problem class. Such systems may view the user as a non-expert in the area under consideration. However, in many areas of scientific computing the user may well be considered as an expert. Formulation of the problem as well as selection of the various solution methods available may not be a problem. In this case it is the creation of a suitable driving code for the numerical or scientific software that is required. The assumption about the user as a non-expert compared to an expert is one which will shape the development and focus of the PSEs evolution.

Gaffney et. al. look at PSEs for scientific computing in the NEXUS project [37] and they assume that the user is seen as a non-expert. The PSE then aims to help the user to choose the most appropriate routine in a library of mathematical software. The use of a tree like decision process based on the earlier NITPACK system [39], [38] helps this process by using a natural language approach to aid the selection of acceptable software. The use of inference mechanisms avoids the need to ask explicit questions about the mathematical properties of the problem. This approach has been applied to initial value ordinary differential equations [2]. The NEXUS project aimed to provide generic

tools to allow the construction of information trees which can be applied to any area of scientific computing. The result of traversing the information tree is, hopefully, to produce a complete program capable of driving the numerical software. The NEXUS work stems from the desire to help those who construct computer programs to make the best use of the wealth of available mathematical software.

Many such expert systems and knowledge bases have been constructed to aid the user in the selection of the algorithmic methods for the solution of PDEs and ordinary differential equations, see [5], [56], [55] for details. The use of expert systems is not discussed here for several reasons, the first is that the user is seen as an expert who understands the solution process and the available software parts required. The second is because the numerical solution process involved here has relatively few solution paths. Whilst the use of expert systems is not discussed here the user interface components of such systems can still provide insight into the requirements for the visual interface of PSEs.

When considering PSEs for PDEs the changing aims and the increase in the capabilities of PSEs can be seen by examining the evolution of two such systems. Firstly the //ELLPACK system, described by Stetter as probably the most advanced PSE yet for a class of mathematic problems, is given below. Following this is a description of another PSE, the Visual PDEQSOL system. It can be seen that both systems evolved in a similar way to utilise advances in computer hardware and software.

2.3.1 The //ELLPACK Problem Solving Environment

The //ELLPACK problem solving environment describes itself as a machine independent PSE for the prototyping of physical objects for a large class of partial differential equations (PDEs). The //ELLPACK system stems from the earlier ELLPACK system [72] a very high level system for the solution of elliptic PDE problems. The ELLPACK system utilised a very high level language to specify the PDE equation, boundaries and grid segments in a way that would be closer to the real world specification of the problem. The problem could then be solved using a modular approach, ELLPACK acting as a translator which produces FORTRAN code, from the high level specification utilising ELLPACK modules to solve the various aspects of the problem.

ELLPACK was first constructed as a environment for the evaluation of performance of the algorithms and software parts used in the solution of elliptic PDEs, however,

it was soon recognised as a powerful modular based tool for solving a wide class of problems. So despite the fact that ELLPACK was initially designed to solve second-order linear elliptic problems, the use of a modular structure allowed it to be used to solve problems from other domains. The realisation that the ELLPACK system could become a powerful problem solving tool has led to many enhancements to the original system. The current //ELLPACK system is the latest in a line of improved ELLPACK environments.

The initial ELLPACK system worked in a batch-like fashion and so an Interactive system was developed as an extension to the ELLPACK system. The Interactive ELLPACK system [31] provides the user with the ability to interactively build grids, choose solution methods and analyse computed results. Interactive ELLPACK provides a means to construct menus of ELLPACK commands which allow the user to drive ELLPACK. For example, Interactive ELLPACK provided a powerful tool for the investigation of varying parameter values in problems, the menus providing an easy means to alter the values. Interactive ELLPACK also allows the user to define grid segments with the ability to add and delete vertical and horizontal grid lines over the numerical domain. This grid specification could be done graphically or via a text based system. Interactive ELLPACK also uses established plotting packages which allows the output to be examined via graphical windows or hardcopy devices. The system utilises existing software tools to produce graphical results.

With the increasing development of graphical workstations the utilisation of such capabilities was the next logical step forward for ELLPACK. The resulting system XELLPACK [15] allowed a window based Interactive environment with ELLPACK. XELLPACK seems to provides very little changes to the format of the Interactive ELLPACK system, providing a large dialogue window that allows the user to construct menus of ELLPACK commands in a similar way to Interactive ELLPACK. Once constructed, the environment allows the menus to be displayed as popup windows. The selection of menu items is made using mouse button presses. The addition of the entries *next menu*, *prev menu* and *quit* on each user defined menu provides the ability to manipulate menus. The graphical interactive grid specification became menu driven as well as the production of graphical output plots. The use of the X Window System applied to the capabilities of Interactive ELLPACK provides a standardised way to utilise the ELLPACK system via graphical interaction.

The next stage in the ELLPACK evolution was seen as the construction of an

expert system with the ability to help the user in the choice of the best solution path. The evolution of the ELLPACK system provided an increase in both the range of problems that can be solved and the number of available modules. The inability of the XELLPACK system, as the original ELLPACK system, to help the user was considered a problem. The Elliptic Expert system [30] was constructed to provide this help. The Elliptic Expert paper makes the statement that XELLPACK has 1147 distinct solution paths. With so many possible paths some form of help is required. The knowledge base used initially has been extended for the //ELLPACK system, see [46] and incorporated into the final //ELLPACK PSE.

The next stage in the development was the //ELLPACK PSE [47] [49] which provided the extension to the ELLPACK environments to allow for the increase in computer facilities available to scientists and engineers. Powerful graphical workstations and high performance parallel systems provided a considerable increase in the types of problems that can be solved utilising an increasing knowledge in the area of computational fluid dynamics. The //ELLPACK system provides tools to specify a greater class of PDE problems than its predecessor ELLPACK, to provide the tools to decompose problems automatically, to utilise the capabilities of parallel hardware and to construct parallel algorithms.

The //ELLPACK system provides a powerful PSE, the objectives of the system are given as: to construct parallel algorithms for the simulation of physical objects and to provide a machine independent PSE to support the analysis and design of physical objects. //ELLPACK also aims to support some form of modular programming, allowing new modules to be constructed and the combination of existing modules to solve an increasing range of problems and to support the development of PDE based applications on parallel and sequential computers. The latest //ELLPACK system consists of several subsystems based around three main elements. These elements support the specification of the model, the processing of the model and the visualisation of the results. Figure 2.1 shows the layout of the //ELLPACK system as outlined in [49].

The subsystems are structured around the //ELLPACK expert system tool in a framework derived from the finite element or finite difference solution process used. The six subsystems located around this central //ELLPACK expert system tool are outlined below along with a brief description of the roles they aim to fulfill in the PSE.

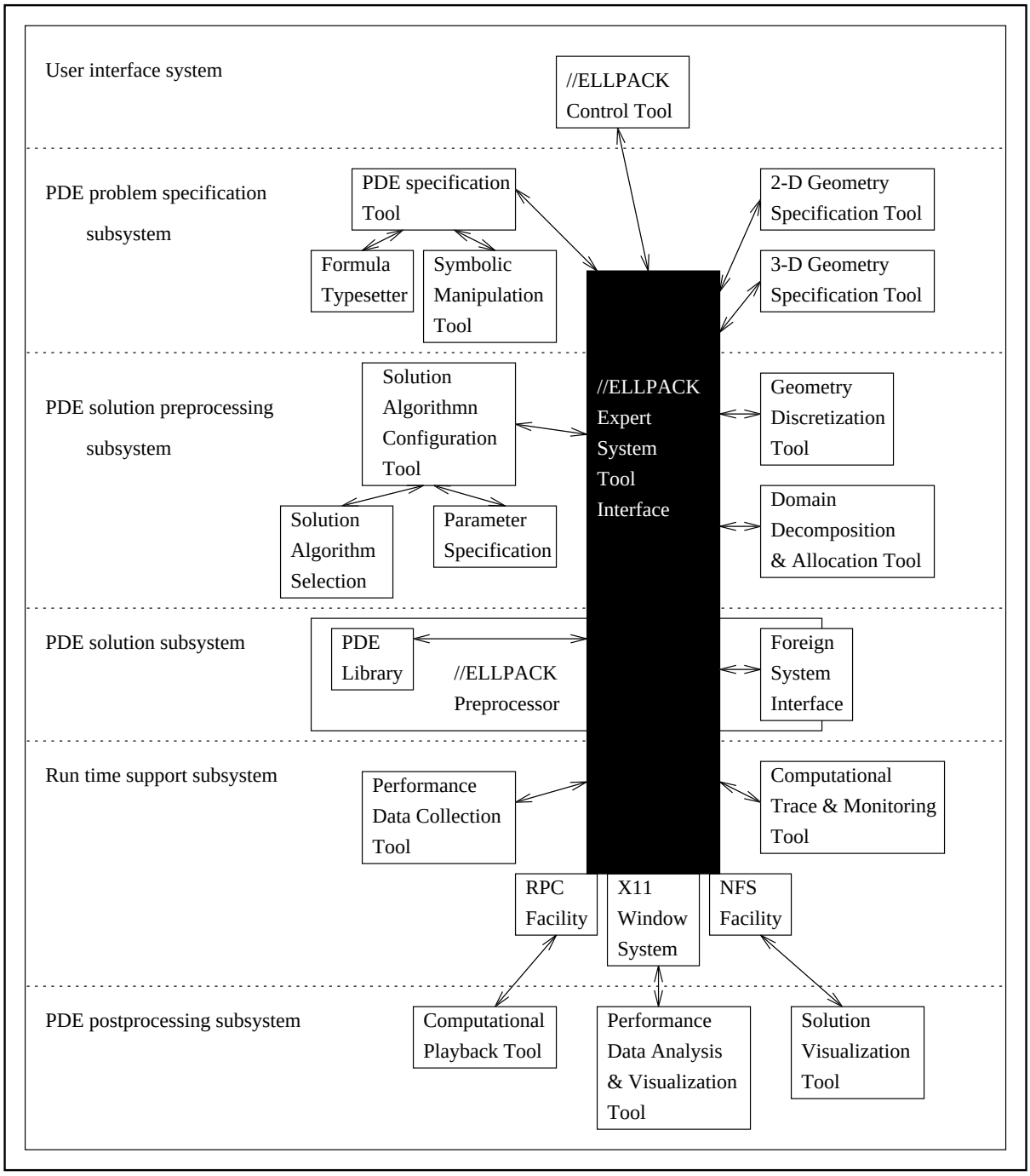

Figure 2.1: The Architecture of the //ELLPACK System

- The User interface subsystem provides control via the //ELLPACK control tool.
- The PDE problem specification subsystem provides a means to specify the geometry of the numerical domain and to specify the governing equations in a graphical and natural way.
- The PDE solution preprocessing subsystem provides means to decompose the domain and to configure solution algorithms.
- The PDE solution subsystem utilises the ELLPACK modular structure to solve the PDE.
- The PDE postprocessing subsystem allows solution visualisation and performance analysis.
- A run time support subsystem collects performance data and allows tracing of the computational process.

The use of a modular approach with fixed interface conditions provides a solid framework for the collection, combination and evolution of modules. It appears to be this, and the exploitation of parallel hardware through decomposition techniques, that are now the driving forces behind the //ELLPACK project. The focus still seems to be aimed at supporting research endeavours although the system provides a powerful tool for the solution of such problems.

2.3.2 The Visual PDEQSOL System

Another system based around a very high level language for the specification of PDEs is that of DEQSOL [59]. DEQSOL stands for the Differential Equation Solver Language and provides a natural way to specify PDE problems. An interactive and visual system has again been fitted round the DEQSOL system and the system itself has been extended to form Visual PDEQSOL [86] which increases the range of problems to include partial differential equations. Again the system generates FORTRAN code for finite difference and finite element methods. This system again provides a further view of PSEs as high level programming language plus code generators for a range of subroutine libraries similar to the ELLPACK system.

The aims of the DEQSOL system were to shorten the time spent in the solution cycle. The aim of fitting a visual/interactive system around DEQSOL was to provide a more user-oriented approach. The authors of the DEQSOL system claim that the environment it provides can shorten the total simulation time by an order of magnitude.

The aims of the Visual PDEQSOL system are given as supporting a total simulation procedure under a graphical environment. The visual PDEQSOL PSE again takes the form of a set of subsystems. The subsystems are listed below.

- PDEQSOL, the high level language and code generation system.
- A model visualiser to specify the physical domain and governing equations.
- A PDEQSOL debugger tool.
- A run time diagnosis system.
- A results analyzer that provides solution and accuracy information.

The design and layout of the systems as given in [86] is shown in Figure 2.2.

Visual PDEQSOL is again built utilising the portability and standardisation of the X Window System to provide a machine independent environment and offers guidance for the choice of numerical algorithms. Some comparisons have been carried out between ELLPACK and PDEQSOL, see [84], however, the main focus was upon the numerical capabilities of both systems. Both systems have led to more generic discussions of PSEs, after the PDEQSOL system see [85], after the initial ELLPACK system, see [71] and after the //ELLPACK development see [48].

The evolution of the Visual PDEQSOL system seems to follow the same path as the ELLPACK system into a graphical problem solving environment. Both systems offer a good basis for determining the desired components for a graphical problem solving environment.

2.3.3 The RPI System

Work by Flaherty et. al. at RPI (Rensselaer Polytechnic Institute) has resulted in the development of a software laboratory of tools to help in the solution of two dimensional PDEs, see [65], [33], [3]. The work here aims to identify the various components that are required to allow researchers with limited scientific computing experience to solve these

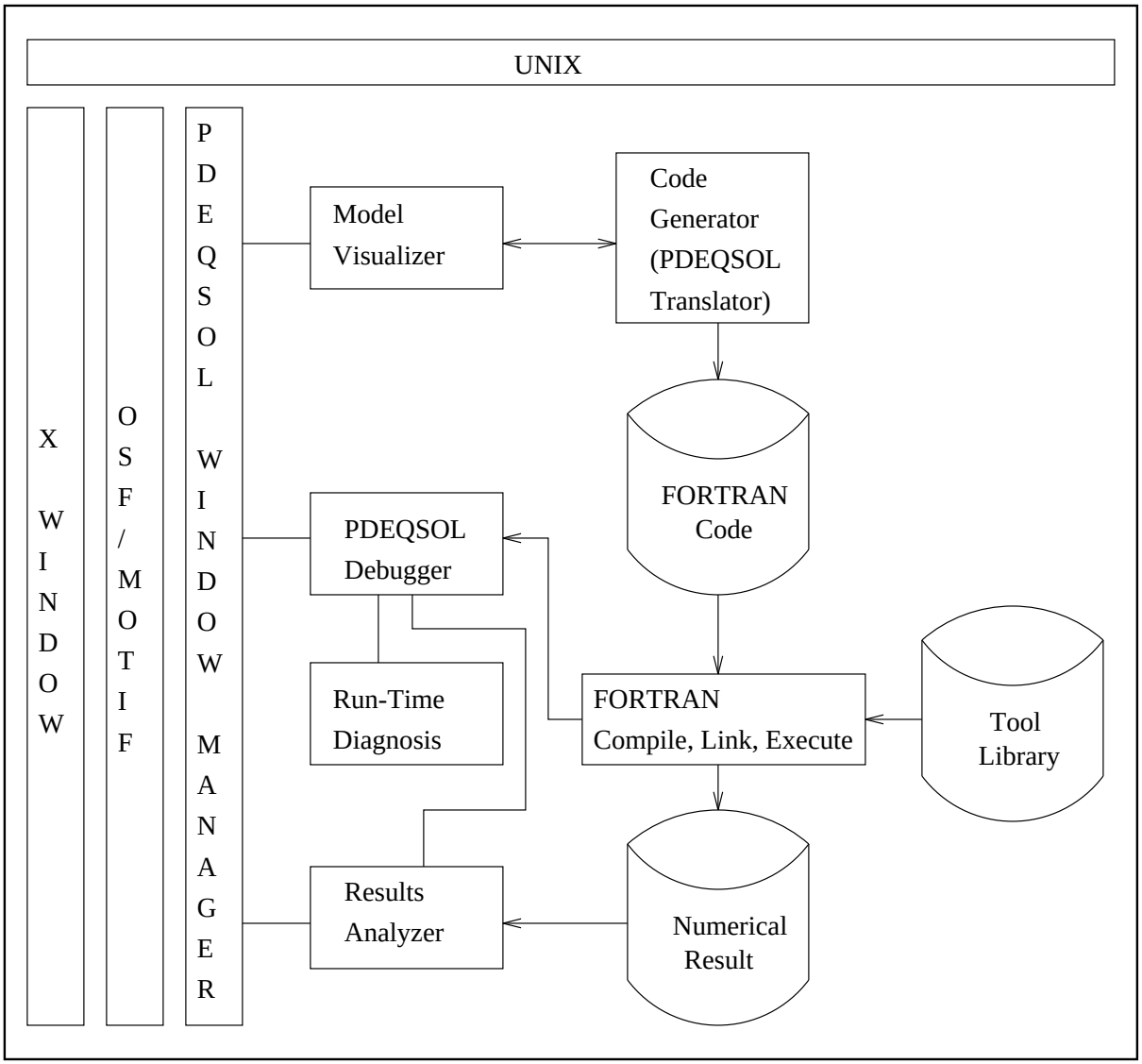

Figure 2.2: The Architecture of the PDEQSOL System

problems. The group identify eight components that a successful system should contain. These are listed below.

- A computer algebra interface to describe the PDE and data in a natural way.
- A geometric modelling system to describe the domain.
- An automatic discretisation package to create a computational mesh over the domain.
- Solution procedure to solve the PDE.
- Error estimation procedures to give local and global accuracy measures.
- Adaptive strategies to improve solution resolution when needed.
- Vector or parallel solution capabilities for increased performance when needed.
- Visualisation tools to analyse and interpret results.

The system is based around either using a finite element method solution process or a finite volume method depending on the nature of the problem. The use of adaptive methods for automatically refining, coarsening and the relocation of mesh points over the computational domain aims to provide reliable and efficient software for the solution of the problems under consideration.

The various components of the system are grouped together to form a software laboratory. This laboratory contains a collection of tools that satisfy the different aspects of the overall problem solving process. The combined tools provide not only a basis for solving problems but serve as building blocks to create more advanced systems as the tools develop and expand. The idea of a collection of tools which may be combined to form a PSE fits closely with the views expressed by Stetter [82] earlier in this chapter.

One key component of the RPI system is the symbolic interface tool. This allows the user to input the PDE, boundary and initial conditions in a natural way. The tool built to perform this task is called *pdefront* and is built using C and the Maple computer algebra system. The use of Maple allows *pdefront* to obtain certain properties of the PDE and allow these to be exploited in the solution method. The system also automatically generates FORTRAN code from the users specifications to compute various functions that

are required in the definition of the PDE. A further advantage is that the *pdefront* system can detect errors and mistakes in the specification of the problem. These can then be remedied at this early stage. A further facility that is provided is the output of the equations, boundary and initial conditions in a document ready form using the Unix troff document processing system.

Other software tools provide the ability to create a computational mesh over the numerical domain. This is achieved with the use of a finite quadtree mesh generation method. The domain is enclosed via a square and then quartered recursively until the prescribed accuracy is reached. The data associated with each quadrant is managed using a hierarchical tree structure. A solution is then obtained on this unstructured grid. The solution process is helped by an adaptive method with refinement and coarsening of the mesh, movement of mesh points and the variation in the order of accuracy over the mesh. Tools can also be used to split the computational domain and exploit parallel machines. The use of visualisation tools is also examined and the problems of using standard graphics packages is outlined. These include the lack of support for certain data structures and the machine dependent nature of such systems. The solution to this problem was to build some primitive graphical software tools capable of interpreting the results produced by the system.

The combined use of various software tools to provide a PSE seems to provide a comprehensive system in this case. The further integration of the different software parts into a more cohesive system is seen as one of the major steps forward in the RPI system. However, this aim is closely linked with that of developing and enhancing the software tools themselves.

2.3.4 The NAG/AXIOM System

The Numerical Algorithm Group (NAG) has possibly the most well established numerical algorithm libraries in scientific computing. The range of the library covers many areas from minimisation, linear algebra, curve and surface fitting to the solution of ordinary differential equations and PDEs, although much of the PDE software is limited to one spatial dimension. New routines are constructed and existing routines continually improved. NAG aims to provide software that is robust, reliable and portable across a wide range of hardware. The current NAG software library, release Mk16, has over

1000 user-callable subroutines. NAG also provides other software for scientific computing including software for visualisation and symbolic computing.

The AXIOM system (see [50], [26]) is a symbolic solver and is a relative newcomer to the area of computer algebra systems. Like many of the other computer algebra systems AXIOM has a high level interactive language and graphical capabilities. It can be used either as a desk calculator or a mathematical tool solving complex problems.

The AXIOM system was developed by IBM but the copyright belongs now to NAG. The AXIOM system works on the principle of a small kernel and high level modules defining the algebraic facilities present this view is similar to that taken by the Maple system. AXIOM, unlike Maple, compiles the modules into machine code to gain speed and performance. The user also has the ability to add new modules to the existing AXIOM library.

AXIOM also contains a built-in hypertext system that provides on-line help, introductory text and tutorials. The hypertext system, built with the X Window system, also has a facility for the user to call the NAG software library [28]. This facility uses a *form like* interface system. The user is required to fill in the blank gaps. Each *form* will have an **Ok** button which can be pressed when the user has filled in the required blanks. The information specified in the first *form* is used to determine what the user needs to specify in the next. Each routine has three pages and the last page contains a **Do It** button. This button will take the information and generate a FORTRAN program that will call the desired library routine.

The combined use of both the AXIOM system and the NAG libraries provide a powerful problem solving tool. This combined with the use of an X based hypertext system provide another example of the combination and integration of software tools to form a PSE.

2.4 Symbolic-Numeric Computing in PSEs

Much attention has been directed towards the symbolic-numeric interface for the PSEs in scientific computing. Such systems utilise the capabilities of symbolic packages, such as Maple [22] and MACSYMA [60], to generate numerical information.

The use of symbolic systems to generate FORTRAN code which can utilise numerical libraries is discussed by Davenport in [25] and more recently using the AXIOM

system to drive the NAG libraries, see Section 2.3.4.

When looking at the solution of PDEs, early work on generating FORTRAN code using symbolic tools and finite differences has been carried out by Steinberg and Roache [79], [78], [81], [80]. They again use MACSYMA to generate FORTRAN code.

Many of the PSEs in the PDE area use solution preprocessors to generate information that is needed or that can aid the solution process. Such systems have been integrated into packages for the solution of PDEs. The RPI system with *pdefront* see Section 2.3.3 and Boubez et. al. [16] both use the Maple computer algebra system. Other systems including the //ELLPACK system attempt to integrate symbolic manipulation tools into PSEs [89], [90] using MAXIMA, part of the MACSYMA package.

The common problem experienced in this area is the interaction between the numerical code and the symbolic system, especially at run time. The problem stems from the development of symbolic tools as stand alone user-level systems, unlike the library approach of numerical software. The effect of this is that interaction taking place forces the client of the symbolic tool to pretend to be human. In effect the PSE must generate instructions for the symbolic tool and some way must be found for the return of the information requested. The report on future research directions in PSEs [41] also highlights this problem. The ability to integrate existing software systems, such as symbolic algebra packages, into larger PSEs is seen as one of the key ways forward.

The //ELLPACK system also acknowledges these problems and resorts to, what it describes as ad-hoc means to get round this problem. Others comment on the unreliability of such systems to produce expected results due to possible conflict of simplification rules. This problem is highlighted by Garbey et. al. [42] when looking at differential equations. The observations made are equally applicable to other areas of scientific computing. The need for symbolic systems to interface to other software systems is seen as a main issue for the future direction of computer algebra systems, see Davenport [25]

Symbolic computer systems have not only been used to preprocess PDEs but also provide a suitable framework for the specification of the governing equations. The requirement to specify the PDE in a natural way has been seen as a way to speed up the overall time involved in the solution process. One early description of such a language is discussed in [21], many of the ideas and notation here are still applicable today. The use of special languages and notation is seen in many of the interfaces of the systems discussed above.

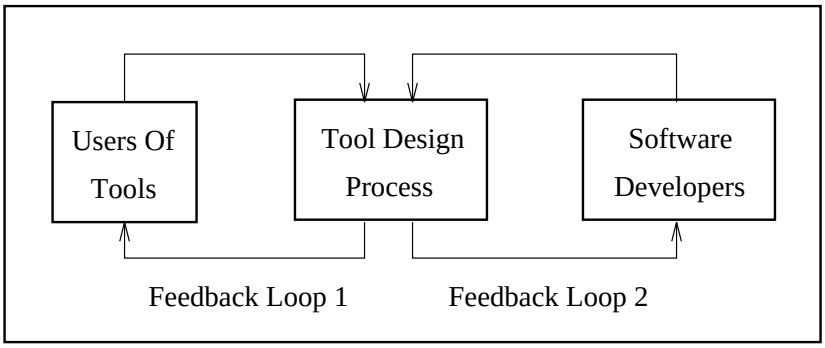

Figure 2.3: Design Environment

With the increased use and availability of symbolic manipulation packages the inclusion of these in PSEs seems a logical step forward. They can be used not only to input the problem but also to process it. This can lead to a better formulation of the problem and to highlight certain properties of the problem. However, the lack of a back door into such systems has resulted in ad-hoc usage of such systems. Any interaction undertaken by the software pretending to be a human user.

2.5 Design Environment

The aim of this thesis is to extract the basic principles behind such PSEs and to apply these techniques to construct a set of tools to aid the solution of two-dimensional convection-dominated partial differential equations using numerical software under development at Leeds [88], [11]. As the numerical software changes the surrounding PSE must also adapt to the users needs. It is intended that the PSE will have the ability to extend allowing the inclusion of new facilities in the PSE as the solution process changes.

The aim is to extract and exploit the generic nature of the problem class, to provide self contained visual tools that are not limited to the numerical software they are designed for but are applicable to the broad problem class of two-dimensional partial differential equations.

The design and construction of such tools will be undertaken in an open environment that involves the users of the tools and the developers of the software, see Figure 2.3. The use of an iterative feedback loop with the users aims to ensure that the tools perform the tasks they are designed for in a manner that is acceptable and appropriate to the

users. The involvement of the developers of the code ensures that the tools provide all the information that is required for the successful solution of the problem under consideration. This open environment for the development of the PSE is one which is seen as essential for success, see [64]. The importance of providing self contained tools is also highlighted. This view is further supported in the future directions report [41] along with the need for generic tools to add to those currently available.

2.6 Building Tools for PSEs

The previous sections illustrate several PSEs and highlighted some of the aims or objectives behind such systems. One common theme is the desire to construct a machine independent environment. This is obviously a desirable requirement, enabling the PSE to be used on many different hardware platforms. Such an aim is an obvious requirement for the PSE that contains the two-dimensional convection-dominated partial differential equation numerical code. In order to maintain a machine independent environment the PSE will be constructed from tools that are either industry standard or are readily available and widely used and therefore may be considered as de-facto standards. This section will outline these building blocks.

The rest of the section will give a brief outline of the tools that may be used as building blocks for the PSE. The choice of each tool is explained and similar systems are outlined. All programming is undertaken on a Unix platform using the C programming language [73], itself a standard.

2.6.1 The X Window System

The evolution of a text based PSE to a graphical PSE may be observed in the development of the ELLPACK and PDEQSOL systems. It is obvious that any successful PSE must capitalise upon the increasing usage of graphical workstations and graphical user interfaces. Usage of the X Window System is essential to allow portability in this area. There are no real competitors to the X system in this area. The X Window System is an industry standard for the construction of portable graphical user interfaces. X provides a device independent architecture that allows the user the mechanisms to produce many different styles of interface. High level libraries built on the X protocol provide building blocks to aid the construction of X applications. X operates on a client server model

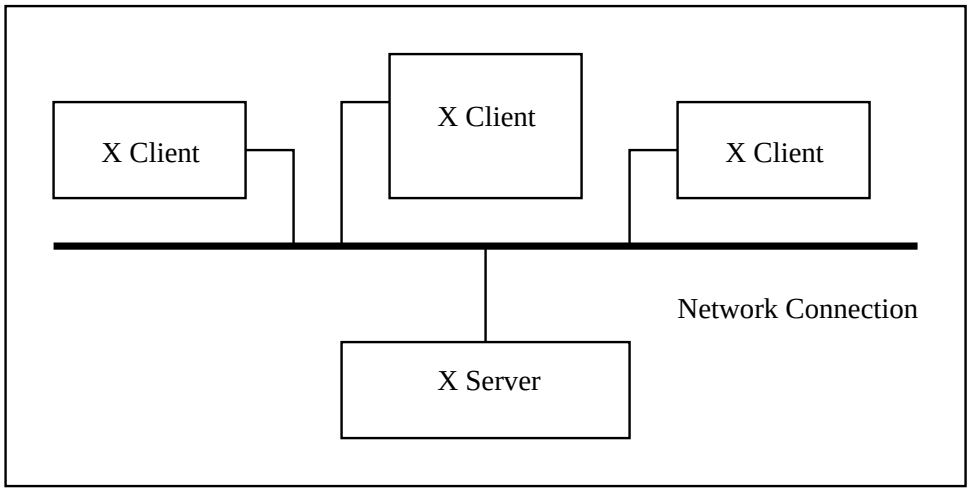

Figure 2.4: Client Server Model for X

with the ability for many clients to be attached to one server. In general the server is a workstation or personal computer. The clients are any other machines that are connected to the server, clients and servers communicate via network protocols, see Figure 2.4.

The basic X resource is a window. The windows have a hierarchical structure descended from the root window, each with its own integer coordinate system. This allows text and graphics to be placed in a window without concern about the position of the window on the screen. The responsibility for displaying and redisplaying the contents of a window rests with the client. The server communicates with the client by means of events. X recognises thirty three different types of events and has the ability for the user to define more. All X applications are event driven the two most common events are the mouse as a pointer and button press device and the keyboard to enter key presses. Events are stored as a first in first out queue. Clients communicate with the server by sending asynchronous requests to display a particular window.

The X Window System provides a set of low level commands via the Xlib libraries and also supports the higher level XToolkit along with higher level widgets sets built on X, see Figure 2.5. Many such widget sets exist and the facilities exist for the application programmer to construct new widgets. These widgets can be combined with the Xlib functions and the XToolkit to build machine independent applications that are supported as an industry standard.

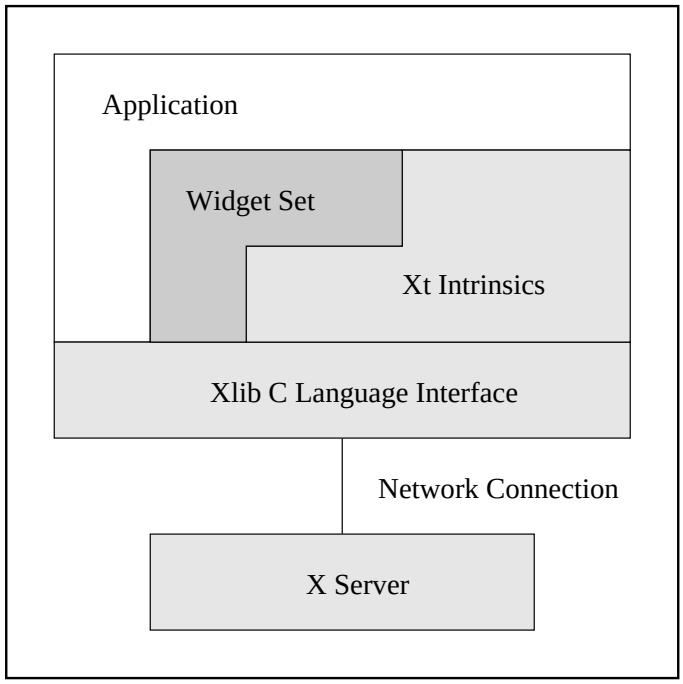

Figure 2.5: Widget Set position in X Windowing System

2.6.2 Building On X — Widget Sets

Widget Sets are high level toolkits built on the X Window System, see Figure 2.5. A widget is defined as an X window with associated manipulation procedures for the window and data structures. Widgets are defined in terms of classes, widgets with common characteristics. Complex widgets are created from simpler widgets by the addition of new manipulation routines or the extension of the data structure.

Many different types of widgets exist which allow the easy construction of basic parts of many X applications for example push buttons, menu, scrollbars and text widgets as well as user defined widgets. Widget sets combined with the X Toolkit allow the easy construction of popup and dialogue widgets. The user can set the attributes or resources of the widgets, with the ability to specify size, colour and placement of the widget.

Facilities exist to allow the user to easily map X events occurring in widget windows to user defined functions. These events may be the mouse pointer entering or leaving a widget or mouse buttons being pressed or released inside the widget. This allows the mapping of the user pressing a mouse button, whilst the mouse pointer is inside a push button widget, to cause the widget library to call a routine which changes the

widgets attributes, then a user defined function may be executed. The combination of these actions gives the user the impression that the button has been pushed causing the application to *do* something.

This ability to push buttons with the mouse and call user defined functions allowing the combination of widgets to form X applications have ensured that the widget sets available have been utilised widely.

One such widget set is the Athena Widget Set [69]. This provides many of the facilities outlined above. Each widget set has its own policy, style and consistency. The Open Software Foundation (OSF) Motif Widget Set [94] is another of these toolkits. The OSF/Motif Widget set has a distinctive look and style which gives a three dimensional appearance to the widgets created. It is this widget set that will be used in the construction of the tool.

2.6.3 Document Processor Tools

Two document processing tools are outlined below, the troff family of packages [66] and the LaTeX system [61]. Both are similar but use of the LaTeX system seems to be widespread and seen as a replacement for the older troff system. The ability for LaTeX to produce high quality mathematical notation and equations seems to lend itself to the scientific computing community.

The Troff document processor package

The troff system is, in many respects, identical to the earlier nroff system but provides laser quality output, the nroff system producing formatted Ascii. The troff package is provided as part of the Unix operating system and is therefore widely available.

The troff system operates on a suitable file containing the text to be formatted and a set of commands. These commands can control text size and structure. They also allow text to be underlined, centered or indented. This provides the user with full control over the format of the document.

The basic troff package cannot deal with mathematical notation, figures, tables or references. In order for the user to utilise these facilities several preprocessors can be used.

- The eqn package to deal with equations [58].

- The pic package to deal with figures [57].
- The tbl package to deal with tables [62].
- The refer package to deal with references [83].

Combined together a powerful document processing tool is available. It appears that this distributed nature of the system seems to have led to the decline in its use.

The LaTeX document processor

LaTeX is a document processor based on the earlier TeX package. TeX is a typesetting package especially suited to mathematical notation, LaTeX provides all the facilities available in TeX but aims to provide an easier-to-use environment.

LaTeX provides many different document styles, such as letters, reports and articles, each of which may use different text sizes and fonts. The aim of LaTeX is to automatically format the document and allow the user to concentrate on the structure of the document without worrying about the layout. In practice users can control aspects of the layout, if required, and even define new document styles to suit their needs.

LaTeX again does not provide a what-you-see-is-what-you-get system, LaTeX is constructed from a file containing the users text and a set of commands. This file is processed by LaTeX to provide a device independent output file which may be printed. Such commands allow the user to change text size and text properties, such as bold face and italics, to use Greek and other special characters as well as defining mathematical notation such as fractions and partial derivative terms, for example the LaTeX commands

```
\frac{\partial{u}}{\partial{t}}
```

will give the mathematical notation

$$\frac{\partial u}{\partial t}$$

LaTeX also provides the commands to construct tables and figures easily, to include imported postscript files and to produce lists, quotations and footnotes. LaTeX also automatically handles cross referencing within documents for equations and section numbers.

This ability to format text easily and to provide support for mathematical notation and equations combined with the other facilities offered by LaTeX has made LaTeX a

widely used document preparation tool. It is therefore the LaTeX system that will be used here. It is LaTeX that has also been used to format this thesis.

2.6.4 Computer Algebra Systems

There are six main computer algebra systems widely available in the scientific computing area. These six systems are REDUCE [44], MACSYMA [60], Maple [22], Mathematica [92], the Derive system based on the earlier muMATH system [93] and AXIOM [26], a recent newcomer to this area. Each system is a general purpose interactive package that provides similar operations. Much literature is devoted to the study of one particular package or an overview of several, see [43] for example, REDUCE, MACSYMA and Derive are implemented using lisp, while Maple and Mathematica use C. The availability of these systems for different operating systems is very similar except for the Derive package which is only available on a DOS platform.

Each system provides similar basic functions and operation, but the notation and syntax of each does differ. All the systems allow the user to manipulate expressions and formula containing integers, rational and real numbers, they also allow the construction of other mathematical objects such as symbolic formulae, polynomials, sets, lists and equations. The systems then allow these to be symbolically manipulated or evaluated to an arbitrary length.

Another important part of such systems is the ability to enhance the system by the addition of new routines. All the above systems also provide the user with the ability to output numerical expression in FORTRAN or C. Such programming languages allow quicker execution of floating point arithmetic albeit to a limited precision. This facility of the systems will be exploited later. Some systems also provide routines to generate instructions for the typesetting programs, troff and TeX.

The similarity of each of the systems allowed a choice of which package to use. The PSE will be constructed on a Unix platform so the Derive system was not considered. The use of a C based system was also preferred and the Maple system was available and appeared to be used widely in the scientific computing community. The Maple system also provided C and TeX output which provided links to the other building tools used. The Maple package is briefly described below and the basic syntax is outlined.

Maple

Maple is an interactive computer algebra system developed at the University of Waterloo. Maple uses its own syntax to construct these mathematical objects, for example if the user has a function f that is dependent on x and y then the user can define this as

```
f := x ** 2 - y ** 2;
```

The `**` notation is used to denote exponentiation. The user can then manipulate this function. Maple provides routines to perform many mathematical operations on the objects constructed and can give exact values, such functions include integration and differentiation. For example differentiating f with respect to y can be achieved by the command

```
diff(f, y);
```

which gives the result

```
-2y
```

Maple can also compute roots of polynomials and factorise expressions, for example

```
factor(f, y);
```

gives the result

```
(x-y)(x+y)
```

Maple also has many other library packages for mathematical areas such as logic, statistics and linear algebra. Each package contains many functions applicable to that area of mathematics. Maple also provide routines to output expressions suitable for inclusion in FORTRAN or C programs. Maple also provides routines to output simple expressions in LaTeX.

2.7 Summary

In this chapter the use of PSEs in scientific computing is examined. The apparent gap between those who need to use such tools and those that are able to use the tools in their *raw* state is one which PSEs are trying to bridge. The utilisation of software tools by

scientists and engineers to solve real world problems should not involve detailed knowledge of the software.

Looking at PSEs from this perspective provides a set of aims for the development of a PSE. These objectives may be seen as ease of problem specification, the reduction of the time spent on the solution of the problem or the convenience a PSE can provide. However, examining many of the current PSEs available shows that the aims of these systems seem to focus on providing a development environment to enhance and improve the software parts involved in the solution process itself. Such systems still provide powerful problem solving tools.

The layout and nature of a PSE is not clear and several examples are given from early attempts to modern more complex systems developed in some areas of scientific computing. The different views and outlines of these PSEs provide enough information to establish the desired properties that a PSE should possess. The aim is to utilise this information and construct a set of tools to aid in the solution of two dimensional convection dominated PDEs. Several systems exist in this general area of computational fluid dynamics e.g Flaherty et. al. [65]. An in depth look at two of the more advanced systems, the ELLPACK and PDEQSOL systems as well as other systems outline some of the key components required. The need for a portable environment that allows the user to specify the problem in natural terms using visual tools. Examining the symbolic numeric interface outlines the use of symbolic algebra systems in scientific computing and highlights some of the problems that exist in the integration of these tools into large systems.

A brief introductory description of the tools that can be used in the construction of the PSE to meet the required objectives followed the examination of other work in this area. The suitability of the tools was examined, the need for portability and familiarity coming from the use of building blocks that are either industry standard or widely used in the scientific computing community.

Chapter 3

The Numerical Solution of Partial Differential Equations

3.1 Introduction

In this chapter aspects of the numerical solution of partial differential equations (PDEs) relevant to PSE will be discussed. These aspects will be illustrated by describing how one particular PDE solver package is used and the features that are common to most PDE software highlighted. An overview of the solution process is given to show what sort and form of information about the problem is required by the package. The interfaces of a particular package are then shown in more detail and an example driver program walked through.

3.2 Scope of Problems

The package described here is designed to solve systems of two-dimensional convection–dominated PDEs that arise from many engineering applications. These may be time-dependent problems where the solution evolves in time or steady problems where a single solution is produced. Convection-dominated equations are typically solved as sets of *conservation laws*. The generic formula for a scalar conservation law is

$$\beta\frac{\partial u}{\partial t} + \frac{\partial f^x}{\partial x} + \frac{\partial f^y}{\partial y} = \frac{\partial g^x}{\partial x} + \frac{\partial g^y}{\partial y} + S \tag{3.1}$$

where

$$u \equiv u(x, y, t), \beta \equiv \beta(u), f^x \equiv f^x(u), f^y \equiv f^y(u),$$
$$g^x \equiv g^x\left(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right), g^y \equiv g^y\left(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right), S \equiv S(u).$$

For steady problems the term β is set to zero to remove the time dependence of the problem and the independent variable t is ignored. The majority of time-dependent problems are scaled so that β is unity.

The terms f^x and f^y define the advective fluxes which lead to wave-like structures in the solution u. The terms g^x and g^y define the diffusive fluxes which lead to diffusion processes in the solution u. The term S, the source term, can be used to add other processes such as reaction terms including chemical kinetics.

As far as the user is concerned much of the information that needs to be specified is the same for both steady and time-dependent problems. The main differences lie in the solution techniques employed. Different numerical methods are needed to ensure the accurate evolution of a time-dependent solution compared with the convergence of steady problems to an accurate solution. It will be shown later that the information required by the numerical code can be split into four main groups, three of which are common to both steady and time-dependent problems.

3.3 Numerical Solution of PDEs

To solve a PDE numerically the continuous differential operators must be replaced by discrete operators. This process is referred to as discretisation. Schemes can either be fully discrete where all differential operators are replaced or semi-discrete where only some are replaced. One popular technique for time-dependent problems is to discretise the spatial differential operators leaving only the temporal ones. The PDE is therefore reduced to a system of ODEs (Ordinary Differential Equations) which can then be integrated using existing software packages. This is referred to as the Method of Lines.

The main advantages of the Method of Lines compared to the fully discrete approach are the separation of temporal and spatial discretisation and the ability to use existing software. There are a wide variety of techniques for discretising in space and in time. Packages based around the Method of Lines are able to combine different combinations of spatial and temporal discretisation as the user requires. The development of PDE

package software is expensive and error-prone so the ability to make use of existing tested software is of great concern. There exists a large collection of sophisticated and tested software that can be used to solve the resulting ODEs.

In one dimension this discretisation of the PDE requires the domain to be represented by a number of mesh points. So the true solution u becomes the discrete solution $U_i, i = 1 \ldots n$. The discrete solution is then found at these mesh points. Consider the linear heat equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

where $\frac{\partial^2 U}{\partial x^2}$ is approximated by the standard finite difference operator, giving n ODEs of the form:

$$\frac{dU_i}{dt} = \frac{U_{i+1} - 2U_i + U_{i-1}}{h^2}, \text{ where } i = 1, \ldots, n$$

which combined with appropriate boundary and initial conditions can be solved by an ODE integrator.

Three of the main classes of spatial discretisation schemes available are briefly described as:

- Finite difference schemes are the earliest form of spatial discretisation. In finite difference continuous differential operators are replaced by discrete ones, usually derived from Taylor series expansion. The advantages of finite differences are the ease of implementation and error analysis for regular spaced square (or structured) meshes. However many physical domains are not suited to structured Cartesian meshes. Combinations of transformed meshes provide one way for complex geometries to be modelled, eg aerofoils, see [19] for a more detailed description.

- Finite element schemes approximate the continuous solution by some form of polynomial. This polynomial is normally constructed as a combination of simpler polynomials that are local to a few cells. The simplest case (and most popular perhaps) is when piecewise linears are used. Higher order approximations are also used. A variational formulation is then constructed by multiplying the PDE by a test function and then integrating by parts. The minimum of the variational formulation is the closest approximation to the PDE with the chosen polynomial form. The finite element method is not just limited to structured grids. A popular choice for two-

dimensional problems is triangular elements with piecewise linear functions used to approximate the PDE, see [53] for further details.

- Finite Volume schemes approximate the solution as a series of piecewise constant elements. The PDE is integrated over an element but, via the divergence theorem, the area integral for the fluxes is replaced by a line integral around the edge of the element. The flux functions in the PDE are then used to calculate the numerical flux between adjoining elements. In principle finite volume schemes may use any form of spatial elements and a mesh may use several different types. In practise the use of a single element simplifies the design and implementation. Quadrilaterals and triangles are the most common elements for two-dimensional problems. The use of triangular elements allows complex domains to be modelled, as in finite element.

The numerical solution process considered employs the finite volume method so it will be the focus of this chapter.

For convection-dominated PDEs, careful discretisation in both time and space is essential to preserve the physical validity of the solution. Ad-hoc discretisation may result in unphysical oscillations appearing in the discrete solution. A commonly used approach to ensure a stable solution is to introduce an upwind bias into the solution process. The upwinding takes into consideration the directional aspects of the PDE. More emphasis is placed on the information coming from the direction of the flow – the upwind values. Since only the advective parts of the PDE have direction then upwinding techniques need only to be applied to this part of the problem.

Upwinding ensures a stable solution, however, the problem may still display unphysical behaviour in the form of spurious extrema. These result from using too high an order scheme. This can be overcome by using a non-linear scheme that changes order near discontinuity by limiting the numerical flux that passes between cells. It is possible to theoretically validate that such schemes do not introduce spurious oscillations provided certain conditions are met, see [13].

If a one dimensional problem is considered, a time dependent system of PDEs can be written in conservative form as

$$\underline{u}_t + [\underline{F}(\underline{u})]_x = [\underline{G}(\underline{u}, \underline{u}_x)]_x$$

where $\underline{F}$ defines the advective flux vector describing the wave like motion of the PDE and

$\underline{G}$ defines the diffusive flux vector describing the diffusion of the solution.

There is no need to use upwinding when evaluating $\underline{G}$, see [74], so central difference like techniques can be employed. The flux $\underline{G}$ is evaluated by either calculating $\underline{G}$ in the left and right cells and then averaging or $\underline{G}$ is calculated with some average value of the solution in the left and right cells.

The advective flux $\underline{F}$ requires upwinding to ensure correct discretisation. Some process is needed to choose which direction the solution is to be upwinded. For simple problems this is obvious and unchanging, e.g. linear advection. However, for complex systems it is not obvious and the direction may alternate or a combination of both left and right values maybe needed for a system of PDEs, see [67]. Typically a Riemann solver is used that replaces the advective flux in the code and uses a combination of knowledge about the PDE and left and right solution values to construct the flux. It is possible to construct exact Riemann solvers however this is generally complex and costly. Often approximate Riemann solvers that do not compute exactly the correct flux but instead use a series of physical/numerical approximations to ease the task. This is acceptable since numerical errors have already been introduced in the discretisation procedure, see [67].

Although it is possible to construct approximate Riemann solvers automatically using symbolic manipulation packages, the current trend is still the user specifying the Riemann solver rather than the advective flux found in the conservative form of the PDE. One possible strategy is to average the left and right solution values and evaluate the advective flux function using this in a similar way to the diffusive flux. Although this ignores the upwind information presented to the Riemann solver it is a default action that can be taken if the user does not supply a proper Riemann solver.

When dealing with two dimensional problems the current trend is towards unstructured triangular meshes. They have the ability to model complex domains and when used in conjunction with spatial adaptivity provide a powerful modelling environment. The numerical solution process considered here uses a cell–centered finite volume scheme on unstructured triangular meshes, [87].

The physical domain is represented by a triangular mesh, each mesh point is the centroid of a triangle. The conservative form of the two dimensional equation is given in Section 3.2. The advective fluxes are dealt with in a similar way to the one dimensional scheme with an approximate Riemann solver being employed to calculate the advective flux and simple averaging used for the diffusive flux. The Riemann solver is now used to

compute the fluxes travelling across the edge in the Cartesian directions. The left and right values used by the Riemann solver are now *internal* and *external* values. These values are again limited using a flux limiter scheme but now implemented on an unstructured mesh. Only the use of this unstructured flux limiter scheme is covered here, for more details see [13].

As for the one-dimensional case, initial and boundary conditions need to be specified. There are three types of boundary conditions allowed by the package:

- Dirichlet Condition - allows the user to specify the dependent variable(s) at the boundary.
- Neumann Condition - allows the user to specify the first derivative of the dependent variable(s) normal to the boundary edge.
- Flux Condition - allows the user to replace the approximate Riemann solver at the boundary by a specified flux. Due to a feature of the flux limiter scheme this will result in a loss of accuracy at the boundary.

The numerical solution process requires the user to provide several pieces of information in order to solve the PDE. This information will be discussed in the next section.

3.4 The Numerical Code

The numerical software used here is the SPRINT2D package. It is a general purpose solver for systems of PDEs in conservation law form. The PDEs must fit into the following template class of PDEs as given below,

$$\beta(\underline{U})\frac{\partial \underline{U}}{\partial t} + \frac{\partial}{\partial \underline{x}}\underline{F}(\underline{U}) = \frac{\partial}{\partial \underline{x}}\underline{G}\left(\underline{U}, \frac{\partial \underline{U}}{\partial \underline{x}}\right) + \underline{S}(\underline{U})$$

where $\underline{x}$ is $[x, y]$ is the Cartesian position, $\underline{F}(\underline{U})$ defines the advective fluxes, $\underline{G}(\underline{U}, \frac{\partial \underline{U}}{\partial \underline{x}})$ are the diffusive fluxes and $\underline{S}(\underline{U})$ are the source terms. For steady problems the β term is set to zero to eliminate the time-dependent aspect. Typically, for time-dependent problems β is set to unity.

The SPRINT2D package has several features including:

- unstructured triangular mesh spatial discretisation;
- unstructured triangular mesh generation;
- spatial adaptivity;
- time integration with local error control;
- error estimation and error control.

Examples of the types of PDE problems that SPRINT2D has been applied to are given in [10].

The program is driven by a user-supplied driving program. This driver program performs two tasks. Firstly, it specifies the PDE(s) to SPRINT2D. Secondly, it allows the user to specify the solution techniques to be used and pass over any miscellaneous information and *hints* about how to be successful. The software takes the form of a selection of numerical modules controlled by the SPRINT2D main driver. The user, in the driver program, needs to specify the following information:

- A Riemann solver for the advective fluxes $\underline{F}$.
- The flux function for $\underline{G}$.
- The source term function $\underline{S}$.
- Boundary conditions.
- Initial conditions.
- A time interval over which to integrate (for time-dependent problems).
- A file containing a specification of the physical domain.
- Relative and absolute tolerances for the adaptivity routines.

Extensive utility routines allow the user to provide additional information or examine the solution during the solution process.

The SPRINT2D package is implemented on top of two existing numerical packages: SPRINT and NAESOL. After applying spatial discretisation to time-dependent problems, the resulting system of ODEs is solved by the SPRINT integration package.

Spatially discretising steady problems removes all differential operators and results in a system of non-linear equations which are solved by the non-linear solver package NAESOL. The details of these packages are masked by SPRINT2D which provides a unified interface since the two packages have significant differences in their interface.

There are currently two mesh generation software packages that can be used by SPRINT2D. The KSLA mesh generator [35] and the GEOMPACK mesh generator [51]. The TRIAD package provides the routines to perform any spatial adaptivity. An h–refinement method is used to refine and coarsen the computational mesh.

All SPRINT2D routines begin with `S2D_` and specific solution modules have a further set of letters to identify them. So the routine `S2D_FVM_diffusive_flux` has the code `FVM` to show that it is part of the `F`inite `V`olume `M`ethod solution module.

There is a large overlap between the information required for time-dependent and steady problems. The main differences occur internally to SPRINT2D due to the way in which the PDEs are solved. A time-dependent problem requires the user to specify which time integration method is required as well as the information about the time interval. These problems also require the user to specify the linear algebra package for the time integrator. Figure 3.1 shows the structure of a time dependent problem.

The steady problems are solved in a different way and need a solution strategy module, for solving the non-linear equations, to be specified. Figure 3.2 shows the structure of a steady problem.

The modular nature of the software allows additional solution modules to be added to the package. Whilst at present there is often only one option the nature of the package allows for additional routines to be added later. A brief outline of the type of modules available in the current version of the package is outlined below.

- **Spatial discretisation modules**: there is currently only one spatial discretisation module, the finite volume scheme (FVM) outlined previously.
- **Solution strategy modules**: again there is one module, this is the black box solution strategy module (BBOX). This is a simple invocation of the NAESOL package.
- **SPRINT linear algebra modules**: there are currently three linear algebra modules available. A dense direct linear algebra module (SLINPK), a sparse direct linear algebra module (PSPARSE) and a sparse iterative module (SPWATSIT).

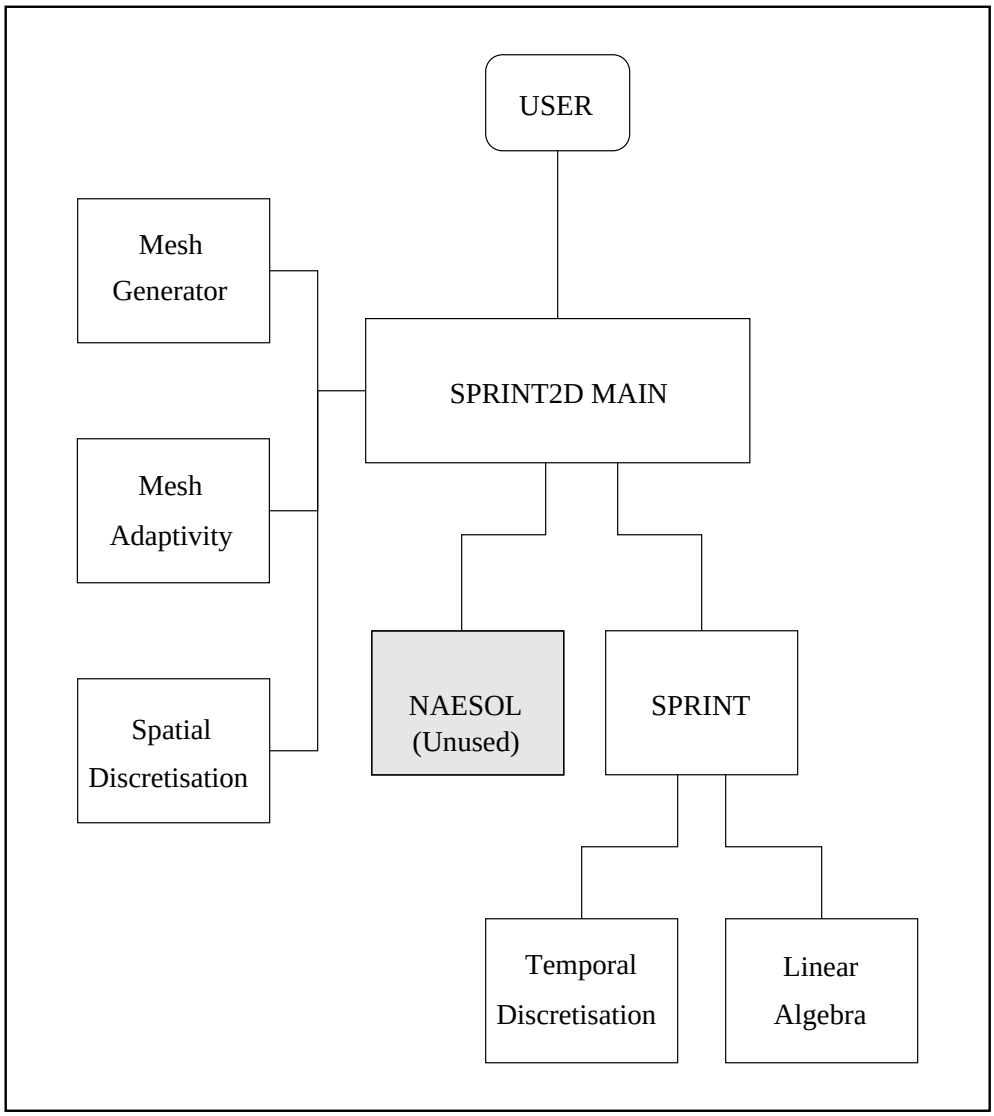

Figure 3.1: The Architecture of Time Dependent Problem

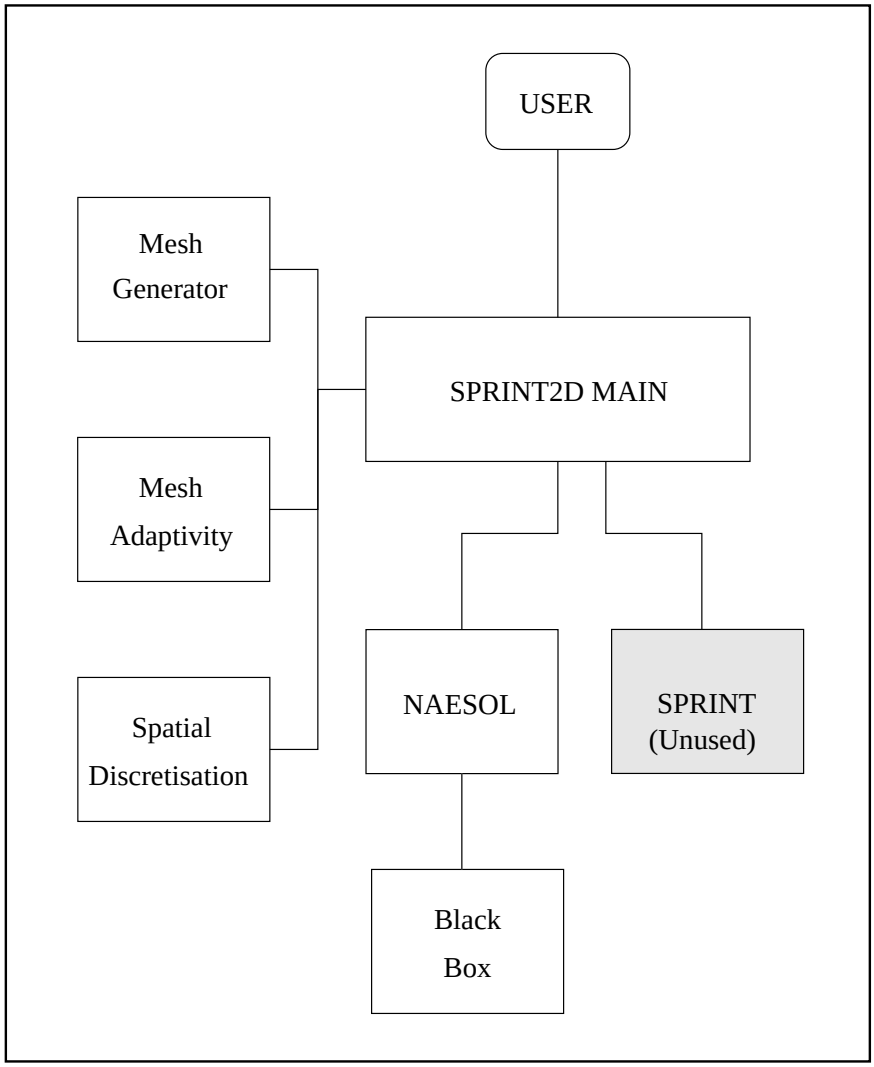

Figure 3.2: The Architecture of Steady Problem

- **SPRINT time integration modules**: there are currently two time integration modules. A Theta method (STHMID) and a backward differentiation formula method (SPDASL).

The next section will examine the way in which the user specifies the relevant information to the package.

3.5 Driving the Numerical Code

The use of a driving program is a common way to drive PDE solvers. The user is required to specify the solution process and the modules to be used. The user also needs to specify tolerances and numerical parameter values. This process is normally done in some high level language. Often the numerical code will have its own language, for example the ELLPACK system [72] and the PDEQSOL system [86] both have an internal high level language to specify the problem. Both then convert this internal language into a traditional programming language (FORTRAN).

The SPRINT2D package driver programs are written in the C programming language [73]. The SPRINT2D modules are selected and initialised from within this C program. The driver program also allows the user to extract information about the numerical solution each time it changes or is updated. This is achieved by the user providing a *monitor* routine which SPRINT2D calls at regular intervals. The layout of the driver program follows the general form given below.

- Include header files for each module to be used by the code.
- Define any functions required by the modules used.
- Define the monitor routine.
- Specify the relevant data needed for the solution process.
- Initialise the modules used by the code.
- Start the solution process.

To better illustrate the structure of the driving program we will consider two example problems. The first is a steady problem and the second a time-dependent one.

Both use the finite volume spatial discretisation module. The steady problem uses the Black Box solution strategy module whilst the time dependent problem uses the Theta integration module and the WATSIT linear algebra module.

The first part of the driver program needs to include the relevant header files for the SPRINT2D package and modules that are to be used. The header files for the steady problem are therefore:

```
#include "S2D.h"
#include "S2D_FVM_finite_volume_discretisation.h"
#include "S2D_BBOX_black_box_soln_strategy.h"
```

The header files for the time dependent problem are:

```
#include "S2D.h"
#include "S2D_FVM_finite_volume_discretisation.h"
#include "S2D_STHMID_theta_temporal_discretisation.h"
#include "S2D_SPWATSIT_sparse_iterative_solver.h"
```

The driver program then specifies the various functions required by the modules to be used. In both cases the finite volume spatial discretisation requires several functions: initial conditions, boundary conditions, diffusion function, Riemann solver. These functions have to follow a specific interface where certain variables have to be returned to SPRINT2D, eg numerical flux between cells in the Riemann solver. SPRINT2D calls these functions with spatial and time information (`x`, `y` and `t`), as well as the number of PDEs (`npde`) and other information relevant to each particular function, eg the boundary conditions routine is given the name of the boundary edge to which it is being applied.

- **Initial conditions**: this function specifies the initial solution values for the PDEs. This function has to set the array `u[npde]` where the user's function must have the following prototype:

```
void problem_ic(TRIAD_Triangle *tri, int npde, double x,
                double y, double t, int sub_name,
                void *users_data, double u[])
```

- **Boundary conditions**: this function specifies the boundary conditions for each PDE. This function has to set the type of boundary condition `type[]` and then,

depending on the type set, one of the following `dubdn[]`, `ub[]` or `fb[]` if the conditions are Neumann, Dirichlet or flux respectively. The user's function must have the following prototype:

```
void problem_bc(TRIAD_Line *line, int npde, double x,
                double y, double time, int edge_name,
                int sub_name, double norm_x, double norm_y,
                double *u, void *users_data,
                S2D_FVM_BC_Type type[], double ub[],
                double dubdn[], double fb[])
```

- **Diffusive function**: this function defines the diffusive flux part of the PDE. The user sets the diffusive fluxes in the Cartesian directions, `g_x[]` and `g_y[]`, given the first derivatives in the Cartesian directions, `dudx[]` and `dudy[]`. The user's function must have the following prototype:

```
void problem_g(TRIAD_Line *line, int npde, double x,
               double y, double t, int sub_name,
               double norm_x, double norm_y, double u[],
               double dudx[], double dudy[],
               void *users_data, double g_x[],
               double g_y[])
```

- **Riemann solver**: this function defines the approximate Riemann solver used by the spatial discretisation code to calculate the advective flux passing between cells. This function has to return the numerical flux in `nf[]` given the left and right values at the edge, `u_l[]` and `u_r[]`. The user's function must have the following prototype:

```
void problem_rs(TRIAD_Line *line, int npde, double x,
                double y, double t, int sub_name,
                double norm_x, double norm_y, double u_l[],
                double u_r[], void *users_data, double nf[])
```

The user can also specify a monitor function. This function allows the user to follow the numerical solution. The monitor function is called at several different stages

during the integration but mainly it is invoked when the next successful solution has been generated. For time-dependent problems this is when the solution has been advanced by a time step. For steady problems this is when the solution has converged on the current adaptive mesh. Typically the monitor function connects up to some visualisation system to allow the user to investigate the current solution.

```
void monitor(int neq, double u[], double udot[],
             double time, double err[], double ewt[],
             S2D_Res_Status_Type res_status,
             S2D_Monitor_Type mon_type,
             S2D_Mon_Status_Type *mon_status,
             S2D_Intgrtn_Obj_Type *integ_obj,
             void *space_disc_data, void *users_data )
```

In the main function, the user passes over the numerical information needed by SPRINT2D and tells SPRINT2D which functions it needs to invoke. The user declares an integration object, `my_integ` for example, and this carries the bulk of the information. Utility routines exist to allow the user to specify different kinds of information. The integration object is initialised with default settings for most parameters. The user can overwrite these later if he/she wishes. There are certain parameters that have to be specified: maximum number of unknowns (e.g. ODEs or non-linear equations), the temporal domain, the initial level of mesh refinement and tolerances to specify the accuracy the solver should try to obtain. Although its possible to specify these values explicitly when invoking the utility, most driver programs query the user for suitable values.

As an illustration of the driving program, this Section will list and briefly describe the SPRINT2D utility routines in a typical driving program. In both cases the integration object needs to be initialised with default values

```
S2D_initialise( &my_integ );
```

The type of problem is then given, steady or time-dependent. For a time dependent problem the user gives SPRINT2D a start time `t_start` and an array, `t_out`, containing specific times at which output is requested. SPRINT2D will invoke the monitor function at these times. The last entry of t_out is the time at which the integration will cease.

```
S2D_time_dependent( &my_integ, 1, neqmax, t_start, n_t_out,
```

```
                                t_out ) ;
```

where `n_t_out` specifies the number of entries in the `t_out` array. The `neqmax` gives the maximum number of unknowns to be allowed by the code. The driving program now sets the temporal tolerances to specify the accuracy to which the solution is required.

```
S2D_temporal_tol( &my_integ, 1.0e-7, 1.0e-7 ) ;
```

where the second argument is absolute tolerance and the third is the relative tolerance. See [10] for an explanation of these types of tolerance. The steady problem is defined in a similar way, except no time information is needed.

```
S2D_steady( &my_integ, 1, neqmax ) ;
```

Both steady and time-dependent problems will need the computational mesh generating from the textual description of the domain. SPRINT2D needs to know the name of the file containing the description of the domain, for example `"problem.dmn"` and the initial level of mesh refinement, `ilevel` to be inserted into the mesh.

```
S2D_ksla_mesh_generator( &my_integ, "problem.dmn", ilevel ) ;
```

The user now needs to specify information about the spatial adaptivity. One option is to disable spatial adaptivity.

```
S2D_no_spatial_adaptivity( &my_integ ) ;
```

Alternatively, if the spatial adaptivity is enabled then absolute and relative spatial tolerance values, `atol,rtol`, are needed to control the spatial error.

```
S2D_spatial_tol( &my_integ, S2D_Scalar_TOL, &atol, &rtol ) ;
```

This concludes the core SPRINT2D information that needs to be specified. Now the different solution technique modules need to be initialised. For example, if the finite volume spatial discretisation module is to be used then it needs to know where to find the functions that specify different parts of the template equation.

```
S2D_FVM_initialise( &my_integ ) ;
S2D_FVM_initial_conditions( &my_integ, problem_ic ) ;
S2D_FVM_boundary_conditions( &my_integ, problem_bc ) ;
S2D_FVM_diffusive_flux( &my_integ, problem_g ) ;
S2D_FVM_riemann_solver( &my_integ, problem_rs ) ;
```

The other solution technique modules to be used are then specified. For steady problem the black box module could be used.

```
S2D_BBOX_initialise( &my_integ, -1.0 ) ;
```

where the value -1.0 implies solve to machine accuracy. For time-dependent problems the integration and linear algebra methods modules require initialisation.

```
S2D_STHMID_initialise( &my_integ, S2D_STHMID_Theta, 4,
                       S2D_STHMID_FI, S2D_STHMID_NoSwitch,
                       0.55 ) ;
S2D_SPWATSIT_initialise( &my_integ, 10.0, 10.0 ) ;
```

The final initialisation is for the user's monitor function, declared earlier in the driving program.

```
S2D_monitor( &my_integ, monitor ) ;
```

Once all the required parts have been specified and any other additional information the user wishes to supply have been processed then the integration can proceed.

```
S2D_integrate( &my_integ ) ;
```

The minimum requirements to drive the numerical software are outlined above. As a further example of a driver programs, the program for a simple time dependent Heat Equation is shown. The PDE is given by the following equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

The initial conditions are set to 0.0 over the domain, the domain itself and boundary conditions are shown in Figure 3.3. A full description of this problem is given later in Chapter 5. The driver program below is a slightly modified version (to conserve space) of one that was produced by the Visual Problem Specification system described in Chapter 5. The aim of this program is to illustrate how the different components described above are used in the final code generated to solve the problem.

```
/* Standard Include Files */
#include <stdio.h>
#include <stdlib.h>
```

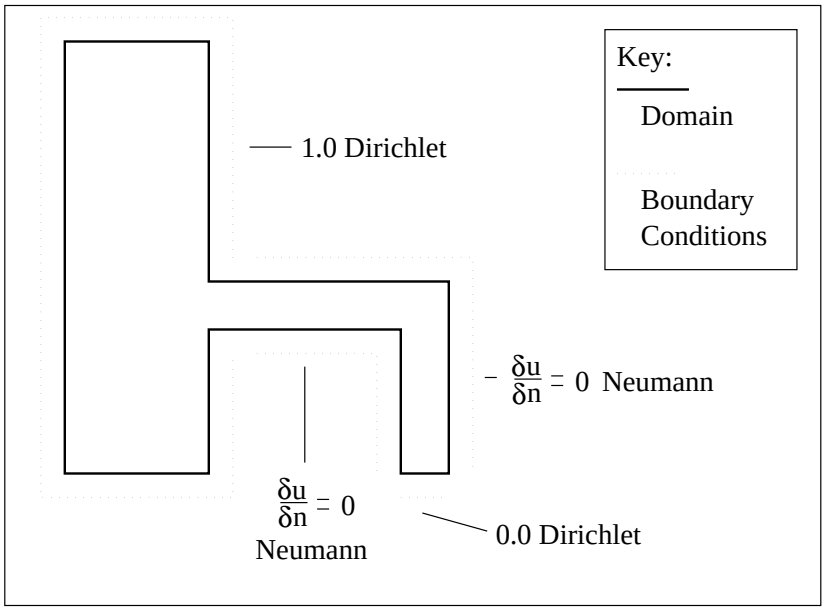

Figure 3.3: The Domain for the Heat Equation and Boundary Conditions

```
#include <math.h>
#include "S2D.h"

/* Include file for FVM */
#include "S2D_FVM_finite_volume_discretisation.h"

/* Time Dependent Problem Include Files */
/* Include file for theta method integrator */
#include "S2D_STHMID_theta_temporal_discretisation.h"

/* Include file for watsit linear algebra package */
#include "S2D_SPWATSIT_sparse_iterative_solver.h"

/* Include file for visual routines */
#include "vwr_comms.h"

/* Monitor Routine */
void monitor (
  int                   neq,
  double                *u, *udot, time, *err, *ewt,
  S2D_Res_Status_Type   res_status,
  S2D_Monitor_Type      mon_type,
  S2D_Mon_Status_Type   *mon_status,
  S2D_Intgrtn_Obj_Type  *integ_obj,
```

```
  void                  *space_disc_data,
  void                  *users_data)
{
  /* ---------------------------------------- */
  /* Monitor Routine for user functions, allows  */
  /* user to monitor process insert things here  */
  /* ---------------------------------------- */

  int        interact;

  /* Viewer routine */
  if ((mon_type == S2D_Initial_Mon) || (mon_type == S2D_PostAdapt_Mon))
    vwr_send_mesh(integ_obj);
  if (mon_type == S2D_Initial_Mon)
    vwr_send_frame(integ_obj,u,u,time,&interact);
  else
  if (mon_type == S2D_Step_Mon)
    vwr_send_frame(integ_obj,u,err,time,&interact);

  /* Return monitor ok status */
  *mon_status = S2D_Mon_Okay;
} /* Monitor routine */

/* Initial conditions routine */
void parabolic_ic(
  TRIAD_Triangle  *tri,
  int             npde,
  double          x, y, t,
  int             sub_name,
  void            *users_data,
  double          *u)
{
  /* ---------------------------------------- */
  /* Initial conditions routine to specify the   */
  /* initial conditions of the problem           */
  /* ---------------------------------------- */

  u[0] = 0.0;
} /* Initial conditions */

/* Boundary conditions routine */

void parabolic_bc(
  TRIAD_Line      *line,
```

```
  int             npde,
  double          x, y, time,
  int             edge_name, sub_name,
  double          norm_x, norm_y, *u,
  void            *users_data,
  S2D_FVM_BC_Type *type,
  double          *ub, *dubdn, *fb)
{
  /* ---------------------------------------- */
  /* Boundary conditions routine to specify the  */
  /* boundary conditions of the problem          */
  /* ---------------------------------------- */

  double    t = time;
  if ( ( edge_name >= 221 )   && ( edge_name <= 225 ) ) {
    type[0] = S2D_FVM_Dirichlet;
    ub[0] = 0.1E1;
  } if ( ( edge_name >= 226 )   && ( edge_name <= 227 ) ) {
    type[0] = S2D_FVM_Neumann;
    dubdn[0] = 0.0;
  } if ( ( edge_name >= 228 )   && ( edge_name <= 228 ) ) {
    type[0] = S2D_FVM_Dirichlet;
    ub[0] = 0.0;
  } if ( ( edge_name >= 229 )   && ( edge_name <= 230 ) ) {
    type[0] = S2D_FVM_Neumann;
    dubdn[0] = 0.0;
  }
} /* Boundary conditions */

/* Diffusive function */

void parabolic_g(
  TRIAD_Line  *line,
  int         npde,
  double      x, y, t,
  int         sub_name,
  double      norm_x, norm_y, *u, *dudx, *dudy,
  void        *users_data,
  double      *g_x, *g_y)
{
  /* ---------------------------------------- */
  /* Diffusive function routine to specify the   */
  /* Diffusive function part of the problem      */
  /* ---------------------------------------- */
```

```
  double    ux = dudx[0];
  double    uy = dudy[0];

  g_x[0] = ux;
  g_y[0] = uy;
} /* Diffusive function */

/* Riemann solver */

void parabolic_rs(
  TRIAD_Line  *line,
  int         npde,
  double      x, y, t,
  int         sub_name,
  double      norm_x, norm_y, *u_l, *u_r,
  void        *users_data,
  double      *nf)
{
  /* ---------------------------------------- */
  /* Riemann solver routine to specify the    */
  /* Riemann solver conditions of the problem */
  /* ---------------------------------------- */

  double u = ( u_l[0] + u_r[0] ) / 2.0 ;
  double f_x, f_y;

  /* ---------------------------------------- */
  /* A Simple Differencing is used to solve   */
  /* this problem. More complex methods may   */
  /* provide better results                   */
  /* ---------------------------------------- */

  f_x = 0.0;
  f_y = 0.0;
  nf[0] = f_x * norm_x + f_y * norm_y;
} /* Riemann solver */

/* Source Term */

void parabolic_src(
  TRIAD_Triangle  *tri,
  int         npde,
  double      x, y, t,
```

```
  int         sub_name,
  double      *u,
  void        *users_data,
  double      *src)
{
  /* ---------------------------------------- */
  /* Source Term routine to specify the       */
  /* Source Term part of the problem          */
  /* ---------------------------------------- */

  src[0] = 0.0;
} /* Source Term */

/* Main */
main(int argc, char *argv[])
{
  S2D_Intgrtn_Obj_Type my_integ;
  int                  ntrimax = 10000;
  int                  ilevel = 4;
  int                  n_t_out = 6;
  int                  t_start = 0.0;
  static double        t_out[] = {0.25, 0.50, 0.75, 1.00, 1.25, 1.50};
  double               atol = 0.050000;
  double               rtol = 0.050000;
  double               max_x = 0.900000, min_x = 0.100000;
  double               max_y = 0.900000, min_y = 0.100000;
  /* End of variable declarations */

  /* Initialise integration object */
  S2D_initialise(&my_integ);

  /* Time Dependent */
  /* Specify a time dependent integration */
  S2D_time_dependent(&my_integ,1,ntrimax,t_start,n_t_out,t_out);

  /* set temporal tolerance */
  S2D_temporal_tol(&my_integ,1.0e-7,1.0e-7);

  /* Spatial tolerance on */
  S2D_spatial_tol(&my_integ,S2D_Scalar_TOL,&atol,&rtol);

  /* Use Finite Volume scheme */
  S2D_FVM_initialise(&my_integ);
  S2D_FVM_initial_conditions(&my_integ,parabolic_ic);
```

```
  S2D_FVM_boundary_conditions(&my_integ,parabolic_bc);
  S2D_FVM_riemann_solver(&my_integ,parabolic_rs);
  S2D_FVM_diffusive_flux(&my_integ,parabolic_g);
  S2D_FVM_source_term(&my_integ,parabolic_src);

  /* Initialise viewing routine */
  vwr_init( min_x, max_x, min_y, max_y, 1.4);

  /* Set KSLA mesh information */
  S2D_ksla_mesh_generator(&my_integ,"parabolic2.dmn",ilevel);

  /* Use theta method integrator */
  S2D_STHMID_initialise(&my_integ,S2D_STHMID_Theta, 4,
          S2D_STHMID_Newton, S2D_STHMID_NoSwitch, 0.55 );

  /* Use watsit sparse iterative solver */
  S2D_SPWATSIT_initialise(&my_integ,10.0,10.0);

  /* monitor routine */
  S2D_monitor(&my_integ,monitor);

  /* Integrate routine */
  S2D_integrate(&my_integ);

  /* close viewer */
  vwr_close();
}
/* End Of Driver */
```

3.6 Output of the Numerical Code

SPRINT2D furnishes the monitor function with a large amount of information each time it is called. The user can choose to use or ignore this information as he/she sees fit. Each triangle has a solution value, an error in space value and, for time-dependent problems, a temporal error value. The code can also provide a large quantity of spatial information about the unstructured mesh such as areas of triangles, lengths of edges, unit normals to edges etc.

This information is used by the visualisation package which complements the SPRINT2D solver. This visualisation package is developed in IRIS GL [91] and runs on a local host whilst the SPRINT2D runs on a computationally intensive platform elsewhere.

Solution frames are sent across the network to the visualisation package within which the user can interrogate the solution whilst the next frame is being calculated. The visualisation package displays the solution values for each triangle in the spatial mesh and error estimates in space and time. New frames are sent from the monitor function they can be every time step or at a fixed times requested by the user in `t_out[]`.

Simple interaction is possible by return values sent to SPRINT2D from the visualisation package. These values can instruct the SPRINT2D application to save the current solution in either a SPRINT2D proprietary format or as a *pyramid* file to be used with the IRIS Explorer visualisation system.

3.7 Summary

In this chapter the numerical solution of PDEs is outlined. The general solution method is discussed via the method of lines approach to constructing PDE software. A brief overview of the types of spatial discretisation schemes available is given. The finite volume scheme is then discussed in greater detail. Initially in one dimension and then in two dimensions on unstructured triangular meshes.

The SPRINT2D package is described as an example of a general purpose solver for a certain class of PDEs. The modular nature of the code is highlighted, a feature common to much PDE numerical software. The information required by SPRINT2D is categorised.

The construction of a suitable driving program to invoke SPRINT2D is explained. The layout of the driver program is shown for both steady and time-dependent problems.

Chapter 4

A Visual Domain Specification Tool

4.1 Introduction

An important part of the specification process for solution of two dimensional convection–dominated PDEs is the definition of the region over which the problem is to be solved. Given the increase in computing power and the speed at which numerical solutions can be found, this aspect of the specification process is one which is taking an increasing percentage of the overall analysis time. Once a numerical specification of a geometry or domain is given, this numerically defined region can then be meshed to provide a suitable domain for the numerical solution process to work with.

The generation of a suitable model for the numerical solution process from a geometric description is one area where automation is desired, see [76]. With the widespread availability of numerical software to solve PDEs, mesh generation methods have changed. The traditional bottom–up mesh generation method, in which the user is required to specify the nodal points, is now seen as outdated. As more and more general purpose solvers become available they aim to automate many of the aspects of the solution process, eg. error control to a prescribed tolerance. An increase in the use of semi–automatic and fully automatic mesh generation packages has accompanied this. Both these methods aim to simplify the generation process of a numerical model that the solution process uses, to complement the increasing use of automation within the solution process.

Semi–automatic techniques require the geometry description and mesh parameters which are used to determine how the geometry is to be split into subregions and meshed. Subregions are constructed such that they can be meshed easily. One example of this type of package is GEOMPACK [51]. A full description of the GEOMPACK system is given later in Section 4.5. The fully automatic mesh generation system requires only the geometry description. The software itself is responsibility for the final mesh. An example of this is the KSLA mesh generation package [35], again a full description of the information required by this package is given in Section 4.5.

This chapter looks at the construction of a visual domain specification (VDS) tool for the easy specification of the initial domain so that it can be meshed, thus reducing the time spent on that part of the problem specification process. The VDS tool is concerned with the specification of geometry that defines the initial domain, the prototype system outlined here does not attempt to provide an interface to a geometric modeller, unlike [76].

The output requirements of the VDS tool are examined here and different input formats that particular mesh generation software requires are discussed. Two mesh generation packages are discussed: one is in the class of semi–automatic mesh generators, the other is a fully automatic package. The different formats required by each are outlined and the VDS tool shown to be subsumed by a generic input medium to both. A third mesh generation package is also considered and the suitability of the VDS tool to provide valid input is examined. The postprocessing system of the tool undertakes the required tasks of producing suitable input for the individual mesh generation software packages.

The design objectives behind the tools and the development of these objectives over time are discussed. The design process of the tool was based on an iterative feedback loop, additional features being added to the VDS tool from user requests. The basic elements such a tool requires are discussed in greater detail in Section 4.4. These elements can be summarised as:

- A suitable way for the user to specify the geometry.
- A way for the user to visualise a coarse mesh defined over the domain.
- The means for the user to control the tool and the underlying data structure required to store the geometry information.

The evolution of the tool is then discussed from a minimal initial specification. The changes to the VDS tools are outlined and related to the key areas of the VDS tools. Justification for the changes will be given in the form of user requests or user observations about the tool.

Finally some evaluation of the tool will be undertaken. This will take the form of user feedback about the VDS tool. The advantages that the tool provides will be highlighted such as the decrease in the time taken in the specification of new domains, the conversion of a set of existing points into a domain and the ability to modify previously defined domains.

4.2 Existing Interfaces to Mesh Generators

Many other PDE systems allow the users of the system to graphically define the geometry over which the problem is to be solved. The Visual PDEQSOL system [86] allows the user to input a region through the Model Visualizer component of its system. The system uses a X–based tool using the OSF/Motif Widget Set to ensure portability. The Model Visualizer part of the system is also concerned with the specification of the PDE and constants involved as well as the initial conditions and boundary conditions for the domain.

The //ELLPACK system [47] also uses geometry specification tools for two and three dimensional domains. Again these tools are built in X to ensure portability. The Eve system [5] also has an input device driven by the mouse for the creation of, what it terms, *simple 2D domains*. The interface is again built in X–Windows and links the boundary conditions to the geometry by means of naming the edges.

Finally the RPI system [65] makes use of the Quadtree mesh procedure. The Quadtree mesh generator interface allows the user to construct complex geometries using many different drawing primitives. These include Bézier curves, splines, quadratic and cubic curves as well as lines, arcs and circles. The user can input points via the keyboard or mouse. The interface also allows the user to specify the model attributes as well as the geometry and to refine the computational mesh near certain points in the geometry.

4.3 Aim of the VDS Tool

The initial objective was to design a VDS tool that allows the user to visually construct a numerical domain specification file for use with the mesh generation package used by the numerical software described in Chapter 3. From this it was hoped that the general principles about the construction of suitable tools for PSEs could be determined. As the VDS tool evolved over time this objective changed to that of producing a portable software tool that allowed the user to visually specify, and manipulate, complex geometries via line and arc primitives.

The VDS tool allows the user to construct, manipulate and modify geometries, thus providing a generic interface for the construction of two dimensional geometries. The internal data structure representing the domain can, if required, be supplemented with additional user defined information, to produce suitable input for mesh generation software packages via postprocessing routines attached to the tool.

The other aims of the VDS tool are to speed up the process of specifying complex geometries, thus reducing the overall time spent on the solution of problems. The VDS tool provides a more natural way to define the geometry through a visual specification system which is easier and more convenient for the users of the numerical software. Meeting these objectives should enable the VDS tool to be a successful component of a possible PSE surrounding the numerical software.

The divorce of the visual specification from the postprocessing required to obtain a suitable numerical specification for the mesh generation software allowed the issues behind the construction of a geometry specification system to be explored. The result of this investigation was a generic tool for the specification of two dimensional regions and a postprocessing system capable of producing information for mesh generation software packages.

The VDS tool uses an internal data structure to construct the geometry. This information is then transferred to an output file suitable for use with mesh generation software via a postprocessing routine. This intermediate step allowed the separation of the visual specification process and the creation of the numerical domain specification file although these two processes are closely linked.

Initially the mesh generation software used was the KSLA mesh generator [35]. This was due to the use of this mesh generation package within the SPRINT2D project

[9]. The KSLA mesh generator is of the class of fully automatic mesh generation packages. If this is the selected mesh generation package the tool can utilise the mesh generation software to display to the user a representation of the resulting mesh produced for the specified domain.

In addition to this another mesh generation software package was also catered for, this was to show that the tool has suitable generic properties. An additional postprocessor was constructed to allow an output file suitable for the GEOMPACK mesh generator [51] [52]. GEOMPACK is a public domain mesh generator and belongs to the class of semi–automatic mesh generators. GEOMPACK represents a very different way of specifying a geometric model. A brief examination of the mesh requirements of the PLTMG package [4], another semi–automatic mesh generator, has also taken place. It will be shown that, in principle, the VDS tool can be used to produce valid input for the PLTMG mesh generation routines, this strengthening the claim that the VDS tool may act as a specification medium for several mesh generation packages given suitable post–processing routines.

The input to the first two mesh generation packages mentioned takes the form of a numerical specification file. This file based output was chosen to enable the separation of the visual specification of the geometry and the production of the numerical specification. Such a system allows both the tool and the mesh generation software to evolve.

To strengthen the generic nature of the VDS tool the X Window system [54] and the OSF Motif Widget Set [94] were chosen to build the user interface part to the tool. The data structure manipulation and postprocessing parts are built using the C programming language and overall the VDS tool is some 6000 lines of code. The use of these building blocks helps to ensures portability the need for which has already been discussed in Chapter 2.

The aim is to construct a prototype with the basic elements of the visual specification system incorporated in it. The VDS tool can then be refined with additional features being added to the tool from its initial state based on user feedback. This process will thus involve the users of the VDS tool providing feedback on features that are required. The next section will identify the basic elements require for the VDS tool, the roles they fulfill and the justification for their inclusion into the initial prototype state.

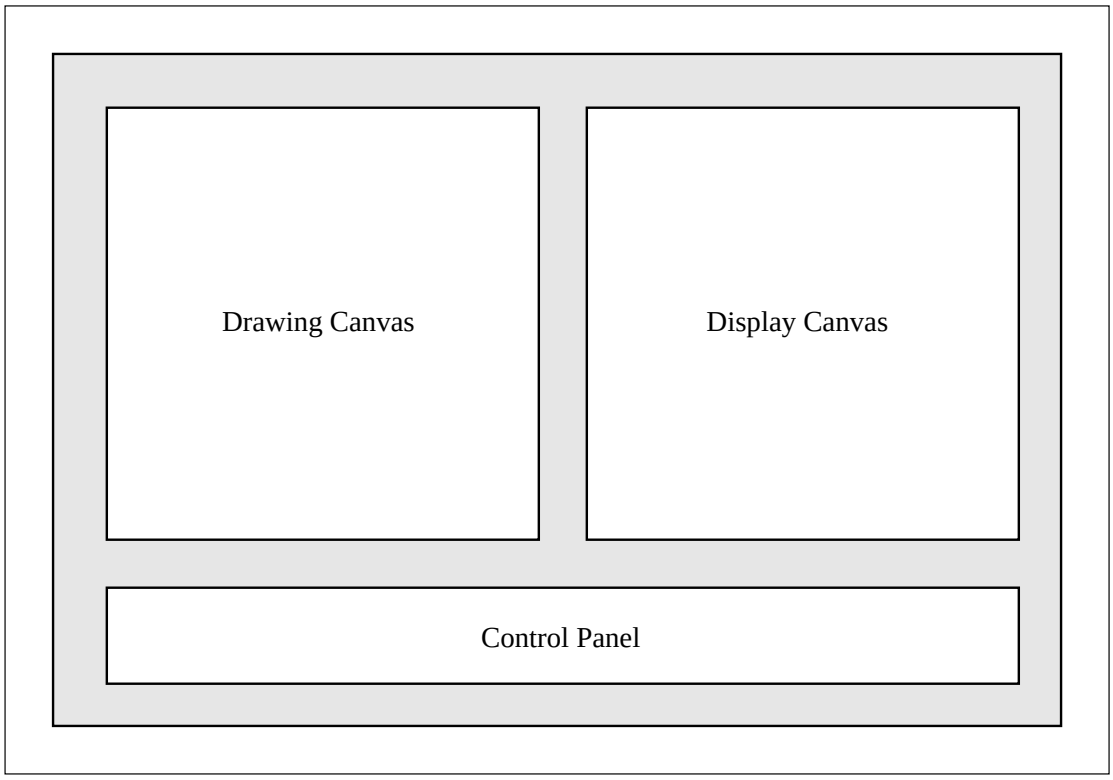

Figure 4.1: The Basic Elements for the Prototype VDS Tool

4.4 Construction of the Prototype

To construct the prototype first requires the identification of the basic elements needed for the domain specification tool. After identifying these elements they can then be combined into a primitive software tool.

This initial design can then be used as the starting point of the iterative loop to refine and strengthen the VDS tool with user feedback providing justification for the changes made.

The domain specification tool is split into three main visual components, see Figure 4.1, these are

- A Drawing Canvas, an area where the user can specify the geometry using the mouse as an input device.
- A Display Canvas, an area that will show the result of calling the KSLA mesh generation software with the user specified geometry.

- A Control Panel, containing a collection of buttons and labels that allow the user to control the tool.

As well as the visual components of the tool a suitable internal data structure needs to be defined. All these areas are explained in more detail below. The evolution of the VDS tool is described in a later section, the changes made from user requests will apply to one of the four areas outlined above.

4.4.1 The Drawing Canvas

The drawing canvas is an area within the VDS tool that will accept mouse movement and mouse button presses as input. The drawing canvas allows the user to visually construct a geometry. As input is received the canvas will provide a visual representation of the geometry to the user. As the geometry is created and displayed the internal data structure will store the information.

4.4.2 The Display Canvas

The Display Canvas is an area that will take as input a KSLA input domain specification file and call the KSLA mesh generation software to produce a mesh using the numerical specification file. The canvas will then display the resulting meshed domain with an additional level of refinement, each triangle in the original mesh represented by four triangles created from bisecting the edges of the triangles. At the initial prototype stage the KSLA mesh generator was the primary mesh generation software package used. The display canvas is therefore driven by the KSLA mesh generator.

4.4.3 The Control Panel

The Control Panel is a collection of buttons that provide the user with the commands needed to manipulate the VDS tool. In the prototype VDS tool the following commands were considered to be the bare minimum for the use of the VDS tool.

- A Quit Button to kill the application
- A Mesh Design Button that will prompt for a filename, produce a numerical specification file and pass this to the Display Canvas which will show the resulting mesh.

- A Clear Button that will reinitialise the tool and allow a new design to commence.

These controls provided a starting point for the VDS tool and many additional command buttons are needed to create an effective tool. However, the emphasis is placed upon the users of the VDS tool to request additions and not to impose unwanted features.

4.4.4 The Internal Data Structure

The problem faced is to design a data structure that fits around the requirements for final output format and allows flexibility for the input requirements. In order to construct this data structure the output required from the VDS tool must be examined. As stated previously, the output of the VDS tool takes the form of a numerical specification file that describes the user defined geometry. The initial mesh generation software used by the VDS tool is the KSLA mesh generation software. The output format supported is a file based system of the form shown in Figure 4.2.

This description of the geometry is a hierarchical list that will specify the domain to be meshed. Each level is built up from the lower levels, the **DOMAIN** and **END_OF_DOMAIN** flags indicate the start and end of the specification file. Other key words specify the different levels and are defined as follows

- **VERTICES** An integer as an unique identifier (id) for the vertex followed by the (x, y) coordinates of the point.
- **ZERO_D_SUBDOMAINS** An integer as an unique identifier (id) for the zero_d_subdomain followed by the id of an associated vertex.
- **ONE_D_SUBDOMAINS** An integer as an unique id followed by either an s and then the id of two zero_d_subdomains for a straight line or a c then the id of a zero_d_subdomain followed by an id of a vertex then the id of a zero_d_subdomain for an arc.
- **TWO_D_SUBDOMAINS** An integer as an unique identifier (id) followed by a list of one_d_subdomains.

The resulting mesh produced by the KSLA mesh generator, with an additional level of refinement, is shown on the left hand side of Figure 4.3. The right hand side of

```
DOMAIN meshfile1
VERTICES
1          0.5000 4.5000
2          4.5000 4.5000
3          4.5000 0.5000
4          0.5000 0.5000
5          1.0000 4.0000
6          2.0000 4.0000
7          2.0000 3.0000
8          1.0000 3.0000
9          3.0000 2.0000
10         4.0000 2.0000
11         4.0000 1.0000
12         3.0000 1.0000
ZERO_D_SUBDOMAINS
101        1
102        2
103        3
104        4
105        5
106        6
107        7
108        8
109        9
110        10
111        11
112        12
ONE_D_SUBDOMAINS
201        s          101        102
202        s          102        103
203        s          103        104
204        s          104        101
205        s          105        106
206        s          106        107
207        s          107        108
208        s          108        105
209        s          109        110
210        s          110        111
211        s          111        112
212        s          112        109
TWO_D_SUBDOMAINS
301        201 202 203 204 205 206 207 208 209 210 211 212
END_OF_DOMAIN
```

Figure 4.2: KSLA specification file

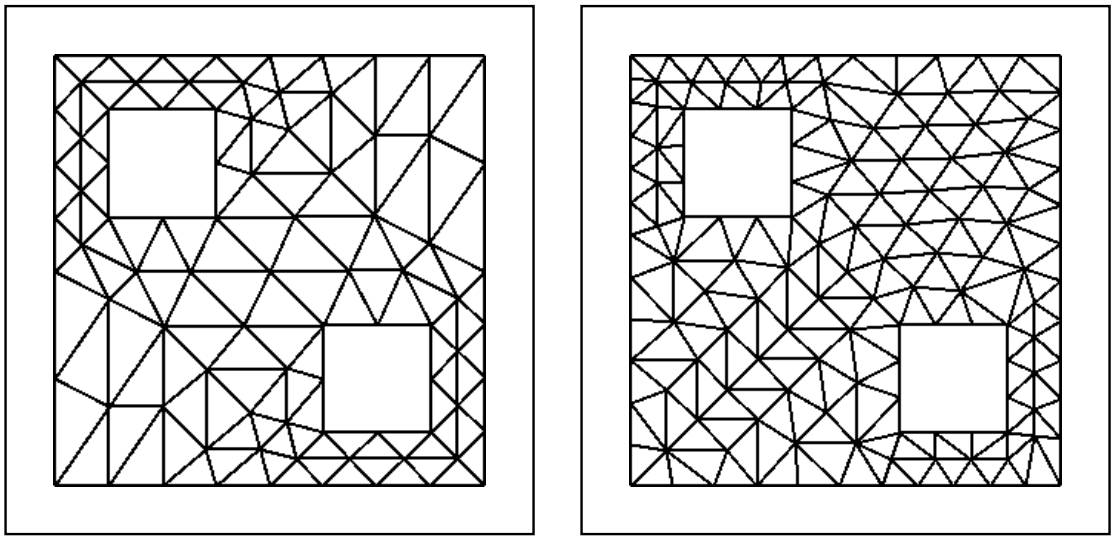

Figure 4.3: The resulting KSLA mesh (left) and GEOMPACK mesh (right)

this figure showing the mesh produced by the GEOMPACK mesh generator which will be discussed later.

A tree like structure will be used to define the geometry, allowing top down and bottom up manipulation. Various elements in the data structure will represent different levels in the tree structure. The data structure created follows the same basic structure as the KSLA output file format. The vertex element representing the information of each user defined point. The one_d_subdomain element representing the basic drawing primitives which are taken as a straight line or an arc. The two_d_subdomain element representing groups of drawing primitives. The zero_d_subdomain element provides an additional link between the vertex element and the one_d_subdomain element, this additional level is perhaps unnecessary, but it provides an easier way for the tool to output the numerical specification file in the form required by the KSLA mesh generation software. Figure 4.4 shows the basic elements of the data structure used in the VDS tool.

In this data structure, the vertex element consists of an id variable and the (x, y) coordinates of the point it represents. It has a pointer up to a zero_d_subdomain and pointers to the next and previous vertex elements to provide a linked list of all vertices used to define the geometry.

The zero_d_subdomain element provides an id variable and another variable for the region the point belongs to. It again provides pointers to the next and previous

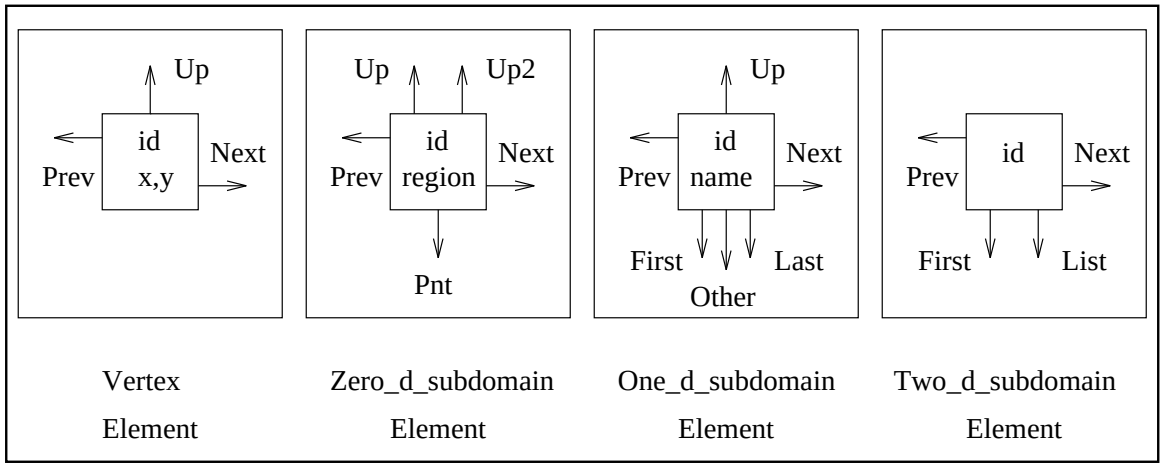

Figure 4.4: The Basic Elements of the Data Structure for the Tool

zero_d_subdomains to provide a linked list of all zero_d_subdomain elements. It also has two pointers up to one_d_subdomains this is because the zero_d_subdomain will be an end for one line or arc and a start for the next line or arc. The domains specified must be closed so this fact is true for all zero_d_subdomains.

The one_d_subdomains have variables for the id and the name, s or c depending on which of the drawing primitives line or arc the one_d_subdomains represents. It also has links to the next and previous one_d_subdomains. The link upwards is to a two_d_subdomain element. The links down are to zero_d_subdomain with first and last and to a vertex with other. The other pointer is used with arcs to specify the additional point needed, this is not needed with the line specification.

The two_d_subdomains consist of an id, pointers to the next and previous two_d_subdomain the links down first and list point to one_d_subdomains.

Whilst the names of the elements in the data structure match the names in the KSLA numerical specification file the data structure used seems flexible enough to allow development of the interface tool. The ability of the data structure to store geometries in a tree like structure, where more complex elements, are constructed from simpler elements is one which has proved successful as it allows top-down and bottom-up manipulation of the information stored in the data structure.

For example, Figure 4.5 shows how a vertex can be deleted from the data structure. Once a vertex has been tagged for deletion new links can be generated and the data structure updated to be as that shown in the right hand side of the diagram. A similar

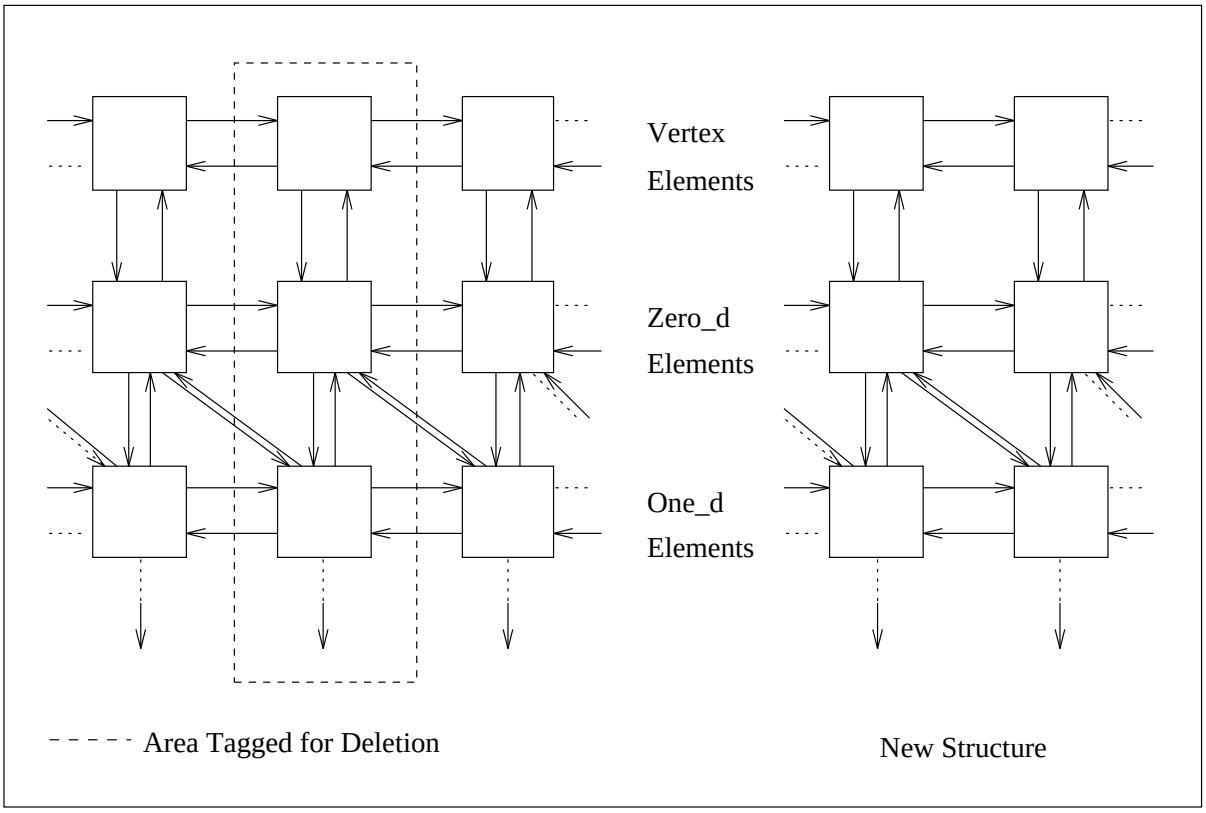

Figure 4.5: The Deletion of an Vertex from the Geometry

situation occurs, but in reverse, if a new vertex is to be created. Care must be taken if the data structure is modified near the start or end of the structure or at the start of a new region, special links to the first elements must be preserved.

The nature of the data structure was useful when the output of the geometry in a form suitable for the GEOMPACK mesh generation software was required. The GEOMPACK package input format is significantly different from the KSLA input format. The conversion between the internal data structure of the VDS tool and the numerical specification files that the mesh generation software packages uses is described next.

4.5 Output of the Tool

The VDS tool provides output in the form of a numerical specification file of the user-defined geometry. The numerical specification file produced by the VDS tool is suitable as input for the KSLA mesh generator [35], the resulting mesh from this is passed

to the display canvas in the VDS tool. The VDS tool also provides the ability to provide a numerical specification file suitable for the GEOMPACK system and can be extended to allow suitable specification files to be produced for other mesh generation packages.

The structure of the KSLA output file is described in the previous section. This section gives information about the GEOMPACK input file format and the conversion from the internal data structure of the tool for both numerical specification files. Firstly the output of the information suitable for the KSLA input file is described. The GEOMPACK input specification is then outlined and the conversion from the internal representation to this input specification file is outlined.

4.5.1 The KSLA input file format

As stated before, the internal data structure is based heavily upon the KSLA mesh generator input specification file. Therefore the transfer of information from the internal data structure to the numerical domain specification is straightforward. The specification file has the format given by

- The keyword DOMAIN followed by the name of the geometry, as supplied by the user, is output.
- The keyword VERTICES followed by the linked list of the vertex elements are output with the vertex identifier (id) followed by the x and y coordinates of the point.
- The list of zero_d_subdomain elements follow the ZERO_D_SUBDOMAIN keyword, the id of the zero_d_subdomain followed by the id of the vertex element attached to it.
- The list of one_d_subdomain elements follow the ONE_D_SUBDOMAIN keyword, the id of the element is output followed by the name of the element, an **s** for a straight line or a **c** for an arc. A straight line is represented by the id of the two zero_d_subdomain elements pointed to by first and last. An arc is represented by the id of the zero_d_subdomain pointed to by first followed by the id of their vertex element pointed to by other and then the id of the zero_d_subdomain pointed to by last.
- The list of two_d_subdomain elements follow the TWO_D_SUBDOMAIN keyword. The id of the two_d_subdomain element followed by a list of the id's of the

```
meshfile1
0.0 30.0 20.0 0.25 0.5 10 200
1 12 3 0
4
4
4
0.500000 0.500000
4.500000 0.500000
4.500000 4.500000
0.500000 4.500000
1.000000 3.000000
2.000000 3.000000
2.000000 4.000000
1.000000 4.000000
3.000000 1.000000
4.000000 1.000000
4.000000 2.000000
3.000000 2.000000
```

Figure 4.6: GEOMPACK specification file

one_d_subdomain elements that point to the same two_d_subdomain element.

- The keyword END_OF_DOMAIN denotes the end of the specification file.

The resulting mesh can be seen in Figure 4.3. As well as the file structure explained above the KSLA input specification file allows optional additional keywords in the input file. These keywords are NVMAX, NZMAX, NOMAX and NTMAX these allow the user to specify a maximum number of vertices, zero_d_subdomains, one_d_subdomains and triangles in the numerical specification of the domain.

4.5.2 The GEOMPACK input file format

The output of the internal data structure in a form that is acceptable to the GEOMPACK mesh generation package is now discussed. Figure 4.6 shows the GEOMPACK numerical specification file equivalent to the KSLA file shown previously in Figure 4.2. The layout and structure of the file is very different from that of the KSLA input file, it is much more compact, for example.

The resulting mesh from this numerical specification file using the GEOMPACK mesh generator can be seen in the right hand side of Figure 4.3. The major differences between the KSLA input specifications and the GEOMPACK specification highlight the

ability of the data structure to extract the relevant information from user defined geometry. The additional information required also illustrates the difference between semi–automatic and fully automatic mesh generation packages. Apart from the additional information required by GEOMPACK the first major difference is that the GEOMPACK package only considers geometries specified by a list of points. This fact forces GEOMPACK to only consider geometries specified by straight lines. This is very different from the hierarchical approach taken by the KSLA mesh generator. A further requirement is that the points must be specified in anticlockwise order. The KSLA mesh generation package makes no such assumptions about the information that it receives. These requirements create additional work that needs to be expended to ensure that the geometry is given in the required form.

GEOMPACK constructs the mesh by splitting the input geometry into simpler polygons and then meshing the resulting polygons. As a semi–automatic mesh generator GEOMPACK requires additional information at the beginning of the specification file to accomplish this. This information provides the user with the ability to control various aspects of the final mesh such as desired number of triangles, mesh smoothness and the way in which the geometry is decomposed into simpler polygons. Taking the specification file shown in Figure 4.6 as an example the various parameters needed are explained below.

- **meshfile1** This is the name of the domain.
- **0.0** This parameter, TOLIN, is the user supplied relative tolerance. It is used with the machine tolerance to determine the tolerance factors used in the construction of the mesh.
- **30.0** This parameter, ANGSPC, is the angle in degrees used to control the spacing of new vertices that may be created as the geometry is decomposed into simpler polygons.
- **20.0** This parameter, ANGTOL, is the user supplied angle tolerance; it provides a lower bounds on the angles permitted when decomposing the geometry.
- **0.25** This parameter, KAPPA, is the mesh smoothness parameter, it must lie in the range of [0..1]. The higher this factor is the smoother the mesh. Mesh smoothness controls the rate of change in the size and properties of triangles between the polygons the geometry has been split into.

- **0.5** This parameter, DMIN, controls the variation of the mesh distribution function in each polygon. The mesh distribution function depends upon the lengths of the edges of the triangles in the polygon and the desired number of triangles in each polygon. The variation of the function and therefore the variation of the distribution of the mesh is controlled by this parameter.

- **10** This parameter, NMIN, is a parameter to ensure that each polygon has a sufficiently large number of triangles. It is the minimum number of triangles that GEOMPACK will try to place in each polygon.

- **200** This parameter, NTRID, allows the user to specify the desired number of triangles in the final mesh.

- **1** This parameter, CASE, determines the remainder of the input file, if case is set to 1 then the file contains the description of a simple region, if case is set to 2 then a complex region is considered. A complex region requires a more complicated description of the geometry with the input file containing more information. GEOMPACK, rather strangely, defines the geometry in terms of curves, a curve consisting of a set of points. The additional information takes the form of specifying the type and location of each curve and also a list of the vertices used to define the curves. The additional information required for a complex case, if this value is set to 2, will also be outlined below.

- **12** This parameter, NVC, gives the total number of vertices used in the definition of the geometry.

- **3** This parameter, NCUR, gives the number of boundary curves used to define the geometry.

- **0** This parameter, MSGLVL, is used to pass error messages during the mesh generation process.

- 4 4 4 These three values are the number of vertices in each of the three boundary curves. The three comes from the NCUR parameter given previously. These values are read into an array, NVBC.

- If a complex case is chosen the next item in the input file will be the array containing NCUR elements describing the type and location of the curves. This value specifies if the curve is a boundary curve, a hole inside a subregion or a hole interface inside a subregion. This information is stored in the array ICUR.

- **0.50000 0.50000 ...** This part of the input file reads in the x and y coordinates in counter clockwise order. This is stored in the array VCL.

- If a complex case is chosen, after the vertex specification, the boundary curves are defined in terms of the indices of the array of the vertex array VCL. This information is stored in another array, IVRT.

In order to output the data stored internally in the VDS tool several issues had to be addressed. The first was how to deal with the geometries that the user had designed using arc drawing primitives. The decision here was to either disallow the use of arcs if the GEOMPACK input file format was required or to approximate the arc by a series of points along the arc. The second option was chosen because it allowed the user more flexibility in the construction of the geometries. The number of points used to approximate the arc, could if required, be a user specified number. The other main problem was to ensure that the points were output in a counter clockwise order, as needed by GEOMPACK.

The linked list of vertex elements in the internal data structure provided the bulk of the information needed here. This information was obtained via the one_d_subdomain elements, thus allowing arcs to be treated in a different way from straight lines.

The two meshes produced by the different mesh generation packages show many differences, especially since the KSLA mesh is shown with an additional level of refinement. This difference and the apparent lack of control over the various aspects of the final KSLA mesh compared to the GEOMPACK mesh seem to occur from the principles behind the design of the mesh generation software. The KSLA mesh generator appears to be built to provide a basic structure that can be refined and adapted by the numerical packages using the mesh. The GEOMPACK work is the result of looking at the triangulation of polygons for the finite element method where adaptive techniques may not be employed. The added control that GEOMPACK provides may not always be an advantage the more information that is needed then the greater the chance of making a mistake. Sensible defaults may be needed for the GEOMPACK system to help ensure successful results.

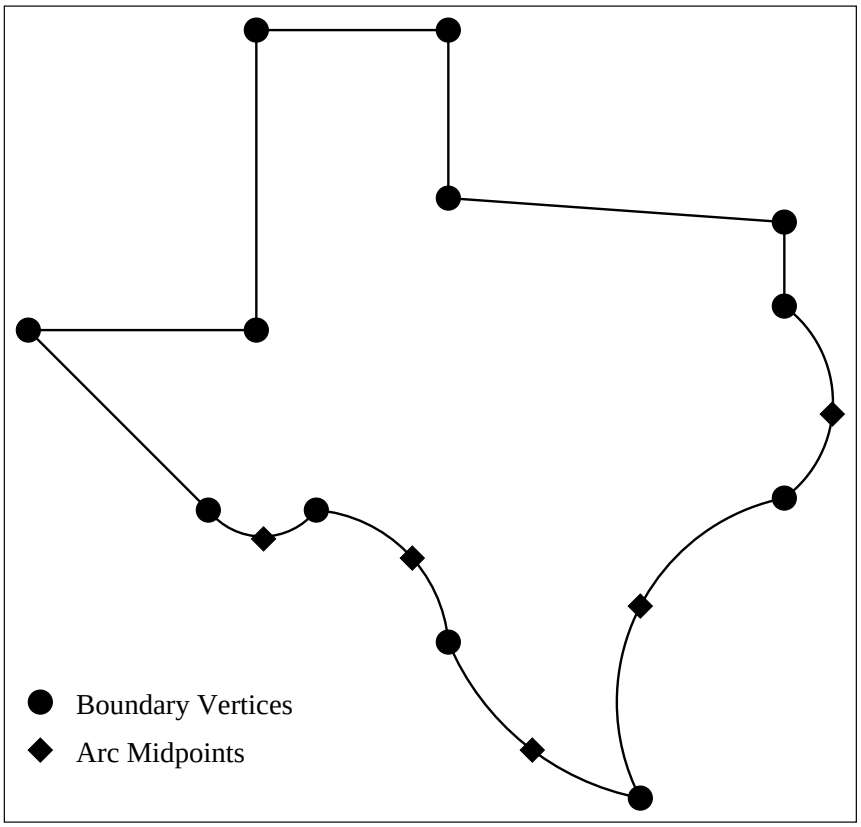

Figure 4.7: Geometry file for Texas

4.5.3 The PLTMG input file format

The PLTMG system [4], is a software package for solving elliptic PDEs, using a Piecewise Linear Triangle Multi-Grid method, hence the name PLTMG. It was developed to allow multi-grid methods and adaptive local mesh refinement algorithms to be applied to elliptic PDEs. The PLTMG domain is two dimensional and is constructed from a set of simple closed curves that do not intersect. The user is required to either input a crude triangulation of the domain or to define a skeleton of the domain which routines within PLTMG can take and produce a mesh. The regions defining the skeleton are a combination of either straight lines or arcs. PLTMG also requires that these are defined in a counter clockwise order, as in GEOMPACK.

The user defines this region by constructing a set of arrays to store the different information, each region is defined by a list of boundary curves, each boundary curve is defined by a start vertex and an end vertex for a straight line, each arc is defined by a start

vertex an end vertex and an additional third point. The user is also required to specify certain mesh parameters to control the attributes of the final mesh. This level of user input places the mesh generation routine is the class of semi–automatic mesh generators.

As an example, consider the domain in Figure 4.7, The mesh generation routine in PLTMG is called by the following line of FORTRAN code.

```
TRIGEN( VX, VY, XM, XY, ITNODE, IBNDRY, JB, GRADE, HMAX, IP, W);
```

The boundary vertices are stored in the arrays VX and VY, the midpoints are stored in XM and YM. The IBNDRY array contains, for each boundary curve

- The position in the VX, VY array of the start vertex.
- The position in the VX, VY array of the end vertex.
- The position in the XM, YM array of the middle point if the curve is an arc or 0 otherwise.
- The boundary condition, either -1, 0 or 1 for Dirichlet condition, an internal edge or natural condition respectively.
- A label to identify which region the curve belongs to.

The JB array stores, in counter clockwise order, the indices of the IBNDRY array. The ITNODE array stores, for each region, the start curve in the JB array, the end curve in the JB array, some symmetry information and a label to identify the region. GRADE and HMAX define two mesh control parameters to control the relative sizes of the triangles in the mesh and the largest edge length. The IP array is used to pass information such as the number of vertices, curved edges and the lengths of each array.

It would therefore appear that the way PLTMG represents the geometry would closely fit in with the VDS tool. The use of lines and arcs match, the counter clockwise ordering is already required by GEOMPACK, the additional information required could be obtained in a similar way to the GEOMPACK mesh control parameters. However, PLTMG has no concept of holes within domains, PLTMG represents such domains by two subregions with cuts in the domain, see Figure 4.8, this may cause problems for anything other than simple domains.

However the relative merits of each mesh generation package or the way in which the domains are specified is not discussed further here. The emphasis here is placed on

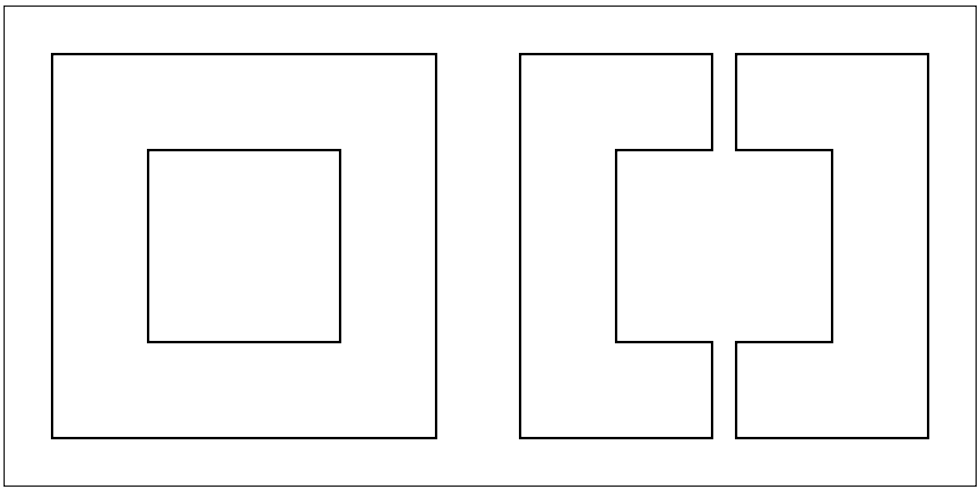

Figure 4.8: Hole Representation by PLTMG

the ability of the visual domain specification tool to produce input files for two very different mesh generation packages, and, in principle, be extended to other mesh generators, PLTMG being one example.

4.6 Evolution of the tool

As stated earlier the aim of the VDS tool is to construct a prototype and then to use an interactive loop involving the user to enhance the VDS tool. The following section gives details of the refinements, additions and changes made to the VDS tool from the initial state to a state that perhaps represents a powerful tool for visually specifying complex geometries quickly and easily. However, the VDS tool by no means is complete and the *wish list* of features the user would like to see still has a few outstanding items. As a starting point the initial state is described, then the list of changes prompted by user feedback are described.

4.6.1 Initial specification

The VDS tool, in its initial state, had five areas of user interest. These are the Drawing Canvas, the Display Canvas, The Quit Button, the Mesh Design Button and the Clear Button. The functionality of these is described below.

- The Drawing Canvas, this area allows the user to specify the geometry, the initial state included one drawing primitive, the line. The mouse is used to move the cursor around this canvas, pressing different mouse buttons will have different actions. The action of specifying a geometry is referred to here as a drawing session.

  - The Left Mouse Button will create a new point at the current cursor position. After this action mouse movement will rubber band a line from the current cursor position to the last user specified point in the drawing canvas.
  - The Middle Mouse Button will join the last user specified point and the first point in a region. For example, pressing the middle mouse button after specifying the fourth corner of a square will complete the square. Pressing the Middle Mouse button will not finish the drawing session but allow the user to specify the first point of another region.
  - The Right Mouse Button will perform the same action as the Middle Mouse Button but will terminate the drawing session. This allows the geometry to be output to a file and passed to the Display canvas via the Mesh Design Button.

- The Display Canvas, this area displays the resulting mesh when a numerical domain specification file is passed to the KSLA mesh generation software.

- The Quit Button, this kills the application.

- The Clear Button, this reinitialises the tool and allows the user to commence another drawing session.

- The Mesh Design Button, this allows the user to specify the name of the file in which to output the numerical specification of the geometry as required by the mesh generation software. The button also passes this file to the Display Canvas.

The initial design was built using the Athena widget set [69], the XToolkit and Xlib routines. This provided the portability and standardisation required, the conversion of this to utilise the OSF/Motif Widget Set [94] was one of the first changes requested by the users of the VDS tool.

4.6.2 Changes To The Tool

The following section gives a description of the changes made to the visual domain specification tool. A full listing of all the changes is given in Appendix B. The changes made were largely driven by user feedback on the VDS tool, however in some situations user requested changes led to other changes in the VDS tool. As an example the initial drawing canvas had a coordinate system of $[0..1, 0..1]$ and a scale button provided a simple means to translate this to say a $[0..5, 0..5]$ region. The next stage in the development process allowed the user to specify the top right and bottom left coordinate pairs of the drawing area, therefore the scale factor became simple to calculate and the scale button became unnecessary.

Many of the changes made to the visual specification tool appear to be part of a longer path moving from the initial state to the current state of the VDS tool. Others appear to be additions to the functionality of the tool as the user requests. To illustrate this a list of changes are described and the way in which these changes are connected. The example given above shows a good example of this. The initial geometry design was carried out on a $[0..1, 0..1]$ region, the inclusion of a scale button allowed this region to be expanded uniformly. The Co–ords button allowed even greater control over the coordinates of the drawing canvas superseding the scale button.

A similar stream of events started from the inclusion of a grid on the drawing canvas. This aimed to provide the user with guidelines to help in the design process. The increase in the ability to control the spatial coordinates of the drawing canvas with the Co–ords button created the problem with the spacing of the grid lines. One solution to this was to provide a fixed grid over the drawing canvas. This grid could then be controlled via a grid button. This button could be set to one of three possible states off, coarse and fine. The off state turned the grid off, the coarse state produced a sparse grid over the drawing canvas and the fine state producing a finer grid. The addition of two labels in the control panel informed the user of the actual position of the mouse pointer in the spatial coordinates of the drawing canvas. Given this condition the grid lines represented guide lines only and had no significant positioning in relation to the spatial coordinate system chosen by the user. However, this system did not appear to provide enough help for designing geometries, the users expressed a wish that the grid line spacing in the x and y direction could be specified. This facility was added, expanding the Co–ords box

to include a x and y increment for the grid. The facility changed the status of the grid button, the button now could have one of two states these being on and off. A default value for the grid spacing was needed next, this was taken such that the grid mimicked the previous fine state. The result of this action increased the significance of the grid lines with respect to the spatial properties of the drawing window. This allowed for two additional buttons to be added that could control the design of the geometry. The first of these buttons, the snap to grid button, this button could be set to on or off, when set to on the creation of new points will automatically be placed at the point of the nearest intersection of the grid lines. The second, snap all to grid button, allowed the user to, at any time in the design process, move all the points to the nearest grid lines intersection point. This allowed the user to redefine the grid lines and then snap the points to the new grid.

Other changes made to the VDS tool seem not to follow a longer path through the evolution of the tool as those described above. These changes provided additional functionality to the VDS tool. These changes are described below, starting with the changes to the drawing canvas, the first step was to allow the user the ability to edit the spatial position of the points used to define a geometry after the drawing session was complete. This move from the drawing session to an editing session was controlled by the completion of the geometry. The signal of a completed geometry is controlled via the right mouse button. Pressing the right mouse button closed the geometry and tidied up the representation of the geometry in the internal data structure. The editing session allowed the user to click the left mouse button near a point and the x and y coordinates of that point would be displayed. The user could then change these values and that would change the position of the point in the drawing canvas. This allowed the user to initially produce a rough outline of the geometry and to then refine the geometry later. The second was to implement the ability of the user to specify arcs in the design process. This required the inclusion of a button in the control panel that could control the drawing primitive in use. This button allows the user to switch between using lines and arcs. The addition of a graphical indication at the location of the user defined points was also included, this allowed for easier selection of the point in Edit mode and also for the user to judge better the middle point when using the arc drawing primitive. The inclusion of a marker to indicate the location of the user defined points provided further help in this respect.

The remaining changes were made to the control panel. The first was the addition

of a refresh button that forces the tool to redisplay the contents of the drawing canvas and the display canvas. This was done early in the iterative process. Many X applications appear to have a similar button. The next change was to provide the user with some online help, the help button provided a brief explanation about how to create a geometry. The help system was upgraded later to provide a more comprehensive and up-to-date help. Also an additional quick help button was added. The aim of this button was to provided the user with a reminder about the action of various mouse button presses during the drawing and editing sessions. This quick help also appears on start up of the tool. This was prompted by user feedback. The users experienced difficulty in remembering which button closed the domain and finished the drawing session if they had not used the tool for some time. On a similar principle several warning messages were added that provided the user with information if no action could be taken. For example, the VDS tool will only allow the user to produce a mesh file if the domain has been closed with the right mouse button. Pressing this button at any other time would result in a message saying this.

The ability to load existing mesh files was expressed by users. This ability to save a rough outline and refine later or to modify existing mesh files was added to the VDS tool. The user, when entering a file name to be loaded, had the ability to specify the spatial coordinate bounds or alternatively the VDS tool will attempt to work out suitable values. The user specified coordinates can be ignored if the data in the file lies outside this range. With the addition of this facility a wish was expressed by users that the drawing canvas and the display canvas be proportional to the coordinate system used. This was implemented but provided a problem concerning resizing the windows in the tool. Previously the user had the ability to resize the window to any size. In order to preserve the aspect ratio of the coordinates this control had to be changed. The solution to this problem was to include two additional buttons in the control panel. These buttons could be used to increase or decrease the size of the drawing canvas and display canvas.

A further feature requested by users was to read in a set of points from a file. This file would contain the x and y coordinate pairs used to specify all or part of a geometry. This was required because other packages may supply information in this form. This provided a greater flexibility in the design process. The tool will automatically change the coordinates of the drawing window if the input data required it. Also the snap to grid lines facility would automatically be turned off. Warning messages are displayed to the

user if the tool changed the coordinate system when loading a file or reading points from a file. The inclusion of a Show Points button allowed the user to switch off the marker indicating the user defined points.

Another feature requested was the ability to add a new point to the geometry or to delete a rogue point. This has already been discussed in Section 4.4.4 showing how the internal representation of the geometry can be manipulated to add or delete new vertices. The decision here was which vertices should the user be allowed to delete or where could new points be added. The choice with deleting a point was to allow any vertex which was the end point of one line and the start point of another to be removed. To create a new vertex, the new point could be created along a straight line, the new point created half way along the edge. The user could then edit the position of the point. Both operation are allowed in Edit mode only and are triggered by the pressing the middle mouse button and right mouse button respectively.

The VDS tool allows the user to quickly define a geometry which is converted into a numerical mesh by whichever mesh generation package is used. The numerical solution process then requires various boundary conditions which are applied to each edge. Although the VDS tool provides no mechanism to specify these boundary conditions at the design stage (the Visual Problem Specification Systems described in Chapter 5 performs this task) the user may still want the VDS tool to display the unique identifiers assigned to each edge. The ability of the VDS tool to display this information is provided via the Show Information button.

The final addition considered here provided the user with some control over which part of the geometry is meshed. This is combined with an explanation of the resulting mesh file that should provide enough information for the user to edit the file if the resulting mesh does not satisfy the users requirements. This is only currently available for use with the KSLA mesh file format, the mesh status flag is set to Simple for the GEOMPACK format. The mesh status button is part of the mesh design popup. The state of the mesh status button will change the numerical specification file and hence change the resulting mesh. The mesh status button can be set to one of the following Mesh Status : Simple, Mesh Status : Mesh Holes or Mesh Status : Mesh Interior. To explain how this will change the mesh two examples are given below. The first is the example outlined earlier which produced the meshes in Figure 4.3, a large outer square with two interior squares, the second is a large outer square with a smaller square inside and a third smaller square

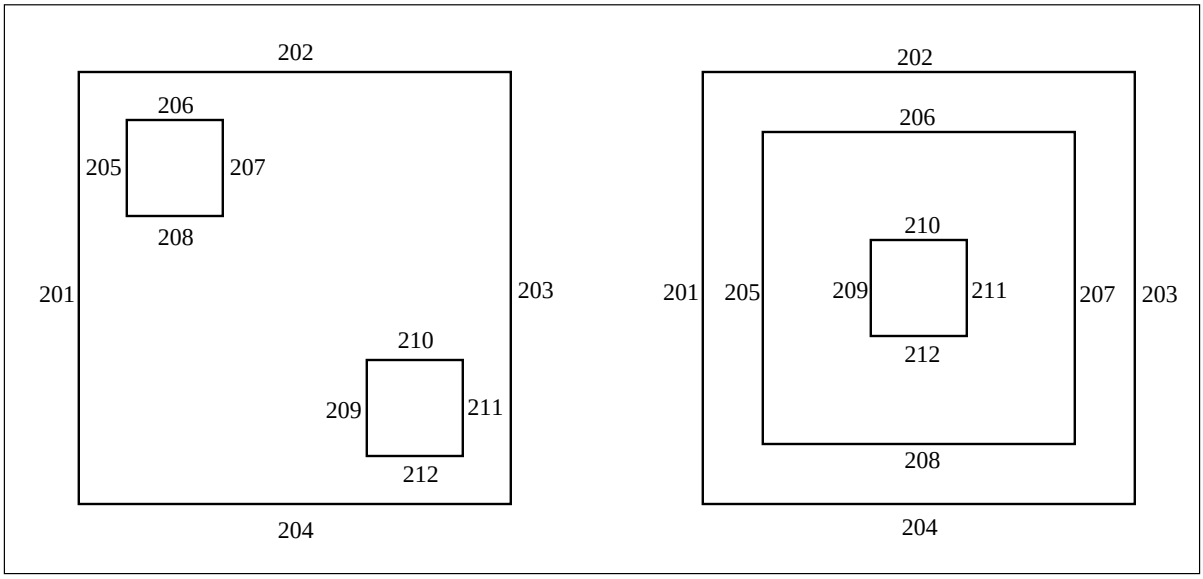

Figure 4.9: The Geometry for Example 1 and Example 2 with Edge Names

inside both. These two geometries can be seen in Figure 4.9.

The area of interest is the TWO_D_SUBDOMAINS section of the numerical specification file. In the example file given in Figure 4.2 the Mesh Status was set to Simple. To illustrate the different states of the Mesh Status the right hand of Figure 4.10 shown the same domain with the Mesh Status set to Mesh Holes. The changes made to the numerical specification file are that the TWO_D_SUBDOMAINS section was

```
TWO_D_SUBDOMAINS
301     201 202 203 204 205 206 207 208 209 210 211 212
```

with the Mesh Status set to Mesh Holes this now becomes

```
TWO_D_SUBDOMAINS
301     201 202 203 204 205 206 207 208 209 210 211 212
321     205 206 207 208
322     209 210 211 212
```

The two additional lines will cause the mesh generator to mesh the regions specified by the list of one_d_subdomains given. This means that the two_d_subdomain 321 will mesh the upper left hole and the two_d_subdomain 322 will mesh the bottom right hole.

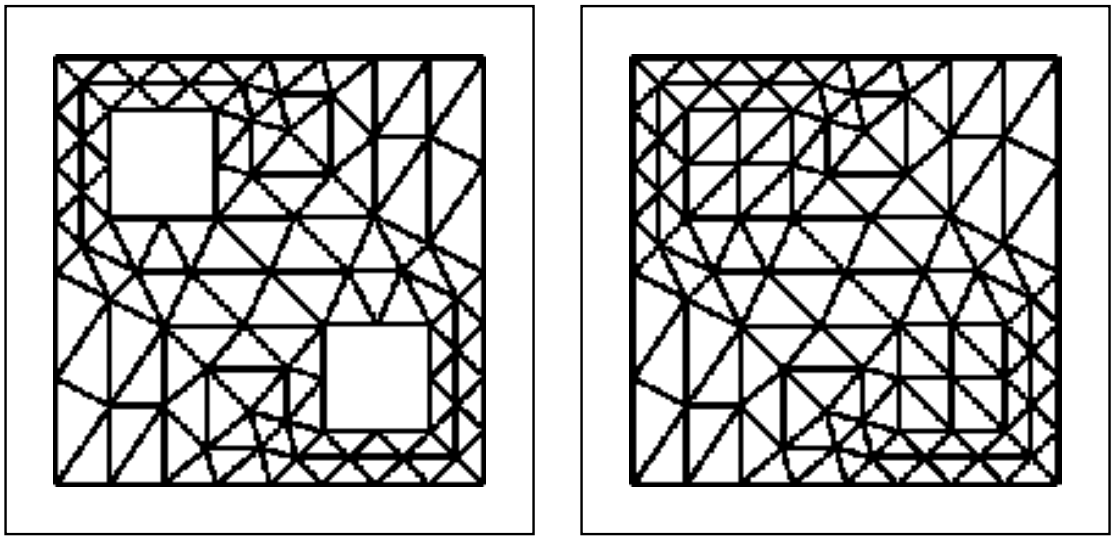

Figure 4.10: Mesh Status : Simple and Mesh Status : Mesh Holes for example 1

The information given in the two_d_subdomain line can be traced back using the numerical specification file. In this example two_d_subdomain 321 has four one_d_subdomains, 205, 206, 207 and 208. Going back further the one_d_subdomain 205 is a straight line from zero_d_subdomain 105 (a pointer to vertex number 5) and zero_d_subdomain 106 (a pointer to vertex number 6).

The second example shows the use of the Mesh Status : Mesh Interior, the geometry is shown in the right hand picture of Figure 4.9. The difference between the Mesh Status : Simple and setting Mesh Status : Mesh Interior is shown in Figure 4.11. For the Mesh Status : Simple the TWO_D_SUBDOMAINS section will be

```
TWO_D_SUBDOMAINS
301     201 202 203 204 205 206 207 208 209 210 211 212
```

For the Mesh Status : Mesh Interior this will become

```
TWO_D_SUBDOMAINS
301     201 202 203 204 205 206 207 208 209 210 211 212
321     205 206 207 208 209 210 211 212
```

It is hoped that the three possible Mesh Status types will cover the majority of the users needs. However, it is envisaged that in some circumstances the user may

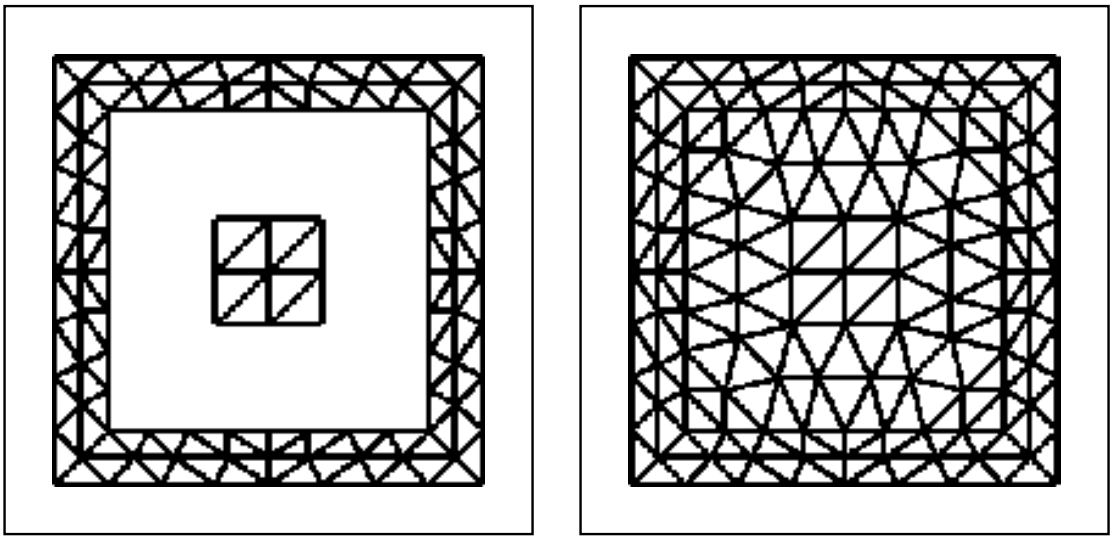

Figure 4.11: Mesh Status : Simple and Mesh Status : Mesh Interior for example 2

construct a geometry with holes and interior regions. It is advised in this case to set the Mesh Status to Simple and to change the numerical specification file. The VDS tool can be used to view the mesh file via the load file button.

4.6.3 Current Status of the Tool

The current status of the VDS tool is described in the users guide, which is also the on line help. The guide, in a slightly modified form, is shown in appendix (A), the information about the mesh status button and editing the mesh file is not repeated.

The initial state of the VDS tool is shown in Figure 4.12. This shows the Quick Help facility that appears on startup. As previously mentioned this help gives the user a reminder of the actions of each button press. The figure also shows the default state of the drawing primitive as well as the default settings for the grid and snap facilities.

The ability to edit points used to specify the geometry is shown in Figure 4.13. This figure also shows the mesh design popup allowing the user to specify a filename and to select the mesh status state. Figure 4.14 shows the GEOMPACK parameter specification popup. All popup windows have a cancel button that allows the user to remove the popup and to continue with no action taken. Finally the ability to alter the spatial coordinates of the drawing window and the spacing of the grid lines is shown in Figure 4.15. This show the Co–ords box popup which allows the user to specify the bottom left and top

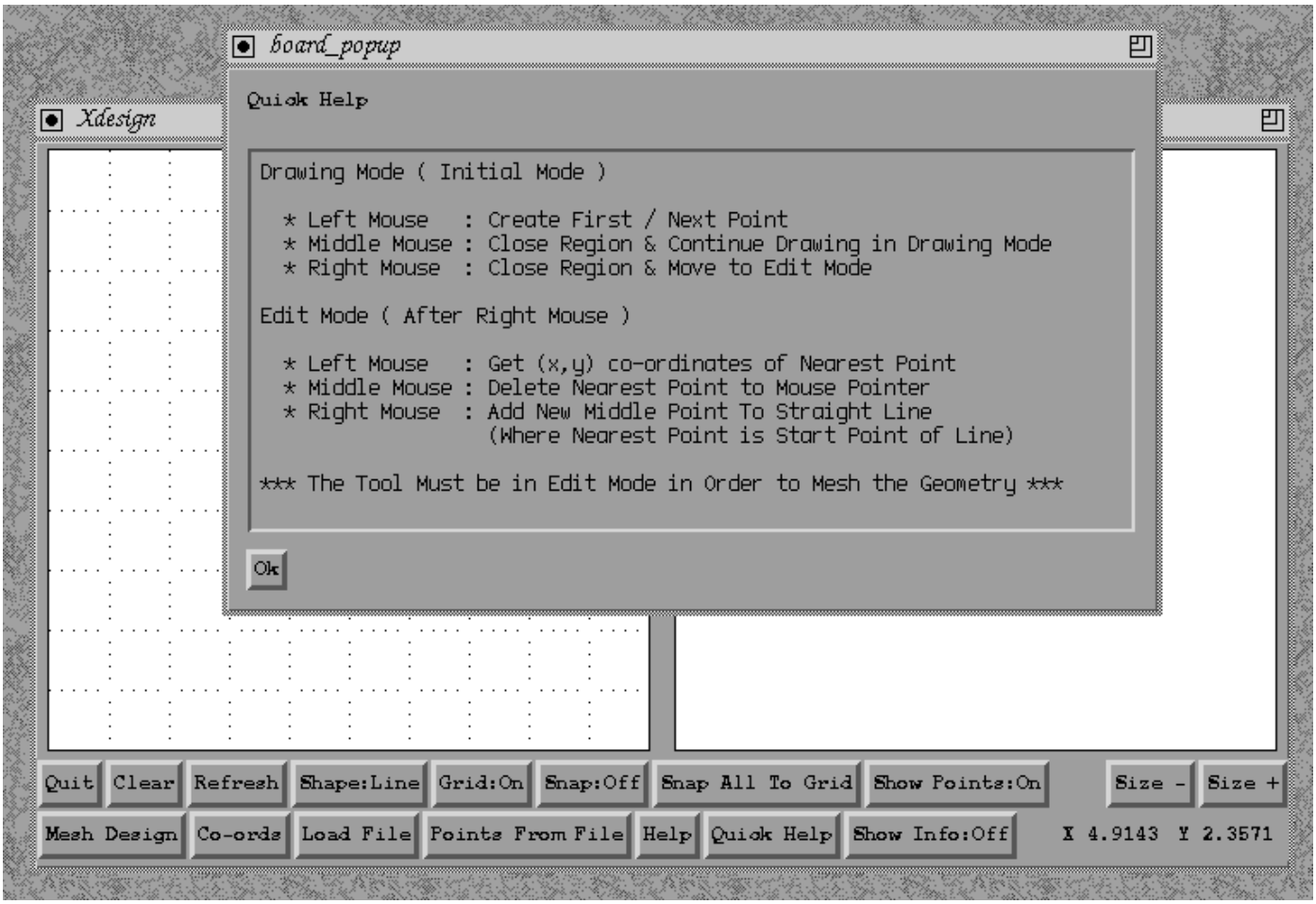

Figure 4.12: The Visual Interface Tool on Startup with the Quick Help Window

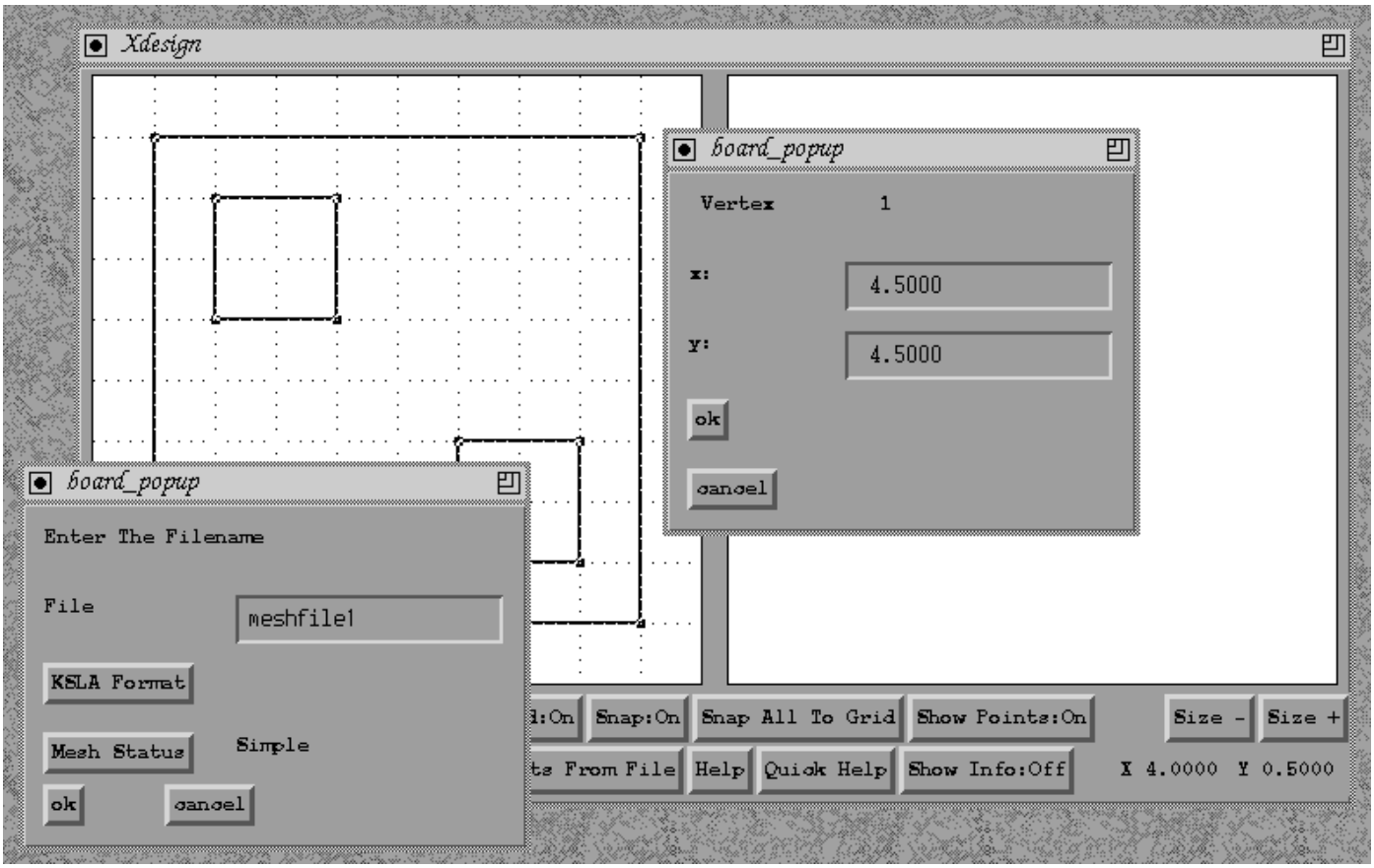

Figure 4.13: The Mesh Domain and Edit Point Facilities of the Tool

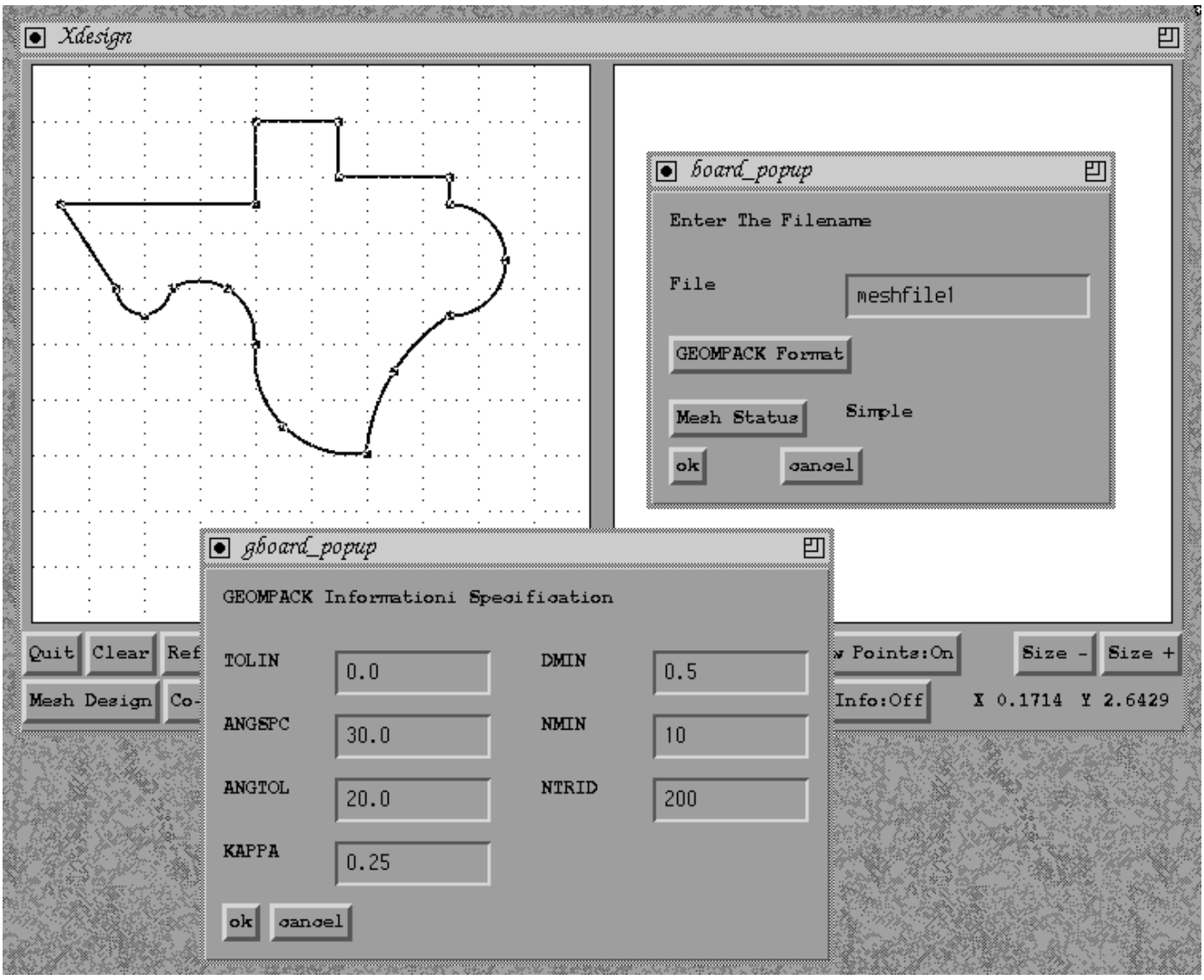

Figure 4.14: The GEOMPACK Parameter Specification Popup

right coordinate pairs as well as the x and y increment of the grid lines.

As stated earlier the tool was constructed with the help of a user feedback loop. Given this situation there is still a users *wish list*, this list will be given at the end of the next section. This next section also provides some user evaluation of the VDS tool.

4.7 Evaluation of the Tool

Writing a domain description file can be troublesome for anything other than the simplest cases. The creation of this numerical representation can be one of the most daunting tasks that face a new user and usually one of the first. This is the view of one of the experienced users of the numerical software, SPRINT2D. The VDS tools is capable of producing a numerical domain description file for SPRINT2D and this section will provide some user evaluation about the VDS tool.

As mentioned earlier production of a domain description file, or domain file, that can be used by numerical software may represent a large percentage of the overall problem specification process. Current techniques, as described by users, involve the drawing of the domain on graph paper then the conversion from this paper representation into the file format required by the mesh generation package. This is either done all at once and often requires several attempts before the software correctly reflects the paper diagram or this can be undertaken incrementally where the domain file is built up in stages adding more detail when the user is happy with the previous stage. A further alternative to this would be to extract the geometry from another computer package, however this still has to be converted into the correct file format. The simplification of the process is one of the primary aims of the VDS tool.

The initial drawing stage, or sketching stage, was seen by the users as an *easy and natural way* in which to work. It reflected the way in which the user would draw the domain on graph paper. It was seen as superior to a method where the user would have to enter all the points first and then specify the connectivity between them. The edit stage was at first seen as a bit awkward, the ability to pick up a point with the mouse and then move it was seen as a more natural way to edit the file. However, often the user was working with a technical design where the vertices would have to be placed at specific positions, often the positions would be such that they may be unobtainable, or difficult to place, given the resolution of the workstation screen. The snap to grid facilities were

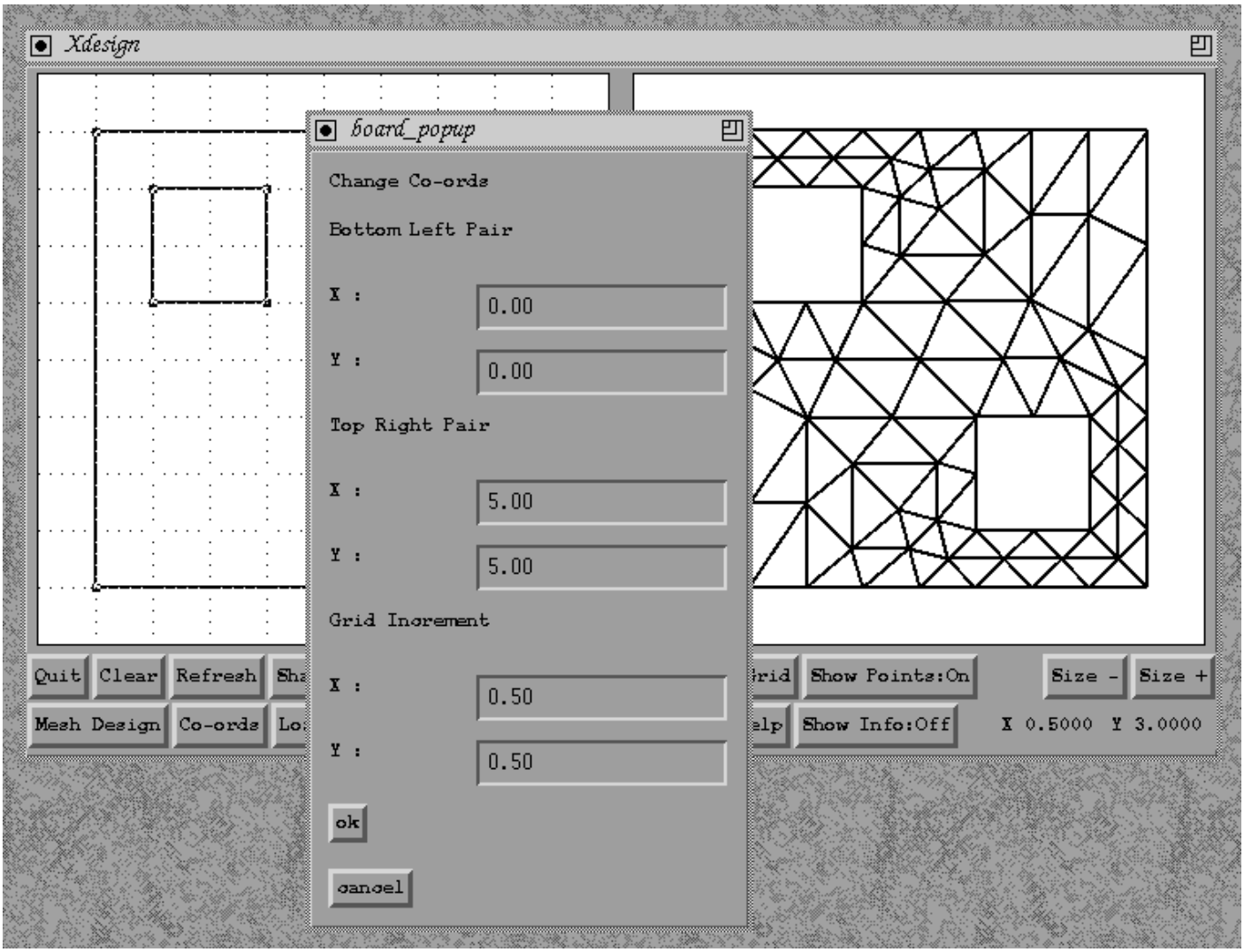

Figure 4.15: The Co–ords box of the Visual Interface Tool

seen as a useful way in which the edit stage became unnecessary for simple domains.

The ability to save a design and then reload it was seen as useful along with the ability to load existing domain files. These files could then be modified by the edit stage. Another useful feature was the ability of the tool to read a list of coordinate pairs. This allowed the user to read in points from another package. Figures 4.16 and 4.17 show this. Figure 4.16 shows a domain description file constructed from a file containing 344 coordinate pairs, this domain represents the River Axe flowing into a bay at Weston-Super-Mare. Figure 4.17 shows another domain constructed from 175 coordinate pairs, this represents a proposed canoe slalom course. The hand conversion of these files would have taken considerably longer than the seconds required by the VDS tool to do the conversion.

The VDS tool also allowed the user to view the names given to the boundary edges by the KSLA mesh generation software. This was seen as an essential part of the VDS tool by the users. The quick help was also found to be a useful aid to jog the memory about how to use the tool. The online help was seen as well laid out and the short tutorial helpful.

The user evaluation also consisted of a *wish list* of features that they would like the VDS tool to have. It was noted that with any software tool once the user had seen an example of what was possible they are keen to suggest extensions. These extensions included the ability to snap to a function of the grid lines, for example having the grid lines every 1.0 but snap to every 0.2, the ability to annotate the grid lines was also mentioned as a possible extension to compliment the x and y position indicators at the bottom of the tool. Another wish was to be allowed to change the current coordinate scale but keep the same spatial positions of the points already defined. At the current time the tool will respect the relative position of the points.

Overall the VDS tool was seen as a useful addition to the problem specification process, one which automates the complex and monotonous task of creating a domain description file by hand. It successfully reduces the time spent on this process and provides an easier method for one of most daunting tasks facing new users of the software.

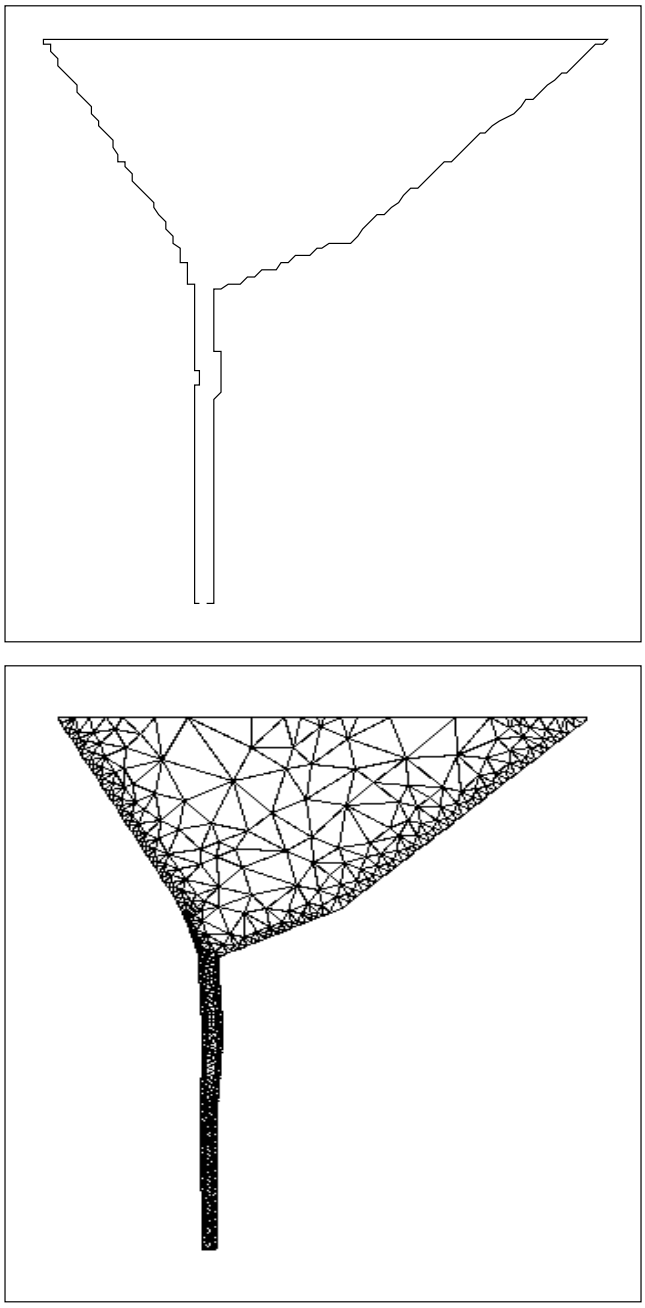

Figure 4.16: Example Geometry 1 and Mesh

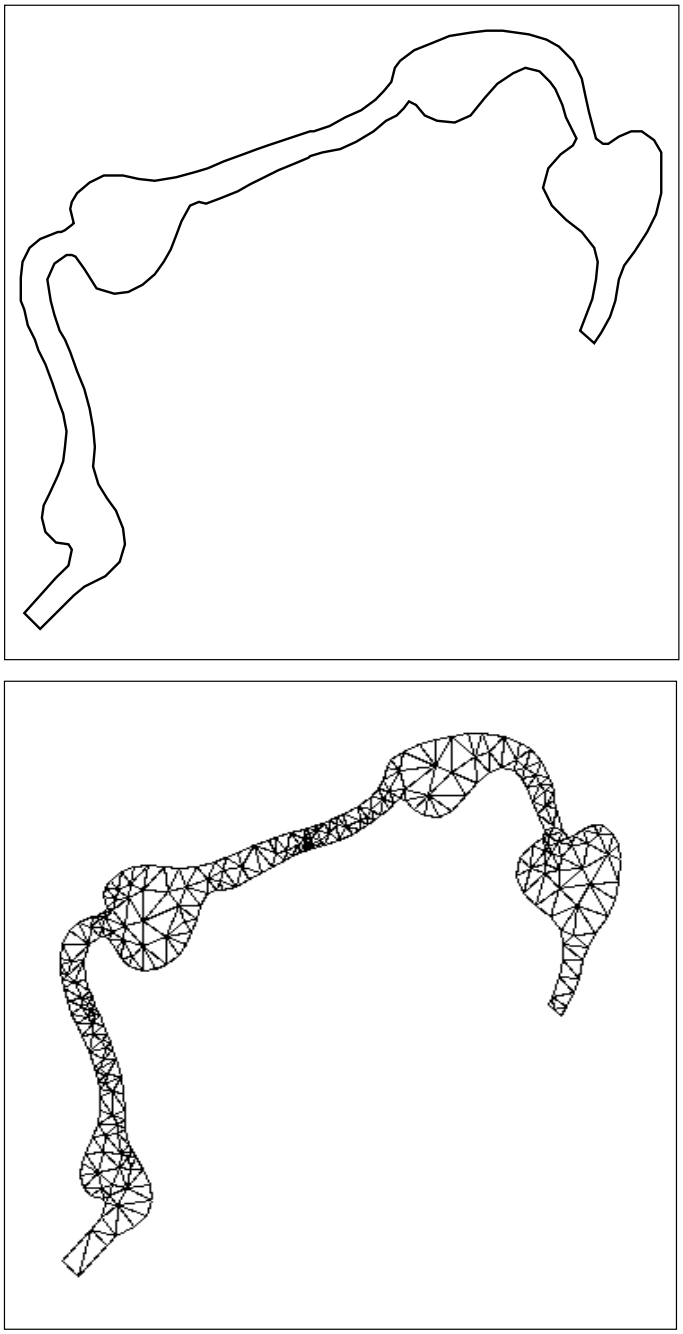

Figure 4.17: Example Geometry 2 and Mesh

4.8 Summary

The specification of a domain or geometry is an important part of the specification process for two dimensional PDEs. The use of mesh generation software allows the user to solve problems over complex geometries. The task of specifying these regions is one which is usually done manually and is therefore time consuming and prone to error. In this chapter the construction of a visual tool to aid the user in the definition of these regions is discussed. The aims of the tool along with the properties it should have such that it may be considered part of a PSE surrounding general purpose software for the solution of two dimensional PDEs are discussed. The ability to reduce the time taken to specify these regions and to provide a more natural and convenient way for the user to define the geometry are essential to achieving this goal. The divorce of the visual front end from the postprocessing system allows the tool to be independent of the numerical software used. The postprocessing system deals with the transformation of data. The internal representation of the geometry is converted to a form suitable for input to the mesh generation software under consideration.

The visual interface and postprocessing systems are constructed using building tools that are considered as standards and will ensure the portability of the tool, another requirement of a PSE. The design process of the tool is discussed and the use of an iterative loop involving the users, this design process took the form of the construction of a prototype with minimal functionality followed by the evolution of this system into a more comprehensive tool. The changes made to the tool are discussed and the reasons behind the changes are given. This process produced a general purpose software tool for the construction of two dimensional geometries.

The visual specification system can be considered as a generic front end and the postprocessing systems can be constructed to generate suitable input files for the mesh generation packages. To show this two different mesh generation packages are considered and their input formats are discussed and shown to be very different. The relative merits of the mesh generation packages are not discussed but the process of producing the input requirements for these packages as well as the ability of the tool to act as a specification system for both.

Chapter 5

A Visual Problem Specification System

5.1 Introduction

This chapter is concerned with the construction of a visual problem specification system which combines a set of visual interfaces allowing the user to specify the information needed to solve two dimensional convection-dominated PDEs. These interfaces are combined with a postprocessing system that will convert this information to a suitable driver program for numerical software. A family of interfaces is used to allow the different parts of the problem to be defined independently, thus allowing the user easily to alter one without affecting the other.

This chapter will look at some existing visual specification interfaces in the area of two dimensional PDEs. The information obtained from looking at these interfaces, combined with the solution process used for PDEs in two space dimensions, will allow a generic description of the information needed for the solution of the these PDE problems to be devised.

This framework, for the class of problems considered, is outlined and the logical grouping of the information within it is discussed. This devised framework provides a starting point for the construction of the interfaces. The aims of the interfaces are examined to ensure that they meet the requirements of a PSE surrounding the PDE software. The layout of each interface is described and its implementation outlined. The evolution of

each is then discussed along with the output of each module. The postprocessing system for combining this information to produce a suitable driver program for the numerical code is then examined.

Three case studies are then considered, two time dependent problems and a steady state problem. Each of the three problems has different characteristics and so demonstrates how the information supplied is utilised to produce suitable driver programs for each. Finally some user evaluation of the system is presented.

5.2 Other Visual Specification Tools For PSEs

Many other PDE systems or PSEs surrounding numerical PDE solvers make use of visual interfaces to allow the easy specification of the information needed to solve the problem. The interfaces used by such systems take many different forms and often focus on one particular part of the information that needs to be specified. Many concentrate on the specification of the governing equations combined with initial and boundary conditions of the problem. The hope is that certain properties may be deduced from this information. Other systems provide a more comprehensive interface which also looks at the solution process and the algorithm used.

The combination of equation specification and symbolic computer algebra tools has been discussed previously in Chapter 2. A good example of this is the equation interface *pdefront* of the RPI system [65]. The pdefront system utilises Maple [22] to deduce properties of the equations specified. Unlike many other equation interfaces pdefront requires the user to enter the full equation in Maple like syntax. The RPI system solves problems in a general form, which is the case for most numerical codes. The system allows the user to specify the independent and dependent variables used or default values may be used. The properties of the system may then be determined. The initial conditions, and the boundary conditions on each of the boundary segment are specified in a similar way. The system also generates a *troff*, [66] formatted specification of the data provided and uses this to display the information, in mathematical notation, to the user. Finally the system generates FORTRAN or C procedures to calculate the functions and matrices contained within the solution. The *pdefront* system has no geometric modelling facility so the domain is limited to a rectangle. Visualisation capabilities are provided via MAPLE or NCAR graphic facilities. The pdefront system is constructed using the X Window system

[54] for portability.

The Eve system [5] is built around a knowledge based system (KBS) containing information about the algorithmic methods available for the numerical solution of PDEs. The system accepts as input a description of the PDEs. There is no predefined set of equations Eve can solve, the ability to solve a system depending upon the contents of the knowledge base. However, the class of equations acceptable is again given by a generic formula. The input of the equations and the domain over which the system is to be solved involves the use of a graphical user interface. The interface allows the choice of one of the known domains or a new two dimensional domain may be constructed using straight lines via an interactive geometry tool. The equation interface consists of several icons depicting numerical operators, the equation is constructed from these. As the icons are selected they are inserted into the equation to match the generic form allowed. This rewriting in the generic form allows easier extraction of any mathematical properties that the equation has. A similar process is applied to the boundary conditions. The system then uses this information with its knowledge base to determine a solution strategy. The decision process is displayed providing an explanation facility. The graphical interface to the system is again built using the portability of the X Window system. The knowledge base itself is built using LISP and relies upon a frame–based model to represent the knowledge.

The visual interfaces for two other PSEs are discussed next. The //ELLPACK system [47] and the Visual PDEQSOL system [86] both utilise the portability of the X Window system to construct interfaces allowing the specification of the information needed for the solution of PDEs.

The PDEQSOL system provides a series of interfaces, these are controlled via an interactive control panel. The control panel is split into three groups. Each group having its own visual interface, described as an input guidance template, to specify the information required. Selection of one the buttons on the top level interface opens the guidance template for that particular aspect of the specification process. For example the PDE button on the top level will open the interface allowing the user to define the governing equations for the problem.

The first group of buttons allows the user to specify the physical aspects of the problem. This deals with the physical domain, the variables used and governing equations as well as initial and boundary conditions. An interactive geometry tool allows the specification of the domain. The equation information, boundary and initial conditions

are specified by selecting and combining variable names and numerical operators from an equation interface. These are then built into equations and functions respectively. The second group deals with the mathematical model to carry out the numerical simulation. This provides information about meshing schemes and numerical algorithms. Solution schemes can be specified, the user highlighting one from a list provided by the system, for each solution part. The guidance information offered by the system is obtained by examining the input PDEs. The choice of, for example, dealing with the time dependent part of the problem may then fix the choice of one of the other solution parts. The third group deals with the execution of the computation and calculation of numerical results. The Visual PDEQSOL system makes the claim that through this interactive visual specification system the total simulation procedures are shorten considerably compared to what they consider as conventional methods.

The //ELLPACK system also utilises visual specification tools to obtain information. The //ELLPACK system describes these as intelligent editors to deal with all the aspects required to specify the problem. The initial input to the system is done via a PDE specification editor combined with geometry tools which are used to specify the region under consideration as well as boundary conditions. The governing equations are again constructed via a combination of numerical operators and permitted variables. The specification of equations in a natural form is a main objective of this step. The PDE specification editor also provides lists of possible selections for various aspects of the solution process, such as discretisation method, indexing method and solution method. Other parts of the specification process include obtaining time step information with the user required to specify starting time, stopping time and time step values. The PDE specification editor also generates a //ELLPACK program from the information supplied.

All of the above interfaces appear to group together information in different ways, the PDEQSOL interface splitting the physical aspects of the problem from the numerical aspects. The //ELLPACK system splitting the geometric modelling from the rest of the specification process. The distinction between the physical and numerical aspects seem valid as does the separation of the geometric modelling. The system discussed here has already had the geometric modelling removed, however, the system will aim to separate the physical nature of the problem from the numerical algorithm selection part of the solution process.

The information gathered by these interfaces often takes one of two forms, a *fill*

in the blanks for certain parts and a *select one item from a list* for others. The fill in the blanks method is used when numerical information is needed and also when governing equations and boundary and initial conditions are defined. The lists are used when a choice is needed, for example, discretisation methods. Both options aim to provide the most natural way to specify the problem and remove the need for explicit programming. The time saved by this facility is a main driving force for such visual interfaces but not the only one. The other advantages provided are to ensure that all information is input, sensible defaults are chosen and that errors, that can occur from the hand conversion of the information to a suitable driving program for the numerical software, are eliminated.

The information obtained from the examination of other work in this area can be combined with feedback from the developers of the numerical code considered in this thesis and from the users of the code. This can then be utilised to outline a series of aims for a set of visual specification systems to complement to visual design tool, described in Chapter 4. This package provides an easy to use layer surrounding the numerical software for the solution of two dimensional PDEs that can eliminate the need for explicit programming.

5.3 Aim of the System

The objective when designing the visual problem specification system was to construct a portable set of interfaces that, when combined with a suitable postprocessing system, could act as part of a PSE surrounding the numerical software. There are several advantages that a visual interface can provide, many of which have been touched upon in the previous section. The interface can represent a significant decrease in the time spent in the specification process, this allows the user to concentrate on the results the code is supplying. Therefore one aim of the visual problem specification system constructed here is to reduce the time taken to specify a problem and produce a valid driver program for the numerical software.

The user interface part of the system will ensure that the user has supplied all the information needed for the problem specification. A fill in the gaps, form-like interface, indicates which information is required. The interfaces can also provide sensible default values for different numerical and physical solution parts which will provide guidance for the users. Sensible default values are essential for novice users of the system. This also enables users with limited experience of the numerical solution of PDEs to utilise the

software and obtain results. The aim of a visual user interface to the specification system must be to ensure that all the relevant information is specified. This information is given either by the user or by the system in the way of sensible default values.

The use of a visual interface will also enable the user to specify the information in a natural form. Equations can be specified in a natural way, the functions needed by the numerical code constructed by utilising one of the many computer algebra systems available, Maple in this case. Boundary and initial conditions can be defined in a similar way. The choice of methods for the various solution parts are presented in list form for user selection. Any numerical information that is required, such as tolerance values, can be requested when needed. A visual problem specification system should therefore allow the user to define the information in a natural way which is quicker, easier and more convenient than the previous method.

Finally the visual specification system will reduce the chance of mistakes from the translation of the problem information to a valid driver program. The use of a postprocessing system to automate this process again speeds up the overall problem specification process. The production of the driver program should be detached from the visual interface dealing with the specification process. This will allow the visual interface part of such a system to be constructed as a generic interface for two dimensional PDE problems. This is another important aim; to construct a visual user interface around the generic nature of the problem under consideration. The validity of the system will be demonstrated by using such a system to specify the information needed for the numerical code considered in this thesis. A suitable postprocessing step from the visual user interface producing the final driver program. The visual interface constructed here will therefore be tailored to accommodate the numerical code although many aspects will be generic and equally applicable to other numerical software in this area when combined with a suitable postprocessing subsystem.

The aims of a visual problem specification system as an integral component of a PSE for the numerical solution of PDEs are therefore given as

- To decrease the time taken from the specification of the problem to the creation of a valid driver program.
- To ensure that all the information needed by the numerical code is provided.
- To guide the user in the specification process.

- To have sensible default values for the solution parts.
- To allow the user to specify the problem in a natural way and avoid the need for explicit programming

Overall these aim to provide a more convenient and easier way to specify the problem.

The next section will look at the construction of the visual specification system, the high level structure of the user interface and the implementation of it to ensure portability. A description of the postprocessing subsystem dealing with the actual construction of the driver program will follow.

5.4 Construction of the Interfaces

The starting point for the construction of the user interface to the problem specification system was to consider the nature of the problem to be solved. The information needed to specify the problem was examined and this was split into four main groups. The mesh information including boundary and initial conditions, the solution information including adaptivity, the equation specification and finally the problem information looking at the nature of the problem, a time dependent or steady problem. The information required by the numerical code for a time dependent problem is given in Figure 5.1.

The figure shows the information needed to solve a time dependent two space dimensional PDE. The system can then use the information required to create a suitable driver program for the numerical software via the postprocessing step. The aim is to create a user interface for the visual problem specification system that is independent of the postprocessing subsystem and the numerical software under consideration. This would only appear to be possible at the high level outlined in Figure 5.1, the actual implementation having to consider the numerical code and the different ways it is able to solve various parts of the problem. The remainder of this section will look at each part of the overall specification, describe the information required and look at the construction of an interface for each group to obtain the information from the user. In this way the numerical and physical aspects of the problem can be separated.

Each user interface is constructed using the X Window system and the OSF/Motif Widget set [94] to ensure that portability. Other basic building tools, as outlined in Chapter 2 are also used In the construction of the user interfaces. These include the

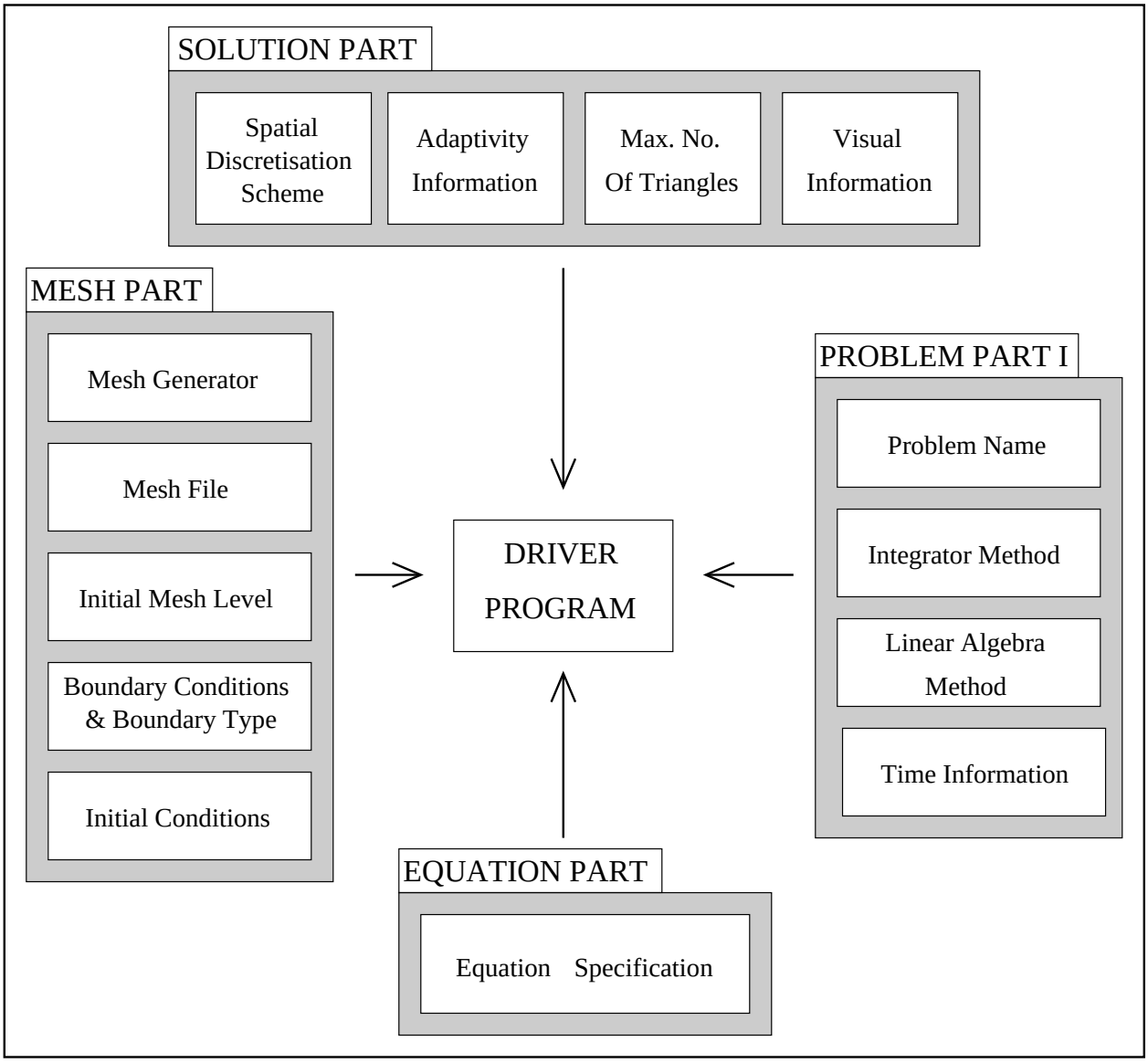

Figure 5.1: Information required for a time dependent problem

LaTeX document processing tool [61] and the Maple computer algebra system. A full description of how these are used is given in the relevant subsection.

The user interfaces described below have many facilities in common, each interface has a help button. The help button provides a text window which will give a brief description of the information required and the structure of the interface. Each interface starts with default values or settings, these values were obtained from the developers of the numerical code. At the end of each user specification process the information supplied will be stored so that the postprocessing subsystem can use it to create the driver program. This information is used as the new default values for the user interfaces when next invoked. This allows the user to then change one or two parameters easily to create a new driver program; to refine a particular problem rather than having to specify the whole problem again. The way in which this information is stored and the creation of the driver program from this is described later. Each interface also has a cancel button, this allows the user to kill the visual interface without changing the previously selected options and values.

5.4.1 The Solution Interface

The solution part of the problem specification deals with four areas: the spatial discretisation scheme, the adaptivity information, the maximum number of triangles and the use of the visual program accompanying the numerical code. The information required for each of these is outlined below along with the way the user interface requests this information. The default values for each are given along with a justification for the choice of each value.

The user interface for the solution part of the problem is in two stages. The user selects the values for the maximum number of triangles, selects the spatial discretisation scheme and sets the adaptivity and visual states to either on or off. If the adaptivity is set to on pressing the ok button on the interface will create another window prompting for the absolute and relative spatial tolerance values needed for the adaptivity routines to control the spatial error. If the adaptivity is set to off no further information is required and no further action is needed. Figure 5.2 shows this interface, the adaptivity is set to on and therefore the interface prompts for the numerical tolerance values required.

- The **Spatial Discretisation Scheme** information is used to determine the numer-

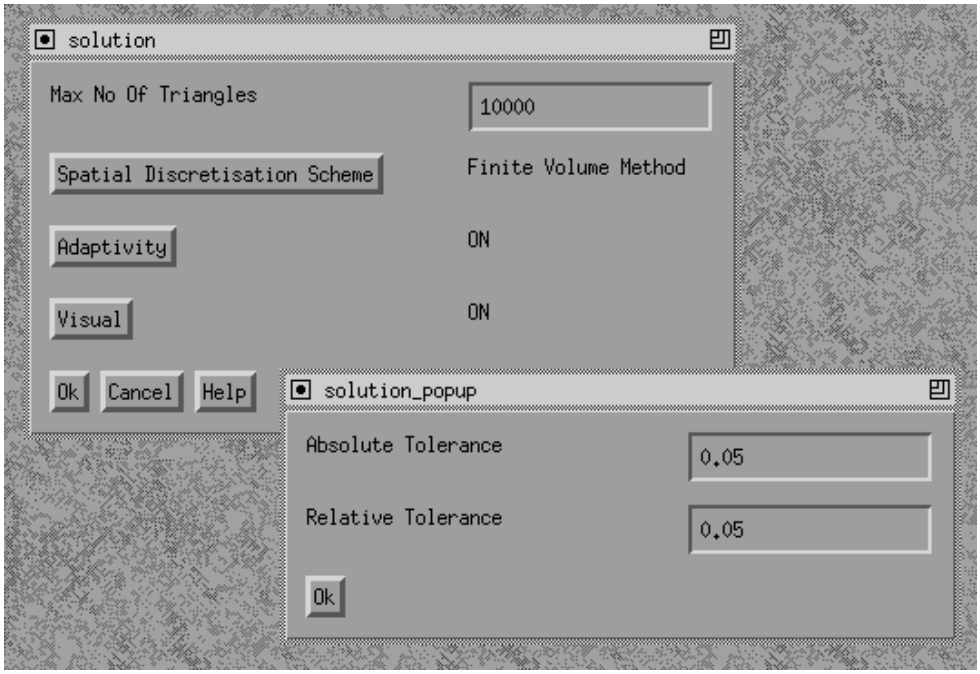

Figure 5.2: The User Interface for the Specification of the Solution Information

ical scheme used to solve the problem. The numerical code under consideration has only one scheme for this, the finite volume method. The default value for this option is therefore the finite volume method. A menu of options is provided for this information, the user selecting one from the menu. This allows the number of methods to be increased easily by simply adding more options to the menu. The output from this interface also allows easy expansion of all menu driven choices.

- The **Adaptivity Information** is concerned with the use of adaptive routines within the solution process. This option may be switched on or off. If adaptivity is required, the on option, the absolute and relative spatial tolerance values are required to control the spatial error. The default value is set to off, if the user switches this on then the default numerical values for the tolerance variables used are 0.005, these values are given as sensible defaults by the developers of the numerical code.

- **Max No. Of Triangles** is used to get a value for the maximum number of triangles that the numerical code may use. This limits the number of triangles in the mesh and therefore the number of points over the numerical domain. This therefore places bounds on the adaptivity routines, hence its inclusion in the solution part rather than

the mesh part. It can also be seen as limiting the overall speed of the computation the fewer triangles that are allowed, the quicker the solution process.

- The **Visual Information** is concerned with allowing the user to specify some form of visual output during the solution process. With the numerical code under consideration a graphics package developed with the code may be used to visualise the solution. The visual information can be switched on or off, the default is off. The visual program with the code requires some information to successfully run but this may be easily obtained from the numerical domain specification file and therefore the user is not required to supply any further information. The visual program also requires the IRIS GL [91] graphics language.

5.4.2 The Equation Interface

The equation part of the problem specification aims to allow the user to define the equation. At the current time only single equations may be specified. This is an obvious area for enhancement as problems consisting of systems of PDEs over a spatial domain are very common. Many problems can be defined in terms of a single PDE and the facilities provided by the equation interface are equally applicable to multiple PDE specification. As with many numerical codes the code under consideration deals with equations given by a generic formula. Figure 5.3 shows the equation interface giving the master equation in terms of a set of functions and the dependency of these functions.

For steady problems the term β is set to zero to remove the time dependence of the problem and the independent variable t is ignored. For the majority of time dependent problems β is unity. The terms f_x and f_y define the advective fluxes which lead to wave like structures in the solution u. The terms g_x and g_y define the diffusive fluxes which lead to diffusion processes in the solution u. The term S, the source term, can be used to add other processes such as chemical kinetics.

The main aim of the user interface for the equation part is to provide a way for the user to define the equation in a natural way. The help button on the interface provides some guidance on the use of the interface and the structure. The interface utilises the Maple computer algebra system to provide suitable input giving the user a standard and easy way to express complicated functions. Maple is also used to convert the function defined by the user into C functions which the driver program requires. A full description

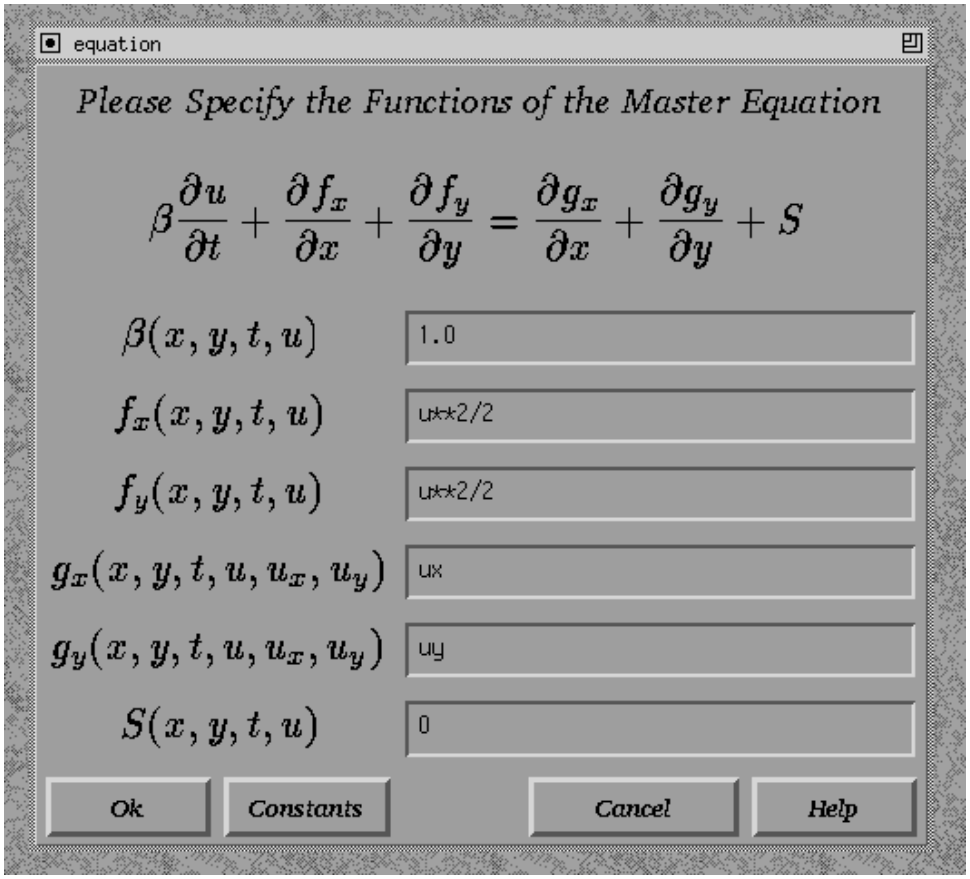

Figure 5.3: The User Interface for the Specification of the Equation Information

of this will be given later.

The equation interface also allows the user to specify constants that can be used to define the PDE and these are then transferred to the driver program. Figure 5.4 shows the subwindow of the interface which is created when the user presses the Const button on the interface and which allows numerical constants to be defined by the user

Another feature of the equation interface is its facility to create a LaTeX document of the equation specified by the user. This provides a high quality copy of the equation in full mathematical notation. Again Maple is utilised in this step, Maple provides routines to convert simple mathematical functions into LaTeX compatible text. When combined with a suitable template file a LaTeX document can be produced. The interface communicates with Maple on a primitive level, see Chapter 2 for a discussion on this, a set of commands are output to a temporary file, Maple executes the commands and outputs the results to

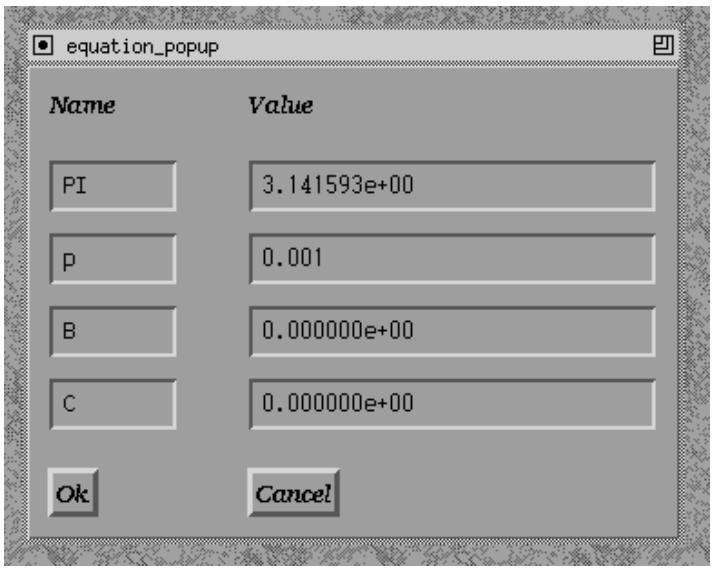

Figure 5.4: The Constant Part of the Equation Interface

another temporary file. The interface program can then read the results from this output file. These are then combined with predefined strings to produce a suitable input file for LaTeX.

5.4.3 The Mesh Interface

The mesh information interface provides a way for the user to enter information related to the numerical simulation of the physical domain and information for the mesh generation software. The interface again is in two stages, the structure of the second dependent upon the information the user supplies in the first stage. The first stage of the mesh interface is shown in Figure 5.5.

The information asked for in the first stage of the mesh interface is given below. The first stage deals with three pieces of information from Figure 5.1 and also requires the user to specify the number of boundary conditions. This information is used in the second stage of the mesh interface.

- The **Mesh File** is the name of file containing the numerical specification of the domain. The default is set to *square_file*; a simple square domain with corners at $(0,1),(1,1),(1,0),(0,0)$.
- The **Mesh Generator** is the software to be used to generate a triangular mesh

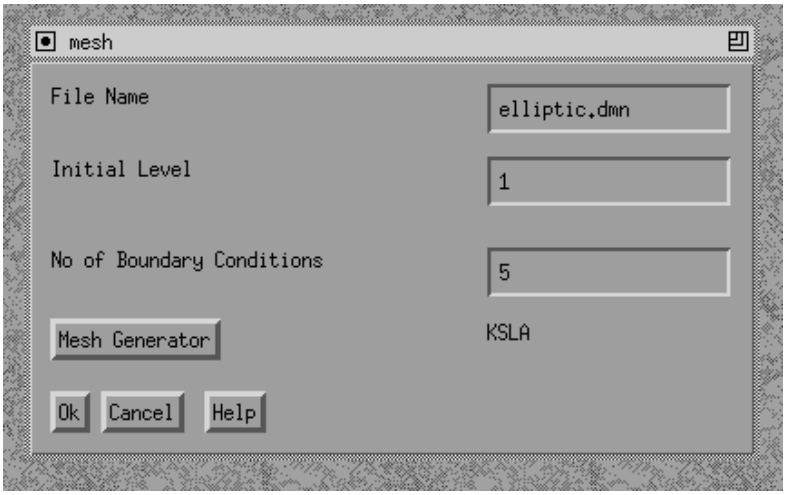

Figure 5.5: The First Stage of the Mesh Interface

from the numerical specification file. At the time of the initial development of this interface the numerical code considered here had the ability to utilise only one mesh generation software package. A second mesh generator has since been added. The result of this is that the mesh interface relies on the filename specified by the user containing the numerical specification of the domain is in the KSLA input format [35]. A full description of this is given in Chapter 4 The default selection for the mesh generator is set to the KSLA option.

- The **Initial Mesh Level** is the initial level of refinement the numerical software will add to the mesh returned by the mesh generation software. The default initial mesh level is set to 1.

The second stage of the interface allows the specification of the initial conditions, boundary conditions and the boundary types, Figure 5.6 illustrates this.

An outline of the boundary is also displayed and the boundary edges are named with a unique integers, see Figure 5.7. This deals with the remaining information outlined in Figure 5.1.

The second stage of the interface needs to know enough about the numerical specification file to extract and reproduce the geometry. Because of this it needs to know about the format of the numerical specification file used by the mesh generation software. This creates problems with regards to the generic nature of this interface. The advantages

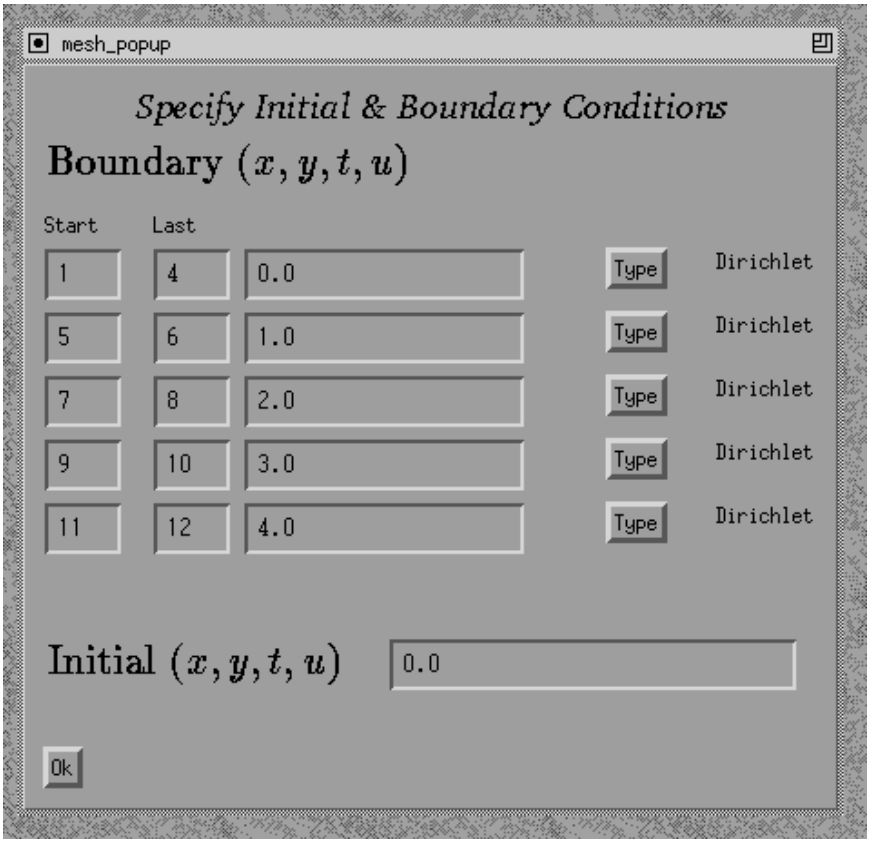

Figure 5.6: The Second Stage of the Mesh Interface I

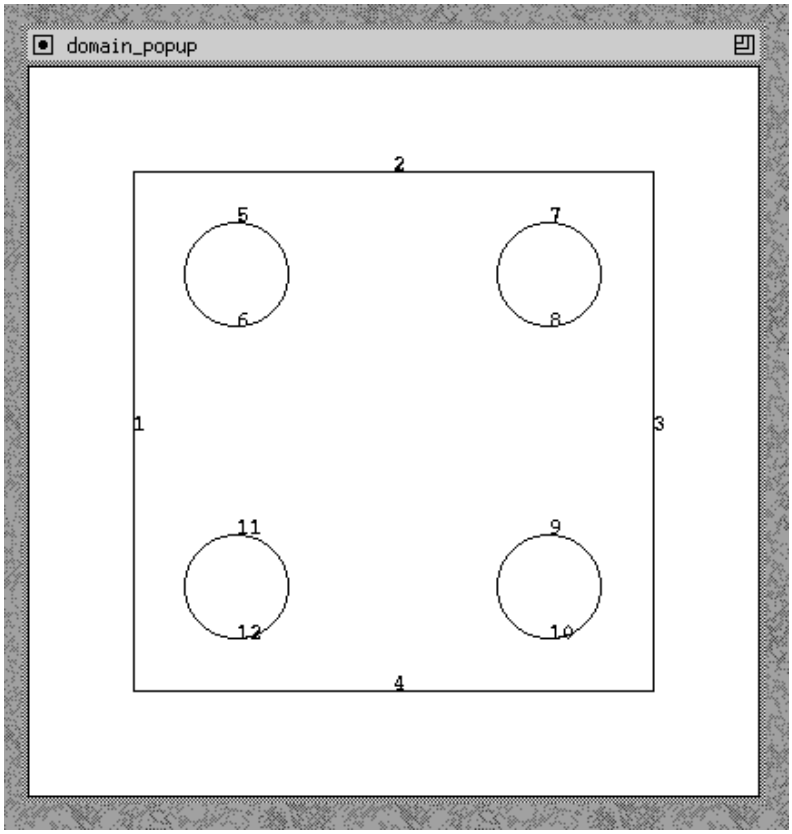

Figure 5.7: The Second Stage of the Mesh Interface II

that a visual representation of the domain bring to the specification of the boundary conditions provide more than enough justification for this however.

Other user interfaces combine the boundary specification with the geometric modelling process or limit the numerical domain such that a text description is adequate. Neither of these options seemed appropriate and so the use of the mesh generation software to extract the geometry was chosen.

- The **Initial conditions** are specified using Maple syntax allowing complex functions to be constructed. Maple is used to convert this information into a function required by the driver program is the same way as the equation information. The default value is 0.0.

- The **Boundary conditions** are specified again using Maple. The boundary condition on each edge is specified by placing conditions on consecutive lists of edges.

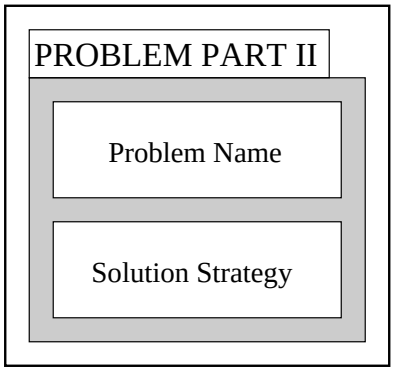

Figure 5.8: The information required for a steady problem.

The start edge and end edge need to be specified. The boundary condition upon those edges can then be defined. The default boundary conditions are set to 0.0.

- The **Boundary type** is allowed to be one of three possible values, Neumann, Dirichlet or Flux. The default value is set to a Dirichlet boundary condition. Other forms of boundary conditions such as combinations of the above conditions are not catered for at this current time.

5.4.4 The Problem Interface

The problem interface can follow one of two paths depending on the nature of the problem. Figure 5.1 shows the information required for a time dependent problem. The other main class is that of a steady state problems in which the solution to the problem does not vary in time. The information for a steady problem is shown in Figure 5.8.

The first stage of the problem interface requires the user to specify the type of problem, either time dependent or steady. The other item requested by the interface at this stage is the **Problem name** name given by the user. The default string for this is given as *test*.

If a time dependent problem is chosen then the interface has two further stages. The second stage will ask for the integrator software module and the linear algebra software module used. The third stage then deals with the time step information that is required. The three stages of the interface can be seen in Figure 5.9. The default values for these items were again obtained by consulting the developers of the code.

- The **Integrator method** looks at selecting a module in the overall solution process

to handle the integration. At present the numerical code supports two methods, the DASSL type integrator [70] and the theta method [87]. The default value at present is the theta method.

- The **Linear Algebra method** looks at the module responsible for the linear algebra part of the solution process. The default selection for this is the Watsit package [87]. Other possible options are to use the NAG sparse matrix routine.

- The **Time Step information** is dealt with in the third stage of the problem interface. The interface prompts the user for the **Start time** of the computation, then the **Number of Time Output Points** where the code will halt computation to show the solution and then the **Time increment** between each output point. The computation will end when the last specified time output point is reached. For example a start time of 0.0 with 10 time output points and a time increment of 0.1 will stop at time 1.0. The default values for these values are set to, 0.000, 20 and 0.050 respectively.

If the problem is a steady problem, Figure 5.8 shows the information required. There are only two levels of the interface, the second level dealing with the solution strategy module used, see Figure 5.10.

- The **Solution strategy** looks at selecting the module for steady state problems. There is currently one module, the black box solution strategy module (BBOX). This is a simple invocation of the NAESOL package. The default for this is therefore the black box module [75].

The problem here as with the other part of the visual specification system is the desire to construct a set of generic interfaces for the specification of two dimensional PDEs. However, in many cases the numerical code considered has a limited number of modules to select, in many cases only one is available.

Due to this fact many of the options appear superfluous to the information required to construct a successful driver program. Whilst this has been the situation with the user interfaces it is hoped that these are constructed such that the expansion of possible choices is provided for. This is true not only in the visual aspect of the user interface but also in the output produced by these. The use of a simple file based system which the

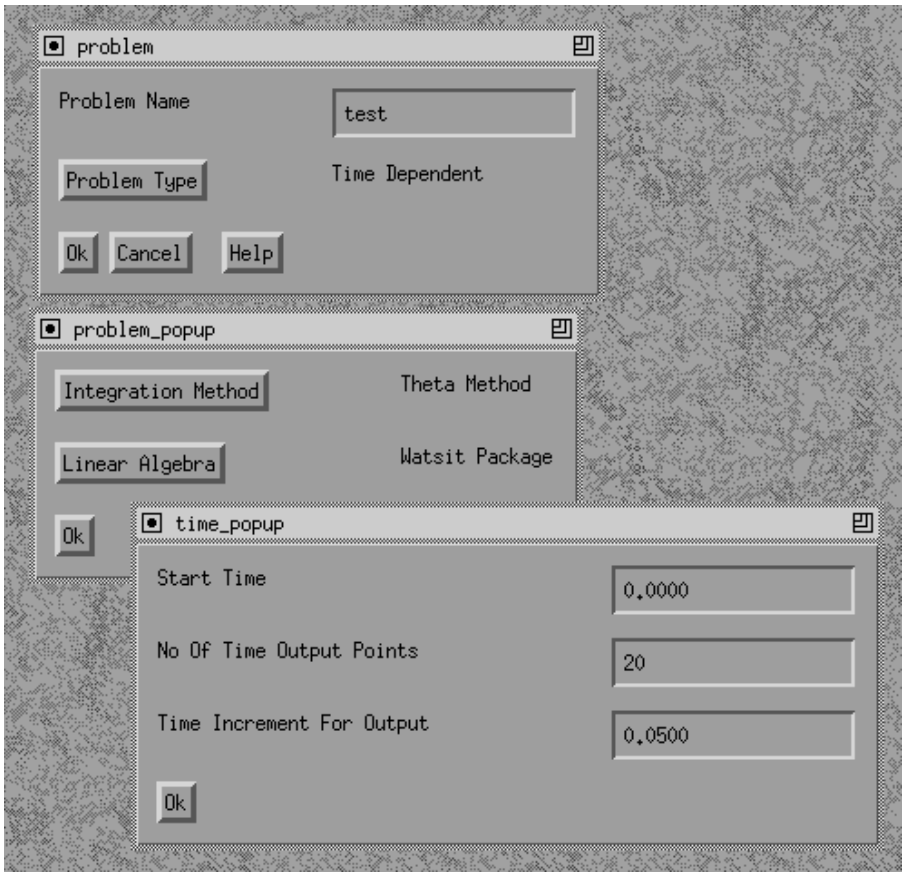

Figure 5.9: The Problem Interface for a Time Dependent Problem.

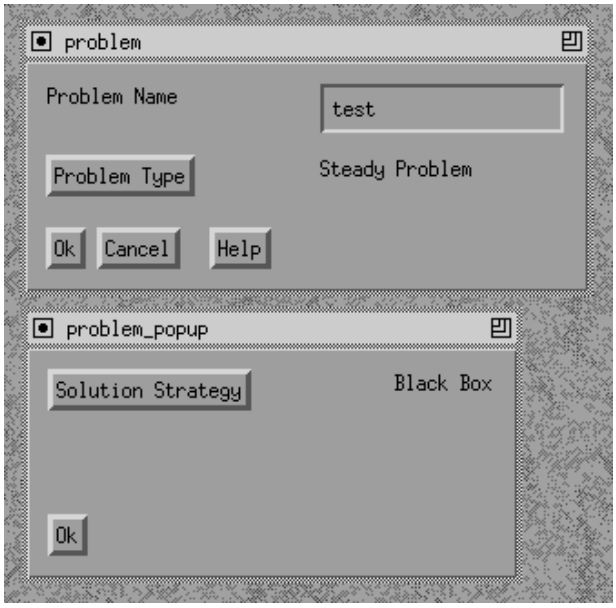

Figure 5.10: The Problem Interface for a Steady State Problem.

postprocessing subsystem uses as input is also able to be easily expanded. This system is explained in the next section.

5.5 Output of the Interfaces

The following section describes the output of each of the user interfaces. This takes the form of a file containing integers, reals, filenames or the Maple notation for a function or mathematical expression.

The files are also used as default values for the user interfaces when next invoked. This system of a visual user interface with a file based intermediate step between them and the postprocessing subsystem responsible for the creation of the driver program allows the two to be kept separate and allows a visual summary window to be fitted to the postprocessing system. The visual summary allows the user, at a glance, to check that all the information is present and correct. If a mistake is spotted or various parts of the problem specification need to be changed then the user can simply reinvoke the required visual user interface, change the value and instruct the visual summary window to rescan the output files produced by the user interface. The following subsections will describe the output files for each interface and demonstrate that this system can be expanded easily.

5.5.1 The Solution Interface

The output from the solution interface takes the following form

- An integer, the value for the maximum number of triangles
- An integer to represent the spatial discretisation scheme used. A 0 to select the default value, a 1 to select the finite volume method. As more solution methods become available they can easily be represented this way.
- An integer to represent the adaptivity state, 0 = OFF, 1 = ON.
- An integer to represent the visual state, 0 = OFF, 1 = ON.
- A real, the absolute spatial tolerance value, if the adaptivity is turned on.
- A real, the relative spatial tolerance value, if the adaptivity is turned on.

5.5.2 The Equation Interface

The output from the equation interface takes a different form, a set of strings represent the different functions of the governing equation, see Figure 5.3. These are followed by the user defined constants.

- The Maple notation to represent the function β.
- The Maple notation to represent the function f_x.
- The Maple notation to represent the function f_y.
- The Maple notation to represent the function g_x.
- The Maple notation to represent the function g_y.
- The Maple notation to represent the function S.
- The name then value of the first constant, if used.
- The name then value of the second constant, if used.
- The name then value of the third constant, if used.
- The name then value of the fourth constant, if used.

5.5.3 The Mesh Interface

The output from the mesh interface takes the following form

- The name of the file containing the numerical specification of the domain.
- An integer, the value of the initial level of mesh refinement.
- An integer to represent the mesh generation package to be used, 0 is the default, 1 is the KSLA package, 2 will, for example, represent the GEOMPACK package [51].
- An integer to give the number of boundary conditions.
- For each boundary condition there is the boundary condition function specified in Maple syntax. This is followed by the names of the start edge and finish edge of a list of consecutive edges which this boundary condition function applies to. These are then followed by an integer to represent one of the three different boundary types, Dirchlet, Neumann or Flux boundary.
- The Maple notation of the initial condition function over the domain.

5.5.4 The Problem Interface

The problem interface output file is dependent upon the problem type, either time dependent or steady problem. The data structure for the time dependent problems is

- The name of the problem, which is user defined.
- An integer to represent the problem type, 0 = Steady, 1 = Time dependent, 1 in this case.
- An integer to represent which integrator method to use.
- An integer to represent which linear algebra method to use.
- An integer giving the number of time output points.
- the value of the start time.
- The value of the time increment for each step.

The output file for the steady problem is

- The name of the problem, which is user defined.
- An integer to represent the problem type, 0 = Steady, 1 = Time dependent, 0 is in this case.
- An integer to represent which solution method to use.

5.6 Construction of the Driver Program

The information provided by the users and encoded by the interfaces is then passed to the postprocessing subsystem responsible for creating the driver program for the numerical code. This postprocessing subsystem has two main stages, the first stage of the procedure is to provide a visual summary of the information such that the user may easily validate the problem definition. The user can then trigger the creation of the driver program, the second stage. This process is triggered from the visual summary window. The final output of the postprocessing subsystem is a valid driver program which can be compiled and linked to run the numerical software. Various examples of this process will be discussed in the next section where three problems are considered as case studies.

The postprocessing subsystem makes use of the Maple computer algebra system to generate C code to include in the driver program. Maple is used by the driver in a similar way to the equation interface when it produces LaTeX output. The interaction with Maple is again via a file based system with commands output to a temporary file. This file is used as input to Maple. Maple produces another temporary file containing the results. These results are then read from the file by the postprocessing subsystem and transferred to the driver program. The utilisation of Maple in this way stems from the development of computer algebra systems as user orientated stand alone systems, see Chapter 2. The postprocessing system is utilising Maple by pretending to be a human user.

The second stage of the postprocessing system deals with building the driver program from the information specified. The structure of the driver program deals first with including the relevant header files for the solution parts considered, followed by the declaration of the user defined constants. The various functions required are then defined; the initial conditions, the boundary conditions and the appropriate functions required for the finite volume method. A monitor routine is also defined and his provides a way for

the user to examine various aspects of the solution process. The numerical information is then defined followed by a set of routines to instruct the numerical code where to find the previously defined functions it requires and which software packages to use. The driver program then starts the solution process.

This idea of fitting information into a template program has been used many times before in scientific computing and PDE software. Enquist and Smedsaas used a mathematical language to define and describe one dimensional hyperbolic and parabolic initial boundary problems [32]. The results of the specification language was to produce a FORTRAN program to solve the problem. More recently Bentley et al. look at template driven interfaces for numerical subroutines [7]. They look at using a mathematical language that can be combined with well established robust software libraries. The result of this language is again to produce a FORTRAN program which will call a library routine and then return the result to the user. This system has been used to solve PDEs in one space variable.

The driver program, as well as having the ability to execute the numerical software, must also be easy to understand, well structured and well documented to allow the user to modify the driver program if required. One example where this is important is the Riemann solver function required by the finite volume method. The current trend is for the user to specify the approximate Riemann solver by one of the many numerical methods available, see Chapter 3 for a discussion on this. A simple differencing scheme is used by the postprocessing subsystem. This uses the advective flux, defined by the f_x and f_y functions of the equation. More complex Riemann solver may need to be employed to give better numerical results, this issue will be addressed further in Chapter 7. The driver needs to advise the user of this fact. The ability of the postprocessing subsystem to recognise where potential problems in the driver program may arise is as important as automating the simplier parts of the process. The driver program can then at least provide a well structured template program which more experienced users may modify.

5.7 Three Case Studies

The following section looks at three different problems. The visual problem specification system is applied to each. The aim is to create valid driver programs for each. The driver programs can then be compared to the driver program manually written

by users of the software. A description of the problem is given, the domain, boundary conditions, initial conditions and solution process. This section aims to show that the examples given will display the ability of the interface to produce valid driver programs.

The summary windows of each problem along with the driver programs produced by the visual specification system are given in Appendix C.

5.7.1 A Parabolic PDE – Heat Equation

The first problem is one already mentioned in Chapter 3, a time dependent Poisson problem given by the following equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

The domain and boundary conditions are given in Figure 5.11 the initial conditions are set to 0.0 over the domain.

- **Problem Name:** Parabolic.
- **Spatial Discretisation Scheme:** Finite Volume Method.
- **Problem Nature:** Time-Dependent.
  - **Integrator Method:** Theta Integration Module.
  - **Linear Algebra Method:** WATSIT Linear Algebra Module.
  - **Start Time:** 0.00.
  - **Number of Time Output Points:** 6.
  - **Time Increment Between Output Points:** 0.25.
- **Adaptivity:** Yes.
  - **Absolute Spatial Tolerance:** 0.05.
  - **Relative Spatial Tolerance:** 0.05.
- **Maximum Number of Triangles:** 10000.
- **Numerical Domain File:** parabolic1.dmn, See Figure 5.11.
- **Mesh Generation Package:** KSLA.

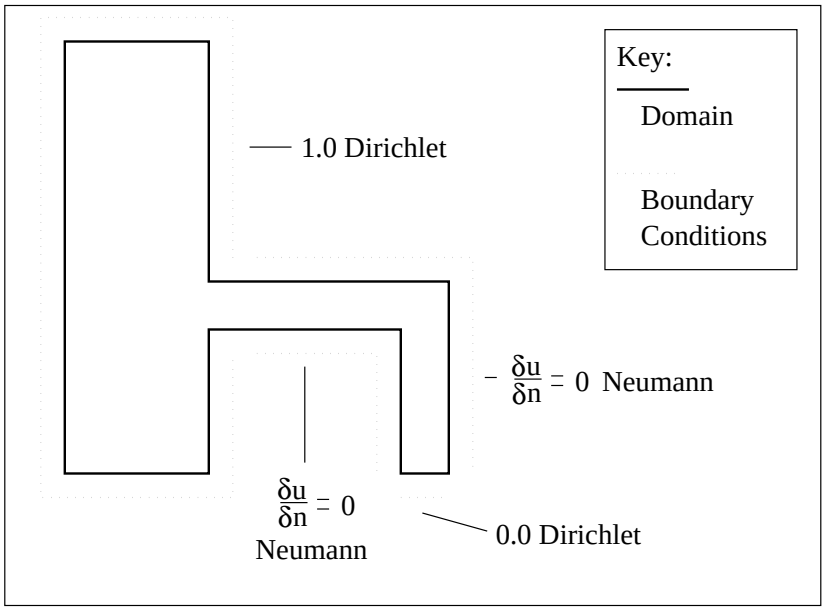

Figure 5.11: The Domain for the Heat Eqn and Boundary Conditions.

- **Initial Mesh Refinement Level:** 4.
- **Initial Conditions:** 0.0.
- **Boundary Conditions & Type:** See Figure 5.11.

The full driver program for this example produced by the visual specification system can be seen in Chapter 3. This example shows the ability of the system to deal with simple time dependent problems. Figure 5.12 shows the solution to this problem at time $t = 6.5361e - 2$.

5.7.2 An Elliptic PDE – Laplaces Equation

The second problem is an elliptic problem defined by the following equation.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

The initial conditions are set to 0.0 over the domain, this and the boundary conditions are shown in Figure 5.13 The problem is a steady-state problem, the specifications required by the visual problem specification systems are

- **Problem Name:** Elliptic.

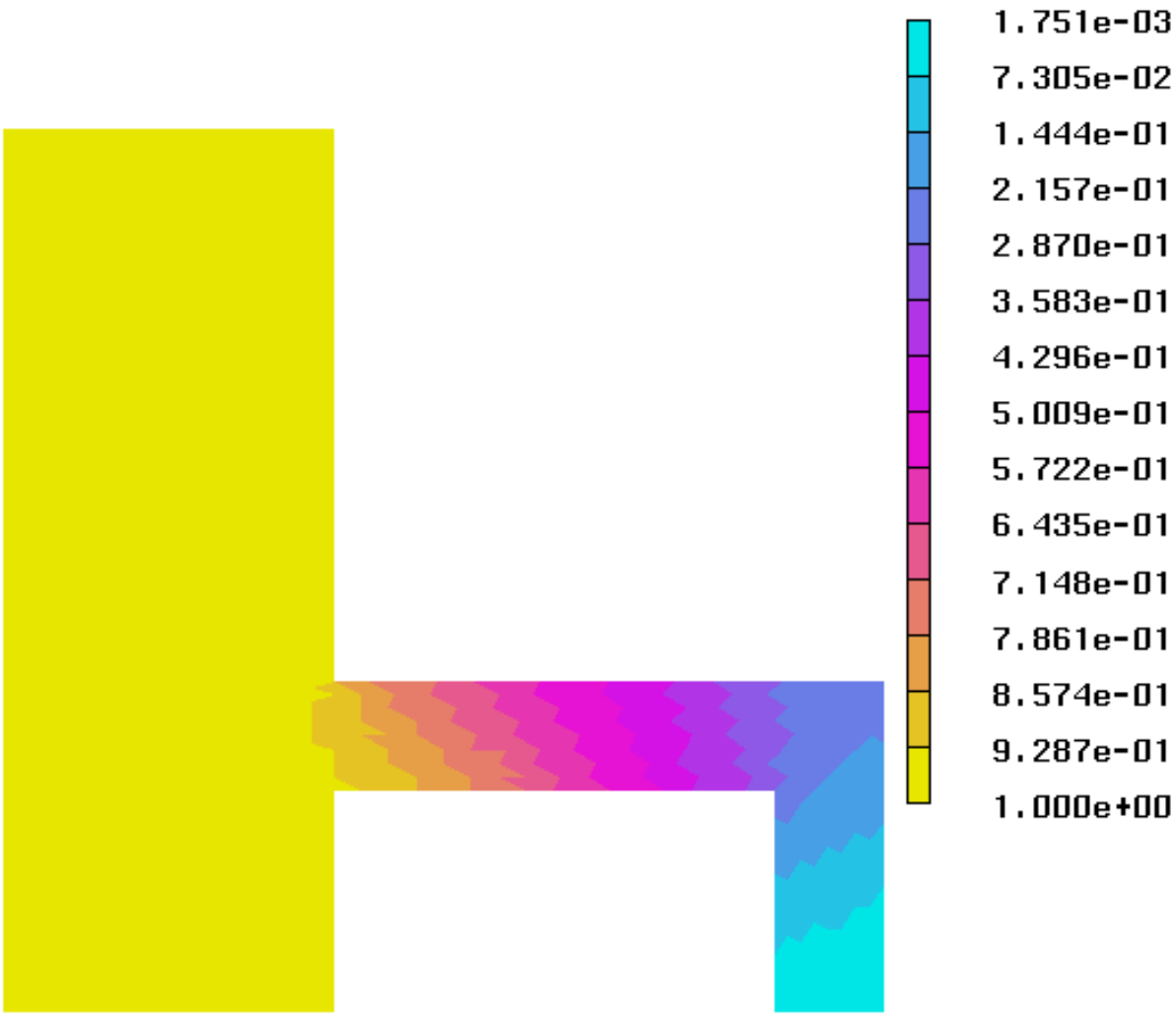

Figure 5.12: The Solution of the Heat Equation at time $t = 6.5361e - 2$

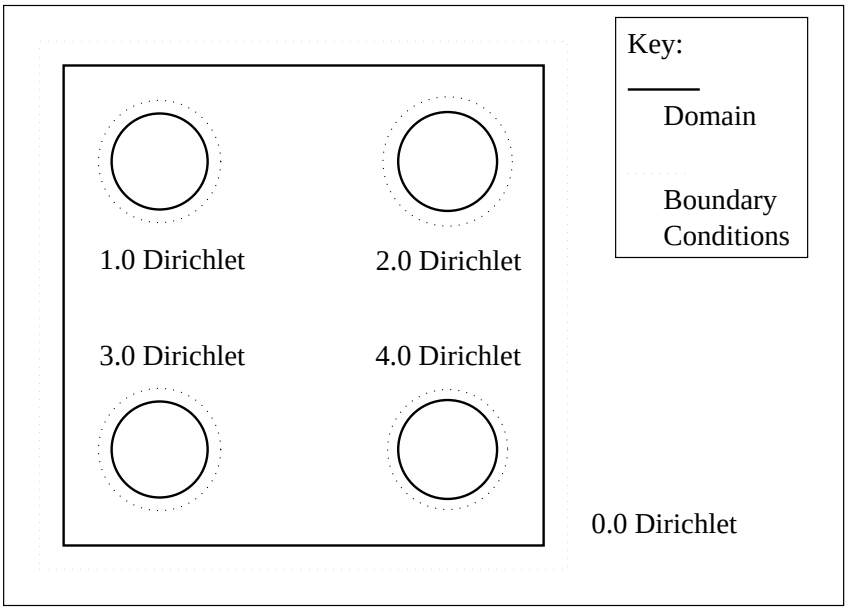

Figure 5.13: The Domain for the Elliptic Problem and Boundary Conditions.

- **Spatial Discretisation Scheme:** Finite Volume Method.
- **Problem Nature:** Steady-State.
  - **Solution Strategy Module:** Black Box.
- **Adaptivity:** Yes.
  - **Absolute Spatial Tolerance:** 0.05.
  - **Relative Spatial Tolerance:** 0.05.
- **Maximum Number of Triangles:** 8000.
- **Numerical Domain File:** elliptic1.dmn, See Figure 5.13.
- **Mesh Generation Package:** KSLA.
- **Initial Mesh Refinement Level:** 1.
- **Initial Conditions:** 0.0.
- **Boundary Conditions & Type:** See Figure 5.13.

Part of the driver program produced by this specification is given below; the include files required, the monitor routine and the main part of the program that will drive the numerical software. The various finite volume routines are omitted, in this case, as the form of these is similar to the example program given in Chapter 3.

```
/* Standard Include Files */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "S2D.h"

/* Include file for FVM */
#include "S2D_FVM_finite_volume_discretisation.h"

/* Steady Problem Include Files */

/* Include File for Black Box Solver */
#include "S2D_BBOX_black_box_soln_strategy.h"

/* Include file for visual routines */
#include "vwr_comms.h"

/* Monitor Routine */
void monitor (
  int                   neq,
  double                *u,
  double                *udot,
  double                time,
  double                *err,
  double                *ewt,
  S2D_Res_Status_Type   res_status,
  S2D_Monitor_Type      mon_type,
  S2D_Mon_Status_Type   *mon_status,
  S2D_Intgrtn_Obj_Type  *integ_obj,
  void                  *space_disc_data,
  void                  *users_data)
{
  /* ---------------------------------------- */
  /* Monitor Routine for user functions, allows  */
  /* user to monitor process insert things here  */
  /* ---------------------------------------- */

  int        interact;
  /* Viewer routine */
```

```
  vwr_send_mesh(integ_obj);
  if (mon_type == S2D_Initial_Mon)
    vwr_send_frame(integ_obj,u,u,time,&interact);
  else
    vwr_send_frame(integ_obj,u,err,time,&interact);

  /* ---------------------------------------- */
  /*                                          */
  /*      Insert Code Here If Required        */
  /*                                          */
  /* ---------------------------------------- */

  /* Return monitor ok status */
  *mon_status = S2D_Mon_Okay;

} /* Monitor routine */

/* Initial conditions routine */
void elliptic_ic( ... );

/* Boundary conditions routine */
void elliptic_bc( ... );

/* Diffusive function */
void elliptic_g( ... );

/* Riemann solver */
void elliptic_rs( ... );

/* Source Term */
void elliptic_src( ... );

/* Main */
main(int argc, char *argv[])
{
  S2D_Intgrtn_Obj_Type my_integ;
  int                  ntrimax = 8000;
  int                  ilevel = 1;
  double               atol = 0.050000;
  double               rtol = 0.050000;
  double               max_x = 10.000000;
  double               min_x = 0.000000;
  double               max_y = 10.000000;
  double               min_y = 0.000000;
```

```
  /* End of variable declarations */

  /* Initialise integration object */
  S2D_initialise(&my_integ);

  /* Steady Problem */
  S2D_steady(&my_integ,1,ntrimax);

  /* Spatial tolerance on */
  S2D_spatial_tol(&my_integ,S2D_Scalar_TOL,&atol,&rtol);

  /* Use Finite Volume scheme */
  S2D_FVM_initialise(&my_integ);
  S2D_FVM_initial_conditions(&my_integ,elliptic_ic);
  S2D_FVM_boundary_conditions(&my_integ,elliptic_bc);
  S2D_FVM_riemann_solver(&my_integ,elliptic_rs);
  S2D_FVM_diffusive_flux(&my_integ,elliptic_g);
  S2D_FVM_source_term(&my_integ,elliptic_src);

  /* Initialise viewing routine */
  vwr_init( min_x, max_x, min_y, max_y, 1.4);

  /* Set KSLA mesh information */
  S2D_ksla_mesh_generator(&my_integ,"elliptic.dmn",ilevel);

  /* Use Black Box Package */
  S2D_BBOX_initialise(&my_integ,-1.0);

  /* monitor routine */
  S2D_monitor(&my_integ,monitor);

  /* Integrate routine */
  S2D_integrate(&my_integ);

  /* close viewer */
  vwr_close();
}
/* End Of Driver */
```

This example shows the ability of the visual specification system to deal with simple steady-state problems. Figure 5.14 shows the solution to this problem.

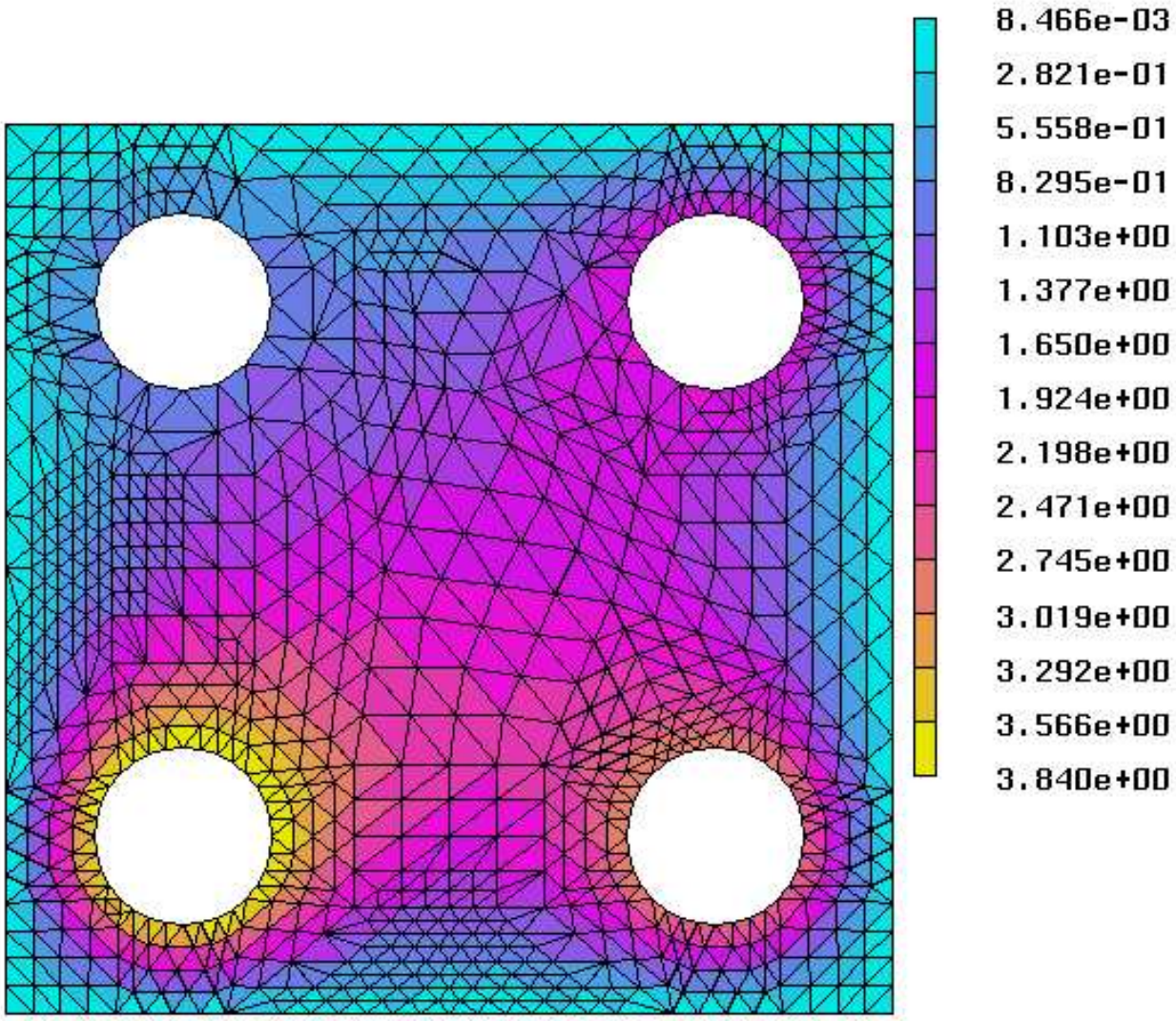

Figure 5.14: The Solution of the Elliptic Equation

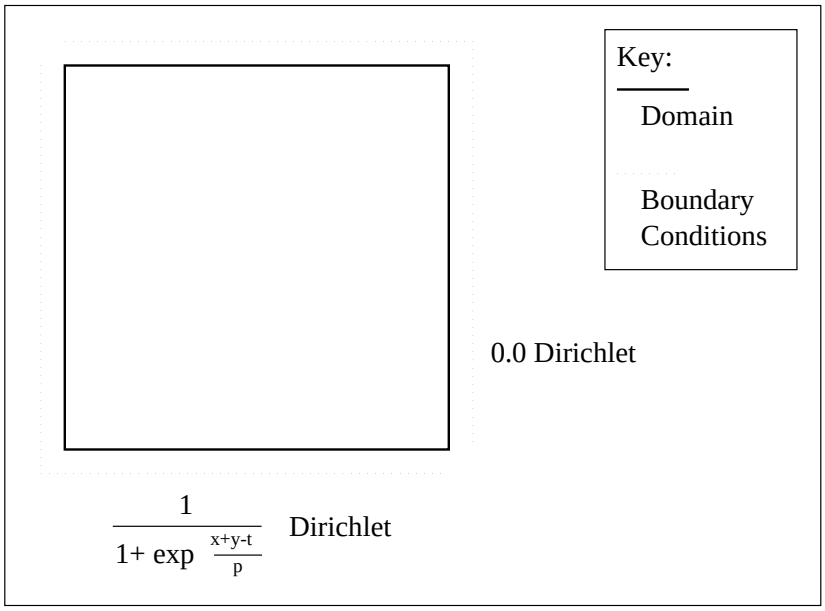

Figure 5.15: The Domain for the Burgers' Problem and Boundary Conditions.

5.7.3 Convection-Dominated PDE – Burgers' Equation

The final problem is a Burgers' equation given by

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) + \frac{\partial}{\partial y}\left(\frac{u^2}{2}\right) = p\frac{\partial^2 u}{\partial x^2} + p\frac{\partial^2 u}{\partial y^2}$$

where p is a constant defined as 0.01. The domain and boundary conditions are given in Figure 5.15, the initial conditions over the domain are

$$\frac{1}{1+\exp^{\frac{x+y-t}{p}}}$$

- **Problem Name:** Burgers'.
- **Spatial Discretisation Scheme:** Finite Volume Method.
- **Problem Nature:** Time-Dependent.
  - **Integrator Method:** Theta Integration Module.
  - **Linear Algebra Method:** WATSIT Linear Algebra Module.
  - **Start Time:** 0.15.

  - **Number of Time Output Points:** 15.
  - **Time Increment Between Output Points:** 0.10.

- **Adaptivity:** Yes.
  - **Absolute Spatial Tolerance:** 0.05.
  - **Relative Spatial Tolerance:** 0.05.
- **Maximum Number of Triangles:** 10000
- **Numerical Domain File:** burgers.dmn, See Figure 5.15.
- **Mesh Generation Package:** KSLA
- **Initial Mesh Refinement Level:** 3.
- **Initial Conditions:** $\frac{1}{1+\exp^{\frac{x+y-t}{p}}}$
- **Boundary Conditions & Type:** See Figure 5.15.

This example demonstrates the construction of the approximate Riemann solver by the postprocessing subsystem for the driver program. It also shows how the system can deal with more complex functions for the initial and boundary conditions. The routines from the driver program for the initial conditions and Riemann solver are given below. It must be noted that due to the naivety of the Riemann solver used in this example the solution is negative close to the wave front, the problems associated with this will be discussed later in Chapter 7.

```
/* Initial conditions routine */
void burgers_ic(
  TRIAD_Triangle  *tri,
  int             npde,
  double          x, y, t,
  int             sub_name,
  void            *users_data,
  double          *u)
{
  /* ---------------------------------------- */
  /* Initial conditions routine to specify the   */
  /* initial conditions of the problem           */
  /* ---------------------------------------- */
```

```
    u[0] = 0.1E1/(0.1E1+exp(0.1E3*x+0.1E3*y-0.1E3*t));

  } /* Initial conditions */

  /* Riemann solver */
  void burgers_rs(
    TRIAD_Line  *line,
    int         npde,
    double      x, y, t,
    int         sub_name,
    double      norm_x, norm_y,
    double      *u_l, *u_r,
    void        *users_data,
    double      *nf)
  {
    /* ---------------------------------------- */
    /* Riemann solver routine to specify the    */
    /* Riemann solver conditions of the problem */
    /* ---------------------------------------- */

    double u = ( u_l[0] + u_r[0] ) / 2.0 ;
    double f_x, f_y;

    /* ---------------------------------------- */
    /* A Simple Differencing is used to solve   */
    /* this problem. More complex methods may   */
    /* provide better results                   */
    /* ---------------------------------------- */

    f_x = 0.5*u*u;
    f_y = 0.5*u*u;
    nf[0] = f_x * norm_x + f_y * norm_y;

  } /* Riemann solver */
```

The solution to this problem at time $t = 9.8070e - 1$ is shown in Figure 5.16.

5.8 Evaluation of the Toolkit

When a user wishes to use the SPRINT2D numerical software a driver program must be created. At best the user will find a driver program for a previous problem with similar properties. At worst the user must construct a driver program from scratch. This

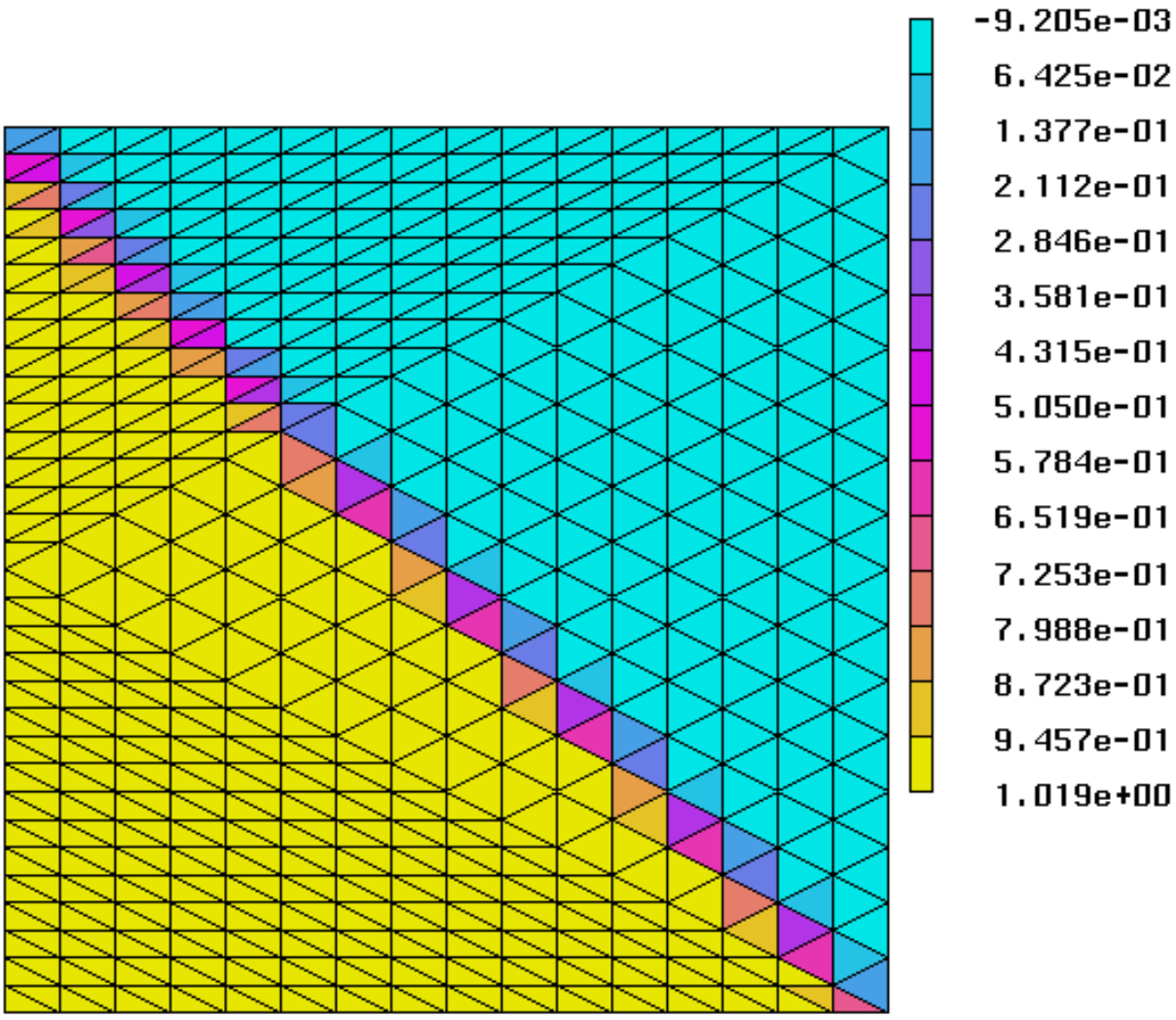

Figure 5.16: The Solution of the Burgers' Equation at time $t = 9.8070e - 1$

can be a lengthly process, the user must first define the problem, often hand drawing the numerical domain, extracting boundary names from the mesh specification file in order to define the boundary conditions. The various modules used need to be determined and then the appropriate SPRINT2D command to initialise them. The numerical code requires all aspects of the problem to be identified, if one item is not defined in the driver program, for example, the mesh generation package to use, the numerical software will fail. This can often be one of the very simple mistakes that are hard to spot. The VPS systems aims to provide an easy and natural way for the user to visually specify all aspects required by the numerical code and to produce a suitable driver program.

This section aims to provide some user evaluation on the VPS system, as with the Visual Domain Specification (VDS) tool described in Chapter 4, the evaluation included some additional features the user would wish to see. Overall the tool was seen as having a positive effect on the solution specification process. The users saw the tool as very easy to use and very accessible and intuitive. The view was that the VPS system provided an easier method than programming from scratch and was also less error prone than taking an existing program and modifying it. The layout of the visual interfaces made it very clear what information was required, the VPS system identifying what the numerical code required, something the code itself fails to do.

The use of a visual interface to display the numerical domain and the boundary edge names was seen as helpful and much simpler than manually performing the same task. The equation interface was also seen as a very natural way to define the PDE.

As mentioned earlier several comments about what additional features could be included were expressed. The specification of the boundary conditions by mathematical function were mentioned and the wish for a mechanism where the condition could be switched depending upon values within the code. It was noted that the current method would cover the majority of users and that with many other aspects of the driver program more complex mechanisms could be hand coded from the driver program produced by the VPS system. The wish to have the ability to change aspects of the numerical domain without specifying the interface displaying the boundary information was also expressed. The wish to save complete specifications and reload them was also suggested in addition to the last specification taken as new default values.

The current state of the VPS system is not as advanced as the VDS tool. It is clear to see that more work needs to be carried out on the initial prototype system to make

it more cohesive, robust and functional. The user evaluation and feedback presented here provides a starting point for this. Loading and saving specification details and switches for boundary conditions. Other possible enhancements, not mentioned by users, may include allowing the changing of the values presented in the summary window and the addition of a button in the summary window for viewing the domain.

Overall the VPS system was seen as a useful addition to the problem specification process, automating the construction of a valid driver program. It can reduce the overall time spent on this part of the problem, identify what the user needs to specify to drive the numerical code and provide an intuitive and natural way to specify the problem.

5.9 Summary

A visual user interface for defining the information needed to solve two dimensional PDEs brings many advantages. Combining this visual interface with a suitable postprocessing subsystem capable of producing a driver program for the numerical software places such a combination into the class of tools that form part of a PSE around the numerical software.

The visual problem specification system provides a considerable decrease in the time spent specifying the problem, thus allowing more time for examining the solution of the problem. The visual system ensures that all the required information is provided, either by the user or by the system itself and produces a driver program. The interface provides a suitable medium for allowing the user to specify the problem information in a more natural way. The logical grouping of the elements of the problem as well as the utilisation of previous information to help in the selection of the latter both help in this respect. The way in which the user is asked to provide the information also allows a more natural approach to the specification process than writing a single driving program.

The translation of the problem information into a driver program for the numerical code via the visual specification system eliminates any errors that may arise if this process is done manually. Such a system provides a more convenient, easier and quicker way to generate the driver program and eliminates the need for explicit programming.

The visual specification system is built around the generic description of two dimensional PDEs using the X Window system and the OSF/Motif widget set to ensure portability. The Maple computer algebra system is also used as well as the LaTeX document

processing system both of which are widely used in the scientific computing community. The validity of the system is demonstrated by using it to produce driver programs for numerical software for solving two dimensional convection–dominated PDEs developed at Leeds. Three case studies have been described to show this. The visual specification system has therefore been modified to accomplish this, but the design and construction of the user interface to the system is hoped to be strongly independent of the numerical software considered.

Chapter 6

Quadratic Interpolants for Triangular Cell-Centered Finite-Volume Schemes

6.1 Introduction

In this chapter the use of interpolation schemes as part of the numerical solution of PDEs is discussed. The desirable properties that such interpolants should have are outlined. The interpolant should complement rather than impose a particular form on the solution process. Such an interpolant is not only required within the numerical solution process, for the recovery of solution values and estimates of spatial derivative values, but is also needed for the visualisation of the solution to the PDE. Good visualisation is an essential aid to understanding the phenomenon being modelled for complex PDE problems.

Initially the focus of the chapter is on an interpolant that will produce numerical and graphical results respecting the underlying properties of the data, for example positivity. The aim is to construct such an interpolant for a two dimensional triangular based PDE solver. However, the conditions required will be constructed from the one dimensional case. Numerical and graphical results are presented. The use of such interpolants to construct spatial derivative estimates is then outlined. The current methods used are described and numerical results given.

6.2 Interpolation Schemes for PDEs

Many areas of scientific computing involve modelling real world problems. The visualisation of the solution to these problems is an essential aid in the understanding of the phenomenon being modelled. Interpolation schemes that will respect the physical properties of the underlying data are thus needed. One example of respecting this physical nature of the data is to produce values within a specific range, for example, to ensure positivity.

Many problems that require such treatment can be modelled by differential equations, either ordinary differential equations (ODEs) or partial differential equations (PDEs). An important feature of some hyperbolic PDEs is that initial smooth conditions may develop into shocks and discontinuities. Some interpolation schemes may produce results that introduce physically unreal values for such problems.

A wide range of numerical software exists for solving such problems in one spatial dimension, for example, the NAG numerical library provides routines to solve PDEs, [67]. However, perhaps more work has been done regarding preservation of inherent properties of data arising from the solution of ODEs, see Brankin & Gladwell for preservation of convexity [17], Higham for monotonicity [45] and Butt & Brodlie for preservation of positivity [20].

In two spatial dimensions, the numerical solution of partial differential equations (PDEs) often use triangular elements because they can accurately represent complex domains and may be used in conjunction with adaptive spatial meshes. The class of convection–dominated PDEs can be used to represent a large number of problems. A number of schemes for solving such problems use a cell–centred finite volume spatial discretisation scheme, see [11], [88], [63] and [29], [6] for details. Such schemes use triangular meshes and generate computed solution values at the centroid of each triangle and, as a by–product, interpolated values at the mid–points of the edges. It is important to stress that no solution values at the vertices of the triangles are known but it is still necessary to use this raw data at the centroid and edge midpoints to construct a satisfactory interpolant to calculate the solution values at non centroid points. This is useful in many areas such as spatial remeshing [87] and the visualisation of the solution.

An important feature of convection–dominated PDEs is that smooth initial conditions in such problems may develop into shocks and discontinuities. Many spatial dis-

cretisation schemes used within the solution process reduce their order of accuracy around shocks and discontinuities to ensure that spurious oscillations do not occur. In particular when the solution to the PDE is dominated by shock–like or steep wave features the accuracy may be only first order, and standard high–order approaches based on *smooth* exact solutions will not necessarily give the expected order of accuracy. This chapter will show that, not surprisingly, this phenomenon also applies to the interpolation method used. In the area of discretisation methods, for example, the scheme of Durlofsky et al. [29] is concerned with the construction of a linear interpolant over each triangle from surrounding centroid values but resorts to piecewise constant interpolation using the centroid value to preserve non-oscillatory behaviour in the PDE solution. The scheme by Devine and Flaherty [27] uses projection limiters to try to overcome the need to flatten extrema to first order accuracy. This is done by limiting the coefficients of Legendre polynomials successively; in this way lower order coefficients are only limited when needed and after all higher order terms have been limited. In this case accuracy is not lost in smooth regions. The scheme by Lin, Wu & Chin [63] uses local solution limiters to ensure that no new extrema are created. Two other important interpolation methods are those of Abgrall [1] and Barth [36]. Both these schemes have the common approach of using adaptive multi-triangle stencils to achieve high order accuracy for problems which may have shocks and discontinuities.

Abgrall's adaptive essentially non–oscillatory (ENO) scheme takes the form of either the centroid of a triangle acting as a control volume for that triangle or the construction of control volumes around each node in the mesh. The method involves the construction of an interpolant of order n by several steps. The initial step involves the construction of a linear interpolant and each following step will increase the order of the interpolant. Several possible combinations of points can be used at each step. The choice of which points to use is made by examining the coefficients of the Lagrange polynomials constructed from each combination. The set chosen is the one in which the sum of the absolute values of the coefficients of the Lagrange polynomial is minimal. For example, if control volumes are constructed around each node in the mesh, then for each control volume there will be a choice of which three points to consider to construct a linear interpolant. The possible combinations will be the node itself and any two of the neighbouring nodes. From the three points chosen for the linear interpolant, three further points can be added, in the next step, to construct a quadratic, then four more can be added to

construct a cubic and so on. The problem is that the number of possible combinations grows rapidly, and the stencil used is potentially large. Abgrall controls the choice of the additional values considered at each step by only considering neighbouring nodes of nodes already chosen. Even so the growth is rapid and the points considered may be far removed from the original node. Abgrall's good results provide a more than adequate justification of the scheme however.

The intention here is to consider a simpler alternative to Abgrall's scheme using a fixed stencil based only on solution values at centroids of triangles and mid–points of edges. These values are obtained by the discretisation method via a 10 triangle stencil for each triangle, (see [88], [11]). The nine additional triangles are either first or second generation neighbours, unlike the Abgrall scheme with its adaptive stencil. Given that the methods used for spatial discretisation and time integration take great care to avoid introducing new extrema, it is important to use an interpolant in the postprocessing step that does not violate this principle. The aim is to construct a quadratic interpolant from the initial data at the centroids and edge mid–points of the triangles. There are two stages in this procedure. The first stage is to construct values at the vertices. This is done by using limited interpolants similar to those used in the discretisation methods, see [11], [88], [63]. This approach gives an interpolant which is multivalued at the vertices and hence discontinuous. A single value at each node in the mesh may then be calculated, such a procedure is desirable when visual results are the primary concern. The forcing of single nodal values produces a continuous surface over the numerical domain. The second step is to construct an interpolant which avoids introducing new extrema, an area in which the standard quadratic interpolant fails. This property is achieved by modifying the standard quadratic shape functions over each triangle. The result of this modification is the new interpolant may not pass through all the data points used to define it. This use of the interpolation information as control points rather than data points, is not uncommon in other forms of interpolation. Bézier curves must lie within the convex hull of the corresponding Bézier polynomial [18]. The justification for this is that other properties of the curve are more important, in this case the curve is aesthetically pleasing, in fact the Bézier curve is constructed from Bernstein basis functions which are all positive and sum to unity over the parametric coordinates. The approach taken here thus has some similarities with Bézier interpolants.

The remainder of this chapter will describe and analyse the way which this is

achieved. Standard interpolation techniques are examined, the principles are discussed with reference to 1D then extended to 2D, the linear cases are examined first to introduce much of the ideas and notation used. Quadratic interpolation is then discussed with the standard 1D case and again its extension to 2D.The quadratic scheme is examined and the failure of the scheme to avoid undershoot and overshoot is shown. The modification of the quadratic shape functions in 1D to create a quadratic scheme that will not introduce any new extrema is then shown and its extension into 2D described. The properties of the new scheme are examined and some numerical justification for the choice of methods given. The construction of the scheme from the initial data is described and the method used to compute the values at the vertices. Some numerical results are given and in particular the numerical results will show that on shock wave type problems the modified quadratic interpolant is as accurate as the original quadratic interpolant without generating new extrema. The extension of the interpolation schemes to generate derivative values at edge midpoints is then given with further numerical results for this. These are compared to current techniques used in cell-centred schemes for determining derivative estimates.

6.3 Linear Interpolation

6.3.1 1D Linear Interpolation

Linear interpolation requires two function values, say f_1 and f_2 at two data points x_1 and x_2. These may be mapped onto a unit scale with the first point at $L = 0$ and the second at $L = 1$, where L is the parametric coordinate, defined by $L = (x - x_1)/(x_2 - x_1)$ see Figure 6.1

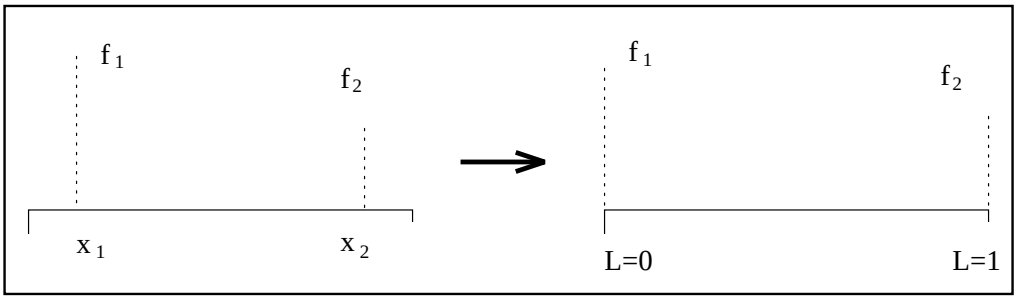

Figure 6.1: Mapping to Parametric coordinates in 1D

Then two standard linear shape functions, ϕ_1, ϕ_2 are $\phi_1 = 1 - L$ and $\phi_2 = L$ and

the linear interpolant is then $\sum \phi_i f_i \; i = 1, 2$. The two linear shape functions are shown in Figure 6.2.

These shape functions also have the following properties, $0 \leq \phi_i \leq 1$, they are always positive and they sum to unity: $\sum \phi_i = 1 \; i = 1, 2$. These properties ensure that the interpolant defined in this way is bounded by the two values used. In the linear case no new extrema are introduced.

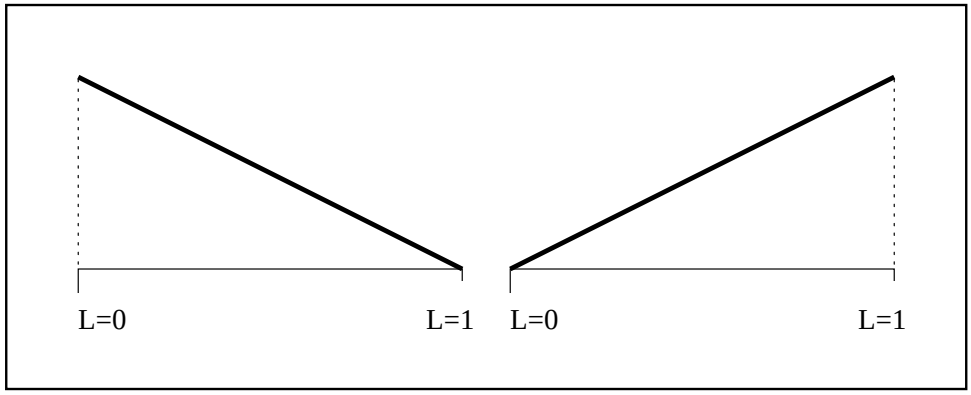

Figure 6.2: The Linear Shape functions

6.3.2 2D Linear Interpolation

Linear shape functions defined over a triangle require three pieces of information. The three values allow three shape functions to be constructed over the triangle. Each triangle is mapped to area coordinates with each point expressed in terms of three factors, L_1, L_2, L_3 which are derived from the ratio of the areas that the point divides the triangle into, see Figure 6.3.

Usually the three values are defined at the vertices of the triangle. In this case the vertices are mapped to the points $(1,0,0), (0,1,0), (0,0,1)$ in (L_1, L_2, L_3) coordinates. The area coordinates are not independent and $L_1 + L_2 + L_3 = 1$. The shape functions have the same properties as in 1D where they are unity at their associated data points and vanish at the others. The method of undetermined coefficients can be used to express these shape functions.

When the vertex values are used to determine a linear interpolant over the triangle the shape functions are $\phi_1 = L_1, \phi_2 = L_2, \phi_3 = L_3$. The shape functions are of the form shown in Figure 6.4. The shape functions have the following properties: $0 \leq \phi_i \leq 1$,

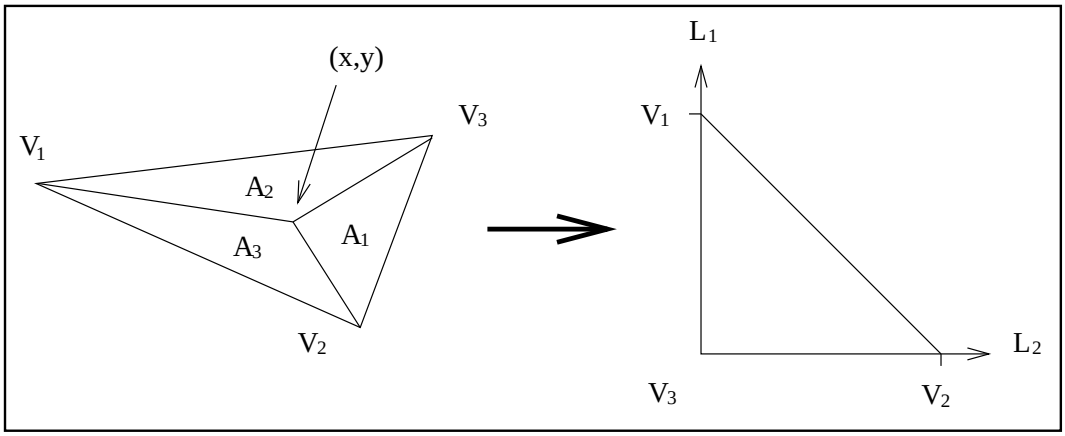

Figure 6.3: Mapping To Area Coordinates

ϕ_i is always positive and $\sum \phi_i = 1,\ i = 1..3$. Therefore any point defined by the interpolant will be a positive combination of the three data points and bounded by the values themselves. Again no new extrema are introduced.

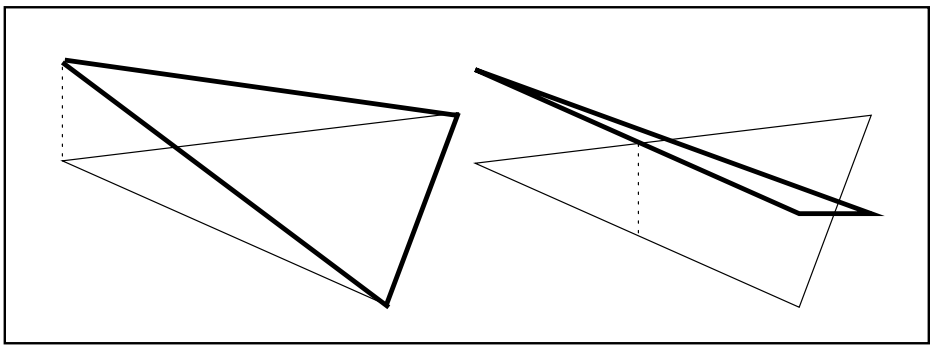

Figure 6.4: 2D Linear Shape function, Vertex Linear (left) & Midpoint Linear (Right)

As the raw data for the interpolant includes edge midpoint values: these may be used in a linear interpolant. In this case the shape functions are $\phi_1 = 1 - 2L_1, \phi_2 = 1 - 2L_2, \phi_3 = 1 - 2L_3$. The shape functions have the following properties: $-1 \leq \phi_i \leq 1$, $\sum \phi_i = 1,\ i = 1..3$, however, ϕ_i is no longer always positive. Therefore any point defined by the interpolant will not be a positive combination of the three data points and the value given by the interpolant is not bounded by the data values used to define it. New extrema may be introduced in this case and for this reason the midpoint linear scheme will be discarded.

6.4 Quadratic Interpolation

6.4.1 1D Quadratic Interpolation

Quadratic interpolation requires three function values, f_1, f_2 and f_3. These are at the mapped to the points f_1 at $L = 0$, f_2 at $L = 1/2$ and f_3 at $L = 1$ using linear mapping such as in Section (6.3.1). Three standard shape functions associated with the three points ϕ_1, ϕ_2 and ϕ_3 such that they are unity at one point and vanish at the other two, see Figure 6.5.

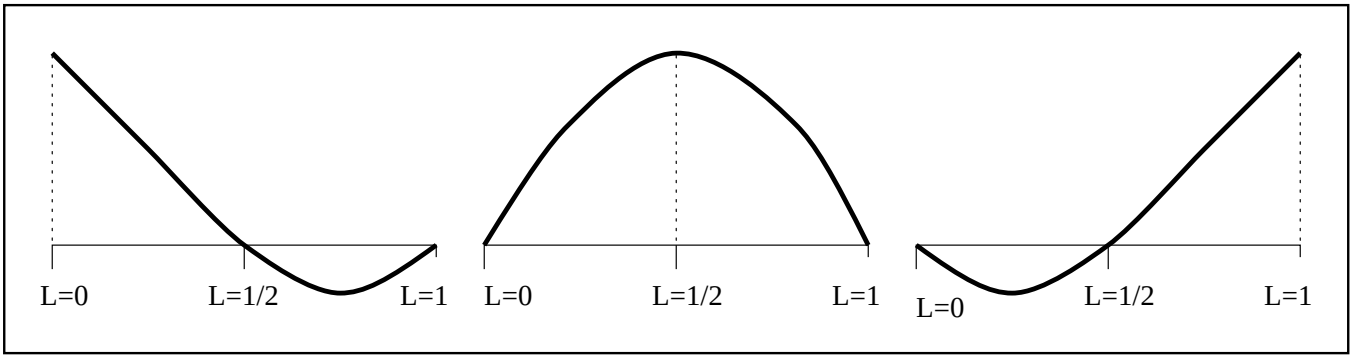

Figure 6.5: The 1D Standard Quadratic Shape Functions

These shape functions, written as a combination of linear and quadratic parts, are defined by

$$\begin{aligned} \phi_1 &= (1-L) + (2L^2 - 2L) \\ \phi_2 &= (4L - 4L^2) \\ \phi_3 &= (L) + (2L^2 - 2L) \end{aligned} \tag{6.1}$$

Unlike the linear shape functions, the quadratic shape functions associated with the end points become negative over the region considered. The shape functions again have the property that they sum to unity, but at particular points, it is possible to create a new maximum or minimum, for example, at $L = 1/4$, the shape functions have values $\phi_1 = 3/8$, $\phi_2 = 6/8$ and $\phi_3 = -1/8$, and ϕ_1 and ϕ_2 sum to greater than unity.

The elimination of the possibility of creating new extrema is forced by ensuring that all shape functions are bounded by the constraint, $0 \leq \phi_i \leq 1$ and also that $\sum \phi_i = 1$ for $i = 1..3$, as in the linear case.

The value produced by a linear interpolant, f_L, will satisfy these constraints. However, the value given by the quadratic interpolant, f_s does not satisfy the required

constraints and can therefore create new extrema. A modified interpolant can be written as

$$f_M = \alpha f_L + (1 - \alpha) f_S$$

then a value of $\alpha = 1$ will give the standard linear interpolant and a value $\alpha = 0$ will give the standard quadratic. The question is what range of values will always ensure the modified scheme displays the desired behaviour ? To guarantee that the shape functions remain positive a value of α between the range of $1/2 \leq \alpha \leq 1$ must be chosen. Under the assumption that a quadratic interpolant will produce better results that a linear interpolant, then the smallest value of α is taken. This giving the largest quadratic part in the modified scheme. Figure 6.6 illustrates these shape functions

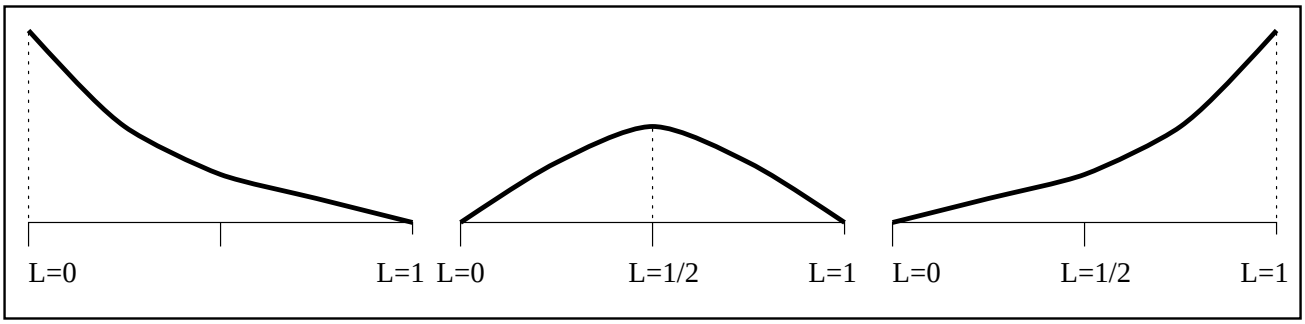

Figure 6.6: The 1D Modified Quadratic Shape Functions

Examination of the shape function identifies the following properties. ϕ_1 no longer vanishes at $L = 1/2$ but is unity at $L = 0$ and vanishes at $L = 1$. ϕ_3 also no longer vanishes at $L = 1/2$ but is unity at $L = 1$ and vanishes at $L = 0$. ϕ_2 vanishes at $L = 0$ and at $L = 1$ but is no longer unity at $L = 1/2$. In fact at $L = 1/2$ $\phi_1 = \phi_3 = 1/4$ and $\phi_2 = 1/2$ this means that the interpolant no longer has the property $f(x_i) = f_i$ at the midpoint, where f_i is the given function value at x_i. However, all shape functions satisfy the constraint $0 \leq \phi_i \leq 1$.

These new shape functions are then

$$\begin{aligned} \phi_1 &= (1 - L)^2 \\ \phi_2 &= 2L(1 - L) \\ \phi_3 &= L^2 \end{aligned} \tag{6.2}$$

All shape functions are positive, therefore

$$f = \sum_{i=1}^{3} \phi_i f_i \tag{6.3}$$

is a positive combination of the f_i's and because of this will create no new extrema. Note however that an error has been introduced at the midpoint. This error can be seen by looking at the shape functions at the midpoint. The modified interpolant is defined as

$$f_1(1-L)^2 + f_2 2L(1-L) + f_3 L^2$$

at the midpoint, $L = 1/2$, the interpolated value, f^*, is given by

$$\begin{aligned} f^* &= \frac{1}{4}f_1 + \frac{1}{2}f_2 + \frac{1}{4}f_3 \\ &= f_2 + \frac{1}{4}(f_1 - 2f_2 + f_3) \\ &\approx f_2 + \frac{h^2}{4}f_2'' + O(h^4) \end{aligned}$$

where h is the distance between data points and assuming that the function defined by f_1, f_2, f_3 has a second derivative. An error is introduced which is proportional to the distance between data points and the second derivative of the function at x_2, the midpoint.

6.4.2 2D Quadratic Interpolation

A 2D quadratic interpolant needs six data points: these points are usually at the vertices of the triangle and the mid–points of the sides. These can be mapped to area coordinates (L_1, L_2, L_3). Six shape functions can be fitted to these points such that they are unity at one point and vanish at the others, see [40] for details.

These shape functions are shown in Table 6.1 and have the property that they sum to unity. However, the shape functions associated with the three vertex values, are negative over large parts of the triangle, see Figure 6.7. Thus it is possible for new extrema to be introduced. This is unsatisfactory for the purpose the interpolant is intended for.

The elimination of the possibility of creating new extrema is achieved by ensuring that all shape functions are bounded by the constraint, $0 \leq \phi_i \leq 1$ and also maintain the condition that $\sum \phi_i = 1$ for $i = 1..6$. To ensure that ϕ_1, ϕ_2 and ϕ_3 are always positive the proportion of the quadratic part of each shape function used in a modified scheme that displays the desired properties is reduced, as in the 1D case. The condition that

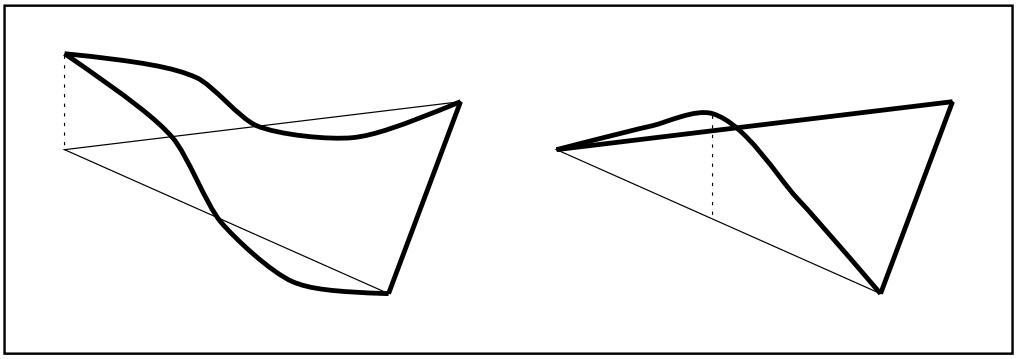

Figure 6.7: Standard 2D Quadratic Shape Functions

$\sum \phi_i = 1$ for $i = 1..6$ still holds true. The new shape functions are shown in Table (6.1) and are shown in Figure 6.8.

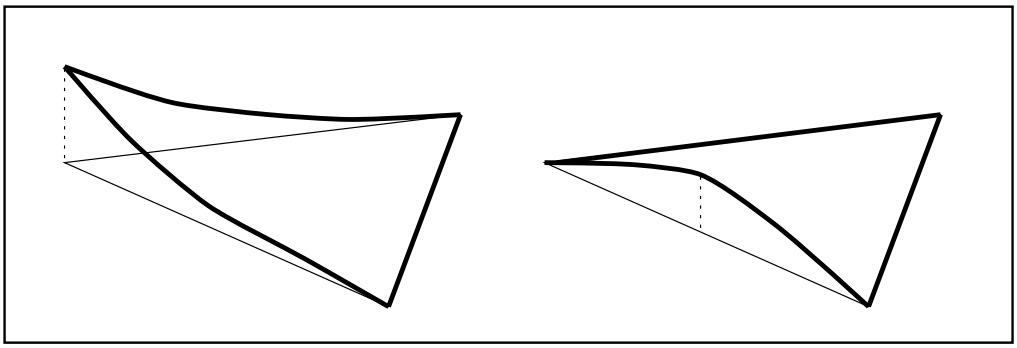

Figure 6.8: Modified 2D Quadratic Shape Functions

The new modified shape functions $\hat{\phi_1}, \hat{\phi_2}$ and $\hat{\phi_3}$ no longer vanish at the adjacent edge mid–points but have a value of 1/4. The mid–point shape functions do vanish at all other points but are not unity at their associated point but in fact have values of 1/2. This is similar to Bézier curve interpolation as described in Section 6.1. The interpolant therefore will no longer exactly match the function value at the mid–points. However, the spatial discretisation schemes used to solve the PDEs [11] [88] [63] create solution values at the centroids which will respect the physical properties of the solution. The elimination of spurious oscillations is a prime concern in the PDE solution process. The modified interpolant will also ensure that no new extrema are created. It is this final property that is of special interest, the ability of the scheme not to introduce spurious oscillations and interpolate solution values outside the range of existing values.

ϕ_1	=	$(2L_1 - 1)L_1$	$\hat{\phi_1}$	=	L_1^2
ϕ_2	=	$(2L_2 - 1)L_2$	$\hat{\phi_2}$	=	L_2^2
ϕ_3	=	$(2L_3 - 1)L_3$	$\hat{\phi_3}$	=	L_3^2
ϕ_4	=	$4L_2L_3$	$\hat{\phi_4}$	=	$2L_2L_3$
ϕ_5	=	$4L_3L_1$	$\hat{\phi_5}$	=	$2L_3L_1$
ϕ_6	=	$4L_1L_2$	$\hat{\phi_6}$	=	$2L_1L_2$

Table 6.1: The standard (left) [40] and modified (right) quadratic shape functions

6.5 Construction Of Vertex Values

When dealing with cell–centred finite volume codes the information given by the code is at the edge mid–points and the centroid of each triangle. In order to use a quadratic interpolant a method is required that will generate values at the vertices of each triangle. There are several ways to accomplish the creation of the vertex values. The aim is to find a way that will ensure that an accurate value is obtained and that this is done without introducing new extrema.

6.5.1 A Two Dimensional Linear Scheme

One method for the construction of the vertex values is to use a two dimensional scheme based on the values at the edge mid–points. The three edge midpoint values provide a way to construct a linear interpolation scheme with three shape functions defined at the mid–points. This method may create new extrema at the nodes so a capping procedure is used to ensure that the values given at the vertices do not exceed the range of the surrounding values. This ensures that no new extrema are created during this phase. The numerical results in Section 6.8 show results using this method.

6.5.2 A One Dimensional Approach

Consider a cross section through a triangle, see Figure 6.9, the values shown are u_0 at mid–point M_0, u_1 at the centroid C and u_2 is the linear interpolated value between M_1 and M_2. The value required is u_3, the vertex value. The distances between these points are h_1, h_2 and h_3.

Given this information a one dimensional quadratic can be constructed through

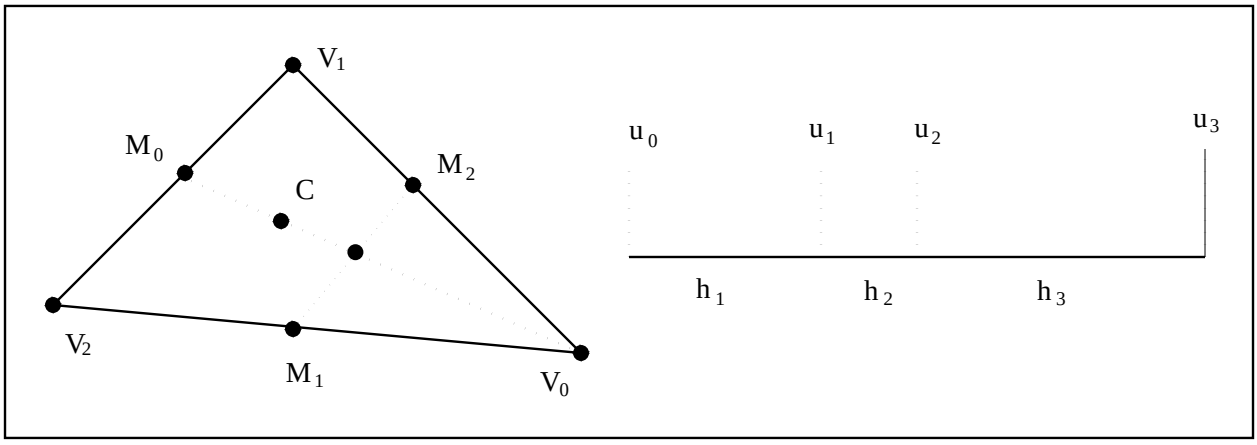

Figure 6.9: Cross Section through a Triangle

u_0, u_1 and u_2 to give a value u_3 at the vertex. Evaluating a quadratic interpolant, $f = ax^2 + bx + c$, at $x = 0, f = u_1$ gives

$$c = u_1 \tag{6.4}$$

at $x = -h_1, f = u_0$ gives

$$ah_1^2 - bh_1 + c = u_0 \tag{6.5}$$

at $x = h_2, f = u_2$ gives

$$ah_2^2 + bh_2 + c = u_2 \tag{6.6}$$

multiplying (6.5) by h_2 and (6.6) by h_1 and adding these gives

$$a(h_1^2 h_2 + h_1 h_2^2) = h_2(u_0 - u_1) + h_1(u_2 - u_1)$$

$$a = \frac{(u_0 - u_1)}{h_1(h_1 + h_2)} + \frac{(u_2 - u_1)}{h_2(h_1 + h_2)} \tag{6.7}$$

substituting this into (6.5) gives

$$b = \frac{(u_1 - u_0)}{h_1} + h_1 \left[\frac{(u_0 - u_1)}{h_1(h_1 + h_2)} + \frac{(u_2 - u_1)}{h_2(h_1 + h_2)}\right]$$

$$b = \frac{h_2(u_1 - u_0)}{h_1(h_1 + h_2)} + \frac{h_1(u_2 - u_1)}{h_2(h_1 + h_2)} \tag{6.8}$$

This gives the quadratic interpolant defined by (6.6), (6.7) and (6.8) as

$$\left[\frac{(u_0 - u_1)}{h_1(h_1 + h_2)} + \frac{(u_2 - u_1)}{h_2(h_1 + h_2)}\right] x^2 + \left[\frac{h_2(u_1 - u_0)}{h_1(h_1 + h_2)} + \frac{h_1(u_2 - u_1)}{h_2(h_1 + h_2)}\right] x + u_1$$

evaluating this at $x = h_2 + h_3$, the value at the vertex, gives the value at u_3 as

$$u_1 + \frac{(u_1 - u_0)}{h_1(h_1 + h_2)}\left[h_2(h_2 + h_3) - (h_2 + h_3)^2\right] + \frac{(u_2 - u_1)}{h_2(h_1 + h_2)}\left[h_1(h_2 + h_3) + (h_2 + h_3)^2\right]$$

$$u_3 = u_1 + \frac{(h_2 + h_3)}{(h_1 + h_2)}\left[\frac{-h_3(u_1 - u_0)}{h_1} + \frac{(h_1 + h_2 + h_3)(u_2 - u_1)}{h_2}\right]$$

defining

$$r = \frac{(u_2 - u_1)}{h_2}\frac{h_1}{(u_1 - u_0)}$$

gives

$$u_3 = u_1 + \frac{(h_2 + h_3)}{(h_1 + h_2)}\left[\frac{(u_1 - u_0)}{h_1}\left(r(h_1 + h_2 + h_3) - h_3\right)\right]$$

$$u_3 = u_1 + (h_2 + h_3)\frac{(u_1 - u_0)}{h_1}\left[\frac{r(h_1 + h_2 + h_3)}{(h_1 + h_2)} - \frac{h_3}{(h_1 + h_2)}\right]$$

$$u_3 = u_1 + (h_2 + h_3)\frac{(u_1 - u_0)}{h_1}\left[1 + \frac{(h_1 + h_2 + h_3)}{(h_1 + h_2)}(r - 1)\right] \tag{6.9}$$

However, there is no guarantee that this quadratic will produce physically sensible values of u_3 and so a linear approximation might be favoured using the values of u_0 and u_1. It would be sensible to construct an interpolant where the quadratic part can be switched off. In Equation (6.9) the terms

$$u_1 + (h_2 + h_3)\frac{(u_1 - u_0)}{h_1}$$

comprise the linear interpolant and the multiplier

$$\left[\frac{(h_1 + h_2 + h_3)}{(h_1 + h_2)}(r - 1)\right]$$

gives the quadratic part of the interpolant. The assumption here is that the linear and constant interpolants will yield appropriate solution values as the edge mid–point and centroid values accurately represent the solution. A limiter based method is then used to ensure that the value generated by the quadratic interpolant lies between the linear and constant values, i.e.

$$0 \leq \left[1 + \frac{(h_1 + h_2 + h_3)}{(h_1 + h_2)}(r - 1)\right] \leq 1$$

this is controlled by limiting r, bounded by

$$\frac{h_3}{h_1 + h_2 + h_3} \leq r \leq 1$$

Given a value of r outside this range the quadratic interpolant is likely to create solution values outside the range of the existing centroid values and because of this the

quadratic part is turned off and the linear or constant interpolant, whichever is closest to the disguarded quadratic is preferred. The operation can be repeated for each vertex of the triangle to obtain the three values required. The underlying assumption is that as u_2 is constructed by linear interpolation it is less accurate than u_0 and u_1, for this reason also the linear interpolant is based on u_0 and u_1 rather than u_1 and u_2. The linear value may still be outside the range of the centroid values adjacent to the nodes and so a simple capping procedure ensures that no new extrema are created by ensuring that each vertex value is bounded by the extrema of the surrounding centroid values. This process will yield a discontinuous representation of the mesh with each triangle defined now by the centroid value, the edge mid–point values and the vertex values.

6.6 Error Analysis

When determining the vertex values the error at these values will be at best a quadratic error and at worst a linear error. In the case where $r > 1$ and so is reset to be one, the second order error is

$$\text{Error } = \frac{h_1 + h_2 + h_3}{h_1 + h_2} \left[\left(\frac{u_2 - u_1}{h_2} \right) - \left(\frac{u_1 - u_0}{h_1} \right) \right] (h_2 + h_3)$$

In the case where $r = \alpha \frac{h_3}{h_1+h_2+h_3}$, where $\alpha < 1$, and r is reset to be $\frac{h_3}{h_1+h_2+h_3}$ the first order error is

$$\text{Error } = \frac{h_3}{h_1 + h_2} (h_2 + h_3) \left[\frac{u_1 - u_0}{h_1} \right] (\alpha - 1).$$

The modified shape functions give an error at mid–point values. If the 1D situation along the edge of the triangle is considered then the modified interpolant along the edge can be defined as

$$f_1(1 - L)^2 + f_2 2L(1 - L) + f_3 L^2$$

at the mid–point, $L = 1/2$, the interpolated value, f^*, is given by

$$\begin{aligned} f^* &= \frac{1}{4} f_1 + \frac{1}{2} f_2 + \frac{1}{4} f_3 \\ &= f_2 + \frac{1}{4}(f_1 - 2f_2 + f_3) \\ &\approx f_2 + \frac{h^2}{4} f_2'' + O(h^4) \end{aligned}$$

where f_1 and f_3 are the vertex values and f_2 is the edge mid–point value and h is the distance between data points and assuming that the function defined by f_1, f_2, f_3 has a second derivative. An error is introduced which is proportional to the distance between data points and the second derivative of the function at f_2, the mid–point. Although this approach gives an error of the same order as the linear interpolant the difference in accuracy is significant, numerical results later showing this.

A further source of error when the interpolant is used with the finite volume scheme of Ware and Berzins, [88], is that the original mid–point and centroid values already contains a second order error. It should be noted that this additional source of error is not present in any of the numerical experiments described below as exact values at the mid-points of edges were used.

6.7 Options and Extensions

6.7.1 Continuous or Discontinuous Interpolants

A further decision when determining the vertex values is that of a unique value at each node of the mesh or a discontinuous representation. The former, as mentioned earlier, would be better suited for visualisation purposes whilst the latter more suitable for the recovery of numerical values. The operation of forcing a single value at each node creates additional errors, given this it will be shown in Section 6.8 that a trade–off may exist between good numerical and good visual results.

The several values at each vertex, one for each triangle, can be converted into a single value, f^* by using a weighted average of the values:

$$f^* = \frac{\sum_{i=1}^{ntri} f_i A_i}{\sum_{i=1}^{ntri} A_i} \tag{6.10}$$

where A_i is the area of the triangle with the vertex value f_i and $ntri$ is a list of triangles that f_i is a member of.

6.7.2 Use of Centroid Values

The interpolation schemes examined considered using data at the edge mid–points and centroids to create vertex values, the vertex values are then capped. The interpolant may not pass through the value at the centroid of each triangle. This centroid value is the primary one calculated by the discretisation method used in the solution process.The condition that the interpolant should pass through the centroid value can, however, be ensured. The approach taken here is to construct an additional shape function that can be used as an added correction to the interpolant. The idea originates from the standard cubic interpolation techniques. A shape function is fitted to the centroid value which is unity at the centroid and vanishes along the edges of the triangle. This shape function is defined by $\phi_c = 27L_1L_2L_3$. To utilise this cubic bubble an estimate of the value at the centroid is obtained via the interpolation scheme and the difference between this and the primary value is multiplied by the bubble and then added as a correction. The numerical results in Section 6.8 will show that the centroid shape function improves the accuracy of the interpolant. However, it must be stressed that, the use of the cubic bubble correction term may result in new extrema being created.

6.7.3 Using Linear Interpolants Over Each Triangle

Two other ways in which undershoot and overshoot can be eliminated is by using linear interpolation over each triangle after the construction of the vertex values has taken place. The first is to simply fit a linear interpolant over each triangle using the vertex value. The second is to fit four linear interpolants over each triangle using the vertex and edge midpoint values. The first three interpolants are defined using each vertex values and the two adjacent edge midpoint values. The final interpolant is defined using the three edge midpoint values. The use of four linears will not introduce any new extrema and will also pass through the edge midpoint values unlike the linear interpolant based on the vertex values alone. This means the four linear scheme, understandably, produces better results than the single linear but at a much higher computational cost since it uses four interpolants over each triangle in the domain.

6.8 Numerical Testing

Several different variants of the interpolation schemes above will now be compared using the standard and modified quadratic shape functions given in Table 6.1. These methods are:

- Scheme A: The standard quadratic shape functions using the true function values at all six points.
- Scheme B1: The modified quadratic shape functions using vertex values obtained by the one dimensional quadratic method.
- Scheme B2: The standard quadratic shape functions using vertex values obtained by the one dimensional quadratic method.
- Scheme C1: The modified quadratic shape functions using vertex values obtained by the two dimensional mid–point linear scheme.
- Scheme C2: The standard quadratic shape functions using vertex values obtained by the two dimensional mid–point linear scheme.
- Scheme D1: Scheme B1 with the cubic bubble correction.
- Scheme D2: Scheme B2 with the cubic bubble correction.
- Scheme E1: Scheme C1 with the cubic bubble correction.
- Scheme E2: Scheme C2 with the cubic bubble correction.
- Scheme F: The standard linear interpolant using the true function values at the vertices of each triangle.

The interpolant was tested on a number of numerical examples using the L_1 norm and a seven point fifth order Gaussian quadrature scheme, see [96], for the numerical integration over each triangle. The quadrature scheme used was chosen because of the location of the quadrature points. These points seem to often coincide with areas of the triangle that are prone to overshoot and undershoot. The summation of the errors over each triangle then provide an error estimate over the whole domain. This can be expressed

as

$$\sum_{tri=1}^{ntri} \left(\left[\sum_{i=1}^{7} |True_i - Interpolated_i| \, w_i \right] \; area_{tri} \right)$$

where $True_i$ is the true function value at quadrature point i and $Interpolated_i$ is the interpolated value at i.

All the numerical results given use a discontinuous representation of the physical domain, no forcing of a single value at each mesh node is made. Several test problems are considered, the first is a simple sine function where the expected rate of convergence is shown. The next three problems are all shock wave problems typical to this area. The final problem is a complex function with many shocks and discontinuties, this problem is presented with graphical results also to demonstrate the ability of the modified quadratic interpolant to accurately capture shocks without the spurious oscillations found when using the standard quadratic interpolant.

6.8.1 Simple sine function

The results for a simple sine function defined by $u(x, y) = \sin(x + y)$ are shown. Table 6.2 gives numerical results, taken from a $[0..1, 0..1]$ domain with a regular triangular mesh with $2N^2$ triangles where $N = 8, 27, 81, 243$. These results indicate that for smooth functions, without severe gradients, the quadratic scheme using exact vertex values out performs the other schemes. Although this is to be expected this quadratic interpolant cannot, of course, be used with the cell-centred discretisation scheme for which no vertex values are available.

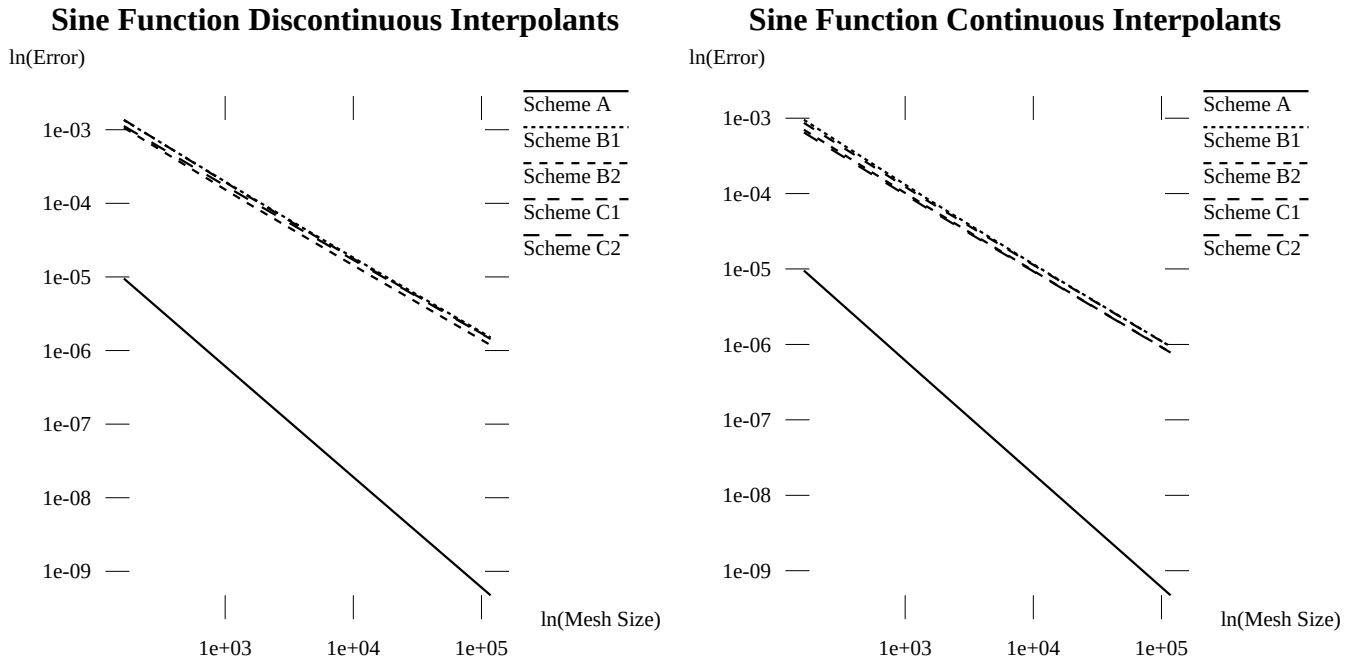

Figure 6.10: Mesh Size Against Error on a log log Scale For Sine Function

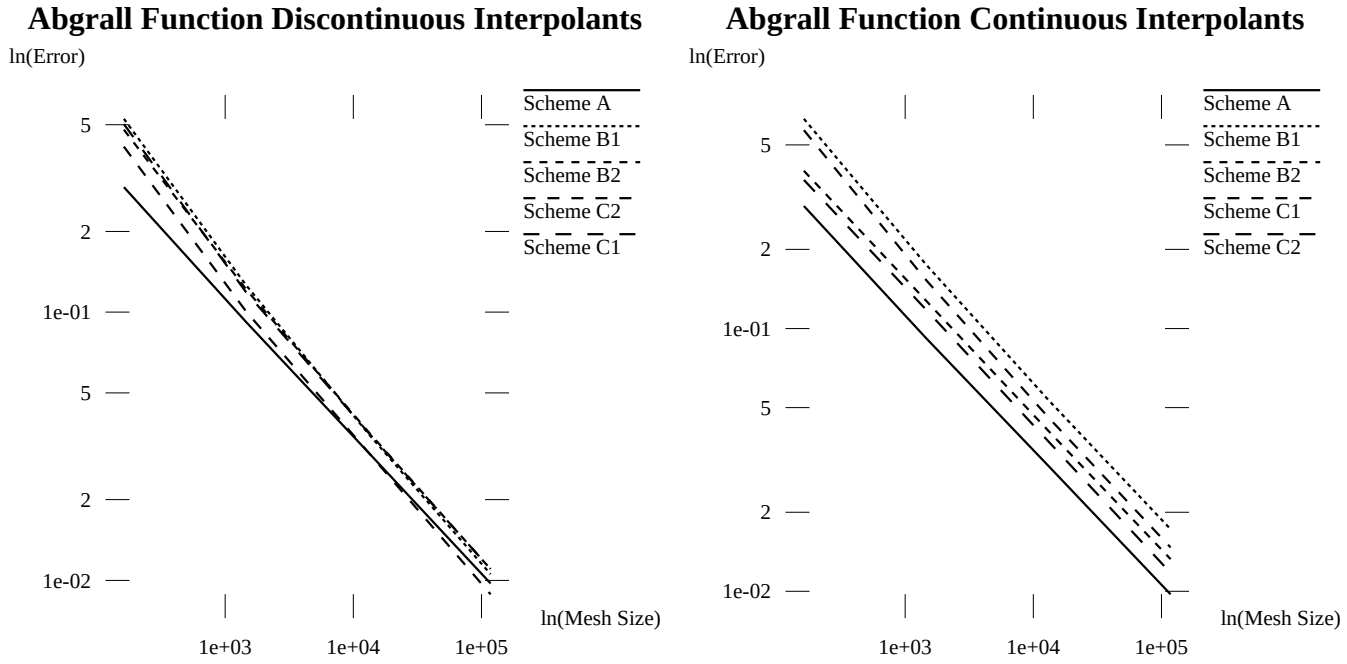

Figure 6.11: Mesh Size Against Error on a log log Scale For Abgrall Function

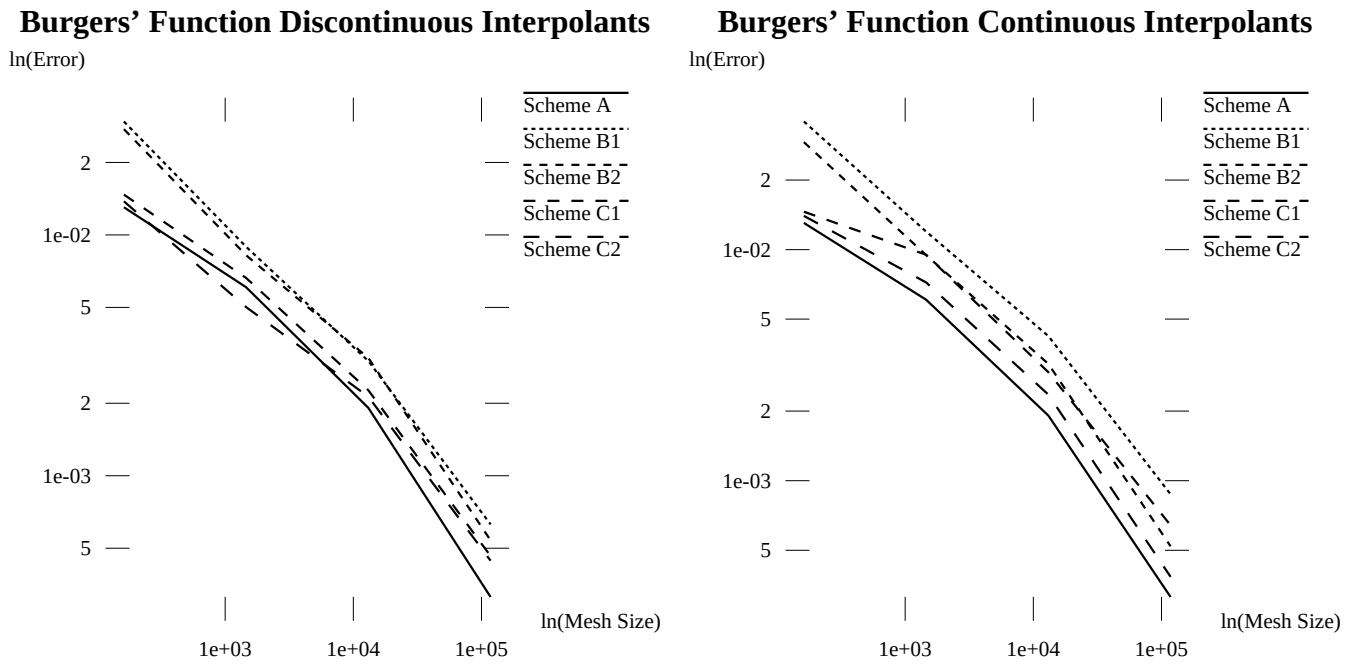

Figure 6.12: Mesh Size Against Error on a log log Scale For Burgers' Function

The differences between the methods A, B1, B2, C1 and C2 are shown in Figure 6.10. The graphs show the error against mesh size on a log log scale for the discontinuous representation (left) and continuous representation (right).

The graphs show little difference between using the two dimensional linear scheme or the one dimensional quadratic scheme for the construction of the vertex values. However Figures 6.11 and 6.12 show a similar analysis for two of the other test problems which contain shocks and discontinuities, see Section 6.8.5 and Section 6.8.2. These indicate more clearly that a discontinuous representation appears to give better numerical results than forcing a unique vertex value.

No of Triangles	162	1458	13122	118098
Scheme A	9.5316e-06	3.4645e-07	1.2745e-08	4.7097e-10
Scheme B1	1.3476e-03	1.3229e-04	1.3856e-05	1.5103e-06
Scheme D1	1.0706e-03	1.0361e-04	1.0833e-05	1.1796e-06
Scheme B2	1.0668e-03	1.0325e-04	1.0883e-05	1.1858e-06
Scheme D2	7.2835e-04	7.0648e-05	7.4451e-06	8.1173e-07
Scheme C1	1.3667e-03	1.2945e-04	1.3381e-05	1.4505e-06
Scheme E1	1.0267e-03	9.8569e-05	1.0340e-05	1.1253e-06
Scheme C2	1.1246e-03	1.1798e-04	1.2991e-05	1.4355e-06
Scheme E2	8.5682e-04	9.1231e-05	1.0079e-05	1.1154e-06
Scheme F	2.3873e-03	2.6530e-04	2.9478e-05	3.2754e-06

Table 6.2: Numerical results for Sine problem (discontinuous representation)

6.8.2 Burgers' Equation Problem

This function is given by

$$u(x,y,t) = w(x,t)w(y,t) \text{ where } w(x,t) = \frac{0.1A + 0.5B + C}{A + B + C}$$

and $A = e^{-0.005(x-0.5+4.95t)/v}$, $B = e^{-0.25(x-0.5+0.75t)/v}$, $C = e^{-0.5(x-0.375)/v}$

This function is the solution to the Burgers' equation, see [8], defined by

$$\frac{\partial u}{\partial t} + w(x,t)\frac{\partial u}{\partial x} + w(y,t)\frac{\partial u}{\partial y} - v\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0, \quad v = 0.0001$$

The Problem time is taken at $t = 0.45$, results are shown in Table 6.3. The numerical results are for a $[0..1, 0..1]$ domain with regular triangular meshes of $2N^2$ triangles where

No of Triangles	162	1458	13122	118098
Scheme A	1.3038e-02	6.0654e-03	1.9130e-03	3.1350e-04
Scheme B1	2.9608e-02	8.9322e-03	2.9958e-03	6.2836e-04
Scheme D1	2.7022e-02	8.1619e-03	2.5298e-03	5.6399e-04
Scheme B2	2.7489e-02	8.2318e-03	3.0856e-03	5.3931e-04
Scheme D2	2.2515e-02	6.7277e-03	2.1070e-03	4.3124e-04
Scheme C1	1.4706e-02	6.6242e-03	2.2643e-03	4.6404e-04
Scheme E1	1.0873e-02	4.6134e-03	1.5714e-03	3.7217e-04
Scheme C2	1.3791e-02	5.0152e-03	2.1307e-03	4.4422e-04
Scheme E2	1.0941e-02	4.6399e-03	1.4616e-03	2.9217e-04
Scheme F	1.4525e-02	1.3600e-02	3.6574e-03	8.9352e-04

Table 6.3: Numerical results for Burgers' problem

$N = 8, 27, 81, 243$. Again it should be stressed that the expected higher order of accuracy of the standard quadratic interpolant is not observed.

6.8.3 Anisotropy test problem

The next test function is defined by

$$u(x, y, t) = \frac{3}{4} - \frac{1}{4 + 4e^B} \text{ where } B = 0.125(-x + y - 0.75t)/v$$

and is the exact solution to a PDE similar to that used by Zegeling [95],

$$\frac{\partial u}{\partial t} + 3u\frac{\partial u}{\partial x} + 3(1.5 - u)\frac{\partial u}{\partial y} - 3v\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0, \quad v = 1.0e - 9.$$

for $t = 0.45$, results are shown below in Table 6.4. These numerical results are taken from a $[0..1, 0..1]$ domain with regular triangular meshes of $2N^2$ triangles where $N = 8, 27, 81, 243$

6.8.4 Burgers' test problem II

The next function is

$$u(x, y, t) = \frac{1}{1 + e^B}$$

where $B = (x + y - t)/v$ which is a nonlinear version of Burgers' problem I, see [8],

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

No of Triangles	162	1458	13122	118098
Scheme A	4.2894e-03	8.6585e-04	4.2829e-04	1.6442e-04
Scheme B1	1.5432e-03	1.0880e-03	5.1692e-04	5.6803e-05
Scheme D1	7.9691e-04	9.8026e-04	4.3545e-04	2.9333e-05
Scheme B2	3.0864e-03	9.8996e-04	5.0469e-04	1.1361e-04
Scheme D2	1.5938e-03	7.7450e-04	3.4175e-04	5.8666e-05
Scheme C1	4.6296e-03	7.5089e-04	4.0318e-04	1.7041e-04
Scheme E1	2.9110e-03	4.6010e-04	3.0802e-04	1.0715e-04
Scheme C2	3.3675e-03	5.1758e-04	4.2829e-04	1.2395e-04
Scheme E2	2.7892e-03	4.6586e-04	2.6117e-04	1.0267e-04
Scheme E2	8.7448e-03	2.0495e-03	5.4307e-04	3.4011e-04

Table 6.4: Numerical results for Anisotropy problem

No of Triangles	162	1458	13122	118098
Scheme A	1.5975e-02	4.9972e-03	2.0005e-03	4.3140e-04
Scheme B1	1.3198e-02	4.3993e-03	2.0350e-03	6.1499e-04
Scheme D1	1.0864e-02	3.6213e-03	1.6601e-03	5.2542e-04
Scheme B2	1.3639e-02	4.5462e-03	2.7271e-03	6.7450e-04
Scheme D2	8.9703e-03	2.9901e-03	1.9774e-03	4.6269e-04
Scheme C1	1.5841e-02	5.2804e-03	2.9710e-03	7.5680e-04
Scheme E1	9.9676e-03	3.3225e-03	2.0876e-03	5.0006e-04
Scheme C2	1.0976e-02	3.6587e-03	2.3489e-03	5.4759e-04
Scheme E2	9.5381e-03	3.1794e-03	2.0735e-03	4.8442e-04
Scheme F	3.3727e-02	1.0328e-02	2.8169e-03	8.8633e-04

Table 6.5: Numerical results for second Burgers' problem

where $v = 0.0001$.

The time for the problem is $t = 0.45$ results are shown in Table 6.5. These numerical results are taken from a $[0..1, 0..1]$ domain with regular triangular meshes of $2N^2$ triangles where $N = 8, 27, 81, 243$

6.8.5 Abgrall's function

A complex test function with many discontinuities was also considered and the performance of the modified scheme was compared with the standard interpolation techniques. The function used was taken from Abgrall [1] and provides a very challenging example. All results for this function are taken on a domain of $[-1..1, -1..1]$ as in [1]. The

function used by Abgrall is defined as follows

if θ is any angle, let f_θ be

$$f_\theta = \begin{cases} \text{if } r & \leq -1/3 \\ & f_\theta(x,y) = -r\sin(\frac{3\pi r^2}{2}) \\ \text{if } r & \geq 1/3 \\ & f_\theta(x,y) = 2r - 1 + \frac{\sin(3\pi r)}{6} \\ \text{if } \mid r \mid & < 1/3 \\ & f_\theta(x,y) = \mid \sin(2\pi r) \mid \end{cases}$$

where $r = x - \frac{\cos(\theta)}{\sin(\theta)}y$ and let $u(x,y)$ be:

$$u(x,y) = \begin{cases} \text{if } x & \leq 1/2\cos(\pi y) \\ & u(x,y) = f_{\sqrt{\pi/2}}(x,y) \\ \text{if } x & > 1/2\cos(\pi y) \\ & u(x,y) = f_{-\sqrt{\pi/2}}(x,y) + \cos(2\pi y) \end{cases}$$

A contour plot of the solution is shown in Figure 6.13. To better illustrate the problem encountered when using standard quadratic interpolant, Scheme A, Figure 6.14 shows where the undershoots and overshoots are found, i.e. when the value produced by the standard interpolant is outside the local range of the true function.

The areas of overshoot can be seen to coincide with the discontinuities present in the function. Both plots were produced by Xprism3, part of the Khoros visualisation software suite, using 32 contour lines.

The visual results displayed later, see Figures 6.15, (6.16) and (6.17), show cross sections through the function. These clearly show the absence of overshoot and undershoot when the modified interpolant is used. These graphs illustrate the ability of the modified interpolant not to create any new extrema and to show the magnitude of the failings of the standard interpolant in this respect. The figures again show the challenging nature of the function.

The results compare favourably with those of Abgrall, see [1]. The modified scheme proposed here ensures that in the problems considered undershoot and overshoot are eliminated. Abgrall does not make such a claim for his ENO reconstruction method although it might be expected that this is the case. These numerical results, see Table 6.6, are taken with regular triangular meshes of $2N^2$ triangles where $N = 8, 27, 81, 243$. Again

Figure 6.13: Contour Plot of the True Function

we stress that the expected higher order of accuracy of the standard quadratic interpolant is not observed.

Figure 6.14: Contour Plot of the Overshoot and Undershoot when using Standard Quadratic Interpolation

No of Triangles	162	1458	13122	118098
Scheme A	2.9262e-01	9.1925e-02	3.0001e-02	9.7310e-03
Scheme B1	5.2601e-01	1.2606e-01	3.5235e-02	1.0542e-02
Scheme D1	4.3515e-01	1.0293e-01	2.9004e-02	8.8092e-03
Scheme B2	4.7890e-01	1.2153e-01	3.5812e-02	1.1009e-02
Scheme D2	3.6250e-01	8.8959e-02	2.5456e-02	7.8131e-03
Scheme C1	5.0248e-01	1.1867e-01	3.5556e-02	1.1025e-02
Scheme E1	3.6240e-01	8.3194e-02	2.4874e-02	7.5536e-03
Scheme C2	4.1424e-01	1.0056e-01	2.9981e-02	8.8636e-03
Scheme E2	3.3316e-01	8.0133e-02	2.3962e-02	7.2855e-03
Scheme F	8.3655e-01	2.1235e-01	5.8062e-02	1.8089e-02

Table 6.6: Numerical results for Abgrall problem

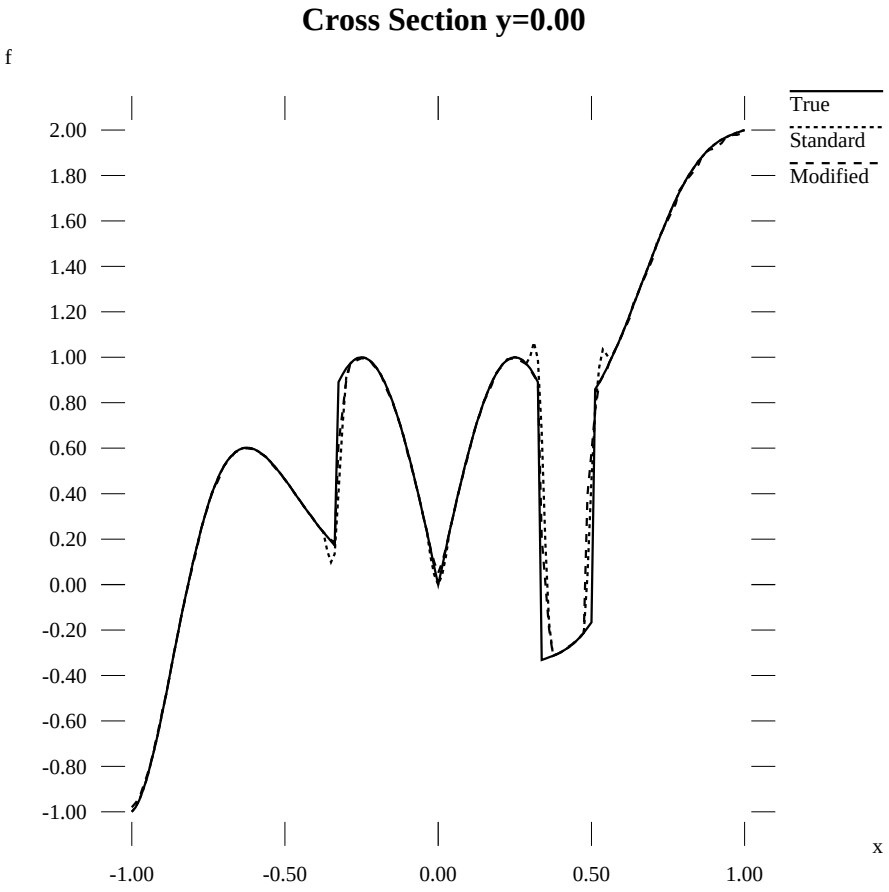

Figure 6.15: Cross Section using a Triangular Mesh with 1458 Elements

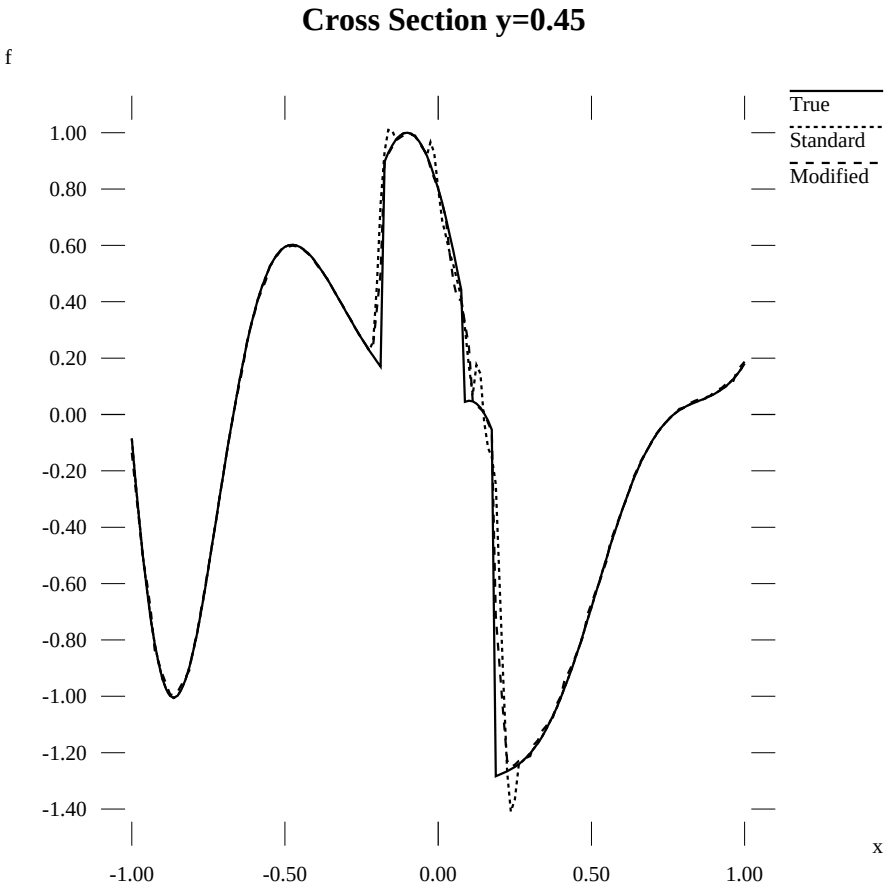

Figure 6.16: Cross Section using a Triangular Mesh with 1458 Elements

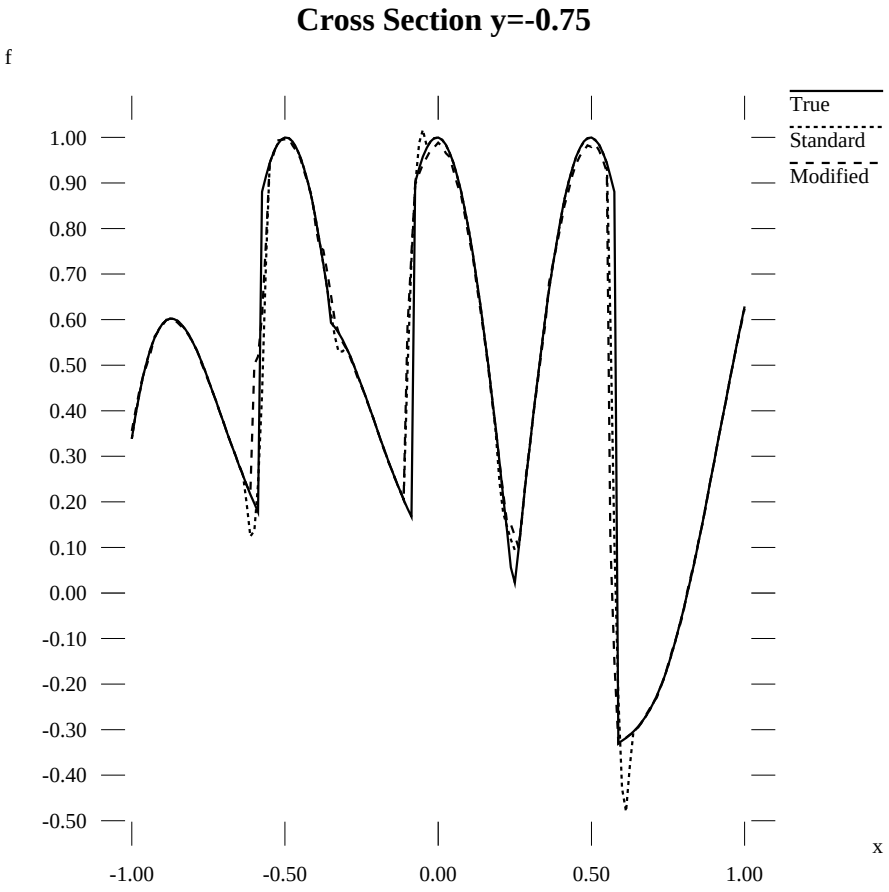

Figure 6.17: Cross Section using a Triangular Mesh with 1458 Elements

6.9 Calculating Derivative Values

When dealing with the numerical solution of PDEs the scheme needs to deal with the discretisation of advective and diffusive fluxes. When dealing with the diffusive terms we require some approximation of the spatial derivatives at the midpoints of the edges.

Conventional methods for the construction of spatial derivative values in cell–centred schemes use the centroid values of the triangle and surrounding centroid values to construct the approximation. Three points allow a linear interpolant to be constructed over a triangle which can be used to give an estimate of the spatial derivatives. The scheme by Durlofsky et al. [29] uses linear interpolants as described above to construct these derivative values. Given that the derivative value at the edge midpoint is required then the choice of the first two points are the centroids of the triangles sharing the edge. The third point is taken from one of the neighbours of these triangles, see Figure 6.18. The centroid of *e, f, c* or *d* is chosen depending upon the spatial positioning of the centroids. The triangle, formed by the three points, that has the closest to a right angle is deemed to provide the good estimate.

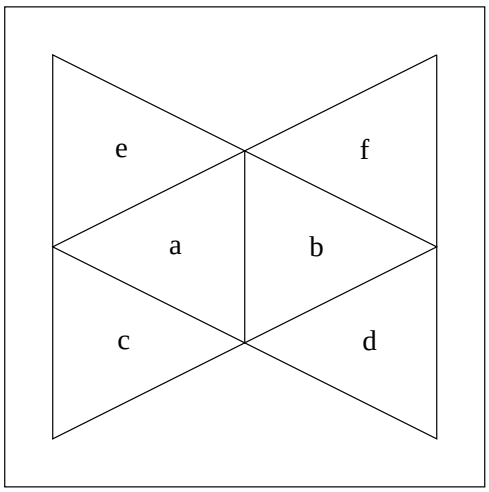

Figure 6.18: Triangular Stencil for Derivative Calculations

An alternative to the use of a linear is that of Berzins & Ware [12]. In this scheme an additional point is used and a bilinear interpolant is constructed. The four points are either *abef* or *abcd* depending on the position of the edge midpoint. When the triangle is at or near a boundary then the linear case may be used if a bilinear cannot be constructed.

An alternative method is provided by the recovery of sensible vertex values for each triangle. This allows the use of linear or quadratic interpolation schemes to be used to estimate these spatial derivatives. Therefore, given a linear interpolant defined by the vertices of a triangle, a linear interpolant can be defined, see Section 6.3.2. This interpolant is expressed as

$$f = \sum_{i=1}^{3} \phi_i f_i$$

where the f_i are the vertex values of the triangle and $\phi_i = L_i$ are the shape functions, where L_i are the area coordinates of the point.

This interpolant can be differentiated with respect to the two spatial coordinates x and y to give

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{3} f_i \frac{\partial L_i}{\partial x}$$

$$\frac{\partial f}{\partial y} = \sum_{i=1}^{3} f_i \frac{\partial L_i}{\partial y}$$

The area coordinates, L_i, are defined as the ratio of areas that a point within the triangle divides the triangle, see Figure 6.3. These are algebraically expressed as

$$L_i(x, y) = \frac{x(y_j - y_k) + y(x_k - x_j) + (x_j y_k - x_k y_j)}{x_j y_k - y_j x_k + x_i y_j - y_k x_i + x_k y_i - y_i x_j}$$

for a cyclic permutation of (i, j, k) where

$$x_j y_k - y_j x_k + x_i y_j - y_k x_i + x_k y_i - y_i x_j$$

is twice the area of the triangle formed by the vertices of $(x_i, y_i), (x_j, y_j), (x_k, y_k)$ and

$$x(y_j - y_k) + y(x_k - x_j) + (x_j y_k - x_k y_j)$$

is twice the area of the subtriangle A_i formed by the vertices $(x, y), (x_j, y_j), (x_k, y_k)$ which are associated with L_i.

Therefore the following expressions can be obtained

$$\frac{\partial L_i}{\partial x} = \frac{y_j - y_k}{P}$$

$$\frac{\partial L_i}{\partial y} = \frac{x_k - x_j}{P}$$

for a cyclic permutation of (i, j, k) and P is defined as

$$P = x_j y_k - y_j x_k + x_i y_j - y_k x_i + x_k y_i - y_i x_j$$

As higher order shape functions are also expressed in terms of area coordinates spatial derivative approximations can also be determined. Therefore, the standard quadratic defined by

$$f = f_1(2L_1^2 - L_1) + f_2(2L_2^2 - L_2) + f_3(2L_3^2 - L_3) + f_4(4L_2L_3) + f_5(4L_3L_1) + f_6(4L_1L_2)$$

gives an approximation to the spatial derivative in the x direction as

$$\frac{\partial f}{\partial x} = f_1 \frac{\partial L_1}{\partial x}(4L_1 - 1) + f_2 \frac{\partial L_2}{\partial x}(4L_2 - 1) + f_3 \frac{\partial L_3}{\partial x}(4L_3 - 1) +$$

$$f_4 \left[\frac{\partial L_2}{\partial x} 4L_3 + \frac{\partial L_3}{\partial x} 4L_2 \right] + f_5 \left[\frac{\partial L_3}{\partial x} 4L_1 + \frac{\partial L_1}{\partial x} 4L_3 \right] + f_6 \left[\frac{\partial L_1}{\partial x} 4L_2 + \frac{\partial L_2}{\partial x} 4L_1 \right]$$

with a similar expression for the $\frac{\partial f}{\partial y}$ term, but with the $\frac{\partial L_i}{\partial x}$ replaced by $\frac{\partial L_i}{\partial y}$. The modified quadratic interpolant defined by

$$f = f_1 L_1^2 + f_2 L_2^2 + f_3 L_3^2 + f_4(2L_2L_3) + f_5(2L_3L_1) + f_6(2L_1L_2)$$

gives a approximation of the spatial derivative in the x direction as

$$\frac{\partial f}{\partial x} = f_1 \frac{\partial L_1}{\partial x}(2L_1) + f_2 \frac{\partial L_2}{\partial x}(2L_2) + f_3 \frac{\partial L_3}{\partial x}(2L_3) +$$

$$f_4 \left[\frac{\partial L_2}{\partial x} 2L_3 + \frac{\partial L_3}{\partial x} 2L_2 \right] + f_5 \left[\frac{\partial L_3}{\partial x} 2L_1 + \frac{\partial L_1}{\partial x} 2L_3 \right] + f_6 \left[\frac{\partial L_1}{\partial x} 2L_2 + \frac{\partial L_2}{\partial x} 2L_1 \right]$$

with again a similar expression for the $\frac{\partial f}{\partial y}$ term.

6.9.1 Numerical results

Several interpolation schemes will be used to obtain the numerical results considered here. These schemes are

- Scheme A: The standard quadratic shape functions using the true function values at all six points.
- Scheme B1: The modified quadratic shape functions using vertex values obtained by the one dimensional quadratic method.

- Scheme C1: The modified quadratic shape functions using vertex values obtained by the two dimensional midpoint linear scheme.
- Scheme B2: The standard quadratic shape functions using vertex values obtained by the one dimensional quadratic method.
- Scheme C2: The standard quadratic shape functions using vertex values obtained by the two dimensional midpoint linear scheme.
- Scheme D: The standard linear shape functions using the true vertex values.
- Scheme E1: The standard linear shape functions using vertex values obtained by the one dimensional quadratic method.
- Scheme E2: The standard linear shape functions using vertex values obtained by the two dimensional midpoint linear scheme.
- Scheme F: The scheme of Durlofsky et al. [29] using linear interpolants constructed from the surrounding centroid values.
- Scheme G: The scheme of Berzins & Ware [12] using a bilinear interpolants constructed from the surrounding centroid values.

Three problems will be considered, the first is the simple sine function defined in Section 6.8.1. The second is a simple Poisson equation given in [12]. The third is the complex function from Abgrall [1] given in Section 6.8.5.

Three error analyses are calculated using a three point second order quadrature rule, [23]. The points located at the edge midpoints, each with a weighting factor of 1/3. This scheme is used as the edge midpoints are the points in the domain that the numerical solution process requires the spatial derivative values.

6.9.2 Simple sine function

The first problem is that given in Section 6.8.1, a simple sine function. The results are given for the spatial derivatives in the x direction only as the function is symmetric. This is not the case for the results provided by the linear and bilinear interpolants constructed from the centroid of the triangle and the surrounding centroid values and

thus both the x and y are shown. The results are for a $[0..1, 0..1]$ domain with a regular triangular mesh. Each mesh has $2N^2$ triangles where $N = 8, 27, 81$ in these results.

No of Triangles	162	1458	13122
Scheme A	8.9534e-04	9.9610e-05	1.1068e-05
Scheme B1	4.1186e-02	1.2594e-02	4.0520e-03
Scheme C1	4.1108e-02	1.1769e-02	3.7001e-03
Scheme B2	3.9110e-02	1.2266e-02	4.0168e-03
Scheme C2	4.2077e-02	1.2645e-02	4.0580e-03
Scheme D	4.3077e-02	1.4337e-02	4.7767e-03
Scheme E1	4.6847e-02	1.4097e-02	4.4824e-03
Scheme E2	4.2917e-02	1.2776e-02	4.0708e-03

Table 6.7: Numerical derivative results for sine function

No of Triangles	162	1458	13122
Scheme F (x)	9.3348e-02	3.1835e-02	1.0713e-02
Scheme F (y)	9.8595e-02	3.2729e-02	1.0945e-02
Scheme G (x)	5.5201e-02	1.9550e-02	6.1416e-03
Scheme G (y)	5.4101e-02	1.9495e-02	5.7389e-03

Table 6.8: Numerical derivative results for sine function using schemes F & G

The expected order of convergence for the quadratic with the true values at the vertices is observed, however, this not the case for the quadratic interpolants using the constructed values. The linear interpolant using the true values also gives the expected order of convergence with the linear interpolant using the constructed values giving a similar error. In fact the linear interpolants, schemes E1 & E2 appear to give a similar error to the quadratic interpolants with less computation.

6.9.3 Simple Poisson function

The second function is a simple Poisson equation given by the analytic solution.

$$u(x, y) = 3e^{x+y}(x - x^2)(y - y^2).$$

The results are given for the spatial derivatives in the x only except again for those provided by schemes F & G. The results are given for a $[0..1, 0..1]$ domain with a regular triangular mesh. Each mesh has $2N^2$ triangles where $N = 8, 27, 81$.

No of Triangles	162	1458	13122
Scheme A	1.3466e-02	1.5050e-03	1.6731e-04
Scheme B1	1.9822e-01	7.7606e-02	2.7417e-02
Scheme C1	1.7993e-01	7.0594e-02	2.4957e-02
Scheme B2	2.5103e-01	1.0173e-01	3.6322e-02
Scheme C2	2.3801e-01	9.4077e-02	3.3307e-02
Scheme D	1.2303e-01	4.0906e-02	1.3604e-02
Scheme E1	2.2064e-01	8.5316e-02	3.0084e-02
Scheme E2	1.6547e-01	6.8723e-02	2.4748e-02

Table 6.9: Numerical derivative results for Poisson function

No of Triangles	162	1458	13122
Scheme F (x)	2.6829e-01	9.6712e-02	3.2731e-02
Scheme F (y)	3.0672e-01	1.0403e-01	3.4906e-02
Scheme G (x)	1.7759e-01	6.3530e-02	2.0881e-02
Scheme G (y)	2.0738e-01	7.4691e-02	2.2735e-02

Table 6.10: Numerical derivative results for Poisson function using schemes F & G

As expected, the quadratic interpolant using true solution values at all six points outperforms the rest. The linear interpolants also give comparable results to the quadratic when using constructed vertex values.

6.9.4 Abgrall's function [1]

The third problem is that given in Section 6.8.5, a complex problem from Abgrall. The function has many shocks and discontinuities. The domain considered is $[-1..1, -1..1]$ with a regular triangular mesh of $2N^2$ triangles where $N = 8, 27, 81$. The function is not symmetric so the x and y derivative estimates are shown. The x and y results will be given in separate tables for clarity and the results from schemes F & G will be shown separately.

The linear interpolation schemes E1 & E2 give similar errors to the quadratic interpolants. The triangular based scheme also seem to be comparing favourably with the schemes currently used in the numerical process, F & G.

No of Triangles	162	1458	13122
Scheme A (x)	1.1063e+01	9.7435e+00	9.5171e+00
Scheme B1 (x)	9.0377e+00	5.9248e+00	4.8047e+00
Scheme C1 (x)	9.7830e+00	7.0967e+00	6.0303e+00
Scheme B2 (x)	1.1478e+01	8.1140e+00	7.1376e+00
Scheme C2 (x)	1.0493e+01	7.8437e+00	6.8951e+00
Scheme D (x)	1.0064e+01	8.6168e+00	7.6777e+00
Scheme E1 (x)	7.5616e+00	4.2109e+00	2.6671e+00
Scheme E2 (x)	9.4244e+00	6.4923e+00	5.2328e+00

Table 6.11: Numerical derivative results (x direction) for Abgrall function

No of Triangles	162	1458	13122
Scheme A (y)	5.2240e+00	4.8617e+00	4.9823e+00
Scheme B1 (y)	6.3855e+00	3.9724e+00	3.0443e+00
Scheme C1 (y)	5.5467e+00	3.4395e+00	2.8341e+00
Scheme B2 (y)	8.4761e+00	6.0752e+00	5.0813e+00
Scheme C2 (y)	6.0580e+00	3.8589e+00	3.3032e+00
Scheme D (y)	6.4901e+00	4.5566e+00	3.7463e+00
Scheme E1 (y)	6.4901e+00	4.5566e+00	3.7463e+00
Scheme E2 (y)	5.6615e+00	3.1276e+00	2.3939e+00

Table 6.12: Numerical derivative results (y direction) for Abgrall function

6.10 Summary

When looking at calculating solution values for convection–dominated PDEs, this new method based on modified quadratic shape functions will not create any new extrema in the data since they are always positive and sum to unity. The method appears to work as well as a standard quadratic interpolant when the nodal values are created in both cases by interpolation, as in Section 6.5. In the applications of interest, the new method has the same rate of convergence on *difficult* problems i.e. those with steep gradients, as an unmodified quadratic. It is important to stress that for the situation considered here the unmodified quadratic cannot be used directly as solution values at the vertices of triangles are not known. The modified quadratic has the important advantage that the interpolant created will be bounded by the maximum and minimum function values used to define it. The interpolant can also be used an an essential aid to the visualisation of the problem,

No of Triangles	162	1458	13122
Scheme F (x)	1.4653e+01	1.2955e+01	1.1305e+01
Scheme F (y)	1.3806e+01	1.0916e+01	9.3596e+00
Scheme G (x)	1.1221e+01	9.3464e+00	8.5086e+00
Scheme G (y)	1.0127e+01	7.3450e+00	6.3749e+00

Table 6.13: Numerical derivative results for Abgrall function using schemes F & G

respecting the physical properties of the underlying data. These factors have been a key requirement when considering the problem class the interpolant is designed for.

The interpolant can also be used to give spatial derivative values over each triangle only using values local to that triangle. This is compared to current techniques for obtaining spatial derivative values in cell–centred schemes. Such schemes use the surrounding centroid values to construct interpolants. Numerical testing shows favourable results when comparing these two methods.

Chapter 7

Summary and Conclusions

The previous chapters in this thesis have outlined the various components of a prototype Problem Solving Environment (PSE) surrounding a general purpose PDE solver for two dimensional convection-dominated problems. In this chapter an attempt will be made to draw together what has been learnt so far, to try and make an initial assessment of the PSE by considering a real problem, and to make some conclusions.

In Chapter 2 of this thesis the role and nature of PSEs in scientific computing was examined. A PSE was seen as trying to provide a bridge between the gap of those who wish to take full advantage of the tools available in the scientific computing community and those who are actually able to use these tools in their raw state. A PSE was then shown to need to satisfy one or more of the following conditions

- To ease the problem solution process.
- To reduce the overall time spent on the solution of a problem.
- To be a more natural or convenient way to solve a problem.

The focus of this thesis then moved to look at the area of the numerical solution of partial differential equations (PDEs) and in particular two space dimensional problems. Several numerical codes capable of solving such problems were examined. The user-friendly environments surrounding these solvers were outlined. Several of these may be considered to meet the criteria of PSEs. The aims and driving forces behind these PSEs were examined and, whilst diverse, the systems constructed can be seen to fulfill the requirements of a PSE. However, the time, effort and expertise expended on such PSEs is large, for example

the //ELLPACK systems [47] and the Visual PDEQSOL system [86]. The way forward is seen by many, [82], as utilising and integrating existing scientific software tools and techniques to construct portable generic software tools as has been shown in Chapter 2. These can be used to aid in the solution of two dimensional convection-dominated PDEs. In order to construct these generic software tools suitable building blocks need to be found. These building blocks should be widely used, portable and familiar to the scientific computing community. Tools that meet these criteria were discussed in Chapter 2 and will be critically examined later in Section 7.2.

Chapter 3 looked at the numerical solution of convection-dominated PDEs in two dimensions and the techniques used to solve these problems. The SPRINT2D numerical software, based on the earlier SPRINT package [9], is a general purpose solver for two dimensional convection-dominated PDEs, was taken as a basis for this discussion. The input requirements for SPRINT2D was shown to take the form of a driving program. This was shown to be common for numerical solvers in this area.

The next three chapters looked at the construction of software tools that can be used to aid in the solution of two dimensional PDE problems. Chapter 4 looked at a visual specification tool to define the numerical domain required. Chapter 5 at a visual problem specification system where a PDE can be specified in a quick and natural way. Both these tools deal with the problem specification side of the overall solution process. Chapter 6 was concerned with interpolation routines designed to compliment the numerical solution of PDEs. These can be used within the numerical solution process itself to recover solution values or used as an aid to the visualisation of the solution. These three areas will be discussed in more detail in the next section.

7.1 Review of PSE Components

This section will give a review of the components of the PSE discussed in this thesis; the visual domain specification tool for constructing geometries in two dimensions; the visual problem specification systems for specifying modules and parameters used within the numerical code and, finally, interpolation routines designed to respect the physical nature of the solution. In particular this section will aim to highlight what can be learned from constructing PSE components in an open environment working along side users and developers of numerical software.

7.1.1 The Visual Domain Specification Tool

When solving two dimensional PDE problems, the numerical code requires a mathematical representation of a domain over which the problem is to be solved. Mesh generation packages provide this model but, in turn, they require a numerical representation of the domain. This representation is usually carried out manually, a pen and paper method is used to draw the domain, this is hand converted into suitable input for the mesh generation software. This is a simple enough task but prone to error and time consuming. The simplification of this task for the user was seen as an advantage. This was achieved by providing a visual tool to replace the pen and paper and a post-processing system to replace the hand conversion. The visual domain specification tool (VDS tool) provided this.

The layout and form of the VDS tool has developed over time, to best focus this development, feedback from the users of the tool were included. In this way the VDS tool would meet the users needs and perform the tasks the users required of such a tool. This was an important part of the VDS construction process and ensured that the tool did what the users needed rather than restricting or imposing its own nature onto the users.

The VDS tool also needed to be generic, in order for it to be included in a PSE. It could not afford to be tied down to one particular mesh generation package. The visual pen and paper part should be as in real life. The geometry constructed should be equally applicable to any mesh generation package. The visual interface part of the tool stores the domain in a generic format, the post-processing system is responsible for converting this into suitable input for mesh generation software. The tool was shown to produce valid input to two mesh generation packages and to be capable of producing input for others this was another key part in the construction of the VDS tool.

The VDS tool and post-processing system needed to be portable, the visual interface of the tool utilises the X Windowing system [54] and the OSF/Motif Widget set [94] to obtain this. The internal aspects of the tool and the post-processing system used C in an UNIX environment to gain this portability. This is essential for the VDS tool be considered part of a PSE.

The VDS tool does provide a simple but powerful tool for specifying the numerical domain required by the numerical code. The VDS tool simplifies the construction process, provides a more natural way to specify the domain and its use leads to a considerable

decrease in the time spent on this part of the specification process. In this way the VDS tool is shown to be a valid part of any PSE surrounding numerical code for solving two dimensional PDE problems.

7.1.2 The Visual Problem Specification System

Many scientific software packages require the user to construct a program that will initialise and drive the code. The use of these driver programs is common. This is true when looking at the numerical solution of PDEs. The usual method is to find an existing program and modify it or to construct a program from scratch. Both are error prone, time consuming and require the user to understand the programming language used. One of the aims of PSEs is to allow those who know about the problem to be able to solve it. The removal of this need to manually create a driver program, or at least to create a well structured template which the user can use, was one of the main aims of the VPS system. To make this specification process easier, quicker, less error prone and more natural were also of prime concern.

As with the VDS tool, described in the previous section, the VPS system must aim not be tied exclusively to the code. The generic problem class of two-dimensional PDEs was then used as the framework upon which to build the interfaces. The interface was split into four logical groupings, the equation, the numerical domain, the spatial discretisation method and the nature of the problem. The requirements of numerical software in this area was matched to these groups. An open design environment talking with the developers and users of the code helped to determine these groups.

As well as the generic structure of the VPS system the visual interfaces and post-processing systems needed to be portable. The visual interface was written in X and Motif for this purpose, the remaining programming was in C on a UNIX platform. Other packages used needed to be widely used and accessible to the scientific computing community, the Maple symbolic algebra package [22] and the LaTeX[61] document processing tool were also used. The portability of these tools will be examined later in Section 7.2.

The VPS system offers sensible default values for the information that is required. This is essential as it allows novice users to construct a valid driving program, again the developers and users of the numerical code provided essential advice when choosing these values. It also ensured that the numerical code has all the information it requires.

For the more experienced user, the VPS system is by no means complete, the scope of problems that can be successfully solved is small. More complex problems may require user modification to the driver program, however, the VPS system provides a suitable template driver program with which the user can work.

This gap between what can currently be done in this area and what is required to solve a more complex problem is highlighted in Section 7.3.1. Despite these limitations the VPS system simplifies the specification process for the problem. It provides a more natural way to specify the problem and can eliminate the need for explicit programming. It provides sensible default values, allowing novice users to use the system. It also gives a considerable saving in the time spent in this part of the specification process. The open design environment under which this was done utilised the expertise of users and developers of the numerical code to great benefit.

7.1.3 Quadratic Interpolation for Triangular Cell-Centered Finite-Volume Schemes

When solving two dimensional convection-dominated PDEs great care must be taken to avoid introducing physically unreal values into the solution. For example physical values such as density should always be positive. The solution to PDE problems are often characterised by shocks and discontinuities present in the solutions. Numerical PDE solvers take great care to respect the nature of the solution, this is a prime concern. However, such conditions can lead to undershoot and overshoot around these features if standard interpolation techniques are used, such interpolants fail to respect these properties, in effect destroying the efforts of the numerical solver.

In Chapter 6 a triangular based interpolant was described. The interpolant achieved the desired properties by bounding the values it produced between the maximum and minimum values used to define it. This was made possible by modifying the standard quadratic shape functions such that they were always positive and summed to unity. In this way any interpolated solution value in the triangle will be bounded. This will always respect the nature of the solution and in doing so compliment the numerical code. This can be used during spatial remeshing to recover solution values and in the visualisation of the solution.

The interpolant requires six functions values for each triangle, the three edge mid-

points and the three vertex values. As several numerical schemes produce only centroid values and edge-midpoint values this chapter also looked at a way to construct sensible bounded vertex values from these known values. This took one of two forms a one dimensional quadratic approach or a two dimensional linear scheme. Numerical results show that similar errors are obtained when comparing the new method with standard interpolation methods. Graphical results show that undershoot and overshoot when using the new method were eliminated.

Another area where interpolation techniques are used in the numerical process is to determine spatial derivative values. The interpolant allows a way for spatial derivative values to be constructed from solution values local to each triangle. Current techniques use surrounding values to gain these estimates. This can cause problems if the triangle is close to a discontinuity. Again numerical results show favourable results when compared with current techniques.

The interpolant described fits into the general PSE framework by providing an aid to the solution process and a more natural way for the user to view the solution. In this way it can be seen to add to the overall solution process. It can also reduce the time spent in the solution process by preserving the nature of the solution data thus allowing the user to easily interpret the results.

7.2 Portability Issues

In this section the portability of the various components described above will be examined. One of the aims of these software tools was to construct them from highly portable building blocks. This was achieved by using industry standard or de-facto standard software tools and programming languages. This section will give a brief description of these building blocks and discuss how portable they are, along with problems that might occur and possible remedies.

As mentioned earlier the aim is to produce software tools that are generic in nature and also portable. When looking at graphical user interfaces the X Window system [54] stands alone in this area. The X Windowing system is an industry standard and is available in a wide range of hardware, from top of the range workstations to personal computers. Using X as a backbone to the system gives the portability desired of the software tools. The X system provides various libraries and toolkits that allow the user to

construct portable graphical interfaces. Sitting on top of the X Windowing system is the OSF/Motif widget set [94]. The Motif Widget set is one of many widget sets that sit on top of the facilities offered by X, allowing easier construction of graphical user interfaces. The Motif widget set is a a commercial widget set developed by the Open Software Foundation, it is widely-accepted and is also available on a wide variety of hardware platforms.

The software was developed on a UNIX platform using the C programming language. All UNIX machines will have a C compiler, both UNIX and C are widely used and it is unlikely that the use of these will create any serious portability problems. The numerical code considered here also requires a FORTRAN compiler, this is again widely available. It is possible however to install and run the interface software on one platform and then to transfer the driver program and numerical mesh specification file onto another system capable of running the numerical code.

The remaining building blocks may present more problems when looking at the portability of the system. The first is the use of the Maple symbolic algebra system [22], this is used to convert user input into C functions for the driver program and also to produce LaTeX output, [61]. The ability to switch off the typesetting facilities of the tools is possible, therefore the use of the LaTeX system is not an essential prerequisite. Maple is one of many symbolic algebra systems available, many of which have similar routines to produce functions in a programming language or instructions for typesetting programs, see Chapter 2. It would also be feasible to utilise another symbolic package to convert the users input to a programming language. The tools to do this are written in such a way that this would simply involve changing two routines in the post-processing system.

The final two tools that are used are for the visualisation of the results from the numerical code. At present the code uses a visualisation package based on IRIS GL, [91]. This package may be switched off. The system also has the ability to produce files compatible with the IRIS Explorer visualisation system [77], this is not required to successfully run the numerical software.

In summary, the majority of the system is highly portable, however, problems may occur with the symbolic algebra system Maple and the typesetting system LaTeX. The latter is not required for the code to produce a driver program but the former is, although the substitution of another symbolic package is possible.

7.3 A Critical Evaluation of the PSE

In this section the PSE described in this thesis will be critically examined. This will be done by looking at a combustion problem relating to the modelling of knock in car engines. This problem is taken from [14] where the SPRINT2D numerical code discussed in this thesis is used to solve this problem.

It must be stated that in order to solve this problem additions were made to the way the numerical software is driven from those given in Chapter 3. The evaluation will take the form at looking at the gaps between the PSE outlined and the requirements of this combustion problem. It is hoped that this evaluation will show that whilst the PSE has been shown to allow users to solve simple PDEs there is still a large gap between what can currently be offered and what the ultimate aims of a PSE in this area should be. It will also be shown that in the case of many of the shortcomings the additions required to remedy them are non-trivial.

The rest of this section will first describe the combustion problem. The remainder will outline the main areas where additions to the driver program were made, the reasons for these changes, and possible enhancements to the software tools in the PSE. It must be stated that these changes will apply to the visual problem specification interfaces discussed in Chapter 5. It is this part of the specification process that is perhaps most closely linked to the numerical software. It is also encouraging that the other areas of the PSE, the visual domain specification tool and the interpolation routines, remain unchanged. This would seem to highlights the generic nature of these components.

7.3.1 The Knock-Modelling Problem

The problem is specified by a system of five PDEs. The first four represent the conservation of mass, momentum and energy. The fifth is a species equation dealing with burning fuel.

$$\frac{\partial \underline{u}}{\partial t} + \frac{\partial \underline{\underline{f}}}{\partial x} + \frac{\partial \underline{\underline{g}}}{\partial y} = \underline{S}$$

where

$$\underline{u} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \\ \rho z \end{pmatrix}, \underline{f} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u[E+p] \\ \rho uz \end{pmatrix}, \underline{g} = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v[E+p] \\ \rho vz \end{pmatrix},$$

$$\underline{S} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -\rho s \exp\left(\beta\left(1 - \frac{1}{T}\right)\right) \end{pmatrix}$$

where $T = p/\rho$ and $\beta = 20.0$.

The variable ρ, u, v, p are the density, velocities in the x and y dimensions and the pressure, z represents the scaled fuel concentration. The energy E is defined by the equation of state

$$E = \frac{p}{(\gamma - 1)} + \frac{\rho u^2 + \rho v^2}{2} + \alpha \rho z$$

where $\gamma = 1.2$ and $\alpha = 8.0$.

The geometry of the problem and the initial conditions for the problem are given in Figure 7.1. The irregular solid line represents the initial position of the flame front. This is obtained from experimental data. The dotted concentric circles indicate temperature hot spots which will lead to autoignition. These will cause pressure pulses that will travel across the cylinder and lead to knock. The four numbered points on the circumference of the cylinder are pressure transducers where pressure histories from experiments are available.

7.3.2 Solving the Knock-Modelling Problem

The following section will highlight the gaps between what can currently be offered by the PSE described here and the way in which the developers of the numerical code are solving this practical problem. There are several areas discussed here from the numerics of the solution process to the use of new visualisation routines to view the results.

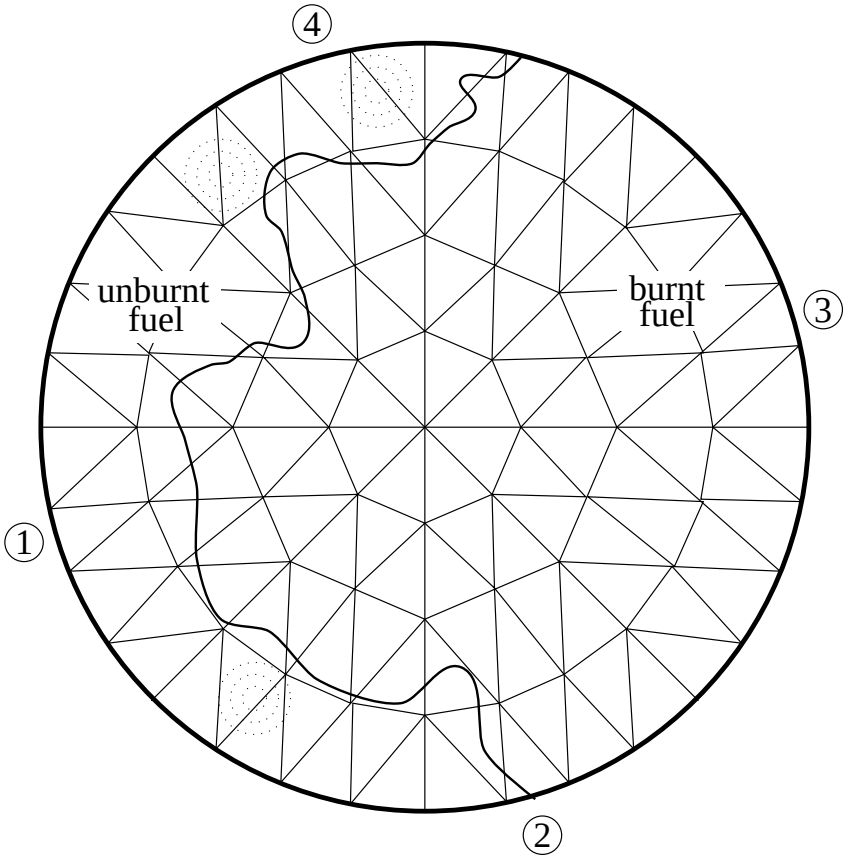

Figure 7.1: Domain and Initial Conditions for the Knock Problem

System of Equations

The first major step up from the current state to solving complex problems is the ability of the system to handle systems of equations. This issue has been briefly touched upon when describing the equation interface in Chapter 5. The combustion knock-modelling problem is defined by a system of five PDEs, the ability for the user to enter systems of equations in a natural way is an obvious extension to the PSE.

Initial Conditions

The initial conditions for the combustion knock-modelling problem are more complex than the form allowed in the PSE. The initial conditions cannot be specified by a mathematical function but are specified on a triangle by triangle basis. These are taken from camera data. The driver program for this problem reads in these results from a data file. This routine involves approximately 450 lines of code. It is not easy to see how this may be done visually in an effective way. One possible way would be to extract the internal representation of the numerical domain from the numerical code and allow the user to specify a triangle or group of triangles and specify the initial conditions of the unknowns. Another way would be to perhaps simply display the *name* of each triangle and then provide a template file for the user to work with. Both methods would require the PSE to communicate with the numerical software and extract information about the internal representation of the numerical mesh.

Numerical Domain

As the initial conditions are taken from observational data, at the start there is no error present. However, numerical errors will quickly appear at the location of the flame front, see Figure 7.1. Heavy refinement of the computational mesh will be required. The ability to instruct the numerical code to refine around these areas is required. This is also true for areas where other phenomenon have been observed in experimental runs, or are likely to occur. To combat this the numerical code has been modified to allow the increased mesh refinement around a specific location. The user has the ability to specify the x and y coordinates of the point and a refinement level. The user can also allow the code to do one of the following

- Derefine this area when the surrounding area has reached a similar refinement level.

- Refine the initial mesh to this level then allow normal spatial adaptivity routines to refine/derefine when necessary.

The addition of these new functions to the numerical software would appear to lend itself to a visual interface. The user could view the numerical domain, with or without a mesh, and then click on a specific location. A popup could then request the level of refinement when the code is allowed to derefine.

Riemann Solver

Perhaps the largest problem faced with producing a driver program capable of giving a valid solution is that of the Riemann solver. The problems seen here have already been highlighted in Chapter 5. The current method used can perhaps be best described as naive. A central differencing scheme is employed which ignores all the directional information about the flow. A naive Riemann solver can lead to a poor solution and may breakdown with complex problems. For the combustion knock-modelling problem the Riemann solver function within the driver program is approximately 650 lines of code. The Riemann solver took an experienced user of the numerical software with a full understanding of the numerical processes involved about 5 days to construct. This is from initial theory via an intermediate one dimensional problem to the final version used in the two dimensional problem. It is easy to see that this is not a trivial task, often requiring symbolic manipulation of the equations.

It is perhaps this area that presents the greatest barrier to the construction of, what might be described as, a complete PSE surrounding numerical codes using the finite volume method. Much can be achieved by providing the users with a template driver program and automating much of the specification process. However, for problems that require more complex Riemann solvers the user is still required to construct an appropriate function.

Switch Off Diffusion

As the combustion knock-modelling problem has no diffusion terms then the numerical code can be instructed to switch off that part of the calculations. This function could easily be added to the visual specification system or be deduced from the equation specification part.

Operator Splitting

A feature added to the numerical code in order to solve this problem much more efficiently is operator splitting. When problems containing chemistry and flow terms are encountered the PDEs that model the flow can often take on the characteristics of the chemistry system. The problem here is that the chemistry requires the use of an implicit ODE solver. Typically implicit solvers use linear algebra to solve non-linear systems and this can be very expensive in terms of computational power. Some savings may be made if the structure of the ODE system is exploited or iterative methods are used but there is still a substantial overhead associated with such solvers. Explicit ODE solvers are much more effective when looking at flow problems, but have problems when dealing with the chemistry terms as the stability of the system limits the step size. In general, the chemical reactions will operate on a much smaller time scale than the flow reactions. This implies that chemical reactions inside a cell (or triangle) is much more important than the interaction between flow and chemistry. It is therefore possible to ignore the chemistry-flow interaction and use an explicit ODE solver which treats the chemistry terms implicity. This technique is known as operator splitting, see [14] for more details. This process results in a modified "explicit" scheme that is capable of handling the chemistry terms but performs much more efficiently than a fully implicit scheme. It must be noted that this operation does result in a *splitting error* being introduced into the solution of the nonlinear equations that may limit the rate of convergence or the accuracy that can be achieved, [14]. At present this splitting is done manually, however, it may be possible for this to be done numerically within the code. If this is the case then this may be switched on or off in the VPS tool.

Output Routines

Several new output routines have been constructed for this problem. An X program providing contour plots of the solution has been constructed. There has also been other new output routines, created to mimic the results that can be obtained by experimental results. These include producing results at the pressure transducers located around the circumference of the cylinder. The pressure results can also be collected along the circumference of the cylinder, both are measured when experimental results are undertaken. The final output routine to mimic the experimental results is that of particle

tracking. In experimental runs, pepper corns are placed in the cylinder and tracked from camera produced results. The numerical code therefore allows the user to place a particle at a location and track its movement. All these output routines are very much problem dependent.

Summary

In this section the solution of a complex problem such as a combustion knock-modelling problem is considered. The requirements of the numeric code to solve this problem have been highlighted and the gaps between this and what can be currently offered by the PSE. In some areas, as the code evolves, the additions to the tools in the PSE are simple. In other cases the gaps seem vast without an easy solution or a knowledge base upon which to build. One good example of this is the Riemann solver.

7.4 Conclusions

In this thesis we have looked at PSEs in scientific computing. There exists several different views and different perspectives, about the form and nature of PSEs which have changed over time. The aims of PSE system builders have changed, however, is it possible to combine many viewpoints, to give the essence of what a PSE should do. A PSE is probably best viewed as an enhancement to the problem solving process, a bridge between those who wish to use the available facilities offered by modern scientific computing and those who can actually use them. The advantages a PSE can bring are to reduce the time spent on the total problem solving task, to provide a more natural or more convenient way to specify and solve the problem and to understand the results. The PSE should enhance and clarify this process. The view of a PSE as a large complex system has perhaps changed to the opinion that PSEs are best defined by a collection of scientific computing tools which can be combined effectively.

In this thesis PSEs in the area of the numerical solution of two dimensional partial differential equations have been considered. There are again a wide variety of PSEs in this area, most being heavily problem oriented. Many numerical codes in this area have user-friendly environments surrounding them. Some of these environments are both complex and powerful, however, these tend to be expensive in terms of time and effort spent on their development. Two such examples of these are the //ELLPACK system [47] and the

Visual PDEQSOL system [86]. The way forward for PSEs is seen as utilising existing technologies to form a collection of tools which can be combined to form PSEs, see [82], two good examples of this are the RPI system and the NAG/AXIOM system. All of these systems aim to enhance the problem solving process and provide powerful problem solving tools.

The aim here was to abstract the ideas presented by these PSEs and to look at a more generic approach for the construction of PSEs in this area. The utilisation of suitable software packages and systems as building blocks showed how to provide portability. The use of industry standards such as the X Windowing systems and other widely used scientific tools helped to achieve this. The inclusion of the end users of the PSE and developers of the scientific code which the PSE surrounds helping to focus these efforts. An open design environment essential to make best use of the time and resources available.

The question posed is; is it possible to do this ? In order to demonstrate that it is possible this thesis has looked at one general purpose numerical solver and attempted to construct tools that are suited to the generic solution process. The aim was, not to let the code take over and simply produce a user-friendly environment for the code. This has been easier for some areas than others. The Visual Domain Specification (VDS) tool described in Chapter 4 and the Interpolation routines in Chapter 6 perhaps do achieve this. When looking at the Visual Problem Specification (VPS) system described in Chapter 5 it is not as clear that this has been achieved. The very nature of this system and the problems has forced a greater interaction with the numerical code.

The thesis has looked at the construction of tools and systems that can be considered as generic tools which stand around numerical code. Visual user interfaces are combined with post-processing system for problem specification tools such as the VDS tool and the VPS system. The interpolation routines are designed for the problem class to respect the nature of the problems. These go some way to form a PSE for the numerical solution of PDEs in two spatial dimensions. In this respect the aims have been met. The tools can enhance the solution process, reduce the time spent and provide a more natural and convenient way to solve the problem. The open nature of the design environment helped in this task.

Whilst these aims have been met it is easy to see that this is only part of the way to a complete PSE, there is still a long way to go. There are still many difficult areas when looking at the problems that users wish to solve, the combustion knock-modelling

problem described in this chapter highlights the areas where the numerical software itself needs modifying to solve such problems. The difficulties faced when experienced users use the numerical software; such as constructing a Riemann solver and inputting initial conditions show that there are still difficult tasks to be performed before we gain good results. As understanding in these areas increases the PSEs can develop alongside the numerical code, the advantages that an open design environment bring is important in this respect.

This thesis has shown that despite the limitations much can still be achieved. Tools with a generic nature combined with post-processing systems can be constructed. These tools can then be enhanced and expanded. It is possible to utilise current scientific computing technology to build software tools and packages that when combined form an easy-to-use layer surrounding complex computational code. This layer can help both novice and experienced users to better utilise their time, efforts and knowledge, even if the layer provides only partial help for difficult problems. Inclusion of users and developers in the design process of such tools can help to focus the development of the PSE and the tools it encompasses.

Appendix A

Users Guide and Help for Xdesign (VDS Tool)

A.1 Introduction

The Xdesign tool is designed to help the user quickly and easily to create a geometry specification for a mesh domain file. The tool is an interactive graphical user interface that produces a numerical domain specification file. This users' guide will explain the features of the Xdesign tool that allow the user to produce a domain specification file. The guide will also explain the output file format and provide the user with advice on how to edit the file if required.

The Xdesign tool consists of three main parts, the Drawing window (right window), the Canvas window (left window) and the Control panel (bottom part of tool).

The drawing window provides the user with an area where a geometry can be specified quickly and easily using the mouse as an input device. The canvas window allows the user to see the resulting geometry meshed with an additional level of refinement. The control panel provides several buttons and labels which give the user control of the tool. Each of these will be described in detail in this guide and an example user session is given.

A.2 The Control panel

The Control panel consists of several buttons that when pressed will affect the behaviour of the tool. The actions of the buttons are described below.

- **The Quit Button** This will terminate the Xdesign tool.
- **The Clear Button** This will reset the Xdesign tool to the default state, all windows will be cleared, the drawing shape will be set to Line and the Grid will be switched on. The default coordinates for the drawing window are [0.0–5.0, 0.0–5.0] with the default grid lines at 0.5 intervals in the x and y direction.
- **The Refresh Button** This will redraw the content of the drawing window and the canvas window.
- **The Shape:Line/Arc Button** This will toggle the drawing shape between Line and Arc. To define a line the two end points are needed. When the Shape type is set to line the application will *rubber-band* a line from the current mouse pointer position and the last user defined point, whenever relevant. Three points are needed to specify an arc, the start point, an interior point and the end point. The drawing of the arc will not be *rubber-banded*.
- **The Grid:On/Off Button** This will switch the grid in the drawing window on and off.
- **The Snap:On/Off Button** This will switch the snap to grid points mechanism on and off. When the snap is switched on the creation of new points will automatically be placed at the point of the nearest intersection of the grid lines.
- **The Snap All To Grid Button** This allows the user to draw with the snap off, then at any point in the drawing or editing session, move all the points to the nearest grid intersection point. This facility allows the user to change to grid and then snap the points to the new grid.
- **The Show Points Button** This switches On and Off the markers to show the location of the user defined points.
- **The Size – and Size + Button** This buttons allow the user to increase and decrease the size of the drawing window. The window itself is sized according to the ration of the x and y coordinates specified by the user. These buttons will resize the window and keep this aspect ratio.

- **The Mesh Design Button** This button activates a popup that allows the user to specify the name of the domain file the geometry will be specified in. If the KSLA option is chosen it will also automatically pass the file to the Canvas window to be displayed. It also provides the user with the option to select the mesh status. The mesh status can be either mesh simple, mesh hole or mesh interior. The various options are explained later in the section on how to edit the specification file. If the GEOMPACK option is chosen then the mesh status is set to Simple and a popup will prompt the user for the additional information required by GEOMPACK to mesh the domain.

- **The Co-ords Button** This button activates a popup that allows the user to specify the coordinates of the drawing area. The window the user actually draws on has a [0.0–1.0, 0.0–1.0] coordinate scale. The user then specifies a bottom left coordinate pair and a top right coordinate pair which are used to transform these coordinates to the user specified coordinates. The Co-ords box also allows the user to specify the x-increment and y-increment for the grid lines.

- **The Load File Button** This button activates a popup that prompts the user for the name of a previous domain specification file. This file is then loaded into the Xdesign tool, the tool is set to edit mode and the user may then edit the points. When loading in a file the user may explicitly state the top right and bottom left coordinate pairs or the computer will attempt a best guess from the data in the file. This option is set by pressing the Specify Co-ords button in the load file popup. This will toggle between AUTO and USER.

- **The Points From File Button** This button again activates a popup and prompts the user for a file name. This allows the user to read in a file containing x and y coordinate pairs. This enables the user to use coordinates generated by other programs or mathematical functions.

- **The Help Button** This provides the help on how to use the Xdesign tool.

- **The Quick Help Button** This provide a quick help on the operations of the mouse buttons during the drawing and edit stages.

- **The Show Info Button** This can be set to On or Off, when On the names associated with the boundary edges will be displayed

- **The X and Y labels** These labels give the x and y position of the mouse pointer in the drawing window.

A.3 The Canvas Window

The canvas window shows the resulting mesh produced by passing the domain specification to the mesh generator. The mesh shown has an additional level of refinement.

A.4 The Drawing Window

The drawing window allows the user to specify a geometry using the mouse. It also allows the user to edit a geometry depending upon the mode of the drawing window.

The drawing window can exist in one of two modes, the drawing mode and the editing mode. The initial mode of the window on startup is the drawing mode the editing mode is switched on by the user finishing the geometry. The user can create new points via mouse button presses and build complex geometries with lines and arcs.

The editing mode allows the user to change the spatial position of the points used to specify the geometry. The commands for the drawing window in both modes are given below.

A.4.1 In Drawing Mode

- The Left Mouse Button will create a new point at the current mouse pointer position. All new points are created with this button.

- The Middle Mouse Button will close a region by joining the last and first user specified points in that region. The window will remain in edit mode.

- The Right Mouse Button will close a region and will switch the window into edit mode.

The current position of the Mouse Pointer, in the coordinates of the drawing window, is shown in the bottom right hand corner of the tool.

A.4.2 In Edit Mode:-

- The Left Mouse Button will activate a popup that will give the x and y coordinates of the point nearest to the mouse pointer when the button was pressed. The user may then change these x and y coordinates and the location of the point will be changed. The geometry will be redrawn reflecting the position of the new point.

- The Middle Mouse Button will Delete the nearest point to the mouse pointer. This may not be allowed if it is the first point or there are not enough points in the region or the point belongs to an arc.

- The Right Mouse Button will add a new point if the closest point to the mouse pointer is the first point in a straight line. The new point will be added to the middle of the line.

These commands are best explained by providing the user with an example that illustrates how these are used in a drawing session, the right hand picture in Figure 4.9 shows a geometry comprising of square with two smaller interior squares, or holes inside the outer. The following is the actions taken to achieve this

- The Co-ords Button was pressed, this produced the co-ords popup. The Bottom Left and Top Right coordinate pair were selected and the Grid Increment was also selected. Once these were set the ok button on the popup can be pressed, the coordinates of the drawing window and the grid lines are updated and the popup is removed.

- The Shape:Line/Arc Button, the Snap:On/Off Button and the Grid:On/Off Button are checked to ensure that the correct settings are chosen. The Shape is set to Line, the Snap is set to On and the Grid is also set to On.

- The Left Mouse Button is pressed when the mouse is at the position of the top right corner of the outside square. This creates the first point.

- The Left Mouse Button is pressed a further three times, once at each corner of the outside square. This creates the next three points. Since the snap is switched on the points are created at the intersection of the grid lines.

- The Middle Mouse Button is then pressed, this will join up the first point and the last point. This will create the outside square. The drawing session has not ended since the Middle Mouse Button is used.
- The Left Mouse Button is pressed a further four times when the mouse pointer is at the positions of the corners of the top left internal square.
- The Middle Mouse Button is used again to close the square without terminating the drawing session.
- The Left Mouse Button is pressed a further four times when the mouse pointer is at the positions of the corners of the bottom right internal square.
- The Right Mouse Button is now used to close the square and to terminate the drawing session. The drawing Window now moves into edit mode.
- From this point the Mesh design Button may be used and the resulting mesh will be shown in the Canvas Window. The user may now edit the geometry.

Appendix B

Changes To The Visual Domain Specification Tool

The following section gives a list of the modifications made to the tool as well as the reasons behind the changes. Many of the earlier changes are superseded by later changes. The iterative process involving the users of the tool, builds upon the last cycle of the design process to enhance the functionality of the tool. At this stage of the development process the tool was converted from using the Athena widget set to using the Motif widget set. The change was requested by the users of the tool. This made little difference except for cosmetic changes to the tool, the tool still had the portability of the X windows system.

- Scale Button added to tool. The mesh is drawn on a $[0..1, 0..1]$ region, the scale button allows the user to scale this to a positive or negative domain. The need for more flexability was expressed by the users.
- Addition of a Grid on the Drawing Canvas. The grid was a fixed size grid to act as an aid to the design process.
- Refresh Button added. This was added to redisplay the tool in case of missed X expose events.
- Move from Scale Button to Co–ords Button. The scale button was replaced by a co–ords button. The new button allowed the user to specify the top right and bottom left coordinate pairs. The drawing region could then be scaled and translated to fit the user

specified coordinates. This provides an easy means of specifying the spatial domain under consideration.

- Ability to Sketch then Edit design. The ability to place the points of the geometry accurately on the drawing canvas. The ability to sketch the shape and then to look at each of the points and to specify the numerical value of the x and y coordinates. This function was added by switching the drawing canvas from the initial design state to an edit state after completion of a geometry, by pressing the right mouse button. The action of pressing the left mouse button in the edit state would be to produce a popup window with the x and y coordinates of the nearest point to the mouse pointer. The user can then change these coordinates, if required, and by pressing the Ok button the position of the point will move to the user specified point. The Cancel button will remove the popup without effecting the position of the point.

- Addition of Arcs as a drawing primitive. The KSLA mesh generator supports domains specified by arcs as well as lines. The arc is defined by a start point, an end point and a middle point. The ability to use arcs enables the tool to be used to create a wider range of geometries. The increase in the complexity of the geometries was requested by the users to ensure the tool could create geometries currently used. The use of *rubber-banding* for arcs proved difficult since three points are needed, so any *rubber-banding* could only be done after the first two points had been specified. Also the movement of the mouse pointer after the specification of the first two points of the arc would result in large changes in the definition of the arc.

- Shape: Line, Shape: Arc Button. The Shape button will toggle between the two shape primitives line and arc. The addition of arcs as drawing primitives creates the need for this button.

- Help Button, this was added to give details of the three main areas involved, the Drawing Canvas, the Display Canvas and the Control Panel.

- Addition of a Grid: Coarse / Fine / None Button. The Grid button allows the user to toggle between a coarse grid, a fine grid and no grid on the Drawing Canvas. The reason behind this was to provide a visible aid to the user during the design process. It replaced the need to determine a sensible grid spacing value from the user-specified coordinates.

- X and Y spatial coordinates labels. With the grid no longer giving easy reference points in the Drawing Canvas in many situations the addition of these labels gives the user the position of the mouse pointer in the user defined coordinate system of the Drawing Canvas.

- Load Button added to the tool. The user expressed a wish to load previous or existing mesh files into the tool so that these maybe modified. Also this allowed a sketched outline to be stored and refined later. The load button enabled an external KSLA mesh file to be transferred into the tools internal data structure. The tool will then determine automatic values for the maximum and minimum coordinates of the Display Canvas obtained from the data in the meshfile. The tool was then placed in edit mode allowing the geometry to be modified.

- At times the automatic determination of the coordinates of the Display Canvas gave unwanted results. The Load button was therefore modified to allow the user to specify these values or for the tool to obtain these values from the meshfile. The user option allowed the to specify the top right and bottom left coordinate pairs. This allowed the user more control over the loading in of new files but also provided the option for the tool to automate this process.

- The users requested that the Drawing Canvas and the Display Canvas be proportional to the aspect ratio of the coordinate specification. This feature was added such that a coordinate specification of 0..2 in the x direction and 0..3 in the y direction would result in the Drawing Canvas and Display Canvas changing size so that the dimensions of the window would be 1.5 : 1.0 i.e. the y dimension 1.5 times bigger than the x dimension. To allow the user to resize the tool and to still preserve the aspect ratio two additional buttons were added. The Size + button allowed the Drawing Canvas and Display Canvas to be enlarged and the Size − button allowed these to be shrunk.

- Modification to the grid capabilities. The users requested a change in the way in which the grid was defined. The user requested control over the spacing of the grid lines by specifying an x increment and y increment.

- Changing of Grid Button. The ability for the user to specify the grid allowed the Grid Coarse/Fine/None button to be replaced by a Grid On/Off button.

- Snap To Grid Button. The facility was included to allow the user defined points to be placed at the intersection of the grid lines. The Snap to Grid can be switched on and off during the Design session.

- Snap All To Grid Button. This function allows the user to sketch and define the grid spacing and then snap all the points to the grid.

- Load Points From File. The users stated that at times the points used to define a geometry would occasionally be stored in an external file containing a list of x,y coordinate pairs. This function allowed the user to read a series of points in from a file rather than specifying the points with the mouse.

- Addition of a Mesh Status button on the Mesh Design Popup. The mesh generator has the facility to allow holes and internal regions to be meshed or not to be meshed. This facility relies on the specification of the geometry in the meshfile. The Mesh Status button allows the user to specify a Mesh Staus of Simple, Mesh Holes and Mesh Interior.

- Update of Help system. A more comprehensive help system was added. The help system provided information about the controls for the Drawing Canvas, Display Canvas and Controls. It also provided an example design session along with help on the effects of the Mesh Status button.

- The addition of a GEOMPACK mesh button on the Mesh Design popup to allows a numerical specification file suitable for the GEOMPACK mesh generator. The Mesh Status would automatically be set to Simple in this case. A GEOMPACK Information popup requests the additional information required by the GEOMPACK package.

- Addition of a Show Info button. This could be set to on or off. When set to On it would show the unique integer identifier assigned to each boundary edge in the domain. These names are used to assign boundary conditions to the domain.

- The ability to add and delete points in Edit mode. This allowed the user to delete certain points and create new ones. The middle and right mouse buttons were used to control this.

- A show points button was added to switch on or off the markers used to show the user defined points. This enabled domains with many points to be viewed easily.

- The Quick Help and Online help are updated to explain the additional features.

Appendix C

Summary Windows for the Three Example Problems

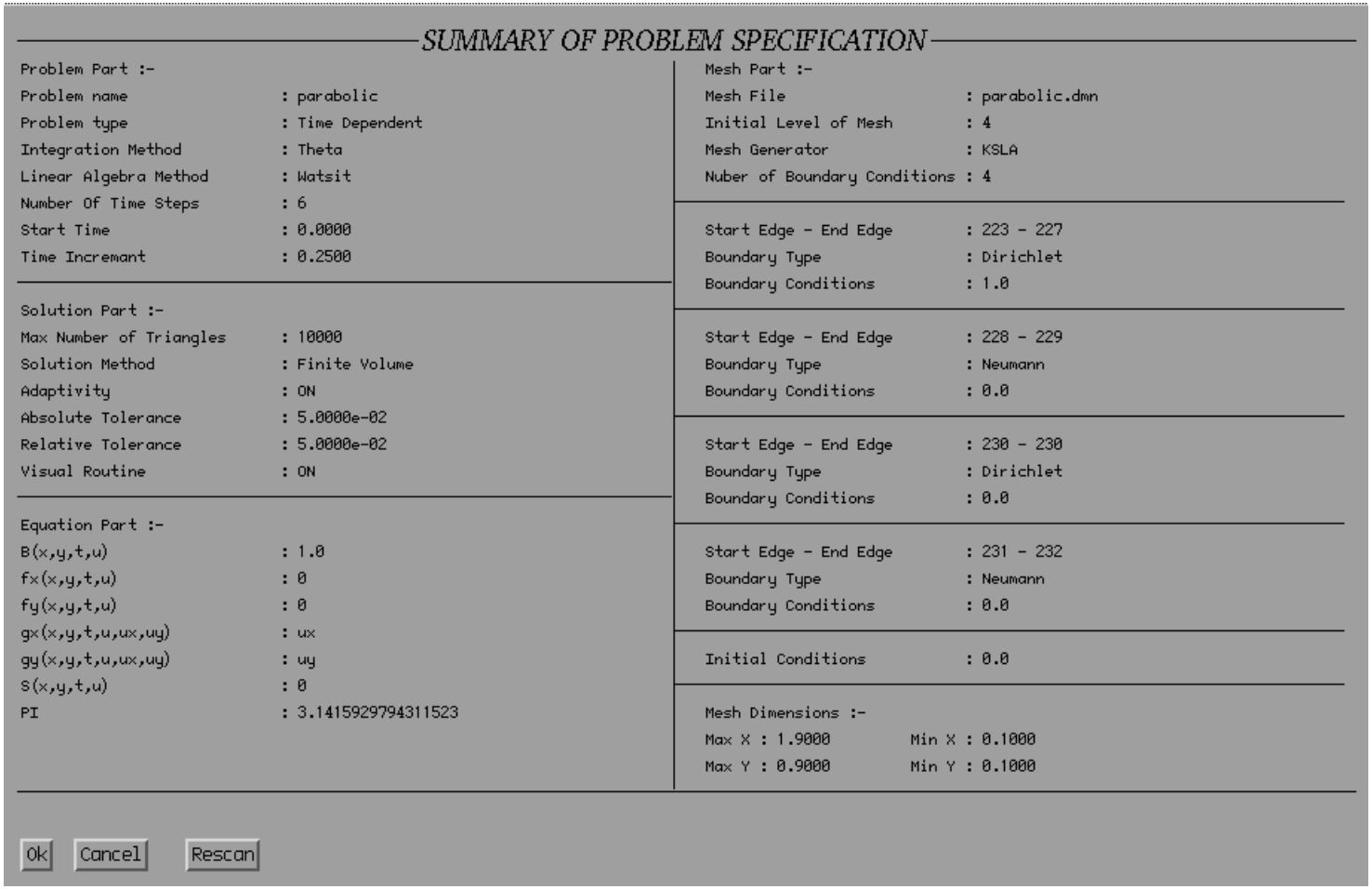

Figure C.1: Summary Window for Heat Problem

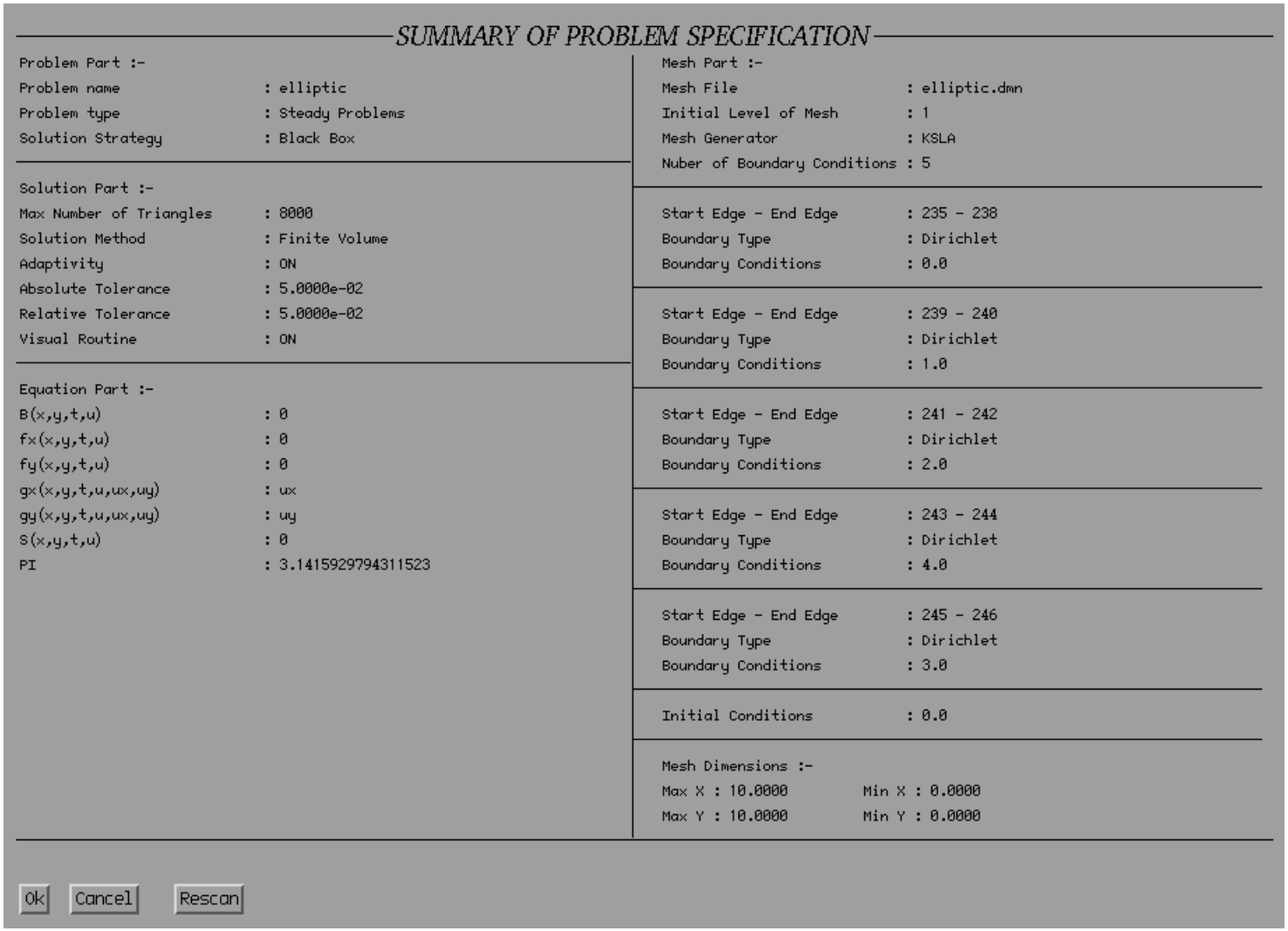

Figure C.2: Summary Window for Elliptic Problem

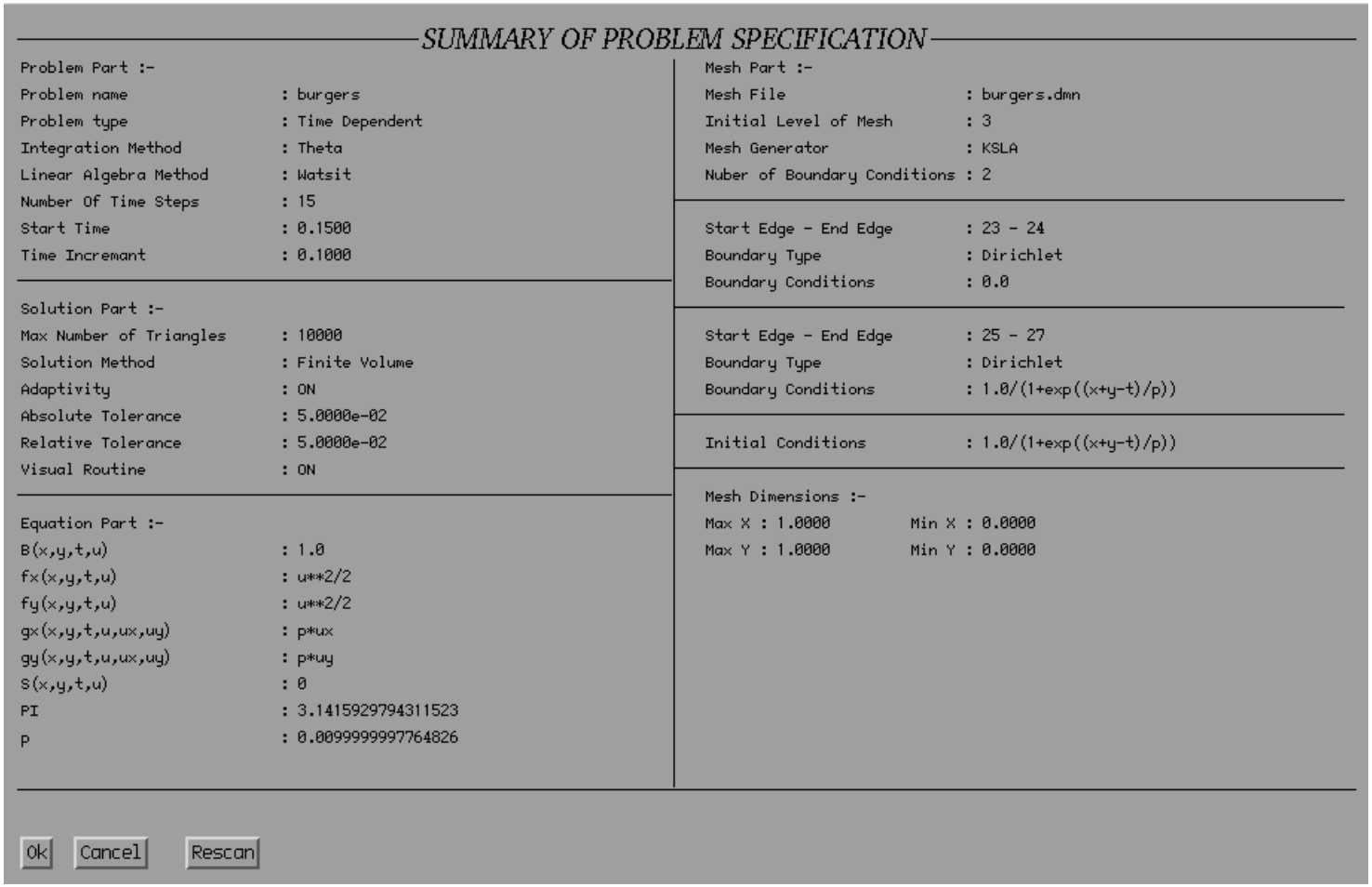

Figure C.3: Summary Window for Burgers' Problem

Appendix D

Interpolation Schemes for PDE problems – Related Work

D.1 Using Natural Logs in Interpolation

A further way in which the spurious oscillations that may occur in interpolation schemes can be eliminated is by confining the range of values the interpolant can produce. In the cases considered here the range is usually given by the data points used to define the interpolant. If all the data points are positive then this provides a logical lower bound of zero for the interpolant. If an interpolant is constructed that will always give a non–negative value then a lower limit of zero is enforced.

Such an interpolant can be defined by utilising the natural logarithm and exponential functions. This interpolant differs from the standard interpolant although the shape functions used in both cases are identical. The values, however, are not, in that the interpolant is constructed using the value given by the natural log of each data point value rather than the value itself. The value the interpolation process then yields will be the natural log of the value at any point in the domain considered. Taking the exponential of this value will give the solution value. For n data points each having a value f_i, $i = 1 \ldots n$, this process is; firstly ensure f_i is positive by some mapping function, then the interpolated value f^* is given by

$$f^* = \exp\left(\sum_{i=1}^{n} \phi_i \ln(f_i)\right)$$

where ϕ_i is the shape function associated with the value f_i at data point i, i.e. $\phi_i = 1$ at

f_i and $\phi_i = 0$ at f_j, $j \neq i$. The reverse mapping function is then applied to f^*

The advantage of this scheme is that, even if the standard interpolation of the log values gives a negative value, then the solution value for that point will still be positive.

The first step in the process is to ensure that all the data point values are positive. This is necessary to derive the natural log value. This will involve some form of mapping function. This mapping function is only used when negative data values are encountered. The purpose of the mapping is simply to ensure that all data points are positive. This is achieved by adding some constant, say k, to all data.

Let $f_{min} = \min(f_i)$, $i = 1 \ldots n$.
If $f_{min} < 0$ Then $f_i = f_i + k$, $i = 1 \ldots n$.

The reverse of the process is then to subtract this floor after the solution value has been found i.e. $f^* = f^* - k$. However, the choice of this value k does present a problem. The natural log interpolant will ensure that all data is positive, however, it is not an ideal situation if the data points used for the interpolant are large. For example, the data set considered here has a maximum value of ≈ 2.6 and a minimum value of ≈ -1.4. If the effect of the mapping is to change this range from $[-1.4 \ldots 2.6]$ to $[3.6 \ldots 7.6]$, $k = 5$ will yield poor results as left graph in figure D.1 shows. This is because, while the scheme will yield positive values, it is unlikely that these values will be close to zero. The advantage of placing a lower bound on the interpolated value is lost.

The idea is to utilise the ability of the scheme to place a lower bound on the interpolant values. To do this it would appear sensible to have the mapping transform the value of the lowest data point used to a value close to zero. In this case the interpolant is bounded by the value of the smallest data point. However, the phrase *close to zero* is unclear, in the example considered a value of 0.5 (right graph figure D.1) gives better results than the value of 5.0. A further reduction in the size of this value to 0.01 (left graph figure D.2) gives better results in some areas of the problem but not in others. However a further drop to say 0.0001 cause problems (right graph figure D.2). In the latter case the natural log value becomes too large, e.g. $\ln(0.01) = -4.6052$, $\ln(0.0001) = -9.2103$ compared to the range of values for $[\ln(0.25) \ldots \ln(2.5)]$, $[-1.3863 \ldots 0.9163]$, with the large difference between data point values the standard interpolation techniques become unstable.

All these examples fail to avoid overshoot, a correction for this deficiency is diffi-

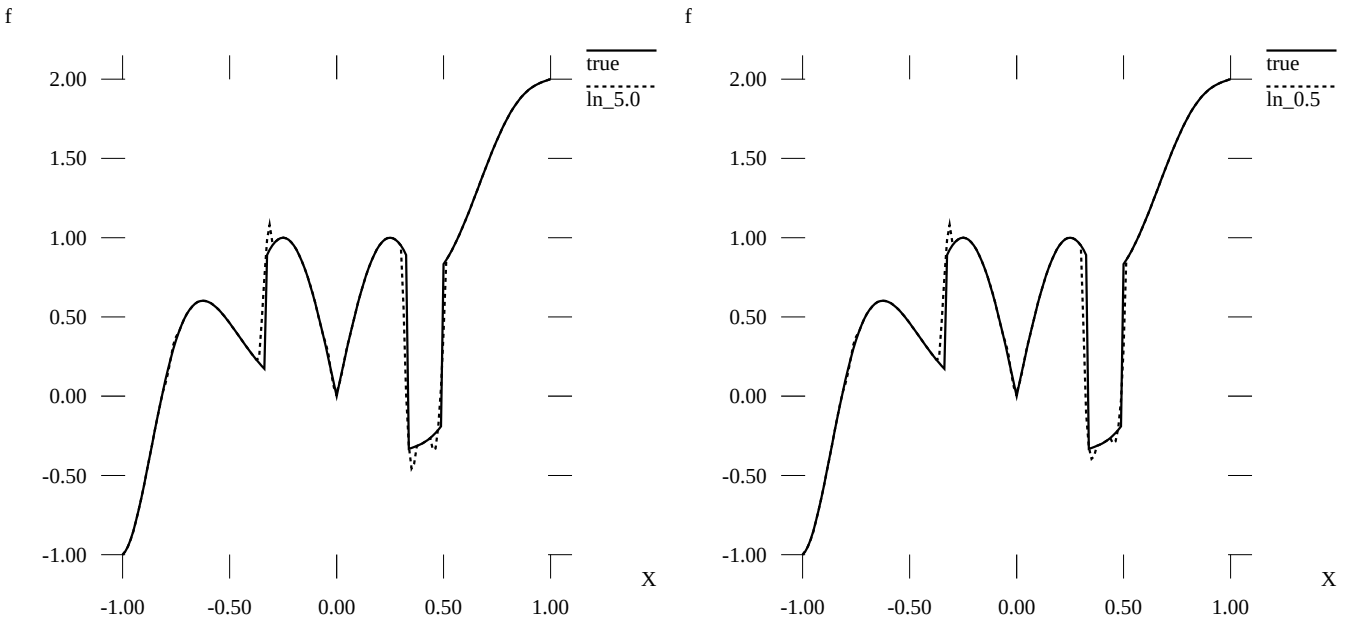

Figure D.1: Cross section at Y = 0.00 on a 1089 Node mesh with floor of 5.0 (left) and 0.5 (right)

cult to envisage. Another disadvantage of the scheme is the increase in c.p.u. time needed to implement, the table below shows the difference in time between this interpolation method and standard interpolation. The time measured is the c.p.u. time spent on calculating the L_1 norm over the domain using a seven point Gaussian quadrature scheme. A $[-1..1, -1..1]$ domain is considered.

No. of Δ's	ln Time	Std Time	ln Error	Std Error
162	0.21	0.09	8.127e-01	2.926e-01
1458	1.77	0.83	2.372e-01	9.192e-02
13122	16.03	8.25	6.749e-01	3.000e-02

As the table shows the c.p.u. time spent on the interpolation method using natural logs is approximately twice that of the standard quadratic interpolant, the errors are larger and undershoot and overshoot have not been eliminated.

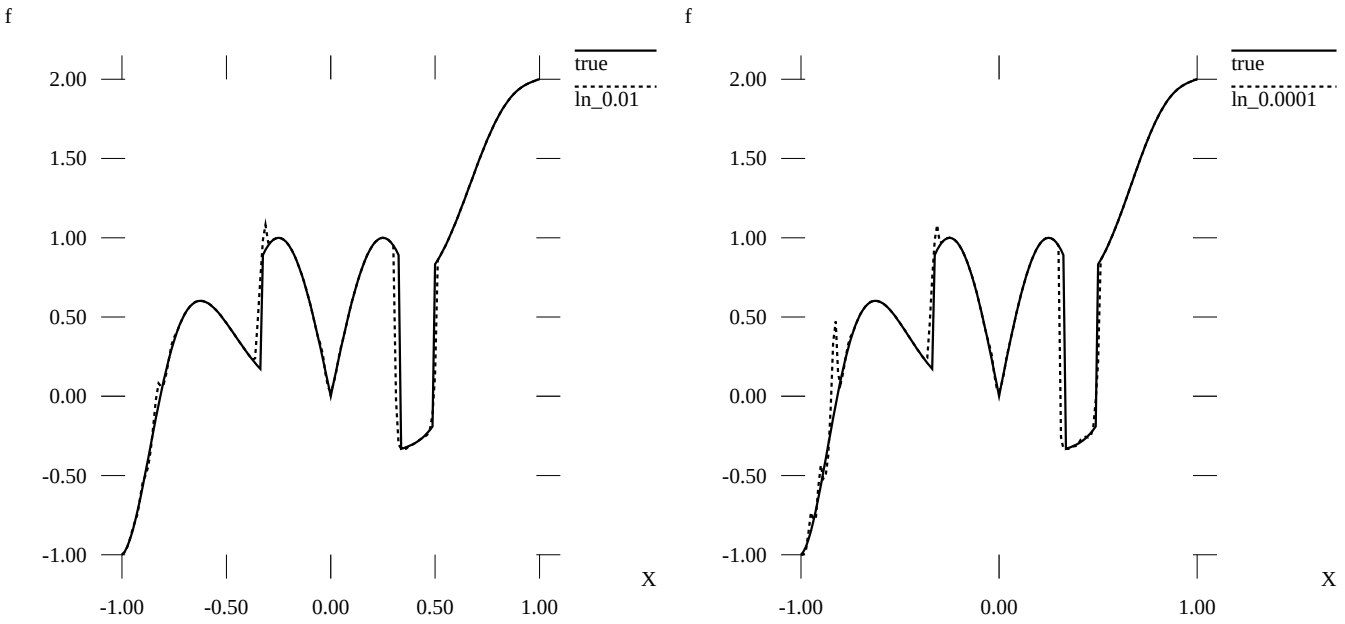

Figure D.2: Cross section at Y = 0.00 on a 1089 Node mesh with floor of 0.01 (left) and 0.0001 (right)

Bibliography

[1] R Abgrall. Design of an essentially non-oscillatory reconstruction procedure on finite-element type meshes. ICASE Report 91-84, December 1991. revised form of INRIA Report No. 1592 January 1992 Math. of Comput., submitted.

[2] C A Addison, W H Enright, P W Gaffney, I Gladwell, and P M Hanson. A decision tree for the numerical solution of initial value ordinary differential equations. *ACM Transactions on Mathematical Software*, 17(1):1–10, March 1983.

[3] S Adjerid, J E Flaherty, P K Moore, and Y J Wang. High-order adaptive methods for parabolic equations. In J M Hyman, editor, *Experimental Mathematics: Computational Issues in Non-Linear Science*, Physics D 60 1-4, pages 94–111. North-Holland, 1992. Proc of 11th Annual International Conference of the Center for Non-Linear studies, 1991.

[4] R E Bank. Frontiers in applied mathematics. In *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations Users' Guide 6.0*, volume 7. SIAM, 1990.

[5] P Baras, J Blum, J C Paumier, P Witomski, and F Rechenmann. EVE: An object-centered knowledge-based PDE solver. In E N Houstis, J R Rice, and R Vichnevetsky, editors, *Expert Systems for Scientific Computing*, pages 1–19. North-Holland, 1992.

[6] T G Barth and Venkatakrishnan V. Application of a direct solver to unstructured meshes for Euler and Navier-Stokes equations, 1989. AIAA Paper No 89-0364.

[7] J L Bentley, M F Fernandez, B W Kernighan, and N L Schryer. Template-driven interfaces for numerical subroutines. *ACM Transactions on Mathematical Software*, 19(3):265–287, September 1993.

[8] M Berzins. Temporal error control for convection-dominated equations in two space dimensions. *SIAM J. Sci. Comput.*, May 1995.

[9] M Berzins and R M Furzeland. A user's manual for SPRINT- a versatile software package for solving systems of algebraic, ordinary and partial differential equations: Part 1 - algebraic and ordinary differential equations. Technical Report TNER.85.058, Thornton Research Centre, Shell Research Limited, 1985.

[10] M Berzins, P H Gashell, A Sleigh, W Spears, A Tomlin, and J M Ware. An adaptive CFD solver for time dependent environmental flow problems. To Appear in Proc. of IFCD Conference on Numerical Methods for Fluid Dynamics, Oxford Universities Press.

[11] M Berzins, J Lawson, and J M Ware. Spatial and temporal error control in the adaptive solution of systems of conservation laws. In *Proceedings IMACS PDE7: Seventh IMACS International Conference on Computer Methods for Partial Differential Equations*, Rutgers University NJ, 1992.

[12] M Berzins and J M Ware. Reliable finite volume methods for Navier-Stokes equations. In F K Hebeker, R Rannacher, and G Wittum, editors, *Numerical Methods for the Navier-Stokes Equations*, pages 1–8. Viewig, 1994.

[13] M Berzins and J M Ware. Positive discretisation methods for hyperbolic equations. *Applied Numerical Mathematics*, 16:417–438, 1995.

[14] M Berzins and J M Ware. Solving convection and convection reaction problems using the M.O.L. *Submitted to Applied Numerical Mathematics*, 1995. Expanded version of a talk presented at the M.O.L workshop Lexington, Kentucky.

[15] J P Bonomo and W R Dyksen. XELLPACK: An interactive problem-solving environment for elliptic partial differential equations. In E N Houstis et al, editor, *Intelligent Mathematical Software Systems*, pages 331–341. Elsevier Science Publishers B V (North-Holland), 1990.

[16] T I Boubez, A M Froncioni, and R L Peskin. A prototyping environment for differential equations. In E N Houstis, J R Rice, and R Vichnevetsky, editors, *Expert Systems for Scientific Computing*, pages 55–68. North-Holland, 1992.

[17] R W Brankin and I Gladwell. Shape-preserving local interpolation for plotting solutions of odes. *IMA J. Numer. Anal.*, 9:555–566, 1989.

[18] Su Bu-Quing and Lin Dind-Yuan. *Computational Geometry Curve and Surface Modeling*. Academic Press Inc., London, 1989.

[19] R L Burden, J D Faires, and A C Reynolds. *Numerical Analysis*. Weber & Schnidt, 2nd edition, 1981.

[20] S Butt and K W Brodlie. Preserving positivity using piecewise cubic interpolation. *Comput. & Graphics*, 17:55–64, 1993.

[21] A F Càrdenas and W J Karplus. PDEL a language for partial differential equations. *Communications of the ACM*, 13(3):184–191, March 1970.

[22] B W Char et al. *MAPLE Reference Manual*. WATCOM Publications Ltd, Waterloo, Canada, 5th edition, 1988.

[23] G R Cowper. Gaussian quadrature formulas for triangles. *Int. Journal Num. Methods in Fluids*, 7:405–408, 1973.

[24] C W Cryer. Expert systems for numerical software. In J C Mason and M G Cox, editors, *Scientific Software Systems*. Chapman and Hall, 1990.

[25] J H Davenport. Current problems in computer algebra systems design. In Miola A, editor, *Proc DISCO*, Springer Lecture Notes in Computer Science 429, pages 1–9, 1990.

[26] J H Davenport. The AXIOM system. Technical Report TR5/92, NAG Ltd, Oxford, 1992.

[27] K D Devine and J E Flaherty. Parallel, high-order finite element methods for conservation laws. In R Vichnevetsky, D Knight, and G Richter, editors, *Advances in Computer Methods for Partial Differential Equations VII*, pages 202–208. IMACS, 1992.

[28] M Dewar. The AXIOM system. Private Communication.

[29] L J Durlofsky, B Enquist, and S Osher. Triangle based adaptive stencils for the solution of hyperbolic conservation laws. *J. Comput. Phys.*, 98:64–73, 1992.

[30] W R Dyksen and C R Gritter. Elliptic expert: An expert system for elliptic partial differential equations. In E N Houstis et al, editor, *Intelligent Mathematical Software Systems*, pages 331–341. Elsevier Science Publishers B V (North-Holland), 1990.

[31] W R Dyksen and C J Ribbens. Interactive ELLPACK: An interactive problem-solving environment for elliptic partial differential equations. *ACM Transactions on Mathematical Software*, 13(2):113–132, June 1987.

[32] B Enquist and T Smedsaas. Automatic computer code generation for hyperbolic and parabolic differential equations. *SIAM J. Sci. Stat. Comput.*, 1(2):249–259, June 1980.

[33] J E Flaherty, M Benantar, R Biswas, and P K Moore. Symbolic and parallel adaptive methods for partial differential equations. In B R Donld, D Kapur, and J L Mundy, editors, *Symbolic and Numerical Computations for Artificial Intelligence*. Academic Press, 1992.

[34] B Ford and R M Iles. The what and why of problem solving environments for scientific computing. In B Ford and F Chatelin, editors, *Problem Solving Environments for Scientific Computing*. North-Holland, 1987.

[35] R M Furzeland, P C Rem, and R F Van der Wijngaart. General purpose software for multi-dimensional partial differential equations. Technical report, KSLA, Shell Research Amsterdam, 1989.

[36] Barth T G. On unstructured grids and solvers. Technical report, von Karman Institute for Fluid Dynamics, 1990. Lecture Series 1990-03.

[37] P W Gaffney, C A Addison, B Andersen, S Bjørnestad, R E England, P M Hanson, R Pickering, and M G Thomason. NEXUS towards a problem solving environment (PSE) for scientific computing. *ACM SIGNUM Newsletter*, 12:13–24, 1986.

[38] P W Gaffney, J W Wooten, K A Kessel, and W R McKinney. Algorithm 606 NITPACK: An interactive tree package. *ACM Transactions on Mathematical Software*, 9(4):418–426, December 1983.

[39] P W Gaffney, J W Wooten, K A Kessel, and W R McKinney. NITPACK: An interactive tree package. *ACM Transactions on Mathematical Software*, 9(4):395–417, December 1983.

[40] R H Gallagher. *Finite Element Analysis Fundamentals*. Prentice-Hall Inc, London, 1975.

[41] E Gallopoulos, R Houstis, and Rice J R. Future directions in problem solving environments for computational science. Technical report, Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry and Artificial Intelligence for Computational Science, Washington, D.C., April 1991.

[42] M Garbey, H G Kaper, and M K Kwong. Symbolic manipulation software and the study of differential equations. In H G Kaper and M Garbey, editors, *Asymptotic Analysis and the Numerical Solution of Partial Differential Equations*, volume 130 of *Lecture Notes in Pure and Applied Mathematics*. Marcel Dekker Inc., 1991.

[43] D Harper, C Wooff, and D Hodgkinson. *A Guide to Computer Algebra Systems*. Wiley, 1991.

[44] A C Hearn. *REDUCE user's manual*. Rand Publishing, 1987. RAND publication CP78.

[45] D J Higham. Monotonic piecewise cubic interpolation with applications to ode plotting. *J. Comp. Appl. Math.*, 39:287–294, 1992.

[46] C E Houstis, E N Houstis, J R Rice, P Varodoglou, and T S Papatheodorou. Athena: A knowledge base system for //ELLPACK. In E Diday and Y Lechevallier, editors, *Symbolic-Numeric Data Analysis and Learning*. Nova Science Publishers, 1991.

[47] E N Houstis and J R Rice. Parallel (//) ELLPACK: An expert system for parallel processing of partial differential equations. In E N Houstis et al, editor, *Intelligent Mathematical Software Systems*, pages 331–341. Elsevier Science Publishers B V (North-Holland), 1990.

[48] E N Houstis and J R Rice. The architecture of PDE solving systems. Technical Report CSD-TR-92-022, Computer Science Dept. Purdue University, Apr 1992.

[49] E N Houstis and J R Rice. Parallel ELLPACK: A development and problem solving environment for high performance computing machines. In P W Gaffney and E N Houstis, editors, *Programming Environments for High-Level Scientific Problem Solving*, IFIP, pages 229–241. Elsevier Science Publishers B.V., 1992.

[50] R D Jenks and R S Sutor. *AXIOM The Scientific Computational System.* Springer-Verlag, 1992.

[51] B Joe. GEOMPACK - a software package for the generation of meshes using geometric algorithms. *Adv. Eng. Software*, 13(5/6):325–331, 1991.

[52] B Joe. GEOMPACK users guide. Technical report, Dept of Computer Science, University of Alberta, Alberta, Canada T6G 2H1, 1993.

[53] C Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method.* Cambridge University Press, 1987.

[54] O Jones. *Introduction to the X Window System.* Prentice-Hall, 1989.

[55] M S Kamel, K S Ma, and W H Enright. ODEXPERT an expert system to select numerical solvers for initial value ODE systems. In E N Houstis, J R Rice, and R Vichnevetsky, editors, *Expert Systems for Scientific Computing*, pages 33–54. North-Holland, 1992.

[56] E Kant, F Daube, W MacGregor, and J Wald. Knowledge-based program generation for mathematical modeling. In E N Houstis, J R Rice, and R Vichnevetsky, editors, *Expert Systems for Scientific Computing*, pages 371–392. North-Holland, 1992.

[57] B W Kernighan. PIC – a graphical language for typesetting users manual. Technical Report 85, Bell Laboratories, 1982.

[58] B W Kernighan and L L Cherry. Eqn - typesetting mathematics - users' guide. Bell labs internal report.

[59] C Konno, Y Umetani, M Igai, and T Ohta. Interactive/visual DEQSOL: Interactive creation, debugging diagnosis and visualization of numerical simulation. In E N Houstis et al, editor, *Intelligent Mathematical Software Systems*, pages 301–317. Elsevier Science Publishers B V (North-Holland), 1990.

[60] Laboratory for Computer Science The MATHLAB Group, M.I.T. Cambridge MA. *MACSYMA Reference Manual*, 1977. Version 9.

[61] L Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, 1985.

[62] M E Lesk. Tbl – a program to format tables. UNIX Programmer's Manual, 1979.

[63] S Y Lin, T M Wu, and Y S Chin. Upwind finite-volume method with a triangular mesh for conservation laws. *J. Comput. Phys.*, 107:324–337, 1993.

[64] M Machura. Problem solving environments for partial differential equations: The users perspective. In B Enquist and T Smedsaas, editors, *PDE Software: Modules, Interfaces and Systems*, pages 171–191. Elsevier Science Publishers, 1984.

[65] P K Moore, C Ozturan, and J E Flaherty. Towards the automatic numerical solution of partial differential equations. Technical Report 88-28, Department of Computer Science Renesselaer Polytechnic Institute, Troy New York 12180-3590, December 1988.

[66] J F Ossanna. Nroff/troff user's manual. UNIX Programmer's Manual, 1976.

[67] S V Pennington and M Berzins. New NAG library software for first-order partial differential equations. *ACM Transactions on Mathematical Software*, 20(1):63–99, March 1994.

[68] R L Peskin, S S Walther, and A M Froncioni. Smalltalk the next generation scientific computing interface. *Mathematics and Computers in Simulation*, 31:371–381, 1989.

[69] C D Peterson. *Athena Widget Set – C Language Interface.* MIT X Consortium, Cambridge, Massachusetts, X11 R5 edition, 1991. X Window System.

[70] L R Petzold. A description of DASSL: a differential algebraic system solver. Technical report, Applied Maths Division 8331, Sandia National Laboratories, Livermore, California, 1982.

[71] J R Rice. Software parts for elliptic PDE software. In B Enquist and T Smedsaas, editors, *PDE Software: Modules Interfaces and Systems.* Elsevier Publishers, 1984.

[72] J R Rice and R F Boisvert. *Solving Elliptic Problems Using ELLPACK.* Springer-Verlag, New York, 1985.

[73] D M Ritchie and B W Kernigham. *The C Programming Language*. Prentice-Hall, 1978.

[74] P L Roe. Finite-volume methods for compressible Navier-Stokes equations. In C Taylor, W G Habashi, and M M Hafez, editors, *Fifth International Conference on Laminar Turbulent Flow*, pages 2088–2101, Montreal, 1988. Pineridge Press.

[75] L E Scales. NAESOL: User's guide. Internal report, Shell Research Ltd, Chester, 1993.

[76] M S Shephard and P M Finnigan. Towards automatic model generation. In *State-of-the-Art Surveys on Computational Mechanics*, chapter 11, pages 335–366. ASME, 1988.

[77] Silicon Graphics Computer Systems, Mountain View, California. *IRIS Explorer User's Guide*, 1993. Document number 007-1369030.

[78] S Steinberg and P J Roache. Using VAXIMA to write FORTRAN code. In R Pavelle, editor, *Applications of Computer Algebra*, pages 74–94. Kluwer, 1985.

[79] S Steinberg and P J Roache. Using MACSYMA to write FORTRAN subroutines. *Journal of Symbolic Computation*, 2:213–216, 1986.

[80] S Steinberg and P J Roache. Automatic generation of finite difference code. In H H Bau, T Herbert, and MM Yovanovich, editors, *Proc. Symbolic Computation in Fluid Mechanics and Heat Transfer*, 1988.

[81] S Steinberg and P J Roache. A toolkit of symbol manipulation programs for variational grid generation. In J S Kowalik and C T Kitzmiller, editors, *Coupling Symbolic and Numerical Computing in Expert Systems II*, pages 103–116. North-Holland, 1988.

[82] H J Stetter. Tools for scientific comptation. *Zeitschrift für Angewandte Mathematik und Mechanik ZAMM*, 73(12):335–348, 1993.

[83] B Tuthill. Refer – a bibliography system. Bell Laboratories, 1988.

[84] Y Umetani. Programming ELLPACK examples in PDEQSOL. Technical Report CSD-TR-92-080, Computer Science Dept. Purdue University, Oct 1192.

[85] Y Umetani. The visual diagnosis on the numerical calculation of PDE problems. Technical Report CSD-TR-92-082, Computer Science Dept. Purdue University, Oct 1992.

[86] Y Umetani, C Konno, and T Ohta. Visual PDEQSOL: A visual and interactive environment for numerical simulation. In P W Gaffney and E N Houstis, editors, *Programming Environments for High-Level Scientific Problem Solving*, IFIP. Elsevier Science Publishers B.V., 1992.

[87] J M Ware. *The Adaptive Solution of Time-Dependent Partial Differential Equations in Two Space Dimensions.* PhD thesis, School of Computer Studies University of Leeds, 1993.

[88] J M Ware and M Berzins. Finite volume techniques for time-dependent fluid-flow problems. In *Proceedings IMACS PDE7: Seventh IMACS International Conference on Computer Methods for Partial Differential Equations*, Rutgers University NJ, 1992.

[89] S Weerawarana, A C Catlin, E N Houstis, and J R Rice. Integrating symbolic-numeric computing in //ELLPACK: Experience and plans. Technical Report CSD-TR-92-092, Computer Science Dept. Purdue University, March 1992. Revised 11/92.

[90] S Weerawarana, E N Houstis, and J R Rice. A software platform for the integrating symbolic computation with a PDE solving environment. Technical Report CSD-TR-94-031, Computer Science Dept. Purdue University, May 1994.

[91] L Williams, M Heinrich, C Lohnes, and K Walsh. *Graphics Library Reference Manual (C Edition).* Silicon Graphics, Inc., Mountain View, California, 4.0 edition, 1990.

[92] S Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, 2nd edition, 1991.

[93] C D Wooff and D E Hodgkinson. *muMATH: A Microcomputer Algebra System.* Academic Press, 1987.

[94] D A Young. *The X Window System Programming and Applications with Xt OSF/MOTIF Edition.* Prentice-Hall, 1990.

[95] P A Zegeling. A note on the grid movement induced by m.f.e. method. Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, 1989.

[96] O C Zienkiewicz. *The Finite Element Method.* McGraw-Hill Book Company, London, 1977.