Mesh generation and visualisation tools for computational fluid dynamics

Michael G. Sarantinos

Thesis submitted for the degree of
Doctor of Philosophy
of the University of London
and the Diploma of Membership of Imperial College

Imperial College of Science, Technology, and Medicine
Mechanical Engineering Department
Exhibition Road, London SW7 2BX

June 1994

BIBL
LONDON
UNIV

Volume 2 is a video
Please apply to
university

Blank

In

Original

To my parents

Στους γονεις μου

Blank

In

Original

Abstract

This study describes the assembly and application of a set of interactive tools for mesh generation and results visualisation, for use with general-purpose computational fluid dynamics codes.

The tools are demonstrated through application in conjunction with the SPEED code, used for simulation of gas motion, fuel injection, ignition, and combustion in reciprocating internal combustion engines. Its use of a CFD methodology embodying unstructured computational meshes with hexahedral cells permits simulation of real engine geometries with uniform meshes. A set of solid modelling, mesh generation, visualisation, animation, and user interface utensils for interaction with SPEED are first presented in the form of algorithms or functions performed with graphics systems. They are then assembled for use with SPEED in two different ways: using a CAD system and using an animation system. In both cases all numerical tasks, definition of engine operation conditions, definition of initial and boundary conditions, checking, and manipulation of data prior to visualisation are accomplished by means of specially constructed pre- and post-processors.

The effectiveness of the proposed system for the definition of the surfaces defining the combustion chamber, generation of uniform, unstructured meshes of hexahedral-cells, and definition of the engine operation conditions and initial and boundary conditions for the simulation is demonstrated in the case of two direct–injection Diesel engines of different combustion chamber configuration. The wide range of visualisation utilities assembled enable interpretation of the SPEED–calculated gas flow patterns, fuel spray behaviour, and combustion development. These include the formation of swirl and squish, injected fuel atomisation and vaporisation, and auto-ignition and flame propagation. These predictions, together with calculations of integral properties characterising engine performance are shown to be in agreement with experimental measurements and the manufacturers' specifications.

Animations of SPEED–calculated air flow and spray results in the chamber of an indirect injection Diesel engine that are accomplished by means of the proposed system demonstrate that dynamic visualisations can enhance understanding of the evolution of in–cylinder phenomena in space and time.

Blank

In

Original

Acknowledgements

This thesis represents the main result of the research which I carried out from 1987 till 1992 as part of the IDEA project of the European Commission, a collaborative project between Fiat, PSA, Renault, Rover, Volkswagen, and Volvo, on the one side, and Imperial College and other research institutions on the other.

My grateful thanks go to my supervisor Prof. A. David Gosman, for coupling expert guidance with a judicious blend of constructive criticism and encouragement throughout.

It has been a priviledge to have worked with Dr. Chris Marooney, whose knowledge and invariable willingness to help was so important in reaching both understanding of the SPEED code and defining the pre– and post–processing needs thereof. Charles Hill and Panos Theodossopoulos have provided particularly useful comments and thought–provocing discussion. I wish to express my appreciation to my colleagues of the SPEED group at Imperial College – in particular Nader Fateh, Henry Weller, Cedomir Kralj, Theodore Theodoropoulos, and Dejan Leskovsek – for enduring my questions and their constant support. Also to Frank Lange and Tim Strauss of Volkswagen AG. Hvala, Danke, Ευχαριστω, Thank you.

Dr. Schweimer and Dr. Schindler of Volkswagen AG and Dr. Zellat of Renault Automobiles have provided constant feedback on the user needs. I would like to thank the former for giving me the opportunity to use the SPEED code in the demanding research and development environment of Volkswagen AG.

Last but not least I would like to thank Nicky Scott-Knight for her continuous encouragement and support.

Table of contents

List of Figures 14
List of Tables 23
List of Symbols 24
Abbreviations 26

1 Introduction 28
1.1 A model of the CFD Engine Simulation Process. 29
1.2 The structure of this thesis. 32

2 Background 34
2.1 Overview of computer systems for CFD engine simulation. 34
2.1.1 Computational methodologies. 34
2.1.1.1 The general transport equation for turbulent flows 35
2.1.1.2 Turbulence combustion modelling 36
2.1.1.3 Software systems for CFD engine simulation 38
2.2 Model building methodologies. 39
2.2.1 Mesh types. 39
2.2.2 Overview of surface modelling techniques. 46
2.2.3 Overview of mesh-generation procedures 47
2.3 Scientific Visualization techniques 50
2.3.1 Overview of visualisation applications 50
2.3.2 Overview of software systems for visualisation of CFD data. 53
2.4 Computer software and hardware for animation 54
2.4.1 Applications of animation to scientific visualisation. 55
2.4.2 Overview of animation languages. 56
2.4.3 Main techniques for animation control. 57
2.4.5 Hardware for animation and video production 58
2.5 User interaction with the simulation process 60
2.5.1 Distributed simulation and visualisation. 62

3 Reciprocating engine analysis and the SPEED code 64
3.1 Engine geometry and operation 64

3.1.1 Engine types....64
3.1.2 Geometric features....67
3.1.3 Charge motion and combustion within the cylinder....67
3.1.4 Intake stroke....68
3.1.5 Compression stroke....72
3.1.6 Combustion stroke....73
3.1.7 Exhaust stroke....74
3.2 Engine flow prediction with the SPEED code....74
3.2.1 Overview of SPEED....75
3.2.2 The SPEED methodology....75
3.2.2.1 Transport equations....75
3.2.2.2 Fuel injection and Diesel combustion modelling....76
3.2.2.3 Handling of boundaries....78
3.2.2.3.1 Inlet or outlet flow boundaries....79
3.2.2.3.2 Wall boundaries....82
3.2.3 Numerical solution....83
3.2.3.1 Discretisation....83
3.2.3.2 Computational mesh....85
3.2.3.2.1 Mesh-addressing conventions....85
3.2.3.2.2 Mesh adaption to boundary change....86
3.2.3.2.3 Mesh-quality characteristics....87
3.2.3.2.4 Mesh generation....89
3.2.3.3 Solution procedure....89

4 Software tools for interaction with the CFD engine simulation process....96
4.1 Geometry definition tools....97
4.2 Mesh generation tools....98
4.3 Visualisation tools....102
4.3.1 Data manipulation tools....104
4.3.1.1 Data contraction....104
4.3.1.2 Interpolation to vertices....106
4.3.1.3 Normalisation....106
4.3.1.4 Filtering....107
4.3.1.5 Mesh rezoning....109
4.3.2 Graphical mapping tools....110

4.3.2.1 Colour mapping 110
4.3.2.2 Computer graphics tools 110
4.3.2.3 Sampling planes 112
4.3.2.4 Cutting planes 113
4.3.3 Tools for visualisation of scalar fields 113
4.3.3.1 One-dimensional tools 113
4.3.3.2 Two-dimensional tools 114
4.3.3.3 Three-dimensional tools 116
4.3.4 Tools for visualisation of vector fields 119
4.3.4.1 Direct vector tools 120
4.3.4.2 Indirect vector field visualisation tools 122
4.4 Graphical user interface tools 122
4.5 Software Engineering Tools 125
4.5.1 Programming tools 125
4.5.2 Data base management tools 125

5 A pre- and post-processing system 128
5.1 Design goals 129
5.2 System structure and user interface 130
5.2.1 System architecture 132
5.2.2 Data structures 134
5.2.3 User interface 136
5.3 Geometry modelling 137
5.4 Mesh generation 140
5.4.1 Segmentation into blocks 141
5.4.2 Mesh construction in each block 143
5.4.2.1 Mesh generation with PATRAN 148
5.4.2.1.1 A semi-automatic mesh-generation program 149
5.4.2.2 Mesh generation with SUPERTAB 150
5.4.3 Definition of boundaries 151
5.4.4 Mesh continuity across block boundaries 153
5.4.5 Geometry model assembly 153
5.4.6 Data manipulation 153
5.4.7 Mesh quality checking 154
5.5 Problem setup 156

5.5.1 The problem file 157
5.5.1.1 Engine operation conditions 157
5.5.1.2 Initial conditions 158
5.5.1.3 Boundary conditions 159
5.5.1.4 Calculation control data 160
5.5.1.5 Fuel injection parameters 160
5.5.1.6 Fuel and combustion model parameters 161
5.5.1.7 Layer addition and removal control parameters 162
5.6 Visualisation 162
5.6.2 Alternative visualisation approaches 164
5.7 Examples 165

6 Dynamic visualisation of SPEED results 180
6.1 Advanced visualisation requirements in SPEED 180
6.2 Software systems for dynamic visualisation 184
6.3 Hardware-imposed approaches to dynamic visualization 185
6.4 Bridging the gap between workstations and video equipment 188
6.4.1 Incompatibilities 189
6.4.2 Bridging the incompatibilities and recording onto a VTR 191
6.4.2.1 Choosing the recording format 191
6.5 An animation system for SPEED 193
6.5.1 Description of IL 193
6.5.2 Description of the AERAS animation system 198
6.5.2.1 Design goals 198
6.5.2.2 Architecture of the AREAS system 199
6.5.2.3 Detailed design of AERAS 200
6.5.2.3.1 Creation of geometry 200
6.5.2.3.2 Data manipulation and visualisation mapping operations 202
6.5.2.3.3 Animation 208
6.5.2.3.4 Recording onto the output device 209
6.5.2.4 Implementation of the AERAS system 211

7 Testcases 218
7.1 Simulation of the VW IDEA 1.9L DI transparent Diesel engine 219
7.1.1 Mesh generation 219

7.1.1.1 Mesh design 219
7.1.1.2 Geometry definition 221
7.1.1.3 Mesh generation. 221
7.1.1.4 Mesh translation and checking 222
7.1.2 Setup of initial and boundary conditions 223
7.1.3 Simulation 225
7.1.4 Interpretation and visualisation of results 226
7.2 Simulation of the VW IDEA 1.9l DI series Diesel engine 268
7.2.1 Mesh design, geometry definition, and mesh generation 268
7.2.2 Setup of initial and boundary conditions 269
7.2.3 Simulation 269
7.2.4 Interpretation and visualisation of results 270
7.3 Dynamic visualisation of results in a Renault IDI Diesel engine 289
7.3.1 Mesh generation 289
7.3.2 Setup of the simulation 292
7.3.3 Analysis with SPEED 293
7.3.4 Dynamic visualisation 294
7.3.4.1 Animation design 294
7.3.4.2 Results interpretation and storyboard of the animation 295
7.3.4.3 Production of the animation with the AERAS system 305
7.3.4.3.1 Creation of geometry objects with the AERAS system 305
7.3.4.3.2 Creation of spray visualisations 305
7.3.4.3.3 Creation of gas flow visualisations 306
7.3.4.3.4 Creation of animation scripts 306
7.3.4.3.5 Lighting and raster tests 307
7.3.4.3.6 Motion development, wireframe and raster flipbooks 307
7.3.4.3.7 Rendering of the animation 308
7.3.4.3.8 Recording of the animation & hardware implementation 309
7.3.4.3.9 Time and disk-space requirements 309
7.3.4.4 Problems encountered 310

8 Conclusions 312
8.1 Achievements and findings 312
8.2 Suggestions for further work 318

Appendices 320
Appendix A 320
Appendix B 327
Appendix C 334
Appendix D 337

Bibliography 342

List of figures

1. Introduction

Fig. No.	Description	Page
1.1	The three phases of the CFD simulation process	30

2. Background

Fig. No.	Description	Page
2.1	Three different types of computational cells	39
2.2	Various structured meshes with hexahedral cells	41
2.3	Various block-structured meshes with hexahedral cells	42
2.4	An unstructured mesh with hexahedral cells for an axisymmetric disc chamber	43
2.5	Transformation from physical to computational domain and reverse transformation	44
2.6	Structured mesh for DI Diesel engine	44
2.7	Structured mesh for DI Diesel engine	44
2.8	Structured mesh for an IDI Diesel engine	45
2.9	Diagram of evolution of animation hardware	58
2.10	Interaction with the simulation: post-processing	60
2.11	Interaction with the simulation: tracking	61
2.12	Interaction with the simulation: steering	62

3. Reciprocating engine analysis and the SPEED code

Fig. No.	Description	Page
3.1	Figurative representation of four types of spark-ignition engines	65
3.2	Three commonly used compression-ignition engine configurations	66
3.3	Indirect injection Diesel engines with (a) swirl and (b) prechamber	66
3.4	Sketches of inlet and outlet flow	69
3.5	Sketch of in-cylinder flow pattern	69
3.6	Rotated flow patterns generated within the cylinder by the intake jet	71
3.7	Sketch of the squish-flow pattern for a DI Diesel engine with omega-shaped bowl in piston	72

3.8	Cylindrical coordinate system of the valve curtain, and definition of the fluid velocity vector **u**, axial flow angle α, and swirl flow angle β at a point on the curtain.	81
3.9	Definition of the geometric weighting factor for cell P	84
3.10	A SPEED mesh for a direct-injection Diesel engine with bowl-in-piston.	88
3.11	A polar mesh for an idealised disc-chamber engine, with one valve.	92
3.12	A bipolar mesh for a disc-chamber engine, featuring one off-centered valve.	92
3.13	A mesh for the combustion chamber of a FIAT direct-injection Diesel engine featuring an offset valve and a lipped bowl.	93
3.14	A SPEED mesh for the combustion chamber of a Renault indirect injection Diesel engine featuring a swirl chamber, glow plug, and a bowl-in-piston.	94

4. Software tools for interaction with the CFD engine simulation process

Fig. No.	Description	Page
4.1	Representation of a simple engine geometry with closed valve and offset square piston (a) CSG (b) B-rep (c) ASM	97
4.2	Example of effect of boundary node spacing on the location of interior nodes (a) Laplacian scheme (b) Isoparametric grid.	99
4.3	Transformation from the physical to the computational domain, at which the mesh is generated, and inverse transformation	101
4.4	Several refinement patterns that are allowed by the mapping isoparametric mesh generation method.	101
4.5	Four hexahedra derived from a tetrahedron	102
4.6	The visualisation pipeline	103
4.7	The visualisation pipeline, showing the different stages in the pipeline covered by visualisation tools.	104
4.8	Sampling out of the data base of those data that is totally included within a rectilinear volume.	105
4.9	Interpolation of data from a fine irregular grid onto a uniform rectilinear of polytriangle-strips grid.	109

4.10	Illustration of the slicing-plane and cutting-plane idioms.	112
4.11	Line-contour plot of the swirl component of mean velocity (taken from Begleris [43])	114
4.12	Different types of contour plots on cutting planes through the combustion chamber.	115
4.13	Flow-velocity vectors and pressure isosurface in the prechamber of an IDI engine (taken from Hill [192])	116
4.14	Multiple cutting planes in the combustion chamber of an IDI engine.	118
4.15	A carpet plot of the swirl component of flow velocity (TDC of compression) at a cutting plane through the swirl chamber of an IDI engine.	119
4.16	Alternative representations of an arrow, used for the visualisations of vector fields.	120
4.17	Vectors display in the prechamber of an indirect injection Diesel engine.	121
4.18	Vectors display in the prechamber of the above indirect injection Diesel engine. (a) cutting plane through the symmetry plane of the prechamber (b) horizontal cutting plane through the prechamber	121
4.19	Streamlines in the combustion chamber of an IDI engine, 10° CA ATDC of compression (taken from Hill [192])	122

5. A pre- and post-processing system for SPEED

Fig. No.	Description	Page
5.1	The two alternative approaches to the interface of the simulation process. (a) single and (b) multiple user interface and database approach	130
5.2	Illustration of the system for interaction with the CFD engine simulation process with SPEED.	132
5.3	Face directions for a SPEED cell and indirect addressing of boundary nodes.	135
5.4	Solid model for the combustion chamber of a DI Diesel engine with bowl-in-piston and open intake valve.	139
5.5	Solid model for the combustion chamber of a pent-roofed petrol engine with four valves, of which two are open.	139
5.6	Data flow diagram of the solid modelling and mesh generation process, with GEOMOD, PATRAN, or KIVA and the numerical pre- and post-processing system.	140

5.7	Data flow diagram of the tasks realised with the numerical pre-processor.	141
5.8	Separation of the combustion chamber in squish, piston, and head-recess regions.	141
5.9	Segmentation of the squish region of the combustion chamber of an idealised IDI Diesel engine by the contours of the liner, the valves, the piston recess, and the prechamber throat.	142
5.10	Segmentation of the geometry of figures 5.8 and 5.9 into blocks (a) squish region (b) bowl region.	143
5.11	Local resolution of the intake jet, by means of mesh refinement.	144
5.12	Mesh generation in the squish region of a DI Diesel engine. (a) top view of contours (b) segmentation into blocks (c) generation of a uniform unstructured mesh.	147
5.13	Generation with PATRAN of the mesh shown in (d) in the region shown in (a). (b): subdivision into blocks, (c) definition of vertices in each block.	148
5.14	Mesh generation with SUPERTAB. (a) region to be meshed. (b) subdivision into blocks (c) generation of structured mesh in block A (d) duplication of vertices in B (e) creation of bricks for the first layer of B (f) generation of the remaining layers.	150
5.15	Boundary regions in a mesh for a pent-roofed petrol engine chamber.	152
5.16	Boundary quadrilateral elements in the case of an IDI Diesel engine, with vertical glow plug. Ghost boundaries are filled gray. Only half the geometry is shown.	152
5.17	Check for deviation from orthogonality.	154
5.18	Method of calculation of the aspect ratio of a cell.	155
5.19	Definition of cell face warpage	156
5.20	Data flow diagram of the problem definition process.	157
5.21	The visualisation pipeline of the pre- and post-processing system.	164
5.22	Data flow diagram of the interface of the pre- and post-processing system for SPEED and the PROSTAR and AVS visualisation packages.	165

5.23	Top view of a mesh for the combustion chamber of the Imperial College DI Diesel engine, with flat piston. (a) coarse mesh. (b) finer mesh.	168
5.24	Computational mesh for the combustion chamber of the Imperial College DI Diesel engine with re-entrant lipped bowl. (a) and (b) coarse mesh. (c) finer mesh.	169
5.25	Top view of an unstructured mesh for the Imperial College model DI engine, with re-entrant bowl recess.	170
5.26	Visualisation of gas flow in the combustion chamber of the Imperial College model DI Diesel engine. Flow is visualised in the form of gas velocity arrows on a cutting plane through the cylinder, bowl, and inlet valve axes.	171
5.27	Visualisation of gas flow in the combustion chamber of the Imperial College model DI Diesel engine. Flow is visualised in the form of gas velocity arrows on a cutting plane perpendicular to the cylinder axis.	172
5.28	Computational meshes for the Imperial College DI Diesel engine, with square bowl.	173
5.29	Computational mesh for the combustion chamber of the FIAT CRF 1930 DI Diesel engine, with bowl in piston.	174
5.30	Visualisations of spray droplets, temperature, and gas flow in the combustion chamber of the FIAT CRF 1930 DI Diesel engine.	175
5.31	Side views of a computational mesh for the combustion chamber of a model DI Diesel engine with rectangular bowl.	176
5.32	Computational mesh for an IDI Diesel engine provided by Renault.	177
5.33	Computational meshes for a Volkswagen DI Diesel engine, with flattened bowl for optical access. (a) coarse mesh. (b) fine mesh.	178

6. Dynamic visualisation of SPEED results

Fig. No.	Description	Page
6.1	The computational domain in the case of an IDI Diesel engine.	181
6.2	Functional diagram of the encoding and the VTR control process.	192
6.3	The stages of the animation process with IL.	194

6.4	The main modules of the IL animation system, FRED, GVIEW, and LAYOFF and their interaction with the user control module LAUNCHER and the database.	195
6.5	User interface of the intelligent light animation system.	196
6.6	Overview of the modelling and animation processes with IL.	197
6.7	Functional diagram of the architecture of AERAS.	199
6.8	(a) Object 'chamber' and the elements – objects – it is constructed with. (b) The organisation of those elements into a hierarchical structure.	201
6.9	AERAS generated geometry for the combustion chamber of an IDI Diesel engine with glow plug, provided by Renault.	205
6.10	(a) Primitive sphere, (b) primitive arrow.	205
6.11	Renault IDI Diesel engine. Visualisation of fuel spray in the prechamber with AERAS.	206
6.12	Renault IDI Diesel engine. Visualisation of flow field with AERAS.	206
6.13	Options for definition of the initial status of the particle system.	207
6.14	Detailed functional diagram of the animation process.	210
6.15	Inter-connection diagram of the workstation and video output systems.	213

7. Testcases

Fig. No.	Description	Page
7.1	(a) Elevation view of the VW IDEA 1.9L DI transparent Diesel engine with a cutting plane along the axis of the bowl. (b) Schematic diagram of the combustion chamber.	229
7.2	Mesh design at valve region.	230
7.3	Mesh topology at TDC of compression.	231
7.4	Geometry contours in the squish region.	232
7.5	Decomposition of the computational domain in blocks.	233
7.6	Solid model for the VW IDEA 1.9L DI Diesel engine at TDC of compression with valves closed.	234
7.7	Gouraud shaded solid model of the VW 1.9L DI Diesel engine at TDC of compression with valves closed.	235
7.8	Coarse mesh for VW IDEA 1.9L DI Diesel engine.	236

7.9	Medium mesh for VW IDEA 1.9L DI Diesel engine.	237
7.10	Fine mesh for VW IDEA 1.9L DI Diesel engine.	238
7.11	Schematic diagram of the combustion chamber, delineating the 6 wall boundaries and the ghost boundary at the entry of the bowl.	240
7.12	Flow rate for each injector hole as a function of crankshaft angle.	240
7.13	Computational mesh at midstroke of induction.	241
7.14	Computational mesh at BDC of induction.	242
7.15	Delineation of vortices formed in the cylinder towards the end of the induction stroke.	243
7.16	Gas flow velocity vectors on cutting planes along the axis of the bowl.	244
7.17	Gas flow velocity vectors at 270°, 290°, 324°, and 350° CA ATDC on cutting planes along the axis of the bowl.	246
7.18	Gas flow velocities at 352.0° CA ATDC on radial cutting planes in the squish and bowl regions.	247
7.19	VW IDEA 1.9L DI Diesel engine.	248
7.20	Calculated pressure trace.	249
7.21	Monitoring of the injected, evaporated, and burnt mass of fuel.	249
7.22	Side view of the fuel sprays at 355.4° and 358.4° CA ATDC.	250
7.23	Unburnt fuel concentration sat 355.4° and 358.4° CA ATDC on axial cutting planes through the axis of the bowl.	251
7.24	Temperature distribution at 355.4° and 358.4° CA ATDC on axial cutting planes through the axis of the bowl.	252
7.25	Temperature field at 359.0° CA ATDC displayed on radial cutting planes through the squish and bowl regions.	253
7.26	Unburnt fuel vapour concentration at 359.0° CA ATDC displayed on radial cutting planes through the squish and bowl regions.	255
7.27	Temperature field at 370.0° CA ATDC displayed on one axial and several radial cutting planes through the squish and bowl regions.	257
7.28	Unburnt fuel vapour concentration at 370.0° CA ATDC displayed on one axial and five radial cutting planes through the squish and bowl regions.	258
7.29	Gas flow velocity vectors at 357.2°, 358.4°, 361.2°, 380.0°, and 400.0° CA ATDC, displayed on axial cutting planes along the bowl axis..	260

7.30	Temperature field at 380.0° CA ATDC displayed on an axial cutting plane through the axis of the bowl and on a radial cutting plane through the squish region.	261
7.31	Unburnt fuel vapour concentrations at 380.0° CA ATDC on an axial cutting plane through the axis of the bowl and on a radial cutting plane through the squish region.	263
7.32	Temperature field at 400.0°, 420.0°, and 450.0° CA ATDC, displayed on axial cutting planes along the bowl axis.	265
7.33	Unburnt fuel vapour concentration at 400.0° and 420.0° CA ATDC, displayed on axial cutting planes along the bowl axis.	266
7.34	Gas flow velocity vectors and temperature distribution on a radial cutting plane through the squish region at 400.0° CA ATDC.	267
7.35	(a) Elevation view of the VW IDEA 1.9L DI Series Diesel engine with a cutting plane along the axis of the bowl. (b) Schematic diagram of the combustion chamber.	274
7.36	Solid model of the VW IDEA 1.9L DI series Diesel engine, indicating the topology of the mesh-blocks.	275
7.37	(a) Gouraud-shaded solid model of the VW IDEA 1.9L DI series Diesel engine. (b) Same as (a) but with superimposed mesh lines.	276
7.38	VW series engine. 'Medium' mesh at 120° CA ATDC.	277
7.39	VW series engine. 'Fine' mesh at 120° CA ATDC	278
7.40	Gas flow velocity vectors at selected instances during compression at axial cutting planes along cylinder axis.	279
7.41	Schematic diagram of squish and rotational motion following the omega shape of the bowl in the combustion chamber of the VW 1.9L DI series Diesel engine.	280
7.42	VW series engine: calculated and measured pressure traces.	281
7.43	VW series engine: Temperature field 1° CA after the start of injection on axial cutting plane.	282
7.44	VW series engine: Fuel monitoring.	283
7.45	VW IDEA series engine: Unburnt fuel concentration and temperature at 358.5° CA ATDC on axial cutting planes.	284
7.46	VW series engine: Fuel sprays at TDC of compression.	285
7.47	Gas temperature filled-area contours at 370.0° CA ATDC in the VW IDEA series engine.	286

7.48	VW series engine: Angular momentum.	287
7.49	VW series engine: Inclination of angular momentum.	287
7.50	VW series engine: Swirl number assuming solid body rotation.	288
7.51	VW series engine: Swirl number with respect to angular momentum	288
7.52	Data flow diagram of the mesh generation process.	289
7.53	Renault IDI engine: Computational mesh	291
7.54	Data flow diagram of the preprocessing and analysis processes.	293
7.55	Storyboard of the animation.	297
7.56	Renault IDI engine: Spray visualisation at 355° CA ATDC.	300
7.57	Renault IDI engine: Spray visualisation at 370° CA ATDC.	300
7.58	Renault IDI engine: Spray visualisation at 385° CA ATDC.	301
7.59	Renault IDI engine: Fuel vapour concentration at 360° and 390° CA ATDC.	302
7.60	Renault IDI engine: Gas flow at 280° CA ATDC.	303
7.61	Renault IDI engine: Gas flow at 340° CA ATDC.	303
7.62	Renault IDI engine: Gas flow at 370° CA ATDC.	304
7.63	Renault IDI engine: Gas flow at 380° CA ATDC.	304
7.64	Data flow diagram of the animation process with AERAS	308

List of Tables

No	Description	Page
3.1	The solution algorithm of the SPEED code	91
5.1	Flag numbers for elements, associating them with SPEED boundaries	152
6.1	Incompatibilities between workstation and video systems	214
6.2	Types of video signals	214
6.3	Recording formats	215
7.1	Main engine operating conditions for the simulation of gas flow and combustion in the VW IDEA 1.9L DI transparent Diesel engine	224
7.2	Spray penetration at selected crank angle positions	271
7.3	Main engine operation conditions for the simulation of gas flow and combustion in the VW IDEA 1.9L DI transparent Diesel engine	273
7.4	Comparison between the transparent and series versions of the VW IDEA 1.9L DI transparent Diesel engine simulations	273
7.5	Main engine operation conditions for the simulation of gas flow and fuel injection in the Renault IDI Diesel engine	292

List of Symbols

A list of commonly used symbols is given here. Other elements, especially indices and computational mesh addresses are explained in the text.

Symbol	*Description*
∇	Gradient operator.
$\nabla\cdot$	Divergence operator.
A	Surface area.
A_L	Area of the surface of the valve curtain.
b_c	Distance of cell centroid from angular momentum axis.
b_i	Specific fuel consumption.
C_1, C_2, C_3	Empirical coefficients of 'k-ε' turbulence model.
C_D	valve discharge coefficient.
C_μ	Constant of the 'k-ε' turbulence model .
C_v	Specific heat at constant volume.
E_T	Kinetic energy of moving gas in the cylinder.
e	Specific internal energy.
I_L	Moment of inertia around the angular momentum axis.
k	Specific mean kinetic energy of turbulence.
L	Angular momentum of moving gas in the cylinder.
l	Turbulence length scale.
$\dot{m}$	Mass flow through a valve curtain.
m_c	Mass of cell.
p	Thermodynamic pressure.
p_i	Mean indicated pressure.
P_k	Rate of production of k.
Q	Total mass of the injected fuel in the cylinder.
r_c	Distance of cell centroid from centre of mass
S	Source.
S_{ed}	Energy interchange rate with droplets
S_{ec}	Energy source due to combustion
S_m	Mass interchange rate with droplets
S_{mom}	Momentum interchange rate with droplets
S_N	Swirl number.
T	Temperature.
t	Time.
u_c u	Velocity (vector); u_c : Gas velocity at cell centroid.
V	Volume.
V_d	Displacement volume

V_p	Piston velocity.
x	Space (vector)
Z_L	Valve lift.

Greek symbols:

Γ	Diffusion coefficient.
Γ_e	Turbulent thermal diffusion coefficient
ε	Dissipation rate of turbulence.
θ	Crank angle,
	Void fraction.
κ	von Karman constant.
λ	Volumetric efficiency.Proportionality constant. λ_c: for cylinder, λ_v: for valve curtain.
μ_{eff}	Effective viscosity of k.
ρ	Density.
ρ_g	Apparent gas density.
$\sigma_\varepsilon, \sigma_k$	Empirical coefficients of 'k-ε' turbulence model
τ	Turbulent shear stress tensor
τ_w	shear stress at the wall, κ is the von Karman constant.
ϕ	General scalar variable.
ψ	General dependent variable.
ω_{crank}	Angular velocity of the crank.

Abbreviations

ATDC	After top dead centre piston position
BDC	Bottom dead centre piston position
CA	Crank angle
CAD	Computer-aided design
CFD	Computational fluid dynamics
CI	Combustion ignition
CT	Computer tomography
DI	Direct injection
EBU	Eddy break-up
FDDI	Fiber distribution data interface
FEA	Finite-element analysis
fps	frames per second
GUI	Graphical user interface
HSC	High-speed channel
IC	Internal combustion
IDI	Indirect injection
NCSA	National center for supercomputing applications (USA)
PDF	Probability density function
SI	Spark ignition
TDC	Top dead centre piston position

Blank

In

Original

Chapter 1

Introduction

The work of designing reciprocating engines has been enhanced in the recent past by the emergence of computer-based techniques for the simulation of the in-cylinder flow and combustion processes. Much of this is due to the growth of computational fluid dynamics (CFD) methodologies, with the development of numerical procedures for solving the governing differential equations and mathematical models for turbulent motion, heat transfer, chemical reaction and combustion. The growth in this field of computational engineering has become possible with the dramatic increase of the speed and size of computers, which has increased the size and complexity of the problems that can be solved, to the extent that engineers can now not only draw upon laboratory-based experimental facilities, but also computer-based systems with capability to model increasingly realistic engine geometries.

However the considerable research effort concentrated on the development of CFD methodologies has not been matched by the development of computer resources for:

- fast and geometrically-accurate generation of computational mesh representations of the combustion chamber of real engines;
- efficient interpretation of the deluge of data produced by the CFD codes, and in general for
- interaction with the whole process of computational simulation of flow and combustion in engines.

On the contrary a considerable absence of such resources has been noticed and documented in studies, like the one by Anderson, commissioned in 1989 by the Ohio State University, in the USA [1]. As a result, engineers spend a considerable amount of their time in generating the mesh or trying to interpret the result of the simulation. In many situations they are forced to turn their attention to programming their own mesh generation, and data interpretation programs, in order to exploit the numerical solution codes they develop or use. Although some engineers consider this task an important and enjoyable part of their work, it remains distracting, if not detrimental, to the attainment of their main research or design goals. In any case, each software solution created in this way is special-purpose, often confined to specific hardware or software platforms, and frequently unable to describe realistic engine geometries in their full complexity.

On the other hand, the use of computer aided design (CAD) systems for finite element analysis (FEA), that has provided a working paradigm for the stress analysis process, has shown that interactive computer graphics techniques can adequately provide data preparation (e.g. geometry modelling, mesh generation) and interpretation means, or *tools*, on the workbench of the simulation engineer.

Starting off from the above observation, this thesis investigates how interactive computer graphics methods can enhance the task of simulation of in-cylinder processes, in the areas of:

- definition of the geometry of engine combustion chambers;
- generation of a computational mesh in the domains thereof;
- extraction of meaningful information from complex CFD datasets through the use of interactive graphics and imaging tools

The availability of such methods would be useful to industrial users of CFD software, and would enrich existing CFD methodology, because

- it would reduce the time involved in mesh generation and simulation conditions definition;
- it would allow efficient modelling of realistic engine geometries in their full complexity;
- it would increase comprehension and insight into the complex results produced by CFD engine simulation codes;
- it would allow better testing and faster development of the simulation models, thereby enriching existing scientific methods.

The requirements for such tools are exemplified by SPEED, a code for simulation of three-dimensional flow, heat/mass transfer, and reaction processes in the combustion chambers of reciprocating engines, which is being developed at Imperial College of Science, Technology, and Medicine.

1.1 A model of the CFD engine simulation process

Computational fluid dynamics simulations in general and CFD simulations of flow processes and combustion in engines in particular, typically involve three main *phases*, illustrated in figure 1.1, that have been termed by Haber [2] and Marshall [3] as:

1. *model building;*
2. *solution* and
3. *interpretation, evaluation* and *steering.*

Multiple steps can occur within each phase; for example the model building phase involves three steps: geometry modelling, mesh generation, and definition of initial and boundary conditions for the analysis. The model building and interpretation phases require human interaction with the system. The solution phase is typically realised in batch mode, due to the computationally intensive techniques it involves. Errors can occur during each step of each phase, in which case the process is interrupted and control is handed to the previous step or phase. The reliability of the simulation depends on error control in all phases of the process [4].

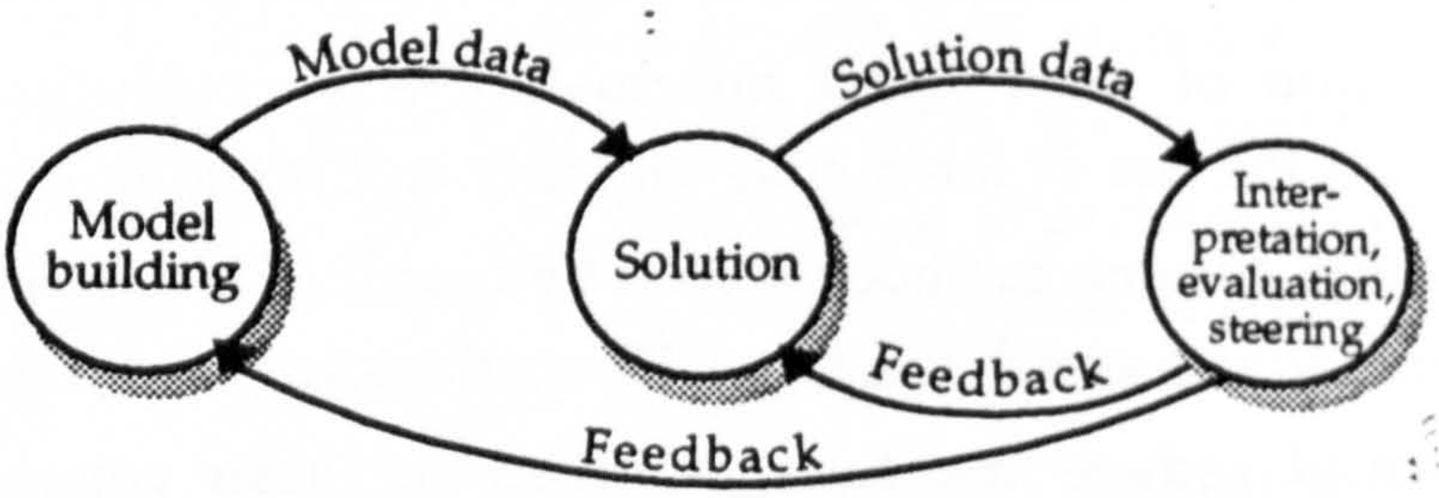


Figure 1.1. The three phases of the C.F.D. simulation process.

An alternative terminology for the above three phases, that is popular in the engineering community is *pre-processing*, *analysis*, and *post-processing*. Also Bankroft [5] uses the term 'software cycle' instead of phase. Haber's and Marshall's terms are preferred because they describe the function of each phase and do not split the simulation into tasks before and after the simulation, as the 'pre' and 'post' prefixes imply. Indeed as it will be shown in following paragraphs, the solution and interpretation phases are coupled through steering. But let us now examine the tasks and requirements involved at each phase.

Model building involves the transformation of an engineering simulation problem for a particular engine into a well-posed mathematical idealisation in terms of geometry and simulation conditions. It is typically realised in three steps:

- definition of the domain of analysis (which for the purpose of the present work is the engine combustion chamber) in the form of a computational surface model,
- generation of a time-dependent computational mesh within the domain enclosed by the surface model, and

- specification of the relevant initial and boundary conditions to be included or excluded.

Human interaction with the system is required in all steps of this phase. As will be shown in later chapters, this interaction can be enhanced through the use of numerical and computer graphics techniques, to define and view the geometry and mesh, and graphical user interfaces (GUI's) for interactive input of data. This phase is exemplified by the graphical pre-processors often used with finite element stress analysis systems, such as PATRAN [6] and IDEAS/CAEDS [7],[8].

Solution phase. Model data available from the previous phase are subsequently input into a CFD code, for the calculation of fluid flow, heat/mass transfer, and combustion in the engine combustion chamber. Several CFD methods exist for the calculation; their key features, according to Gosman [9], [10], include a computational mesh arrangement, discretisation practices used to transform the differential equations of the governing equations into algebraic equations, solution algorithms, and mathematical models for turbulence, fuel injection, and turbulent combustion. Because of the high computational time required by such methods, the solution phase tends to be a batch process. If evaluation of the results points that the computation setup is in error, then it is necessary to interrupt the solution and restart with a different setup. This process, referred to by Marshal [11] as 'steering' is discussed in more detail in chapter 2. It is expected that the advent of more sophisticated computational systems will make it possible to interpret and evaluate the solution while it is in progress, and to modify interactively both the model and the solution strategy in response to the evaluation.

Interpretation, evaluation and *steering* phase. This phase involves the implementation of numerical and computer graphics tools to interpret the results of the CFD simulation. It is commonly referred to by the term 'visualisation', since the 1987 publication of the National Science Foundation report *Visualisation in Scientific Computing* [12]. Visualisation is defined by Haber and McNabb [4] as the use of computer graphics and imaging technology techniques for comprehending data obtained by simulation or physical measurements; in Kaufman's words [14] 'it provides processes for seeing and steering the unseen, thereby enriching existing scientific methods'.

1.2 The structure of this thesis

The foundation of the present work transcends application and field boundaries, depending on the integration of a variety of disciplines such as computer graphics, image processing, computer-aided design, geometric modeling, computer animation, mathematics, computational fluid dynamics, approximation theory, visual psychology, and user interface studies. This interdisciplinary background as well as an overview of the previous work in the areas of user interaction with CFD codes, geometry definition, mesh generation, CFD simulation of in-cylinder flow processes, visualisation, and animation is reviewed in chapter 2 of the present work.

An entire chapter – Chapter 3 – is dedicated to a brief overview of the flow processes in engine combustion chambers, and a summary of the methodologies for the simulation thereof, that are implemented into the SPEED code. The purpose of chapter 3 is not to study the flow-simulation, fuel-injection, and turbulence combustion models provided by SPEED; it is rather to highlight the need for geometry definition, mesh generation, initial and boundary conditions definition, visualisation, and generally user-interaction tools for use with it. A comprehensive list of such tools, suitable for interfacing with the CFD simulation process in general and the SPEED code in particular is presented in chapter 4.

Chapter 5 presents the realisation of a selected set of such techniques in a pre- and post-processing system for the SPEED code that has been developed by the author. This system is based on CAD systems; it employs interactive computer graphics techniques and is suitable for geometry definition and mesh generation in the case of real engine geometries in their full complexity, it caters for definition of initial and boundary conditions for an engine simulation with SPEED, and allows visualisation of solution results.

The software environment presented in chapter 5 provides a satisfactory means of definition of the engine combustion chamber geometry and mesh generation within the domain thereof; however, its dependence on CAD systems limits its application when visualisation of CFD and engine-specific results, such as time-varying fluid-flow data or fuel sprays, is required. In such cases a variety of dynamic visualisation (i.e. animation) techniques, tailored to the visualisation of SPEED results, had to be employed. Such techniques, including moving cutting planes, zooming into regions, and advection of particles seeded into the flow, have been implemented in an animation system, that is presented in chapter 6.

In Chapter 7, the effectiveness of the systems presented in Chapters 5 and 6 is demonstrated in the case of two direct injection and one indirect injection Diesel engines. Definition of the surfaces bounding the combustion chamber, generation of uniform, unstructured meshes of hexahedral-cells, definition of engine operation conditions and initial and boundary conditions for the simulation, calculation of gas flow, fuel injection, auto-ignition, and combustion, and finally visualisation and animation of calculated results, are presented.

Chapter 8 summarises findings of the study and the conclusions drawn therefrom, and makes recommendations for future work.

Chapter 2

Background

The objective in the present chapter is to provide an overview of the main developments that have taken place in the recent past in the fields of mesh generation, computational simulation of flow in engines, scientific visualisation and computer animation. These disciplines comprise the main background of the present work and the major tasks in the three-phase process of computational simulation of flow and combustion in engines.

2.1 Overview of computer systems for CFD engine simulation

2.1.1 Computational methodologies

Early efforts to analyse and predict in-cylinder processes consisted of approximate or *global* techniques based on simplified models of the processes. They provided limited, but often very useful information about them: an example is the one-dimensional model, developed by Fitzgeorge and Allison [15], for the prediction of the inward radial motion or 'squish' which occurs during the compression stroke of engines having bowl-in-piston configurations.

Since the late 70's a number of multi-dimensional methods have been available for simulation of flow processes in the combustion chamber of reciprocating engines. The characteristic building blocks of these are: [9],[10]

- a computational mesh arrangement, defining the number and position of the locations at which the flow parameters are to be calculated;
- the partial-differential conservation equations of physics governing the spatial and temporal variations of velocity, pressure, etc.;
- additional equations representing mathematical models of key sub-processes occurring within the fluid, such as turbulent motion, heat transfer, chemical reaction, etc.;
- computer-based numerical procedures for solving the equations for the circumstances of interest (e.g. discretisation practices used to transform the partial-differential equations into algebraic equations at the locations defined by the computational mesh).

2.1.1.1 The general transport equation for turbulent flows

The mathematical description of fluid flow is based on the differential equations expressing the laws of conservation of mass (continuity equation), momentum and energy, known as the 'Navier-Stokes' equations [16], [17]. Although in-cylinder processes are governed by the same equations as any other fluid flow, they pose some special obstacles to solution. These arise from the existence of non-stationary turbulent motions with small time and length scales, the direct calculation of which are not feasible due to the prohibitive computing requirements. This problem is attacked through decomposing the instantaneous conserved variable ψ of the conservation equation into an average or mean component $\bar{\psi}$ and a fluctuating or turbulent component ψ'' (Bradshaw [18]), so that:

$$\psi = \bar{\psi} + \psi'' \tag{2.1}$$

Ensemble and time-averaging techniques (see Lancaster [19] for definitions) are widely used in engine applications, mass or Favre averaging [20] is often employed in circumstances giving rise to large density fluctuations, such as combustion, while space averaging or 'large eddy simulation' (LES) techniques allow cycle-to-cycle variations. For a detailed evaluation of the relative merits of these techniques the reader is referred to the works of Gosman [21] and Reynolds [22]. Due to averaging of the conservation equations, handling of additional terms, called 'turbulence stresses' and 'fluxes', is required. Several approaches to this problem, conventionally known as turbulence models, exist (see e.g. the survey by Bradshaw for a review), of which two have been widely used in engine studies: the 'k-ε' and the 'Reynolds stress' models. The former, developed at Imperial College by Launder and Spalding [171], and modified by El Tahry [172], assumes that the turbulent stresses and fluxes obey constitutive relations, and has been extensively used and tested (see e.g. the Ph.D. thesis of Ahmadi-Befrui [23]). It involves the scalar fields k and ε, representing 'turbulence kinetic energy' and 'turbulence dissipation rate' respectively, defined as:

$$k = \frac{1}{2} u_i'' u_j'' = \frac{1}{2} (u_1''^2 + u_2''^2 + u_3''^2) \tag{2.2}$$

$$\varepsilon = \frac{c_\mu^{0.75} k^{1.5}}{L_t} \tag{2.3}$$

where u_i'' is the turbulent component of velocity, c_μ an empirical constant, and L_t a turbulence length scale.

The Reynolds stress model is now emerging in engine applications. However it requires the solution of seven partial differential equations (six stresses and ε) compared to two (k, ε) for the k, ε model, which imposes an additional computational load on the mathematical model.

The conservation equations and the turbulence equations obtained by the k-ε model, can be expressed in a common form for dependent variable ψ, that expresses the time evolution of the flow and facilitates the development of a numerical solution. This common form called the 'general transport equation' has the form:

$$\frac{\partial}{\partial t}(\rho\,\psi) \;+\; \nabla\cdot(\rho\,\mathbf{u}\,\psi) \;=\; \nabla\cdot(\Gamma_\psi \nabla\psi) \;+\; S^\psi \qquad (2.4)$$

Rate of change Convection Diffusion Source

where the symbols t, ρ, u, Γ_ψ, S^ψ denote time, fluid density, fluid velocity, diffusion coefficient for ψ and source of ψ respectively. There is one equation of this form for each of the dependent variables $\psi = u_1, u_2, u_3$ (the three components of the averaged velocity field u), e_s (averaged specific internal energy), unity (for the mass conservation equation), k (turbulence kinetic energy) and ε (turbulence dissipation rate).

This equation implies that the rate of change at any location in the flow depends on convection (transport of the conserved quantity with the fluid to or from the location), diffusion (transport towards a uniform distribution of ψ as a result of its gradient), and sources (and sinks) of the quantity related to effects such as pressure distribution and shear action.

The derivation of (2.4) as well as the diffusivity and source terms for ψ can be found in the Ph.D. thesis of Theodoropoulos [24].

2.1.1.2 Turbulence combustion modelling

The high computational cost of calculating chemical kinetics directly, due to the large number of chemical species and chemical reactions involved, has given rise to several models for premixed and Diesel combustion. Such models, called *turbulent combustion models* because they deal with turbulent flames and their interaction with turbulent flows, are based on the simplification of the complex chemical kinetics and on averaged quantities (e.g. average temperature, velocity, reaction rate etc.). For a review of the

building blocks of these methods, the reader is directed to the works of Gosman [25], Weller [26], and Borghi [27].

A first approach to turbulent flame modelling has been the *eddy-break-up (EBU) model*, proposed by Spalding [28]. According to this the burning zone consists of alternating fragments of unburnt and fully burnt gas, and the chemical reactions are infinitely fast; therefore the mean 'reaction rate' - the mass rate of production per unit volume of species, suitably averaged for computational efficiency - is controlled only by the turbulent mixing of the unburnt and burnt gas. Another model of the EBU type, that has been widely used in engineering applications, is the one proposed by Magnussen [29]. This model includes a limited dependency of the mean reaction rate on the fuel, air and burnt gas concentrations in the mixture. EBU models became popular because they are easy to solve and numerically stable. However experience with such models has shown that they are unable to predict correctly the thickness of the flame and its propagation speed, especially in the vicinity of the walls, where the predicted flame accelerates unphysically, except in some special circumstances, often supplemented with ad-hoc measures. This lead to enhancements of the EBU model to include chemical kinetics effects in a way that compensates the flame acceleration near a wall by the reduction of the mean reaction rate in the thermal boundary layer, often by the introduction of a laminar flame speed to represent the combustion process occurring between the unburnt and burnt gases as they mix [30], [31].

Alternative methods combine turbulent mixing and chemical kinetics effects and assume that the flame is a finite-thickness reaction zone between the fully-burnt and unburnt phases. They represent the reaction zone as a distribution of 'flamelets' [32], [33], each of which is considered to be a laminar flame, the behaviour of which is evaluated from the local conditions. Within this framework, an approach that is applicable to engineering problems is the *presumed PDF model* [34], that presumes the probability density function (PDF) of the fuel concentration in the reaction zone, in this way being computationally more efficient. An alternative approach is the 'transported joint PDF' model [35], in which a joint PDF of velocity and turbulent dissipation is solved, albeit at a higher computational cost.

A more physically justifiable way of representing the interaction between flame and turbulence is assumed in the so-called *flame area combustion models*. According to these, a zero-thickness flame sheet propagates locally as a laminar flame. The flame area changes due to heat release and

interaction with turbulent flow, with formulations for flame propagation, stretching, curvature, singularities (e.g. cusps) and self-intersections. Such flame area models have recently been suggested by Pope [36], Poinsot [37], and Weller [38]. A model of the latter has been used by Sarantinos for the simulation combustion and flame propagation in a VW petrol engine [248].

2.1.1.3 Software systems for CFD engine simulation

Several three-dimensional methods for simulation of air-flow in cylinders had initially been applied to the RPM code, developed by Gosman et al. [39], [40], at Imperial College. However, RPM's contribution to the study of formation and evolution of fluid flow was restricted to axisymmetric geometries, because of the low computing processing overheads they required.

The advent of high-end computers, capable of processing several million instructions per second, permitted researchers to implement 3D solution techniques in non-axisymmetric cases, with codes such as KIVA [41], developed at Los Alamos National Laboratory, and EPISO [42], [43], and [44], developed at Imperial College. These codes, although capable of modelling irregular engine geometries, were limited through their use of structured meshes and the provision of mesh-generation and visualisation tools that were not sufficient for realistic engine simulation. The FMCS and FIRE codes [45], developed by AVL in Austria, differed in that they provided dedicated pre- and post-processing programs but they too suffered from the above limitations.

In 1988, the development of the SPEED code at Imperial College [46], established the use of time-dependent, unstructured mesh methodologies [47], that enabled the simulation of real engine configurations [48] and permitted more accurate flow predictions without large storage overheads by use of locally refined mesh regions. However the need for unstructured meshes defined over real engine configurations, required appropriate mesh generation and visualisation systems. Such systems, effectively providing means of interaction with the engine simulation process with the SPEED code, were developed by the author [49], and are presented in subsequent chapters of the present work.

2.2 Model building methodologies

It has been noted earlier that one of the tasks during the in-cylinder flow simulation process is the generation of a computational mesh arrangement. This is realised in a two-step paradigm: first the boundary surface of the problem domain is interactively defined by the user, usually in terms of a computational surface model, and subsequently the mesh is generated in the domain enclosed by the boundary surface. As engine flow analysis codes are applied to increasingly realistic combustion chamber geometries, the need for software systems for construction of 3D meshes in such geometries has grown. This, in turn, has constituted mesh generation, according to Smit and Su [50], the most difficult and time-consuming aspect of the simulation process.

2.2.1 Mesh types

A computational mesh is, for the purposes of this work, a collection of closed polyhedra (e.g. hexahedra or tetrahedra), called *cells*, covering the problem domain; the intersections of these are *faces*, *edges*, and *vertices* (see figure 2.1); we will refer to these collectively as *mesh entities*.

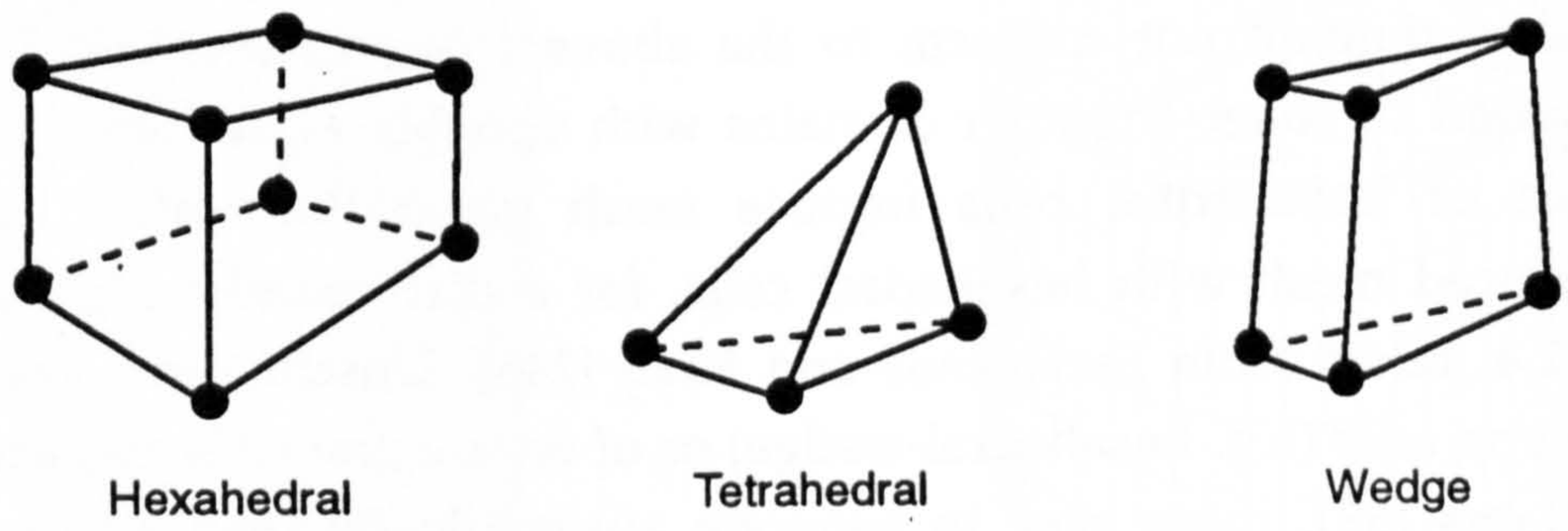


Figure 2.1 Three different types of computational cells. From left to right: hexahedral, (with six faces), tetrahedral (with four faces), and wedge (with five faces). Vertices are denoted by thick points, whereas edges are denoted by lines.

Depending on the cell types and their arrangement in space, meshes can be classified to *structured* or *mapped*, *block-structured*, and *unstructured*.

In the case of *structured* meshes, hexahedral cells are laid out in such a way, that each type of mesh entity has the same connectivity to its corresponding mesh entities. Such meshes in a region R (see figure 2.5) can

be mapped, through an appropriate transformation, into orthogonal meshes in a parellelepiped or cylindrical region R´. As it can be seen from the variety of structured meshes shown in figure 2.2 (courtesy of Institute Francaise du Petrol in the case of 2.2.b, and Renault in the case of 2.2.c, and d [51]), realistic engine combustion chambers can be handled in this way, however not without difficulty during mesh generation, and in most cases by introducing numerical inaccuracies to the CFD solution procedures, due to the distorted cells they contain. The meshes of figures 2.2.b, c, and d display less cells than are actually defined; in the case of mesh 2.2.b, cells are defined throughout the piston region, but only the ones in the rectangular bowl are active. The same applies for the cells around the prechamber in mesh 2.2.c and on each side of the pent roof in mesh 2.2.d. This means that structured meshes impose higher than necessary storage overheads in the case of realistic engine geometries.

Complex combustion chamber configurations can be handled more efficiently by subdividing the domain to be meshed into several sub-domains, known as *sub-regions, blocks* or *macro-elements,* in such a way that structured meshes can be generated in each region independently, and ensuring continuity of mesh lines where the meshes interface. Such meshes are called *'block-structured'*. Several hexahedral-cell, block-structured meshes can be seen in figure 2.3, where the different blocks are highlighted with thick lines.

Meshes that do not conform to the above rules, or *unstructured,* are more suited to cover irregular domains with variable mesh density, but in the case of hexahedral cells impose mesh generation difficulties. An unstructured mesh with hexahedral cells, for a disc chamber, is shown in figure 2.4, taken from Sarantinos and Kralj [245]. Unstructured meshes of mixed-type cells (e.g. hexahedral-wedge) or of tetrahedral cells are, according to Williams [53], more easy to generate automatically than structured or block-structured ones, but impose storage and solution constraints to the CFD solvers.

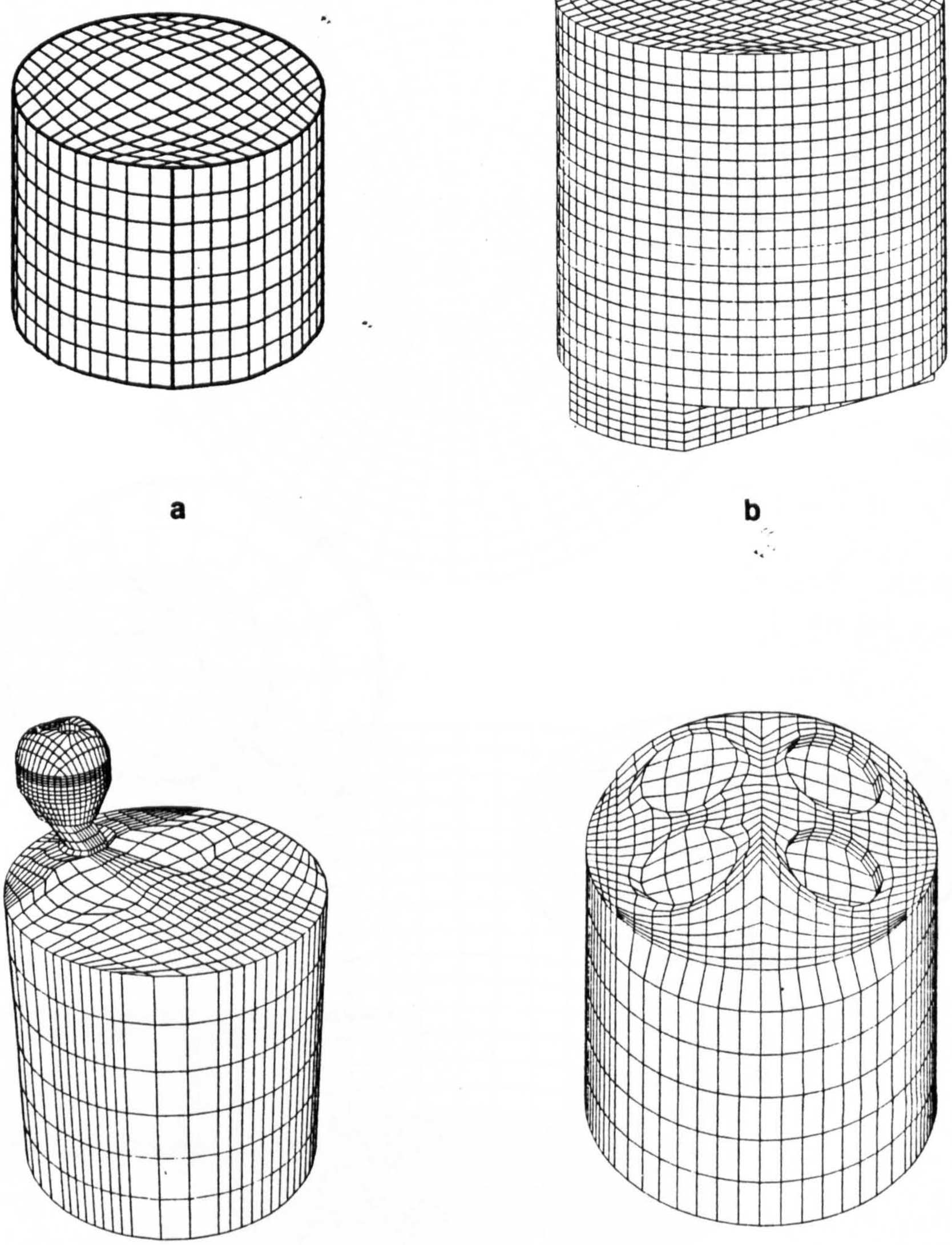
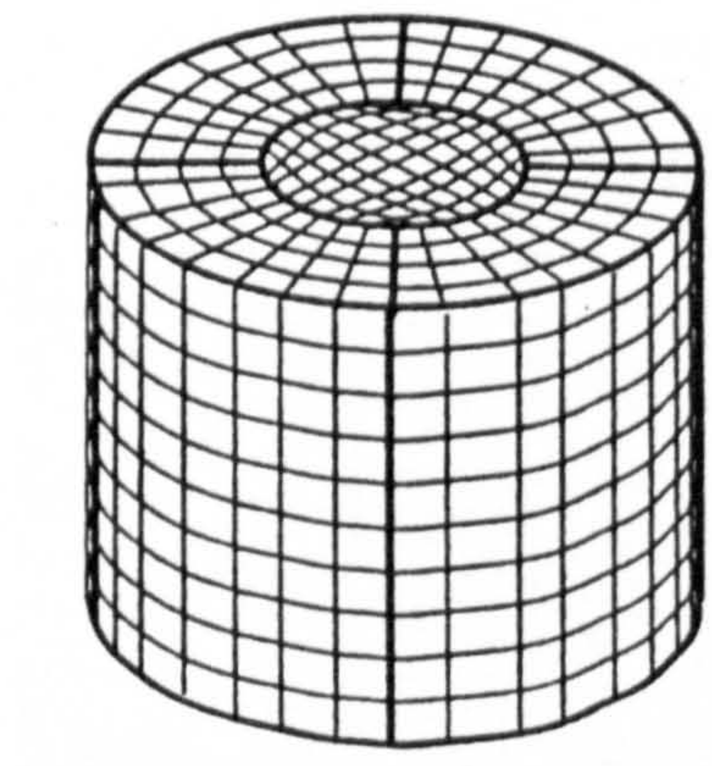


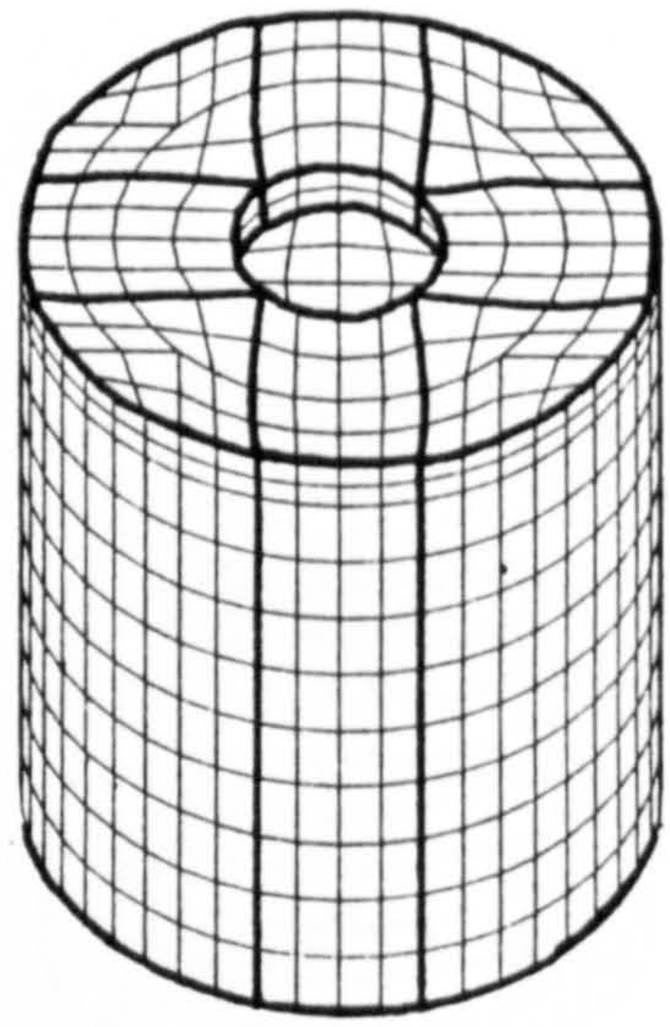
Figure 2.2

Various structured meshes with hexahedral cells.

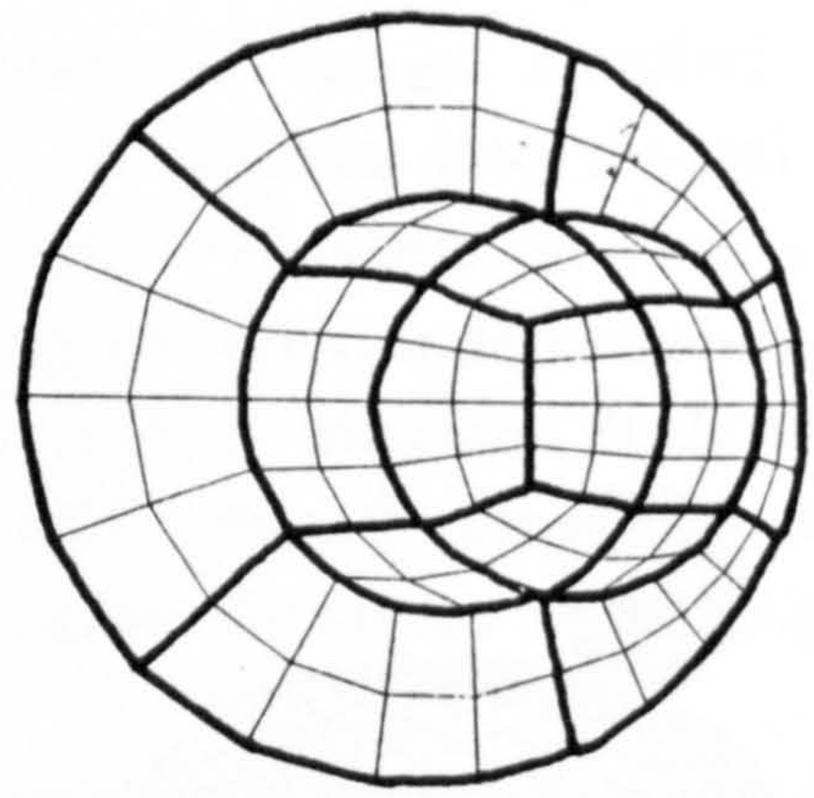
a) disk chamber, b) chamber with rectangular bowl, c) IDI Diesel engine chamber, d) petrol engine chamber with pent roofed cylinder head.



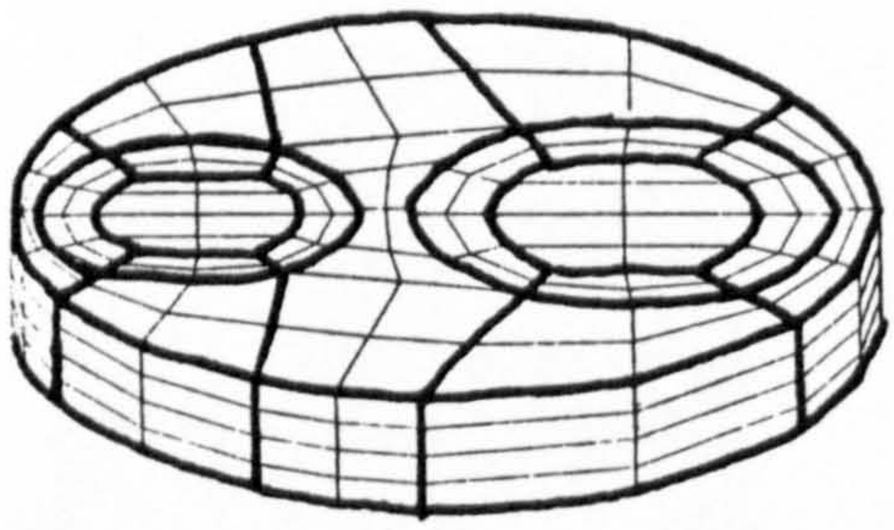
a



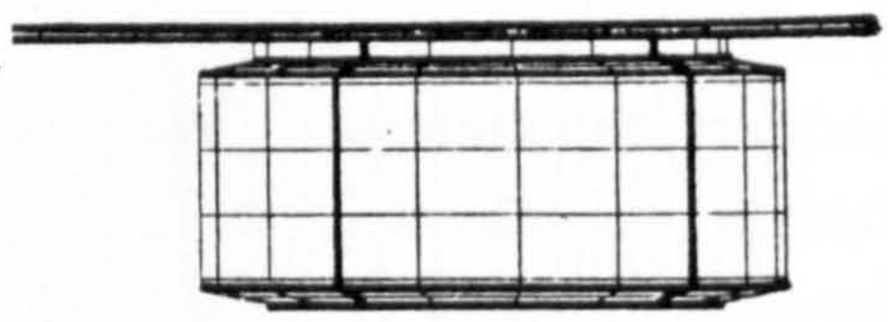
b



d



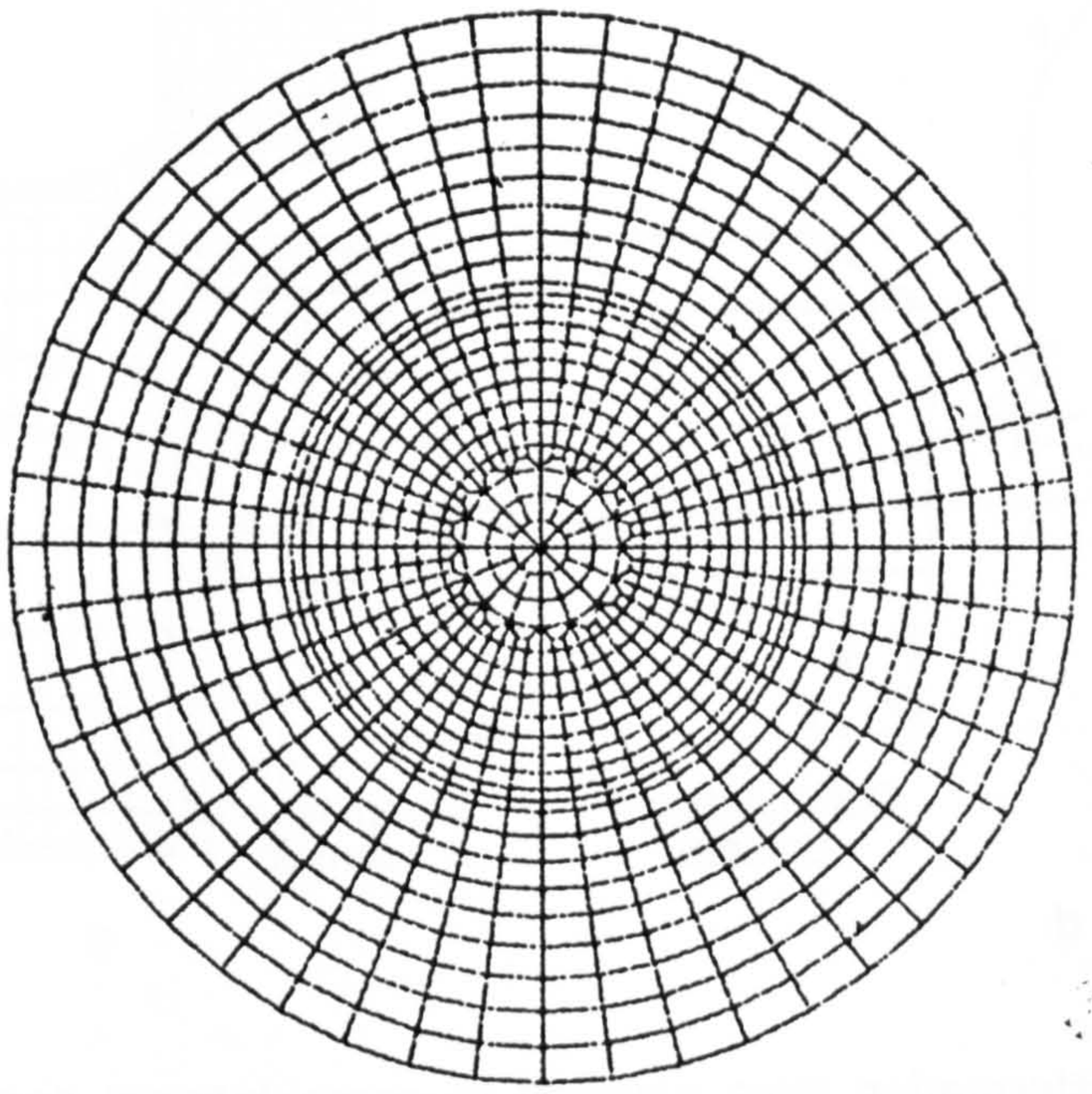
c



e

Figure 2.3

Various block-structured meshes with hexahedral cells generated for model engines.
a) disk chamber with five blocks, b) disk chamber with nine blocks and open intake/exhaust valve, c) disc chamber with two valves, d) DI Diesel engine combustion chamber with lipped bowl and inlet valve (top view) e) same as d, but elevation view.



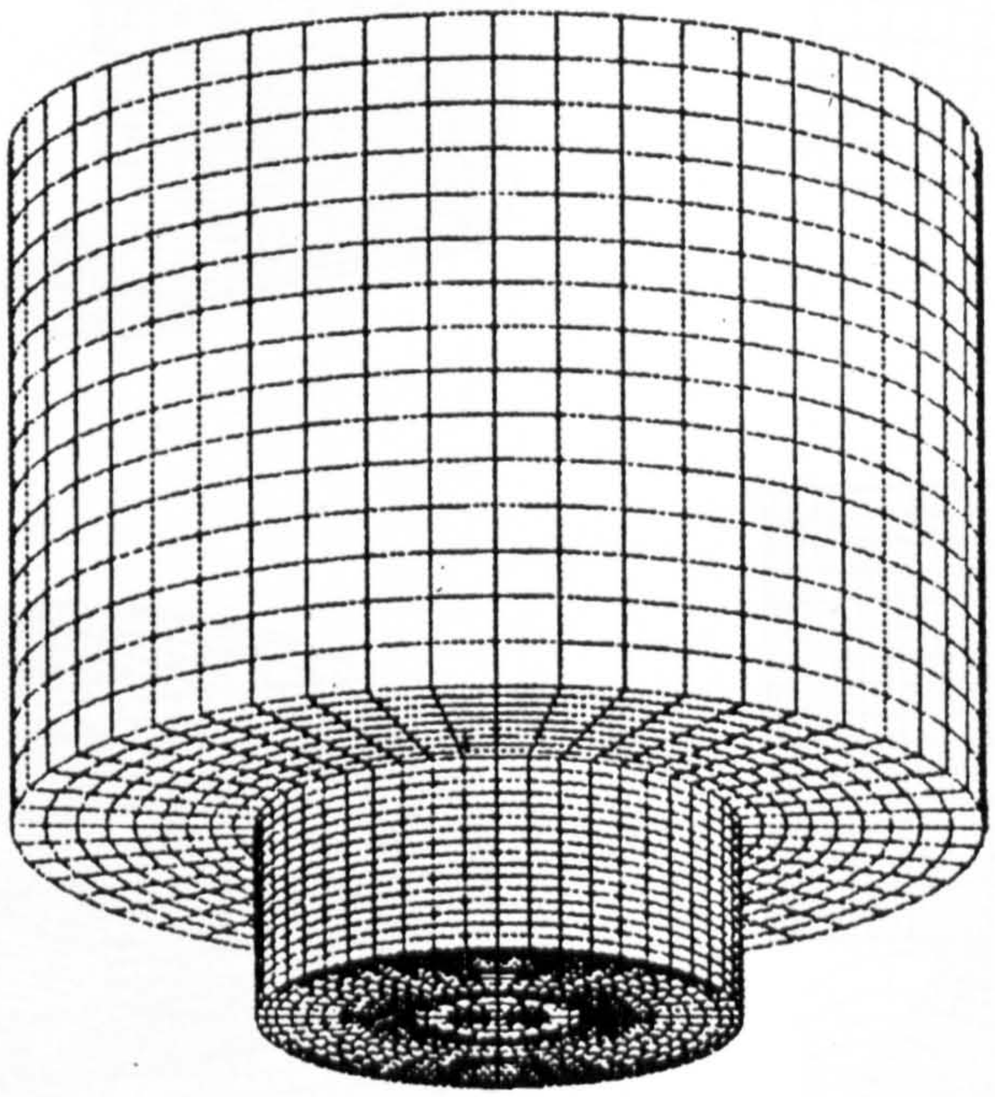


Figure 2.4

An unstructured mesh with hexahedral cells for a model DI Diesel engine with axisymmetric combustion chamber with bowl in piston.Top: top view. Bottom: side view.

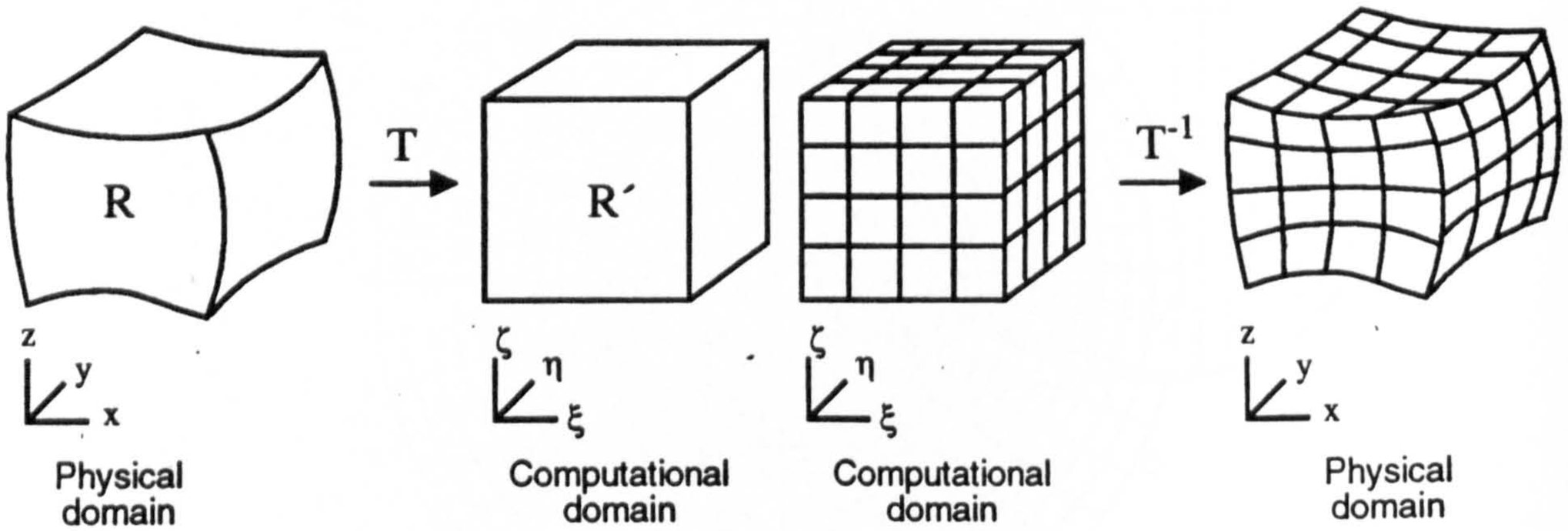


Figure 2.5. Transformation from physical to computational domain and reverse transformation.

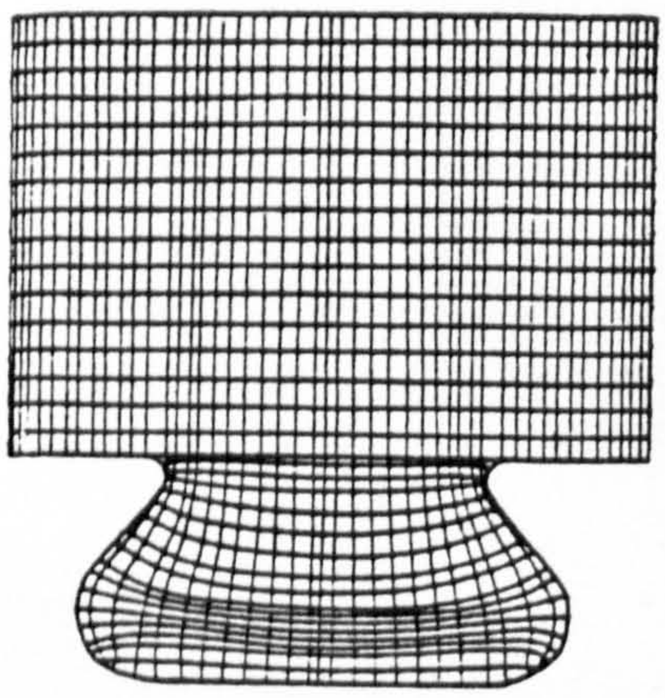


Figure 2.6

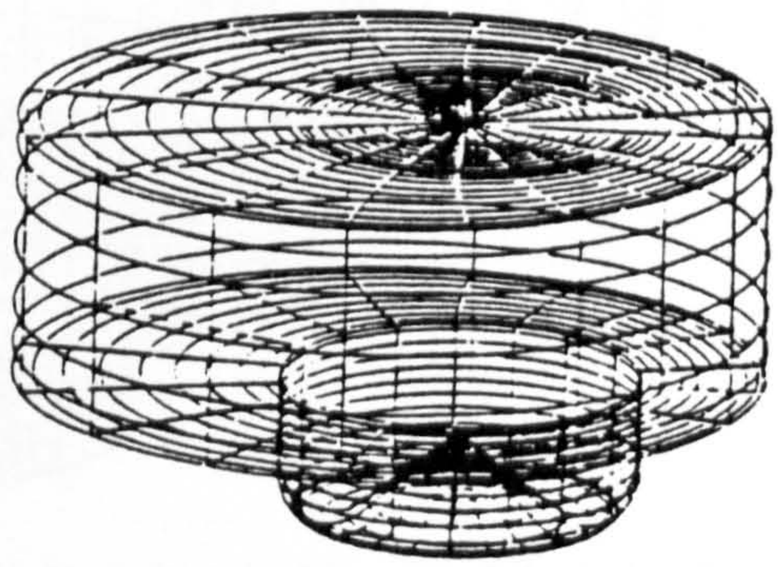
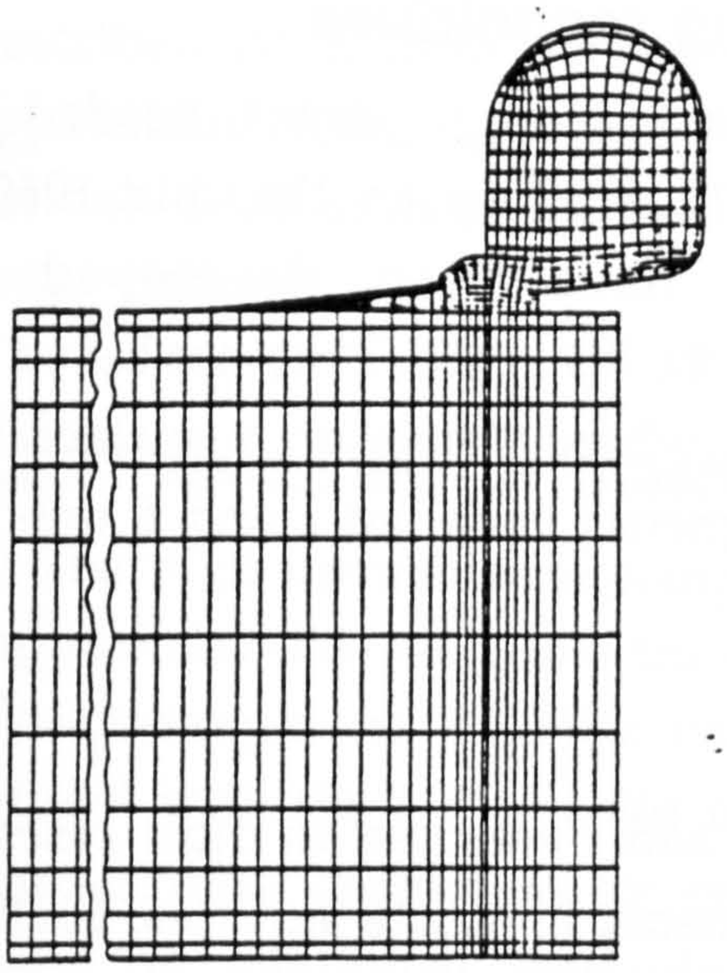
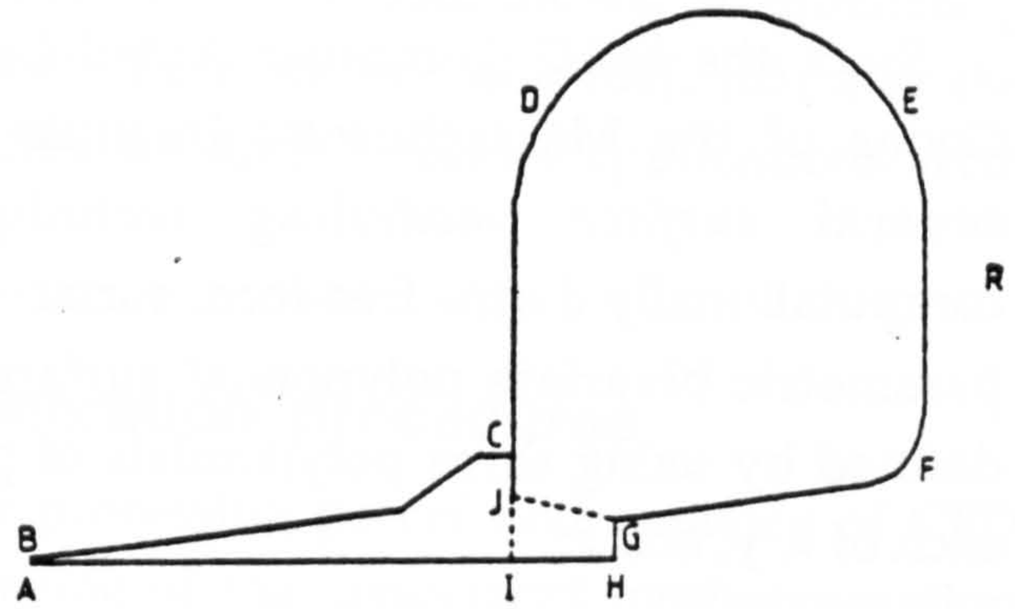


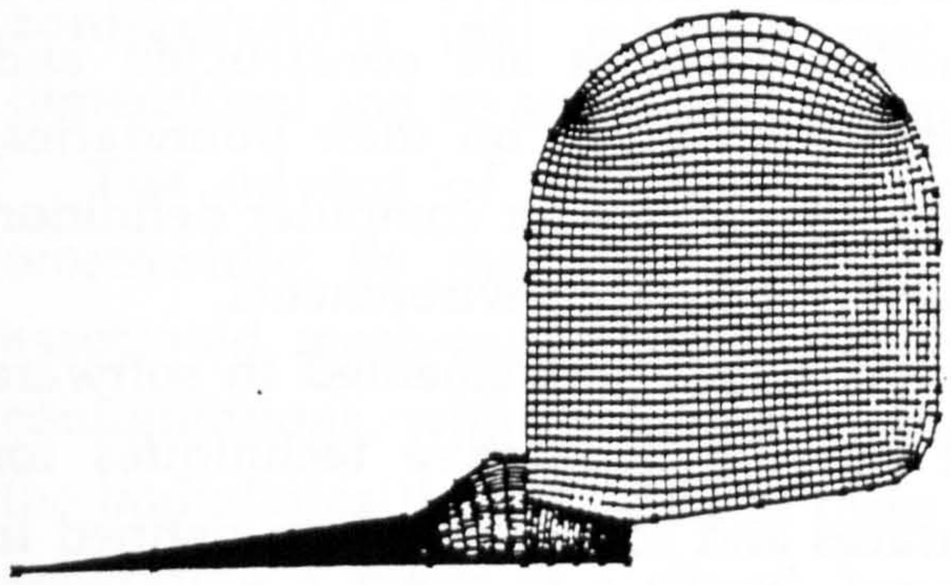
Figure 2.7



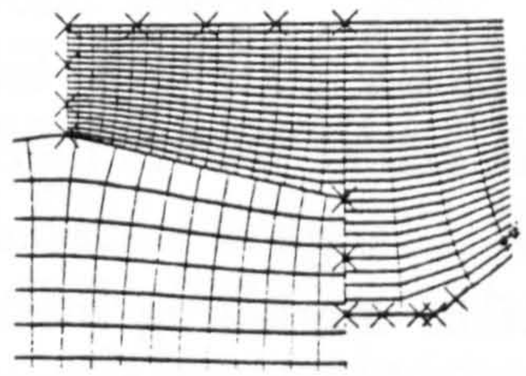
a



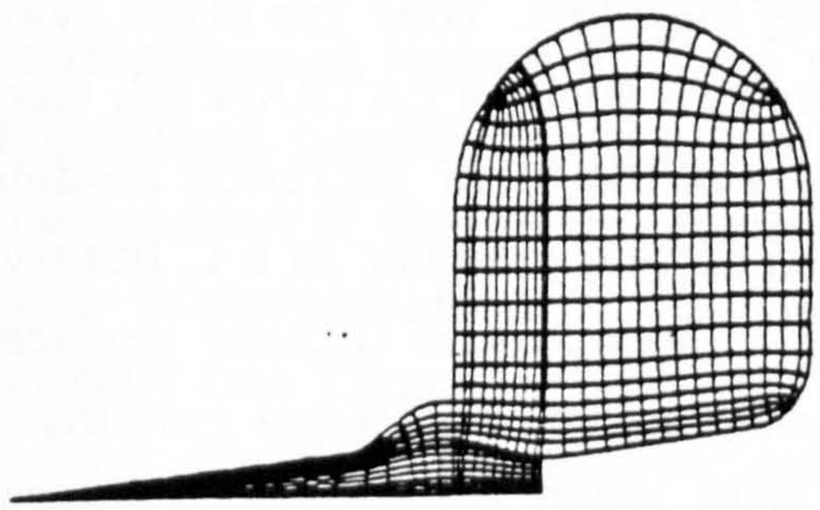
b



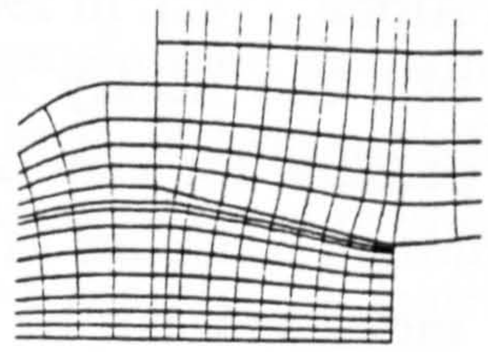
c



d



e



f

Figure 2.8

2.2.2 Overview of surface modelling techniques

Since the MAC Computer Aided Geometric Design project, headed by Coons of the Massachusetts Institute of Technology in the mid-1960's, several *surface modelling* techniques have been developed to computationally define free-form surfaces in the 3D space. They are based on parametric bivariate polynomial surfaces S, where points on a surface are defined by using three polynomials of parametric variables r and t, one for each of x, y, and z:

$$S(r,t)=\big(x(r,t),y(r,t),z(r,t)\big) \tag{2.5}$$

Cubic polynomials in both parameters r and t are used in most cases to define free-form surfaces, resulting to parametric bivariate surfaces, such as Hermite, Bezier, and B-spline ones [54]. Quadric surfaces (embracing spheres, cylinders, cones, ellipsoids, etc.), based on quadratic polynomials have been widely used in specialised applications such as solid modelling of Eucledian solids [55]. When cubic and quadric surfaces are constructed and manipulated mainly in terms of information given on their boundaries, they are called 'Coon's patches', and are appropriate for computer definition of a wide range of surfaces in user-friendly software environments.

Surface modelling methodologies are mainly implemented in software systems, called *solid modelers*, that provide interactive techniques for construction of three-dimensional surfaces and solids; these are defined in terms of 3D free-form Coon's patches, called 'patches' for brevity, and combinations thereof to form 3D objects called 'volumes' or 'solids'. A variety of solid modelling techniques have been developed in the past decade, such as *boundary-representation* or *B-rep* [56], *constructive solid geometry* [57], and *analytic solid modelling* [58], [59]. They are often available within the closed framework of CAD systems such as PATRAN [6], Geomod [60], and Alpha_1 [61]. In 1989, the release of the ACIS [62] solid modeler, in the form of a C++ library, and the SHAPES [63] modeler as a C library, has made it possible for simulation system developers to include surface and solid definition functions as modular components of software systems.

Solid modellers can be combined with mesh generators in interactive model-building systems based on the two-step paradigm mentioned earlier (see section 2.2): first the boundary surface of the problem domain is interactively defined by the user with solid modelling techniques, and subsequently the mesh in the domain enclosed by the boundary surface is

generated with algebraic or numerical techniques, like the ones that will be described later, from point-distributions on the boundaries. A pioneering application of this approach has been the work of Cook [64], [65] and the various CAD systems, both restricted to stress analysis applications with finite elements.

2.2.3 Overview of mesh-generation procedures

The high demands on computer processing power and memory of CFD applications did not allow the extension of the associated mesh-generation methodologies to realistic engine cylinders that have been previously defined with the aid of solid modelling techniques. Indeed, until the early eighties, most techniques for generation of meshes in engine cylinders were based on analytical methods, within the restrictive framework of 2D cartesian or cylindrical-polar coordinate systems. The utility of these methods was restricted to 2D idealisations of axisymmetric engine configurations [66] whereas real engine configurations are three-dimensional and so are the air motions which they generate.

The advent of cheaper and more powerful computers made it worthwhile, by the early eighties, to extend CFD techniques and the associated mesh-generation methodologies to more irregular 3D engine configurations, with *body-fitted* meshes having mesh lines coinciding with the boundaries thereof. Most of these approaches by-passed the problem of generating a mesh in a closed, irregular region R, by transforming this region from the physical domain to a region R´ in a mathematical domain (see figure 2.5). The mesh was generated in the mathematical domain, where the complication of handling irregular boundaries was effectively removed. The mesh generation problem was equivalent to that of mapping region R into a transformed region[1] R´. Since the shape of the transformed region was predetermined, parallelepiped or cylindrical in most cases, what was needed for the solution of the mesh equations in the transformed domain was the inverse transformation[2] . This was determined either analytically or numerically (see Thompson [67] for a general review of these methods).

Several *analytical* or *algebraic methods*, based mainly on mathematical interpolation functions, have been used to fit meshes in simple, idealised

---

1 With a transformation T expressed as: $\xi=\xi(x,y,z)$, $\eta=\eta(x,y,z)$, $\zeta=\zeta(x,y,z)$.

2 The inverse transformation T´ expressed as: $x=x(\xi,\eta,\zeta)$, $y=y(\xi,\eta,\zeta)$, $z=z(\xi,\eta,\zeta)$.

2.2.2 Overview of surface modelling techniques

Since the MAC Computer Aided Geometric Design project, headed by Coons of the Massachusetts Institute of Technology in the mid-1960's, several *surface modelling* techniques have been developed to computationally define free-form surfaces in the 3D space. They are based on parametric bivariate polynomial surfaces S, where points on a surface are defined by using three polynomials of parametric variables r and t, one for each of x, y, and z:

$$S(r,t)=\big(x(r,t),y(r,t),z(r,t)\big) \qquad (2.5)$$

Cubic polynomials in both parameters r and t are used in most cases to define free-form surfaces, resulting to parametric bivariate surfaces, such as Hermite, Bezier, and B-spline ones [54]. Quadric surfaces (embracing spheres, cylinders, cones, ellipsoids, etc.), based on quadratic polynomials have been widely used in specialised applications such as solid modelling of Eucledian solids [55]. When cubic and quadric surfaces are constructed and manipulated mainly in terms of information given on their boundaries, they are called 'Coon's patches', and are appropriate for computer definition of a wide range of surfaces in user-friendly software environments.

Surface modelling methodologies are mainly implemented in software systems, called *solid modelers*, that provide interactive techniques for construction of three-dimensional surfaces and solids; these are defined in terms of 3D free-form Coon's patches, called 'patches' for brevity, and combinations thereof to form 3D objects called 'volumes' or 'solids'. A variety of solid modelling techniques have been developed in the past decade, such as *boundary-representation* or *B-rep* [56], *constructive solid geometry* [57], and *analytic solid modelling* [58], [59]. They are often available within the closed framework of CAD systems such as PATRAN [6], Geomod [60], and Alpha_1 [61]. In 1989, the release of the ACIS [62] solid modeler, in the form of a C++ library, and the SHAPES [63] modeler as a C library, has made it possible for simulation system developers to include surface and solid definition functions as modular components of software systems.

Solid modellers can be combined with mesh generators in interactive model-building systems based on the two-step paradigm mentioned earlier (see section 2.2): first the boundary surface of the problem domain is interactively defined by the user with solid modelling techniques, and subsequently the mesh in the domain enclosed by the boundary surface is

generated with algebraic or numerical techniques, like the ones that will be described later, from point-distributions on the boundaries. A pioneering application of this approach has been the work of Cook [64], [65] and the various CAD systems, both restricted to stress analysis applications with finite elements.

2.2.3 Overview of mesh-generation procedures

The high demands on computer processing power and memory of CFD applications did not allow the extension of the associated mesh-generation methodologies to realistic engine cylinders that have been previously defined with the aid of solid modelling techniques. Indeed, until the early eighties, most techniques for generation of meshes in engine cylinders were based on analytical methods, within the restrictive framework of 2D cartesian or cylindrical-polar coordinate systems. The utility of these methods was restricted to 2D idealisations of axisymmetric engine configurations [66] whereas real engine configurations are three-dimensional and so are the air motions which they generate.

The advent of cheaper and more powerful computers made it worthwhile, by the early eighties, to extend CFD techniques and the associated mesh-generation methodologies to more irregular 3D engine configurations, with *body-fitted* meshes having mesh lines coinciding with the boundaries thereof. Most of these approaches by-passed the problem of generating a mesh in a closed, irregular region R, by transforming this region from the physical domain to a region R′ in a mathematical domain (see figure 2.5). The mesh was generated in the mathematical domain, where the complication of handling irregular boundaries was effectively removed. The mesh generation problem was equivalent to that of mapping region R into a transformed region[1] R′. Since the shape of the transformed region was predetermined, parallelepiped or cylindrical in most cases, what was needed for the solution of the mesh equations in the transformed domain was the inverse transformation[2] . This was determined either analytically or numerically (see Thompson [67] for a general review of these methods).

Several *analytical* or *algebraic methods*, based mainly on mathematical interpolation functions, have been used to fit meshes in simple, idealised

---

1 With a transformation T expressed as: $\xi=\xi(x,y,z)$, $\eta=\eta(x,y,z)$, $\zeta=\zeta(x,y,z)$.

2 The inverse transformation T′ expressed as: $x=x(\xi,\eta,\zeta)$, $y=y(\xi,\eta,\zeta)$, $z=z(\xi,\eta,\zeta)$.

combustion chambers of engines. Cylindrical-bipolar systems have been used by Johns [68] and Begleris [43] to generate meshes in cylindrical chambers having offset valves or offset cylindrical recesses in the piston head. Gordon and Thiel [69] adapted and applied 'transfinite interpolation' techniques to generate meshes in cylindrical or other basic configurations. As shown by Haber and Abel [70] such techniques are computationally efficient and well-suited to interactive graphics implementation. The 'multisurface method' developed by Eiseman [71] and applied in several geometries provided formulae for mesh definition based on mesh descriptions of two boundary surfaces and an arbitrary number of intermediate control surfaces. The techniques described by Smith and Weigel [72] and only recently implemented by Dener and Hirsh in the IGG code [73], are based on the description of two, four or six exterior boundaries and the application of either linear or Hermite cubic polynomial interpolation to compute the interior grid.

The application of the above techniques becomes increasingly difficult as they are applied to engine combustion chambers that cannot be described in terms of Eucledian solids, e.g. including curved bowls or curved head recesses. Meshes in such cases are easier generated by *numerical methods*, such as the ones described by Gosman and Johns [74]. Partial differential equations, in most cases the Laplace or Poisson equation, are taken as the basic generating system, and are solved numerically. Such mesh-generation methodologies have been applied in codes such as TOMCAT [75], GRAPE [76], and EPISO [40]. A general discussion on partial differential systems based on Gauss equations or the Rieman tensor can be found in the paper by Warsi [77]. Marching techniques for orthogonal mesh generation, where the mesh marches away from one boundary, have been described by Eiseman [78]; these techniques involve prescription of orthogonal trajectories or other simpler prescriptions by the user.

The application of algebraic and numerical techniques is limited to structured and block-structured meshes, in which case they are applied in each region or block independently, in ways that ensure continuity of mesh lines where the regions interface. Gosman [74], Johns [79], and Begleris [43] made use of the above method to generate meshes in combustion chambers for flow simulation with the RPM, FMCS and EPISO codes respectively. Gosman and Johns [80], in order to generate the RPM code mesh illustrated in figure 2.6, used:

i. Algebraic techniques to generate for each position of the piston an expanding-contracting mesh in the squish region, defined by the

cylinder head and the moving piston-crown; such meshes are sometimes referred to in the literature as 'concertina' mesh systems.

ii. Numerical techniques to generate the grid in the curved bowl region of the mesh, that was generated once for all and was then translated to the relevant piston position.

Johns [68] used the same technique to generate the mesh depicted in figure 2.7 for a direct-injection, and in figure 2.8a, for an indirect-injection Diesel engine. To generate the latter he had to segment the swirl chamber into the two sub-regions shown in figure 2.8.b, to implement numerical techniques to generate a mesh independently for each subregion (figures 2.8.c and d), and to interpolate to match the interfaces between subregions in such a way as to produce mesh continuity (figures 2.8.e and f).

It was not before the mid-eighties that the above mesh-generation procedures were combined with surface-modelling techniques to generate meshes for CFD analysis in complex geometries. In 1986, Marshall, Eiseman, and Kuo [81] used algebraic procedures to generate block-structured meshes of hexahedral elements. Their method allows interactive user-definition of the bounding surface of the geometry, partition thereof into blocks, and control of the mesh properties. De la Viuda and his co-workers [82] used a CAD system for geometry-surface definition and hexahedral-cell, structured-mesh generation, in combination with separate numerical mesh-smoothing codes. Smit and Su [83] of Boeing Airplane Group, combined surface definition and multi-block, algebraic, mesh-generation techniques in a code named AGPS, to generate hexahedral-cell meshes for aerodynamic modelling use. The INGRID code [84], developed at Deutche Airbus, operates in a similar fashion. The 'interactive grid generator' (IGG), developed by Dener and Hirsch [85] for turbomachinery applications [86], and 'VisualGrid', developed by Cordova [87] but restricted to 2D meshes, are worth mentioning because they provide a large variety of mesh-generation techniques, software support for run-time integration of other programs within the meshing environment, and specially-written user interfaces. Finally, a different approach to hexahedral-cell structured-mesh generation, being used by Tsubui and co-workers [88], and Buratynski [89] is the 'background mesh method', that starts with a boundary surface and a space-filling rectangular mesh, and then distorts and enriches it to accommodate the boundaries.

The application of the above methods has been, in the case of hexahedral-cell meshes, mainly restricted to structured or block-structured meshes. Unstructured grid-generation procedures exist, but they are, in most cases,

BIBL
LONDON
UNIV

limited to meshes with cells of tetrahedral or mixed (hexahedral, wedge, tetrahedral etc.) shape. Such methods include: 'Delaunay triangulation' [90], that uses a set of points chosen from the boundaries and interior of the problem domain, and makes a tetrahedral-cell mesh in a way that polyhedra defined by planes vertical to each edge at its middle, do not contain more than one vertices, and the 'advancing-front method' [91], that makes a 3D mesh of tetrahedral cells from a 2D surface mesh of triangles, by 'growing' elements from it.

2.3 Scientific Visualisation techniques

The advent of raster graphics, in the mid-80's, and scientific visualisation, or visualisation for brevity, in 1987, has made it possible to convert entire fields of information to colour images. This has enhanced understanding of the development of such fields by bringing the eye-brain system, with its great pattern-recognition capabilities, into play in a way that is impossible with purely numeric data. The merits of scientific visualisation were immediately recognised and applied to a variety of disciplines, including medicine, biology, earth sciences, computer systems, space exploration, mathematics, and education, whereas CFD has been a late-comer in the field.

2.3.1 Overview of visualisation applications in various disciplines

A pioneering and primary application area of visualisation and a driving force in its development has been 3D display of medical data or 'medical imaging'. Even before the introduction of the term visualisation, Greenleaf [92] had presented the first use of computer graphics techniques to display blood flow data and Vannier [93] had applied CAD systems to plot and understand 3D data acquired from X-ray scans of the human body from a number of directions, a method known as 'computer tomography' (CT). In 1983, Herman and Udupa [94] presented a medical application of *volume rendering*, a technique of graphically displaying 3D numerical data that involves mapping it to a 3D mesh of cells, called *voxels* (the 3D analog of pixels), and a variety of tools to identify, handle, and display objects in a

voxel structure. Several volume rendering techniques[1] have been reviewed by Kaufman [101],[14], Levoy [102], and Shirley and Neeman [103]. In 1984, Brewster [104] presented an interactive surgical planning system (called ISP), that constructed a voxel system from projections by using CT techniques and employed an object-oriented data base of human organs, that allowed addition and removal of organs (e.g. heart, bones, etc.) In 1985, Farrell [105] made a pioneering use of animation technology in displaying cranial structures by rotating them around the head image, in this way enhancing the spatial perception of disjoint fragments thereof. His system, called IMAGE3D MEDIT (MEdical Diagnostic Imaging Tool), needed 2 minutes to form a crude animation of 8 to 16 images.

Today medical imaging is widely used in what is known in medicine as 'non-invasive vivisection', that is the study of organs and parts of the human body, without having to open it. As noted by Herman and Liu [106], it involves the following steps: At first X-rays are taken of the body from a number of directions. Subsequently a 3D voxel model is reconstructed from the 2D projections. Finally, selected organs are detected and displayed with advanced rendering techniques on the computer screen. These techniques are applied to pre-surgical planning and post-operative follow-up, in areas such as craniofacial, heart, and joints surgery, nuclear magnetic resonance imaging of flow in organs, radiation therapy[2] and diagnostic radiology[3].

Biology and chemistry applications of visualisation include imaging of complex molecular structures and visualisation of properties thereof. The structure of molecules has been visualised by means of the traditional construction set of balls (atoms) and sticks (see the interesting collection of visualisations compiled by Rosenblum [107]). Goodsell [108] has used visualisation to study the electrostatic potentials calculated around the atomic surface of macromolecules (e.g. proteins or DNA decamers), using a colour key to show transition from positive to negative potential.

---

1 E.g. object identification and surface extraction from a voxel structure, 'depth/distance-only shading' (displaying objects in the 3D space with continuous tone shading assuming a virtual light source) by calculating the illumination value of a voxel as a function of its distance from the viewer and the position of the light source.

2 Treatment planning where radioactive implants are embedded in organs.

3 E.g. study of spinal misalignment as possible cause of paralysis.

Earth sciences. Pereyra and Rial [109] have demonstrated the usefulness of visualisation techniques in applied geophysics, using examples from 'seismic prospecting', that is imaging of earth's interior based on reflected, elastic wave energy scans in order to increase the efficiency of locating oil and gas fields or predicting strong ground motions related to earthquakes. Typical visualisations they have made include coloured wave energy plots, and surface representations of geological structures. Wolfe and Liu [110] have used seismic events to identify oil reservoirs; they have transformed reflections from subterranean surfaces to voxel structures and used volumetric rendering to reveal the structure of geological layers. Their system, supported on a variety of platforms, takes six minutes to build an image on an IBM PC AT, equipped with an IBM PGA (professional graphics adaptor) card with 4 bit-planes, as opposed to 15 seconds on the IBM 7350.

Meteorology. Voxel based techniques have been used to visualise atmospheric phenomena from data sets produced either by satellites, radars and other sources, or from simulation programs, such as the LAMPS animation system developed by Papathomas [111]. Haber and McNabb [4] have used packages such as Wavefront [246] to extract, render and animate surfaces of uniform rain density in severe storms, whereas Hibbard and Santek [112] have written the McIDAS package to render and animate contour surfaces of hail and rain as well as wind trajectories, superimposed over topological maps.

Computer systems. Computer graphics techniques have been used to visualise aspects of parallel programming such as 'tasking' (distributing the computation among processors), 'synchronisation' (interaction between computing tasks), and 'data scoping' (how data is shared by tasks). The 'Stategraph' and 'Timeline' [113] systems have represented each task as a node in an ancestry tree, where nodes change representation as events are processed. Specialized graphs called 'multigraphs' [114] have been used to observe the performance of networks, thereby improving conventional statistical analysis.

Space exploration and astrophysics. Computer Graphics techniques have been used to visualise the deluge of data available from planetary study systems. Examples are the animation of the motion of planets and the visualisation of the dynamics of their magnetosphere. Nielson and Shriver [115] have summarised several techniques for combining such data with computational astrophysics simulations to create visual metaphors of abstract physical phenomena, such as black holes and the behaviour of their gravitational field.

Mathematics. Visualisation techniques have been used in exploratory data analysis, especially in multidimensional data, such as census data. Conventional 3D scatterplots have been enhanced with colour, 'glyphs' such as 'Chernoff's faces' [116], where each data point is a stylised drawing of a human face, with the size and shape of facial structures representing variables in the analysis, perspective, stereo pairs, shading or motion [117] to aid the visual perceptual system in detecting structure.

Education. As visualisation methodologies and low-cost computers (IBM PC, Apple Macintosh, and workstations) become more easily available, they are incorporated in the teaching curriculum of schools and universities. Such visualisation systems are 'Mathematica' from Wolfram Research, 'Physics Simulations' from the Stanford Faculty Author Program, and 'Molecular Editor' from Kinko's Software Exchange. Also remarkable here is the work done at the Open University for the visualisation in mathematics, physics and biology for T.V. broadcasts. Nevertheless, a report by Cunningham [118] concludes that powerful visualisation systems are not yet cost effective for educational use.

2.3.2 Overview of software systems for visualisation of CFD data

Although, as noted earlier, CFD has been a late-comer in the field of applications of visualisation, it has been a very dynamic one, with a variety of visualisation programs targeting only the CFD market and CFD options being now available with most generic visualisation systems.

One of the first visualisation programs, PLOT3D [119], had the capability of displaying many quantities associated with a physical flow field obtained from a CFD calculation. Such quantities were of two types, scalar and vector. Scalar quantities were displayed with the use of contour lines and surfaces, profiles, carpet plots and variations of colour. Vector fields were displayed using arrows that give direction and magnitude.

Recent developments, that became available long after the initiation of the research effort presented here, allow immediate access to visualisation techniques, and have been grouped in flexible systems called *toolbox* environments. In 1989 Upson and his colleagues at Stellar computer described the 'Application Visualisation System' (AVS) [120]. The following year Dyer described a conceptually similar system, the 'animation production Environment' (apE) [121]. Both systems were designed to be largely discipline-independent in that they offered a set of general-purpose techniques, or *tools*, for application to a variety of disciplines. They also allowed data exchange between program components that run

simultaneously – known as *inter-process communication (IPC)* – and therefore could be used as an interface to a *distributed processing* system, in which multiple programs and computers are applied collectively and appropriately to a range of tasks; several distributed processing systems are reviewed in subsequent paragraphs of this section. Two other common features of these systems were the provision of animation tools, and their ability to model new user-defined data types and file formats from simple descriptions, thus easing translation of custom-designed simulation data formats.

In the last two years, at least four other notable visualisation systems have been developed. Bancroft and his co-workers at NASA Ames, produced the 'Flow Analysis Software Toolkit' (FAST) [5], based on existing visualisation tools such as PLOT3D [119], GAS [122], RIP [123] and SURF [124]. FAST includes an inter-process communication subsystem which fully exploits shared-memory computer architectures, and provides a 'CFD calculator' – an interactive tool for deriving secondary CFD quantities from the primary simulation results.

Recently 'Explorer', a toolkit from Silicon Graphics Inc., designed by Upson [125] was released. Explorer offers more flexible design of the user interface in order to make best use of limited screen-space, a new approach to data-flow control which minimises communication between the controlling program and other modules, a graphical editor of data models, a graphical aid to module implementation, and an object-oriented programming interface in C++.

Apart from these systems the 'Data Visualizer' [126] and [246] from Wavefront Technologies is noteworthy as a single-process program for visualisation designed primarily for end-users rather than developers, and 'Visual3' [127], another single-process visualisation program mainly for aeronautical applications, offers yet another approach to program integration, this time using FORTRAN only. The Data Visualizer and Visual3 are both claimed to offer much improved performance over their toolkit counterparts.

2.4 Computer software and hardware for animation

The foundation of computer-assisted animation lies within the field of a variety of disciplines, such as computer graphics, geometric modeling, visual psychology, video technology, and film-making. Its history dates back to 1965, when Knowlton, of Bell Laboratories, New Jersey, USA, presented a

computer-animation system that was intended for educational use. Knowlton's system was using 8 tones of colour, produced by three bits of colour information for each pixel (3-bit planes), in a raster image with 252 x 184 pixels and a simple animation control language. A sequence of animated diagrams, lasting 20 minutes and containing 25000 frames of which 3000 were different, was specified by 2000 lines of code and took two man-months to produce.

The advent of raster graphics in the early eighties has given computer-assisted animation a big boost, with applications today ranging from education [118] to the entertainment industry [129], and from industrial control systems [130] to flight simulators [131]. Its scientific and engineering applications are mainly in the area of 'dynamic visualisation', where simulation results are studied by blending visualisation and animation techniques. A variety of methods, alongside the hardware available to the animation scientist, are reviewed in the following paragraphs.

2.4.1 Applications of animation to scientific visualisation

Graphics workstations, video input/output peripherals, and animation software are being used by scientists and engineers to produce dynamic visualisations of simulation data. Two easily discernable approaches have evolved:

- that of recording high-resolution rendered images on tape;
- that of *live video* where sets of images, precalculated and stored in memory or generated and displayed at the same time, are played back in real-time on the workstation screen.

Recording high-resolution rendered images on tape, the only way to produce animations until the late 80's, made it possible to generate high-quality animations without the need for dedicated video boards and large memory. The new generation of graphics workstations and video boards have made it possible to use live video techniques to preview the animation, prior to recording on tape.

Live Video. The most common type of live video, also called 'cine sequences' or 'video windows', is one in which the point-of-view, called *camera,* and a virtual *light source,* are fixed and the object of the animation rotates. The reflections across the surfaces in the data enhances comprehension of the scene. A variation of the above, frequently used in volume rendering, is a sequence in which the object, the light source and the camera are fixed and a cutting plane moves [98]. Fuchs, Levoy, and Pizer [132]

have done live video with fixed object, light source and camera and changing voxel opacities to study in detail the complex structures of the human head. They achieved that by computing a sequence of images in which the opacities of all voxels lying between the observer and the clipping plane are reduced to zero, and each image in the sequence has the clipping plane in a slightly different position relative to the data. Hibbard and Santek [112] have produced cine sequences of isosurfaces of sulfur dioxide concentration and wind trajectories of calculated atmospheric data. The temporal resolution used by their system was 5 to 10 frames per second, from a data set containing 121 time steps, in a 25 x 35 x 17 -point rectangular orthogonal mesh. Kluksdahl and Kriman [133] have produced an automated system for dynamic visualisation of isopotential surfaces in resonant tunneling diodes, by generating each image through a graphics library and passing it to a video control system.

Edwards [134], at United Technologies Research Centre, has used commercial animation software provided by 'Intelligent Light' to visualise flow within gas turbine combustors and at the exit from the gas turbine combustor entering a turbine, as well as in the visualisation of supersonic exhaust jets interacting with an external stream. The display objects that he animated were semi-transparent 3D contour surfaces and cutting planes through the flow domain. Intelligent light, like most other animation systems, provide mechanisms for control of the animation and the dedicated hardware involved. Such mechanisms are reviewed in the following paragraphs.

2.4.2 Overview of animation languages

It is common today for animation software systems to provide a user interface for the creation of the animation. This often takes the form of a script-writing language, that is used to define actions and control the motion of objects, camera, lights, etc. Such languages can be based either on a textual, programming language style paradigm, or a visual one. Since 1965, when the first animation language was designed and used at Bell Laboratories, several new ones have been developed. They are classified by Foley et al. [54] in *linear-list notations*, *general-purpose languages* and *graphic languages*.

A *linear-list notation* system for animation scripting, that became very popular, was suggested by Catmull [135]. In this system, each event in the

animation was described by a starting and ending point and an *event* that is to take place[1].

The ASAS *general-purpose animation language*, developed by Reynolds [136] in 1982, was built on top of the general-purpose programming language LISP. It included primitive entities such as vectors, colours, lights, patches, solids, hierarchical groups of solids called 'objects', geometric transformation of objects (with commands such as up, down, zoom-in, zoom-out etc.), and points of view. The advantage of ASAS over linear-list notations, was its ability to handle hierarchical objects, at the expense of increased skill required by the animator.

However, a problem with the above textual animation languages, especially in the cases where real-time previewing is not available, is that it is difficult for the animator to imagine what will take place in the animation just by looking at the animation language script. This problem was overcome by the *graphic animation languages*, that instead of explicitly writing out commands describing actions, provided a visual metaphor of the action. Some of the earliest work on this area was done by Baecker in 1969, who introduced the notion of curves as a means of motion representation in the GENESYS animation system [137]. Advanced graphical animation languages include 'DIAL', developed by Feiner and his co-workers [138] and 'S-Dynamics' developed by Symbolics [139].

Finally an enhancement to the linear-list notation was the SCEFO (SCEne FOrmat) animation language, developed at Brown University, USA [140]. Scefo supported object grouping and hierarchy, programming language features such as variables and expressions and a whole environment called BAGS, of which SCEFO is the core, surrounded by add-on renderers, lighting-models etc.

2.4.3 Main techniques for animation control

The control of the animation is independent of the language that is used for its description. Most animation control languages can integrate various animation control tools. Such tools range from fully explicit, like the Bbop system [141], where the animator controls explicitly the position and

---

[1] For example the action: move object SPHERE by 0.025, 0.008, and 0.01 metres in the x, y, and z directions respectively, between frames 1 and 30, would be notated in a statement such as:
1, 30, MOVE, "SPHERE", 0.025, 0.008, 0.01.

attributes of every object in the scene, to so-called 'knowledge-based systems', where the animation is controlled by high-level descriptions, that automatically generate explicit controls. Such systems are:

- 'Procedural models' such as particle systems [142];
- 'Constraint-based cinematic and dynamic models', such as the ThingLab system [143]. A very successful application of such models was the animated model of 'Luxo Jr.', from Pixar Animation [144], where dynamic models using constraints were used for the calculation of the path of a five-piece jumping desklamp;
- Systems for tracking live-action [145].

2.4.5 Hardware for animation and video production

The following figure, inspired from Phillips [146], delineates the evolution of animation capability. Although software and hardware developments are tightly bound in such an evolutionary process, only hardware evolutions are sketched in this diagram.

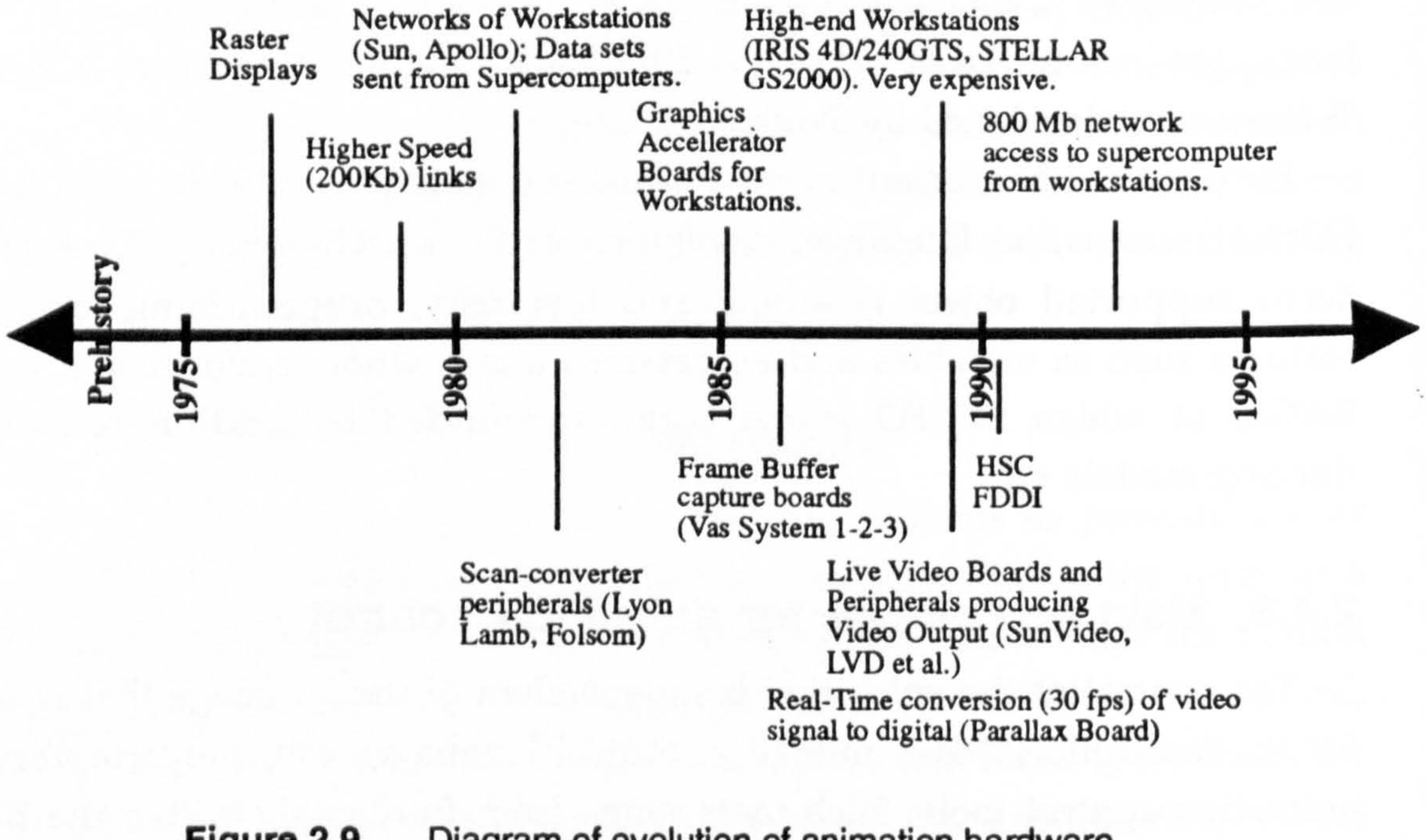


Figure 2.9 Diagram of evolution of animation hardware.

While the use of animation in scientific visualisation is not new, a recent confluence of technologies has revolutionalised the way animation is used in this field. These developments appear on the timeline beginning in the early 1980's. By the mid '80's, the key role in the computer-assisted animation workbench had been taken by the workstation. Large data sets of

complicated simulations were calculated in remotely located supercomputers and transferred at a rate of 200 Kbits per second, to a workstation. There, frames for animation were rendered, scan-converted by peripheral devices such as the Lyon Lamb or Folsom, and transferred, frame-by-frame, on to video, in what was a slow but automated process. The advent of high-end workstations[1] and inboard peripherals for live-video that fit at the back plane of the computer[2], in the end of 1989, allowed live-video animation of simple images and video output at a rate of 30 frames per second (fps) which is sufficient for smooth animation. Such animations could be transferred to a workstation, for local viewing and recording, by means of the 'Parallax' board[3], that is able of digitising video signals in 'real time' (30 fps).

While the graphics performance of such workstations was sufficient for live video, the data delivery rate to them was still limited by network, in most cases of 'Ethernet' type, throughput. This constraint was overcome in 1991, with the announcement of the HSC network standard, for which all major workstation manufactures provide interfaces, and fiber-based FDDI networks, expected to give transfer rates of 800 Mbits per second. This is expected to make it possible to dynamically visualise large data sets by splitting visualisation tasks between different computing systems connected to such networks, in what is called 'distributed visualisation'. This will allow visualisation of results while the solution is still in progress - called 'simulation tracking' - by calculating 1024 x 1024 x 24-bits resolution images on a supercomputer and transmitting them to a workstation at a rate of 30 fps for local viewing. Such a system is the one developed by Bankroft and his co-workers at NASA Ames Research Center [153] for animation of aircraft bodies and lines representing paths of particles seeded in the flow around them; it tracks the simulation by distributing the various visualisation tasks on an HSC network of supercomputer and workstations, using the Graphics Animation System (GAS) for dynamic visualisation and the Abecas system

---

1 Such as the POWER IRIS 4D, and Challenge by Silicon Graphics, with the capability to render more than 100,000 full-coloured polygons per second [147].

2 Such as 'SunVideo' by SUN Microsystems, 'RGB VIEW2000' by RGB Technology, and 'Live Video Digitizer' and 'IndigoVideo' by Silicon Graphics (released in 1993 for the 4D/GTX and IRIS Indigo respectively).

3 By Parallax Graphics.

for recording on videodisc or tape. Distributed visualisation, simulation tracking and in general interaction between the user and the simulation, are addressed in the remaining part of this chapter.

2.5 User interaction with the simulation process

It was mentioned earlier that high-speed networks and graphics workstations will make it possible to visualise the results while the solution is in progress, in this way establishing a way of interaction between the user and the simulation. As far as interactive control of the simulation goes, Marshall, Kempf, Dyer and Yen [3] have classified the different visualisation techniques in three categories: *post-processing*, *tracking*, and *steering*. Each technique varies in the degree of integration between the execution of the simulation and the viewing of the resulting images.

Post-processing is the most common technique used today. According to this, results from the simulation are, in some organised fashion, saved in a file that is at a later time read by the visualisation software system (see figure 2.10). There is no coupling to the original simulation; the data may have resulted from a supercomputer calculation done weeks or months before. Such software systems are Plot3D [119], [148], Surf [124], GAS [122, AVS [120], NCSA, Wavefront Data Visualizer [126], FieldView [149], PV-Wave [150], as well as all the CAD systems. Some systems specialise in the production of high-quality images and animations, such as the Wavefront Data Visualizer [126] and Intelligent Light 3DV [151], [152], but usually at high prices.

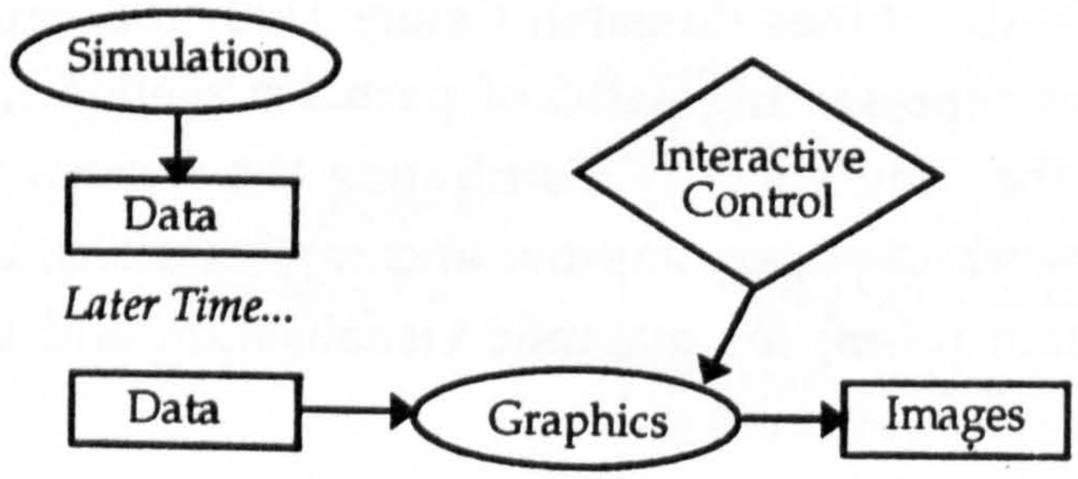


Figure 2.10 Interaction with the simulation: Post-processing.

An advantage of post-processing is that data is stored and hence it can be examined repeatedly using different mappings and techniques for production of high-quality images and animation thereof.

Tracking is the viewing of images as the calculation proceeds but without control over the parameters of the simulation other than termination (see figure 2.11). In the case of calculations requiring large amounts of time,

Marshall, Kempf, Dyer and Yen claim that the resulting images can be collected in a window of the workstation, so that the user can check the results occasionally while doing other tasks.

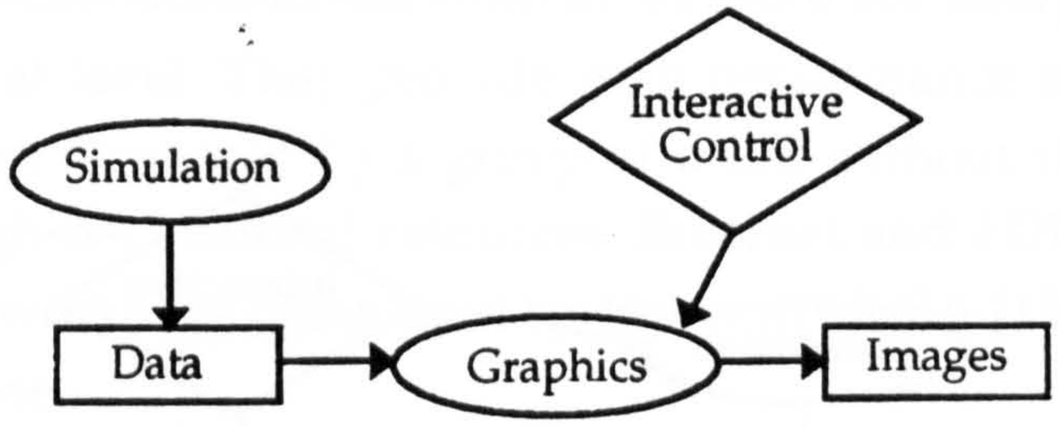


Figure 2.11 Interaction with the simulation: Tracking.

Tracking allows better utilisation of computational resources, because:

i. users can generate images directly from data as they are generated and choose what data to save;

ii. erroneous calculations may be detected and interrupted instead of running to completion;

iii. when the simulation is performed on a remote super-computer, data are sent directly to the local disk on the workstation, instead of first being saved on the supercomputer's disk and then copied across.

A disadvantage of this approach is that data may be available only at the time of the calculation. To avoid this problem, systems for tracking should provide the ability to save the data so that the user can examine the results off-line. Postprocessing systems that have been modified at NASA Ames Research Center for tracking are GAS and RIP developed by Bancroft and others [153].

Hybrid postprocessing - tracking systems have been suggested, where interactive techniques are used during the development and debugging of the model and off-line postprocessing techniques are used for more thorough study of the data or presentations. Such a system is the RIVERS-SDG, created by Haber [154] at NCSA.

Steering is the interactive control of a computational model during execution while viewing the results of the calculation graphically, (see figure 2.12). Steering provides a mechanism for controlling aspects of the model, such as computational model parameters, grid resolution, and time intervals, during execution. It also provides a mechanism for changing the focus of interest in viewing the output of a computation; for example a computation may initially use a course grid and as interesting flow structures develop, the focus may be changed to increasing the grid

resolution and decreasing the timestep. A popular steering application is flight simulation, where the 'pilot' uses a series of visual cues to steer the computation to the desired result. In CFD the underlying concept is the same, only the graphic feedback is not as obvious as in the case of the flight simulator.

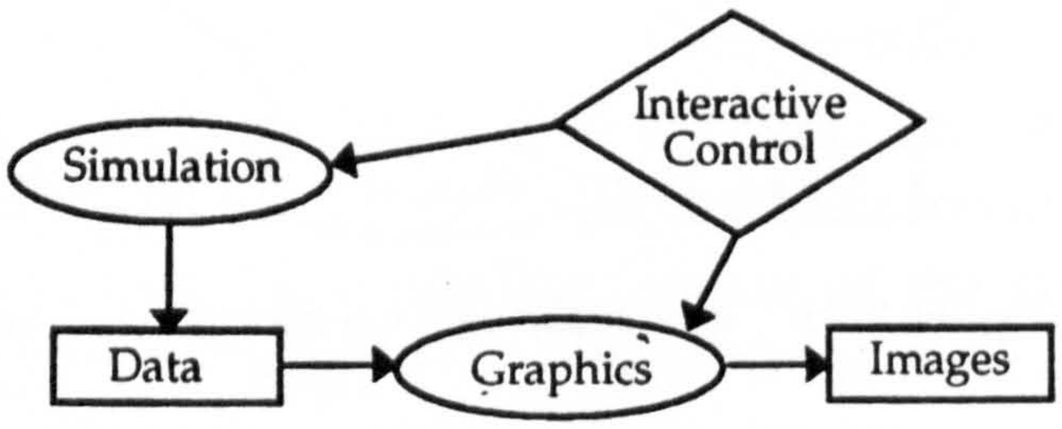


Figure 2.12 Interaction with the simulation: Steering.

Steering involves mechanisms that allow data exchange between user-input, model-building, solution, and visualisation programs that run simultaneously on different processes. A code that provides steering of both the model building and the solution phase is IZ (Interactive Zoner) from NASA [153], an interactive 2D grid generator tool running on a workstation connected to a 2D CFD flow solver running on a supercomputer.

Tracking and steering CFD simulations is a computationally intensive process. It requires high-performance computers and high - bandwidth networks, that at the time of writing are becoming available but at high prices. The current experience of long waiting times on supercomputers, combined with lack of high - bandwidth networks, imposes post-processing as the only realistic option for the present work.

2.5.1 Distributed simulation and visualisation

In the case of large data sets, tracking and steering can be achieved only through splitting tasks of the simulation, such as numerical solution, visualisation, mesh adaptation to the flow, and user-input, to several computing components. This process is called *distributed simulation* and it involves high-, middle-, and low-tier computing machines, connected to high-speed networks.

High-tier machines are supercomputers (e.g. CRAY Y-MP), available at a regional or national level and connected with other resources in high-speed networks, with maximum transfer rate of 10 to 100 megabytes per second. Such networks are reserved for massive data transfer using large packet sizes, for uses such as distributed processing, real-time visualisation and

accessing shared high-performance mass storage systems. Such systems and their application to distributed simulations at the Los Alamos National Laboratory, have been reviewed by Phillips [146].

Middle-tier systems such as the Alliant VFX-80, are computers available at the departmental level. They provide high-performance multi-processing capabilities that can be shared by a group of users without the compromises necessary with high-tier national resources. Ethernet and FDDI (Fiber Digital Data Interface) networks are being used by Haber at NCSA [154] to take on the middle-tier function.

Low-tier machines are workstations and personal computers, used at a desktop level. They are used for interactive user input and graphical operations. Their graphics performance can be improved by the use of peripheral graphics boards, that are used to accelerate the rendering operations. Such boards include the Raster Technologies GX4000 boards and the TAAC-1 application accelerator for volumetric rendering, for Sun workstations, and parallel graphics boards ('elan', 'extreme graphics' et al.) for Silicon Graphics workstations.

Chapter 3

Reciprocating engine analysis and the SPEED code

The objective in this chapter is to review engine types and operating conditions, for the non-scientific reader, and to describe the flow processes in engines, in a way that defines the geometry construction, initial and boundary conditions, simulation, as well as the visualisation tasks for the designer of real engines. Subsequently the methodologies underpinning the SPEED code, as well as its application to reciprocating engine design will be summarised, in a way that defines the mesh generation, problem definition and visualisation needs that will be discussed in later chapters.

3.1 Engine geometry and operation

Since the invention of reciprocating engines in the end of last century, a number of different configurations has evolved. In the subsequent paragraphs a brief review of engine types, operation, and knowledge of in-cylinder flow processes, obtained with experimental probing techniques and three-dimensional computational models, will be made, prior to discussing the computational tasks related to SPEED.

3.1.1 Engine types

There are many different types of internal combustion engines. They can be classified by application (automobile, truck etc.), basic design (reciprocating, rotary), working cycle (four-stroke, two-stroke cycle), valve or port design and location, type of fuel (petrol, Diesel, natural gas etc.), method of mixture preparation (carburation, fuel injection into the intake port or inlet manifold, fuel injection into the engine cylinder). Depending on the method of ignition, they can be classified to:

- spark-ignition (SI) engines and
- compression-ignition (CI) engines.

Normally the former are petrol, whereas the latter are Diesel engines.

In conventional *spark-ignition* engines each cylinder receives fuel premixed with air and the mixture, after being compressed, is ignited with a spark plug. In the so-called 'stratified-charge engines', fuel is injected directly

into the combustion chamber and then ignited with a spark plug, as it mixes with air [155]. Several spark-ignition engine configurations are shown in figure 3.1, taken from Barnes and Moss [156].

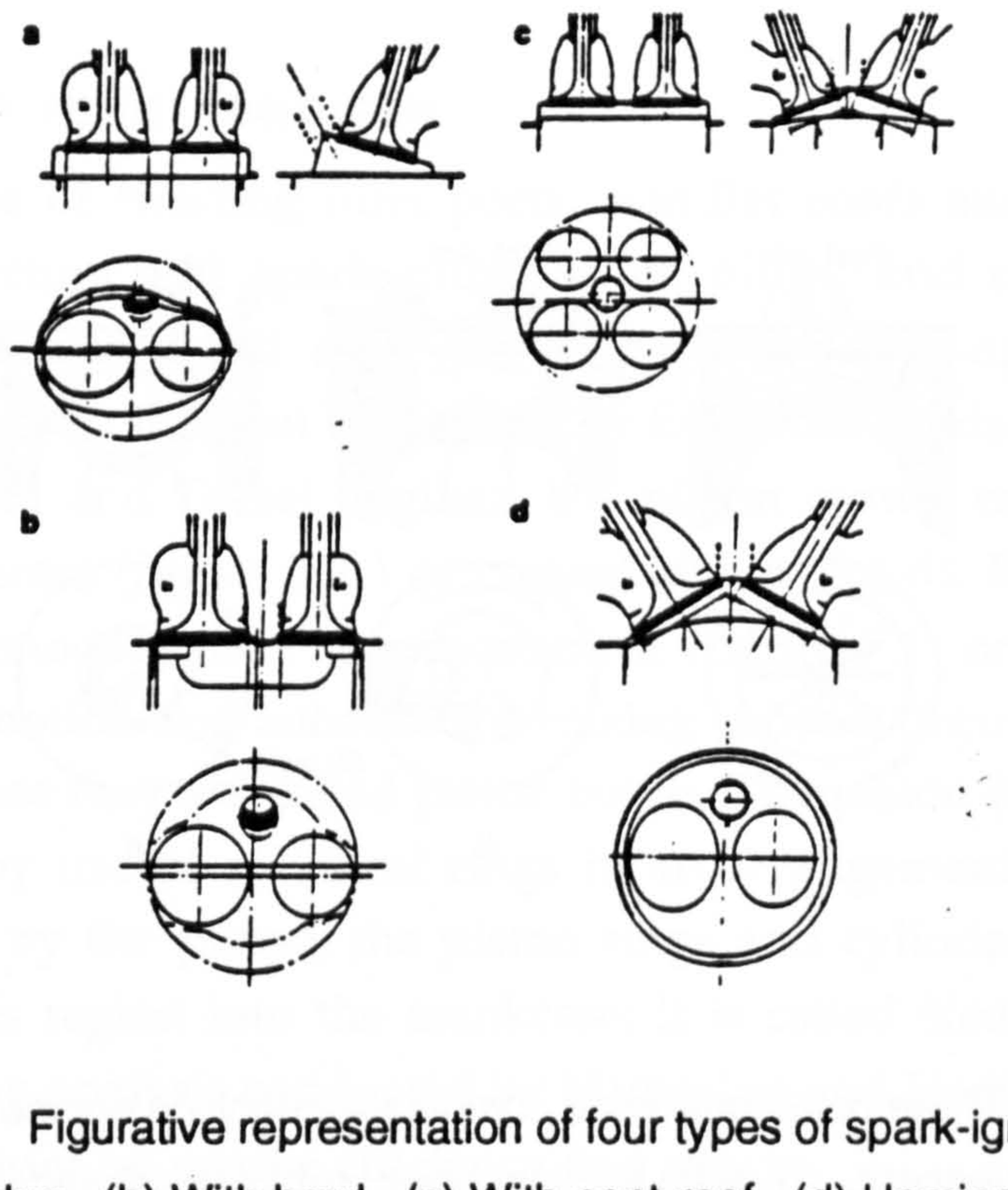


Figure 3.1 Figurative representation of four types of spark-ignition engines.
a) Wedge. (b) With bowl. (c) With pent-roof. (d) Hemispherical.

In *compression-ignition* engines the fuel is injected through a single- or multi-hole fuel injector towards the end of compression in an environment of high pressure and temperature sufficient to ignite it. Popular configurations are shown in figures 3.2 and 3.3, taken from Heywood [157]. In some models a glow plug is used to assist starting in the case of cold weather.

Compression-ignition engines are further classified by their combustion chamber design to:

- Open chamber or direct injection (DI) engines, in which the fuel is injected directly into the combustion chamber which can be of disc, wedge, or bowl-in-piston design, like the ones shown in figure 3.2; and
- Divided chamber or indirect injection (IDI) engines, in which fuel is injected into a smaller chamber communicating with the cylinder through a narrow passage as it is shown in figure 3.3. The smaller chamber is designed in such a way that fuel is injected through a

pintle nozzle as a single spray in an environment of high pressure and temperature, in a flow pattern that permits better fuel-air mixing. If the geometry induces a swirling flow structure inside the smaller chamber, then it is called swirl chamber, otherwise prechamber.

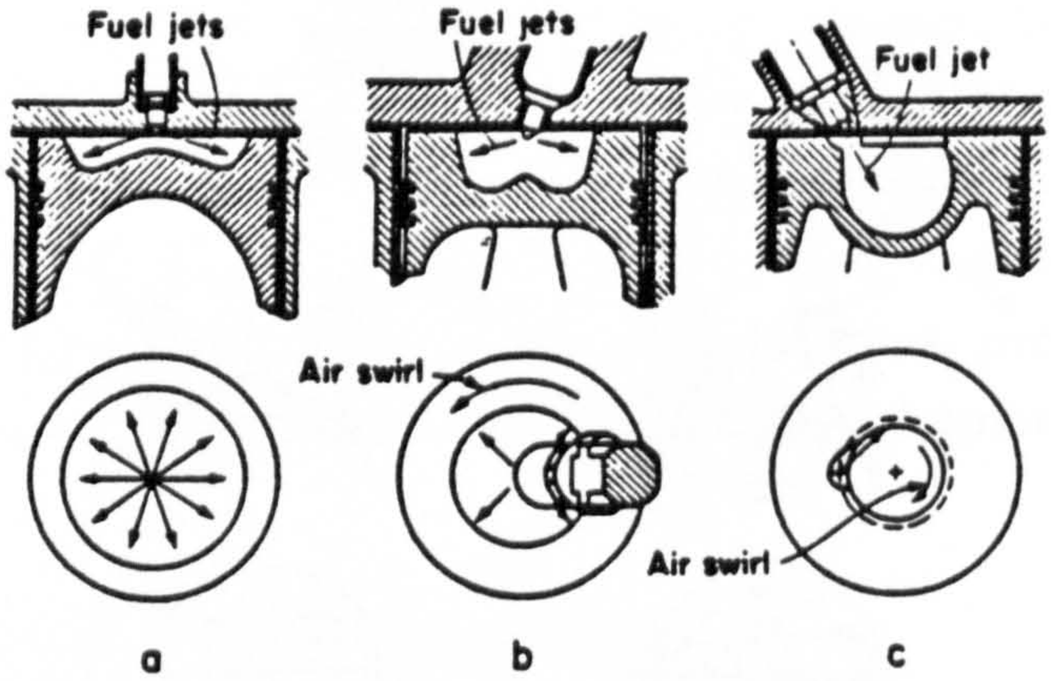


Figure 3.2 Three commonly used compression-ignition engine configurations. (a) Quiescent. (b) With toroidal bowl. (c) With spherical bowl.

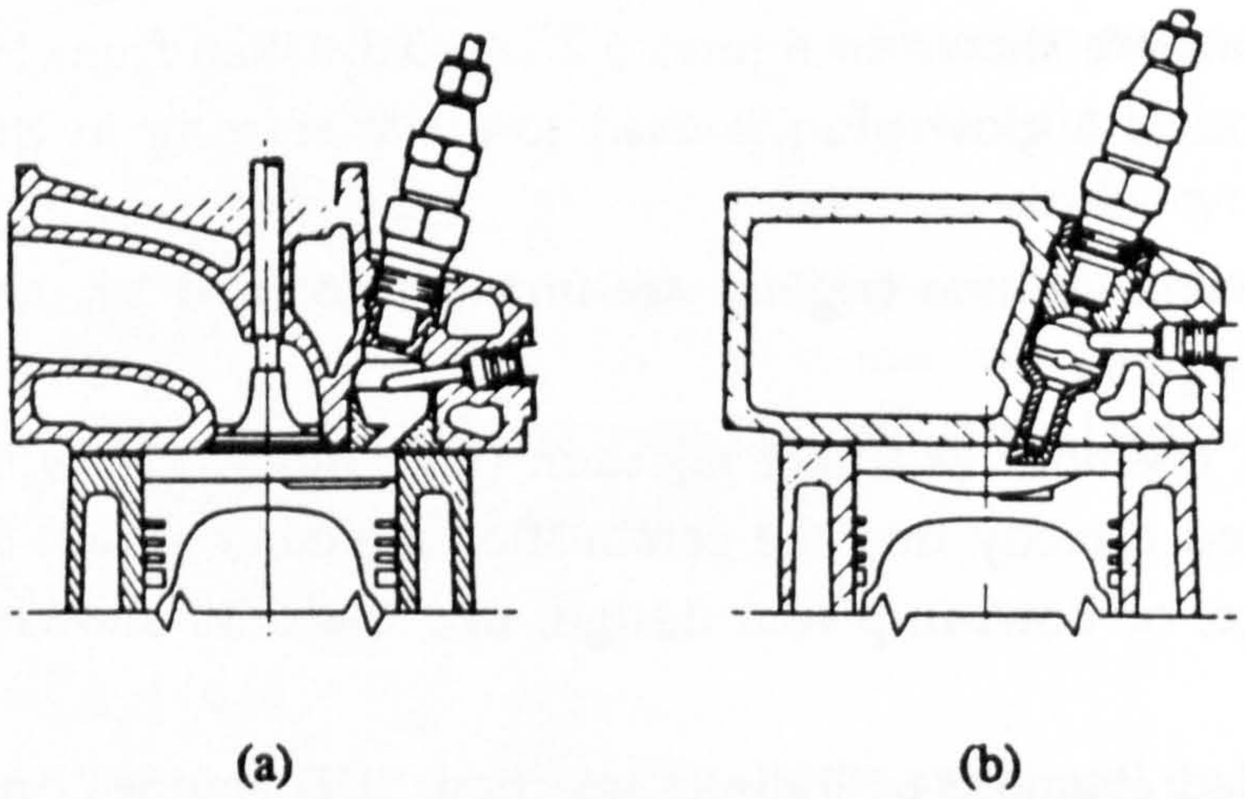


Figure 3.3 Indirect injection Diesel engines with (a) swirl chamber (b) pre-chamber.

Finally, engines can be classified by the type of mixing of the charge to:

- Homogeneous charge engines, in which fuel and air are mixed prior to their entry into the engine cylinder; they are mainly petrol engines.
- Non-homogeneous charge engines, in which fuel is injected into the engine cylinder; they are mainly Diesel and stratified-charge engines.

3.1.2 Geometric features

The presence of twisting inlet ports, non-flat roofs and piston crowns, protruding injectors and spark plugs, glow plugs, and crevices result in combustion chambers and inlet ports that have a shape that deviates remarkably from one that can be defined by Euclidian solids.

In both petrol and Diesel engines, the piston crown can be flat (figure 3.1.a), with a recess (figure 3.2) or curved (figure 3.1.d). Recesses are used either to accommodate the valve movement (figure 7.1) or to influence the in-cylinder air motion and enhance the mixing process (figures 3.1.b, 3.2, and 7.1), in which case they are called piston bowls. The piston is normally sealed from the wall by use of spherical rings in an arrangement which creates a crevice defined by the piston, the piston rings and cylinder wall. Some gas flows out of this region into the crankcase; it is called *blowby* and, as it has been shown in an analysis conducted by Namazian and Heywood [158], it has a significant impact on engine efficiency and power.

In both petrol and Diesel engines the cylinder head may be flat (figures 3.1.b and 3.2), wedged (figure 3.1.a), pent-roofed (figure 3.1.c), hemispherical (figure 3.1.d) or generally curved.

Protruding glow plugs can be found in Diesel engines. These can be horizontal, vertical or of any direction. Usually they are sealed to the cylinder head in a way that creates crevice regions.

Protruding fuel injectors are found in petrol, Diesel or DISC engines. Often fuel injectors protrude in the piston recess region. Usually they are sealed to the head in a way that creates a crevice region.

3.1.3 Charge motion and combustion within the cylinder

The major role of fluid mechanics within the cylinder is firstly to prepare the mixture of fuel and air for combustion, in Diesel engines, where fuel and air are not premixed before entering the chamber, and secondly to control the combustion process through large- and small-scale mixing in SI and Diesel engines .

Insight into in-cylinder flow and combustion has been obtained with experimental probing techniques and recently by three-dimensional

computational models. The measurement of the in-cylinder flow is achieved by a variety of diagnostic techniques, the most established ones being hot wire anemometry, Schlieren photography [159], laser Doppler anemometry (LDA) [160], and the light paddle wheel [161] for swirl measurements. Characterisations of sprays and flames in the cylinder have been obtained by means of laser shadowgraphy [162], high-speed photography [163], and back-illuminated photography [164]. The above knowledge has been recently enhanced by the application of three-dimensional computational models for fuel flow, turbulence modelling, spray formation, combustion and turbulent flame structure, on computational grid arrangements of several thousand points on middle and high-tier computers.

In the subsequent paragraphs the flow processes of the engine cycle are described in sequence of occurrence during the four-stroke cycle. A comprehensive review of current knowledge of in-cylinder flow processes, as well as an examination of experimental and computational techniques can be found in the papers by Arcoumanis and Whitelaw [165] (also containing an accomplished survey of literature on major research efforts in the field) and Gosman [21].

3.1.4 Intake stroke

Gas enters the cylinder through the inlet valve opening, in the fashion shown in figures 3.4 (a to d), taken from Theodoropoulos [24].

Due to the fact that the intake valve exit surface or curtain is the minimum area for the flow, gas velocities here form what is known as the valve jet, featuring the highest velocities set up during the intake stroke. The mean velocity profiles on the curtain may include sign reversal near the edges of the valve curtain, indicating early flow separation, which reduces the effective breathing area, as it has been shown by Vafidis and Vorropoulos [166]. A prominent feature during induction is the annular jet formed by the incoming gas, shown in figure 3.5, taken from Sarantinos and Lange [247]. When emanating from off-centre valves, it often reaches the liner wall on the near side, where it penetrates tangentially, forming a so-called wall jet, while on the diametrically opposite side it usually remains free.

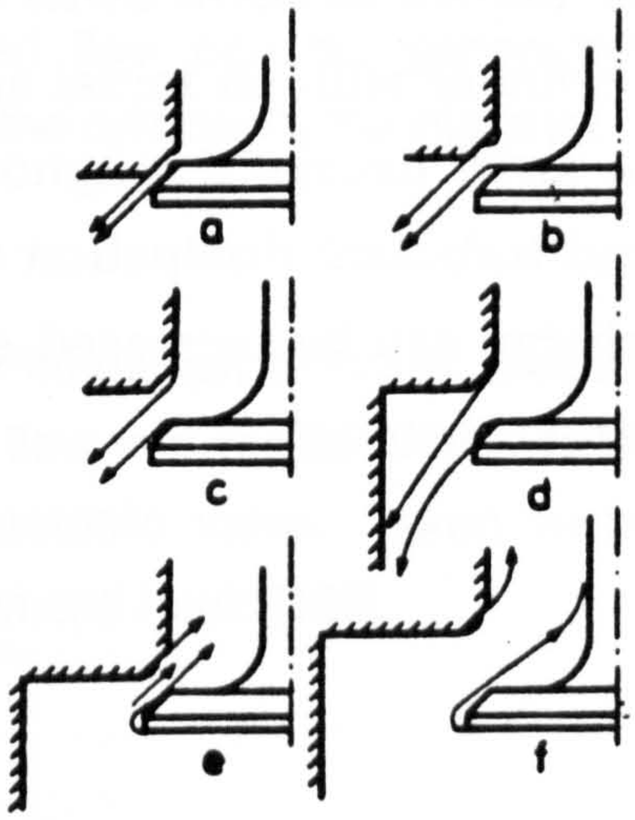


Figure 3.4
Sketches of inlet and outlet flow.

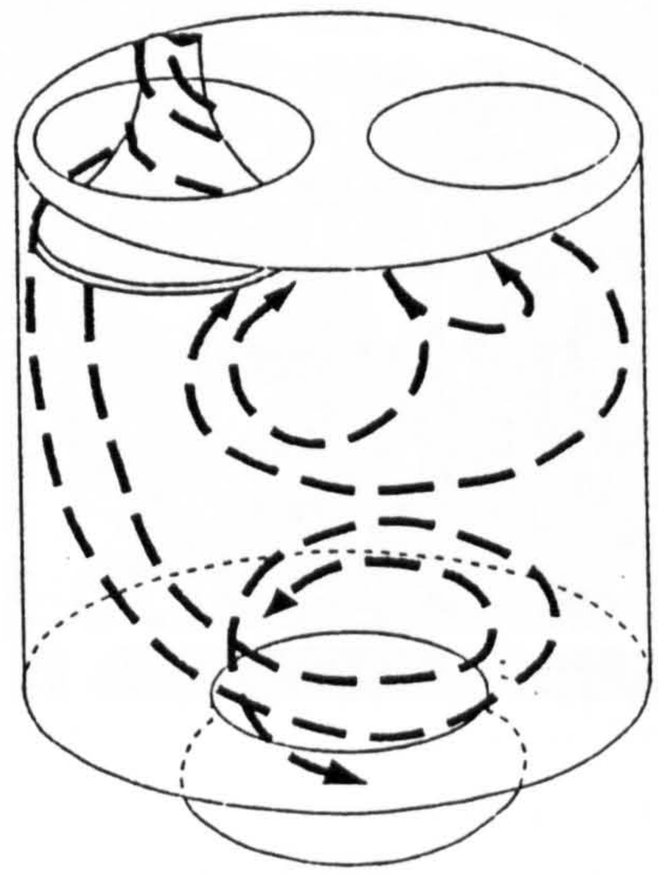


Figure 3.5
Sketch of in-cylinder flow pattern.

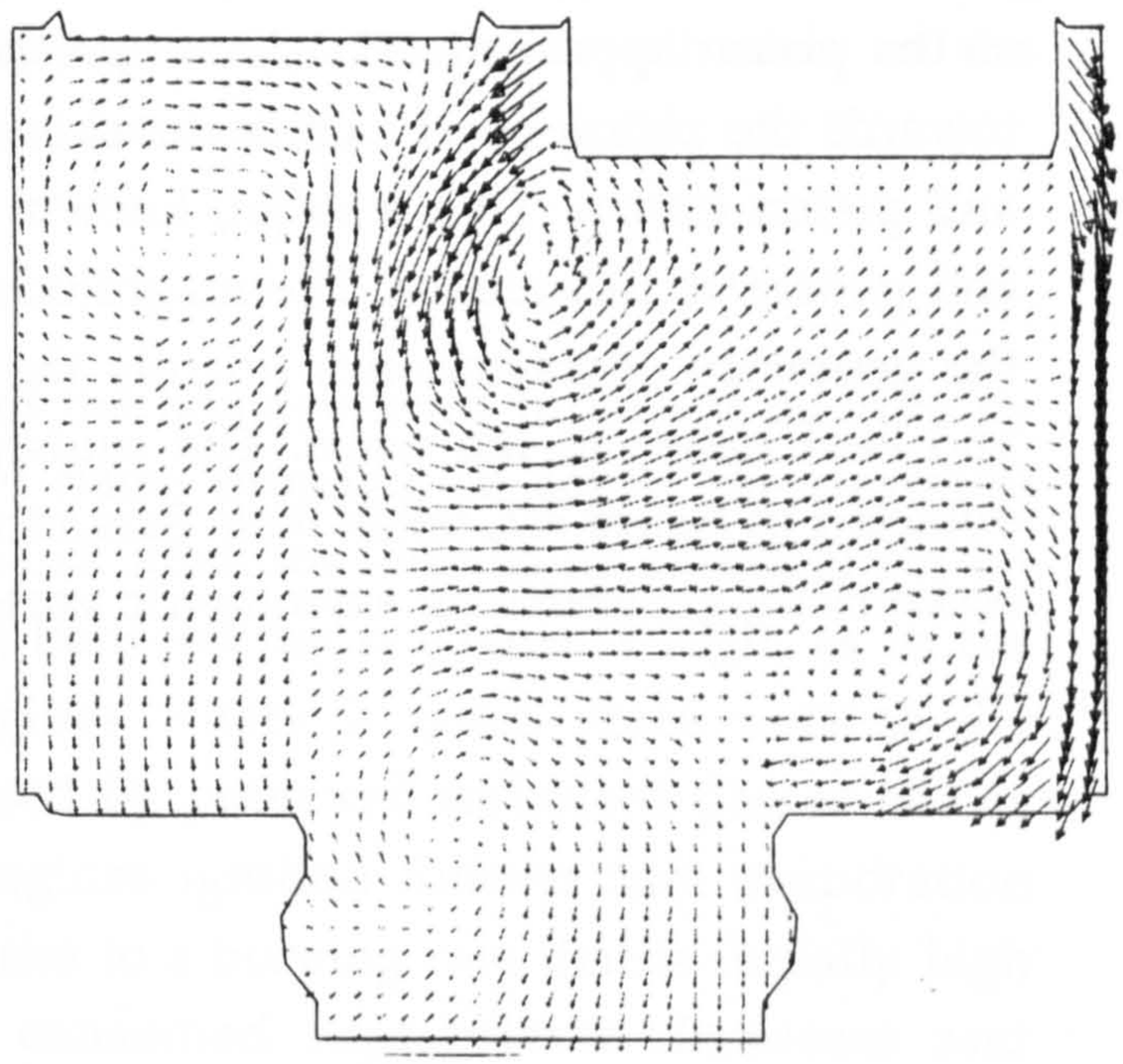
The valve jet interaction with the cylinder walls and the moving piston creates, apart from wall jets, large- as well as small-scale recirculation patterns, like the ones shown at the top of figure 3.6, that depend mainly on the cylinder and inlet system geometries. The evolution of these during the intake and compression strokes has been numerically predicted, among others by Gosman and Johns [40] in DI Diesel engines, Ahmadi-Befrui [23] in an axisymmetric model engine, Begleris [43] in of-centre-valve Diesel engines with re-entrant toroidal bowl-in-piston, Theodoropoulos [24] in a model Diesel engine with square bowl-in-piston, Gosman and Marooney [46] in a DI Diesel engine with bowl in piston, and a pent-roofed petrol engine, Sarantinos and Strauss [167] in a Volkswagen DI Diesel engine modified for optical access, and Sarantinos and Lange [247] in the same engine but in its original form. The above studies have concluded that the interaction of the valve jet with the wall results in counter-rotating axial tumbling vortices during early induction, located in the region where the liner and the cylinder head join and below the valve face (see figure 3.6, middle). Towards the end of the induction stroke, another recirculation zone emerges near the periphery of the piston face (see figure 3.6, bottom).

'Directed' or 'helical' inlet ports [168] or shrouded valves can generate a rotation of the charge about the cylinder axis, known as *swirl*. Swirl can be created by bringing the intake flow into the cylinder with an initial angular momentum. The angular momentum of the swirl decays throughout the intake process, due to friction at the walls and turbulent dissipation within the fluid. However, swirl velocities in the charge can be increased during compression by suitable design of the combustion chamber.

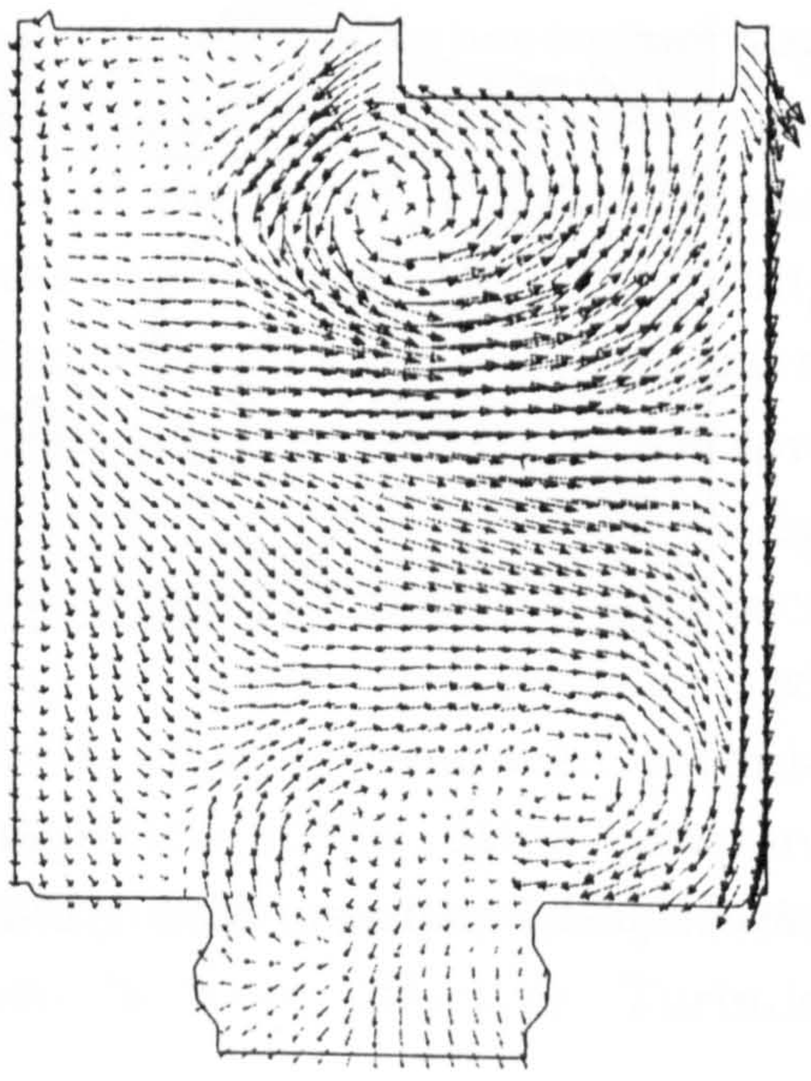
Figure 3.6
Rotated flow patterns generated within the cylinder by the intake jet



Top: Photograph of streak lines in water flow into model engine with axisymmetric valve. Taken from Ekchian and Hould [169].



Middle: Calculated flow field on an axial plane in a VW DI Diesel engine with one offset intake valve at 90^{o}. Calculation performed by Sarantinos and Strauss [167] with the SPEED.DC code.



Bottom: Same as above, but for 150^{o} CA ATDC.

3.1.5 Compression stroke

The induction process terminates with the closure of the inlet valve (IVC) and establishes the initial conditions for the subsequent compression process, reflecting the inlet port and valve geometry configurations and the small effect of piston geometry on the flow development during induction. As the piston approaches top dead centre (TDC) of compression, the effect of inlet geometry diminishes and the piston and cylinder head geometry plays the dominant role in the structure of the mean and turbulent velocity fields. In order to increase induction-generated swirl and turbulence intensity, *bowls* of various shapes have been incorporated into the piston or the cylinder head to enhance the fluid motion at the point of the engine cycle where it is mostly needed, the point of ignition. In bowl configurations and as the piston approaches TDC, compression induces a radial inward motion towards the piston-bowl axis in the clearance gap between the cylinder head and piston crown. This wall jet, known as *squish*, distorts the swirl profile and generates turbulence. The magnitude of the squish velocity depends on the geometry characteristics (mainly the squish area), operating conditions and heat losses to the walls. The squish induced flow in a piston bowl is depicted in figure 3.7, taken from Sarantinos and Lange [247], where the bowl eccentricity generates an asymmetric vortex pattern.

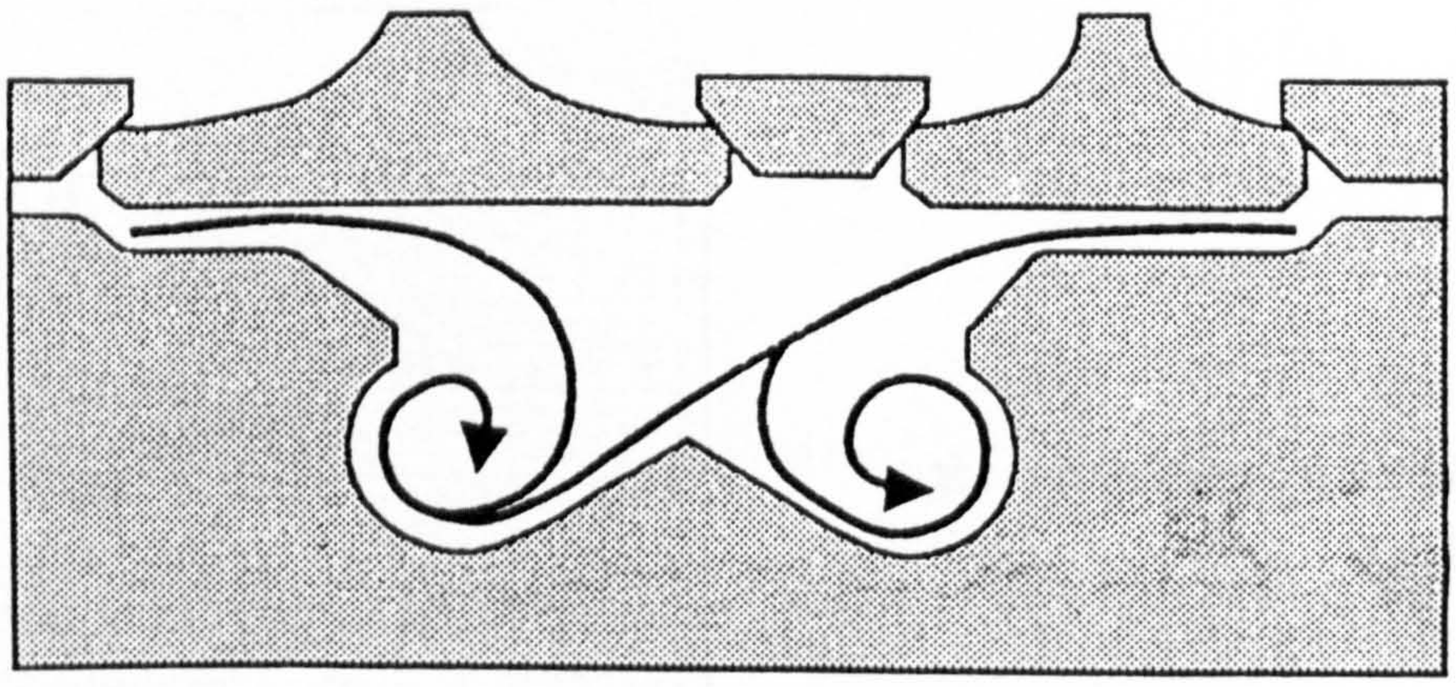


Figure 3.7

Sketch of squish-flow pattern, for a DI Diesel engine with omega-shaped bowl in piston.

In direct- or indirect-injection Diesel engines, fuel is injected in the form of one or more high-speed spray jets penetrating deeply into the combustion chamber, near TDC of compression into a swirling air flow under conditions of high pressure and temperature. In the case of direct injection fuel is injected in the swirl- and squish-dominated flow field of the clearance volume, whereas in the case of indirect injection, fuel is injected in the swirling air flow of the swirl-chamber. The surface of the spray breaks mainly due to turbulence and aerodynamic interaction, resulting in atomisation, a chain break-up process which leads to the creation of droplets whose diameters are much smaller than the nozzle diameter. The droplets become sources of vaporising fuel which then mixes with the surrounding gas. Subsequent combustion may be controlled by evaporation or mixing and these, in turn, depend mainly on the velocity distribution of the swirling air and the fuel-injection characteristics.

3.1.6 Combustion stroke

In petrol engines, the ignition of the homogeneous turbulent mixture by the spark plug and the consequent flame propagation across the clearance volume has been the subject of much attention due to its direct effect on engine performance. If the mixing between the reactants is high, as in the case of homogeneous-charge engines where it takes place outside the cylinder, combustion is characterised by premixed turbulent flames and is called homogeneous. In Diesel engines ignition follows fuel evaporation and its mixing with air and gives rise to a burning rate that is initially high but as the premixed charge is consumed heat release decreases and combustion is subsequently controlled by the mixing between fuel vapour and remaining oxygen. Self-ignition leads to strong pressure waves, high temperatures and velocities of the order of the speed of sound at certain locations [157]. Over-penetration of the spray may occur, resulting to rebound of the fuel from the walls and redistribution of droplets in the swirl field. The region where combustion initiates and the rate at which it spreads to the wall are determined by the fuel-injection rate. Multi-spray arrangements are often applied in direct-injection Diesel engines because they allow better fuel distribution around the upper bowl periphery where swirl velocities are higher and the bulk of squish-induced turbulence production takes place. Combustion is initiated in the region between the fuel jets, where there is a higher probability of evaporated fuel and on the edges of the spray where atomisation is more effective. Turbulence

influences the chemical reaction rates. The levels of turbulence just ahead of the flame front influence the local burning rates. The bulk flow also influences combustion, particularly through its effects on the flame area. Burning rates tend to increase with the squish area through the increase of swirl and increased squish-swirl interaction. In combustion chambers of cylindrical or re-entrant geometry, shortly after TDC, fluid expansion from the bowl region to the lower pressure region created by piston descent, known as reverse squish, enhances turbulence near the bowl lip and improves mixing at this last stage of combustion, when fuel injection has terminated and swirl has decayed.

The combustion process is influenced by the local generation of radical nuclei from the gaseous fuel, which, under conditions not fully understood, may link together to form soot particles, that are combustible and compete with the unburned fuel for oxygen. Reduction in power and efficiency and increase of unburnt hydrocarbon emissions is also caused by the formation of a circumferential wall jet caused by the expansion of gases due to the blowby effect [158].

3.1.7 Exhaust stroke

Near the end of the combustion stroke, the exhaust valve opens and the burned gases near the cylinder head surge out of the cylinder due to the lower pressure in the exhaust pipe. By that time, fuel burning is almost complete. The axial flow during the exhaust stroke is nearly one-dimensional and exhibits characteristics of a sink type of flow close to the cylinder head. The swirling flow persists through the whole engine cycle, although significantly reduced after combustion, and, if trapped inside the piston bowl, may survive the exhaust process, carrying itself to the next cycle as a residual motion.

3.2 Engine flow prediction with the SPEED code

The major findings of engine research in the areas of gas flow, fuel injection and in-cylinder combustion have been summarised in the preceding paragraphs. As it has been mentioned elsewhere in this chapter, in the past 20 years the knowledge attained through experimental measurements has been complemented by computer codes that can simulate engine behavior. The main building blocks of such methodologies and their implementation in codes such as RPM [39], EPISO [42], KIVA [41], FIRE [45], and SPEED [46] have been reviewed in chapter 2. The objective here is to

present how gas flow, fuel injection and combustion are simulated in realistic cylinder geometries in the specific case of the SPEED code .

3.2.1 Overview of SPEED

SPEED is a CFD code designed for internal combustion engine design and performance prediction, that supports:

- airflow;
- homogeneous combustion and NO_x formation; and
- Diesel combustion with sub-models for nozzle flow, spray behaviour, spray wall interaction, auto-ignition, combustion, soot and NO_x formation.

The methodologies underpinning the SPEED code have been presented in detail by Gosman and Marooney [46], together with extracts from predictions for various petrol and Diesel engine configurations as well as comparisons with measured data. The discretisation technique employed in the code has been described by Gosman and co-workers [47], whereas the implementation of the PISO algorithm for solution of the discretised equations is described by Theodoropoulos [24]. The implemented Diesel combustion simulation model has been described by Gosman and his co-workers [170], and so has the the application of the code on various engine geometries [48]. The aspects of the methodologies that are relevant to the interaction of user with the simulation process, namely the computational mesh, initial conditions, boundary conditions, and the generation of four-dimensional property fields , are summarised in the subsequent paragraphs.

3.2.2 The SPEED methodology

3.2.2.1 Transport equations

The mathematical description of in-cylinder flow in SPEED is based on the conservation equations of mass, momentum, and energy, as well as the turbulence equations for turbulence energy k and its dissipation rate ε. The conservation equations are, in vectorial notation:

Continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho_g \mathbf{u}) = S_m \qquad (3.1)$$

Momentum equation:

$$\frac{\partial}{\partial t}(\rho_g \, u \, V) + \nabla\cdot(\rho_g \, u \, u) = -\theta \, \nabla p + \nabla\cdot\tau + S_{mom} \qquad (3.2)$$

Energy equation:

$$\frac{\partial}{\partial t}(\rho_g \, e \, V) + \nabla\cdot(\rho_g \, u \, e) = \nabla\cdot(\Gamma_e \, \nabla e) + S_{ed} + S_{ec} - \theta \, p \, \nabla\cdot u \qquad (3.3)$$

where t: time, u: gas velocity (vector), p: pressure, e: internal energy, θ : void fraction (fractional volume of droplet/gas mixture occupied by gas), ρ_g: apparent gas density, τ : turbulent shear stress tensor, S_m: mass interchange rate with droplets, S_{mom}: momentum interchange rate with droplets, Γ_e: turbulent thermal diffusion coefficient, S_{ed}: energy interchange rate with droplets, S_{ec}: energy source due to combustion.

The k-ε model is used for turbulence modelling, requiring the solution of one equation for the turbulence energy k and one for its dissipation rate ε. These are:

$$\frac{\partial}{\partial t}(\rho \, k) + \nabla\cdot\left[\rho \, u \, k - \frac{\mu_{eff}}{\sigma_k}\nabla k\right] = P_k - \rho \, \varepsilon \qquad (3.4)$$

$$\frac{\partial}{\partial t}(\rho \, \varepsilon) + \nabla\cdot\left[\rho \, u \, \varepsilon - \frac{\mu_{eff}}{\sigma_\varepsilon}\nabla \varepsilon\right] = C_1 P_k \frac{\varepsilon}{\kappa} - C_2 \rho \frac{\varepsilon^2}{\kappa} + C_3 \rho \, \varepsilon \, \nabla\cdot u \qquad (3.5)$$

where C_1, C_2, C_3, σ_ε, and σ_k are empirical coefficients, μ_{eff} is the effective viscosity of k, and P_k its rate of production.

The derivation of equations (3.1) to (3.5) as well as values for the empirical coefficients involved can be found in paragraph 2.1.1.1, as well as in the work of Theodoropoulos [24] and in the user's manual of the SPEED code [173].

3.2.2.2 Fuel injection and Diesel combustion modelling

In Diesel engines the fuel is injected in the combustion chamber in the form of a liquid spray. The spray passes through the processes of atomisation and breakup to form individual fuel droplets which vaporise and produce fuel vapour ready to be burnt. In SPEED, sprays are represented by a 'discrete droplet model' as a set of computational droplets (hereafter referred to as 'droplets' for brevity), each representing a parcel containing a number of real droplets with equal velocity, size, temperature, and density. The histories of the computational droplets, including their trajectory, breakup, and evaporation due to interaction with the gas, are obtained from the

Langrangian conservation equations of mass, momentum and energy. Their formulation is based on the works of Gosman and Kralj [170], Reitz and Diwakar [176], and Borman and Johnson [177], and can be found it the user's manual of the SPEED code [173]. The droplet-wall interaction model of Obermeier [174], including droplet fragmentation – meaning that a droplet hitting the wall is fragmented into smaller ones which stay on the wall – and splashing – meaning that a droplet hitting the wall breaks up into smaller ones which are ejected – is implemented in the code. So are two atomisation models, one developed by Huh and Gosman [175], and another developed by Reitz and Diwakar [176]. Finally, the injection process is modelled with the nozzle flow model, developed by Obermeier [174]. This model calculates the velocity of the injected droplets by solving an orifice-type equation for the volumetric flow rate through the injector. The injection velocity is an important parameter in a spray calculation, since it influences the atomisation and breakup processes, spray penetration, the inter-phase transfer processes, and also the droplet-droplet and droplet-wall interactions.

The 'Shell' auto-ignition model [178], originally developed for the study of knock in gasoline engines, extended for Diesel fuels by Theobald and Cheng [179], is implemented to predict ignition in SPEED. The model is capable of simulating the cool-flame phenomenon as well as the rapid acceleration of the reaction rate after the onset of ignition. It involves three transport equations for the concentrations of the ignition species Φ (see the work of Theobald for descriptions), that are solved in conjunction with the equations for combustion. These are of the form:

$$\frac{\partial \rho Y_\Phi}{\partial t} + \nabla \cdot [\rho \mathbf{u} Y_\Phi - \Gamma \nabla Y_\Phi] = S_\Phi \qquad (3.6)$$

where Y_Φ is the concentration of the ignition species ($\Phi \equiv$ R, B, or Q [179]), ρ is the density, $\mathbf{u}$ is the gas velocity, Γ is a diffusion coefficient, and S_Φ is the local source of each of the ignition species. A new energy value S_{ign} due to the temperature rise caused by the ignition model is also calculated at this stage and, added to the energy produced by combustion (see equation 3.10), is used as a source term in the energy balance equation (eqn. 3.3).

Combustion is simulated using Magnussen's turbulence combustion model [29], which is an 'eddy breakup' model, the basic principles of which have been described in section 2.1.1.2 of the present work. According to this the mean 'reaction rate' - the mass rate of production per unit volume of species - is controlled by the turbulent mixing of the unburnt and burnt gas,

as well as by the fuel, air and burnt gas concentrations in the mixture, according to the following equation:

$$R_f = A\,\rho\,\frac{\varepsilon}{k}\,\min\left(Y_f, \frac{Y_{O_2}}{r_f}, \frac{BY_p}{(1+r_f)}\right) \tag{3.7}$$

where R_f is the reaction rate, ρ is the density, $\frac{\varepsilon}{k}$ is the turbulent mixing, Y_f, Y_{O_2}, and Y_p are the local mean concentrations of fuel, oxygen, and reaction products respectively, r_f the stoichiometric oxygen requirement to burn 1 kg of fuel, and A and B are reaction constants of the Magnussen model. k and ε are the local values for the turbulence energy and its dissipation rate, except for cells adjacent to the walls, where average values are used.

The numerical implementation of the Magnussen model in SPEED, instead of characterising the mixture composition in terms of the above mentioned species, employs the unburnt fuel vapour concentration Y_{fu} and a fictitious 'total fuel' concentration Y_{ft}, defined as the fuel vapour distribution which would have prevailed if the fuel had undergone no chemical change due to combustion. The transport equations for Y_{ft} and Y_{fu} are (taken from [173]):

$$\frac{\partial \rho Y_{ft}}{\partial t} + \nabla\cdot[\rho\,\mathbf{u}\,Y_{ft} - \Gamma_{ft}\,\nabla Y_{ft}] = S_{ft} \tag{3.8}$$

$$\frac{\partial \rho Y_{fu}}{\partial t} + \nabla\cdot[\rho\,\mathbf{u}\,Y_{fu} - \Gamma_{fu}\,\nabla Y_{fu}] = S_{fu} \tag{3.9}$$

where ρ is the density, $\mathbf{u}$ is the gas velocity, Γ_{ft} and Γ_{fu} are the diffusion coefficients for the unburnt and total fuel, S_{ft} is the local source of fuel vapour due to droplet evaporation (obtained from the spray model) and S_{fu} is the rate of production or consumption of fuel.

The contribution of the combustion and ignition processes to the energy balance is given by the source term S_{ec} of equation 3.3, which is:

$$S_{ec} = R_f\,E_{comb} + S_{ign} \tag{3.10}$$

where E_{comb} is the energy of combustion for a given fuel and S_{ign} is the source term for the ignition model..

3.2.2.3 Handling of boundaries

Two types of boundaries are possible in SPEED:

1. Inlet or outlet flow boundaries. The code provides for flow to enter and exit the computational domain through valve opening surfaces or valve curtains, where a normalised velocity is provided by the user, with the overall boundary mass flow characterised by a pressure drop between the boundary pressure and the average chamber pressure. This type of boundary is used to model engine valves when the port and manifold are not modelled, as is currently the case in SPEED.
2. Walls. They can be stationary, move with the velocity of the piston, or move with the velocity of the valve head of each valve. Furthermore, SPEED provides for walls to be adiabatic or heat-conducting, in which case the wall temperature is prescribed by the user and the heat flux through the wall is calculated by the code.

3.2.2.3.1 Inlet or outlet flow boundaries

SPEED provides for flow to enter or exit the computational domain through valve curtains, which are defined by the edge of the moving base of each valve. Hence valve curtains have the time-dependent shape of a cylindrical surface with diameter D_v that of the valve and height Z_l equal to the valve lift. The flow through a valve curtain is described by the user in terms of the pressure and temperature in the port, and in the case of inflow through the valve, the relative velocity, axial flow-angle, and swirl flow-angle distributions, defined in the local cylindrical coordinate system of the curtain. All these values are either available from measurements, or estimated by alternative means. Also needed is the valve lift Z_{li} and the discharge coefficient C_{Di} for each valve, which is characteristic of the inlet system and is available from measurements. It is defined as:

$$C_{Di} = \frac{A_{fi}}{A_{li}} \tag{3.11}$$

where A_{fi} and A_{li} are the flow [22] and geometrical areas of valve curtain i.

The rate of mass flow through each valve curtain $\dot{m}_i$, where i indicates the valve number, is calculated from the pressure drop Δp_i between the average cylinder pressure and the pressure in the inlet or outlet port, as well as the discharge coefficient C_{Di} and the density in the cylinder ρ_{cyl} or the port ρ_{port}, by the following equation (taken from [Begleris 86]):

$$\dot{m}_i = A_{li}\, C_{Di} \sqrt{2\, \rho_{port_i}\, \Delta p_i} \qquad \text{(if } \dot{m}_i > 0 \text{ inflow)} \tag{3.12}$$

$$\dot{m}_i = A_{li}\, C_{Di} \sqrt{2\, \rho_{cyl}\, \Delta p_i} \qquad \text{(if } \dot{m}_i < 0 \text{ outflow)} \tag{3.13}$$

The evolution of the average pressure in the cylinder is thus of primary importance for the characterisation of the flow through each valve. In SPEED it is calculated in a zero-dimensional calculation, the global thermodynamics calculation, prior to the three-dimensional flow solution.

Because the SPEED code permits more than one valves to be open simultaneously, overall continuity in the global calculation is given by:

$$\frac{dM_{cyl}}{dt} = \sum_{i=1}^{N} \dot{m}_i \tag{3.14}$$

where M_{cyl} is the overall cylinder air-mass and N is the number of valves. The (global) energy conservation equation is:

$$\frac{dH_{cyl}}{dt} = Vcyl \frac{dp_{cyl}}{dt} + \sum_{i=1}^{N} \dot{m}_i \, h_{port_i} + \dot{Q} \qquad (\text{if } \dot{m}_i > 0 \text{ inflow}) \tag{3.15}$$

$$\frac{dH_{cyl}}{dt} = Vcyl \frac{dp_{cyl}}{dt} + \sum_{i=1}^{N} \dot{m}_i \, h_{cyl} + \dot{Q} \qquad (\text{if } \dot{m}_i < 0 \text{ outflow}) \tag{3.16}$$

where H_{cyl} is the chamber enthalpy, V_{cyl} is the chamber volume, h_{cyl} and h_{port_i} are the specific enthalpies in the chamber and in port i respectively, and $\dot{Q}$ is the heat flux through the walls, which is calculated from equations 3.23 and 3.24, given in the following section.

The above global thermodynamics calculation results in the mass-flux for each valve. How the velocity and turbulence properties on the curtain are calculated from it, will be described in the remaining paragraphs of this section .

Velocity

When there is outflow through the valve, the velocity on the curtain is calculated by extrapolation from the interior of the mesh. In the case of inflow through the valve, the velocity distribution on the valve curtain is calculated from the mass-flux through the curtain and the user-defined dimensionless radial velocity $u_r(\theta,z)$, axial flow angle $\alpha(\theta,z)$, and swirl flow angle $\beta(\theta,z)$ distributions, expressed as functions of position in the local cylindrical coordinate system of the valve, shown in figure 3.8. The latter three input values are interpolated to the boundary nodes of the curtain, where the fluid-velocity vector $u(\theta,z)$ is calculated by the equation:

$$u(\theta, z) = \frac{\dot{m}}{\rho A_1} u_r(\theta,z) \left(e_1(\theta) + e_2(\theta) \tan(\beta(\theta,z)) + e_3 \left(\frac{1}{\tan(\alpha(\theta,z))} \right) \right) \tag{3.17}$$

where: $\mathbf{u}$ = fluid velocity vector

A_l = area of the valve curtain

$\mathbf{e_1}$ = unit radial-direction vector

$\mathbf{e_2}$ = unit swirl-direction vector

$\mathbf{e_3}$ = unit axial-direction vector

and $\mathbf{e_1}$, $\mathbf{e_2}$, $\mathbf{e_3}$ are defined from the orthogonal unit vectors $\mathbf{d_1}$, $\mathbf{d_2}$, $\mathbf{d_3}$ defining the orientation of the valve curtain as follows (see figure 3.9):

$$\mathbf{e_1} = \mathbf{d_1}\cos\theta + \mathbf{d_2}\sin\theta$$

$$\mathbf{e_2} = \mathbf{d_2}\cos\theta - \mathbf{d_1}\sin\theta$$

$$\mathbf{e_3} = \mathbf{d_3}$$

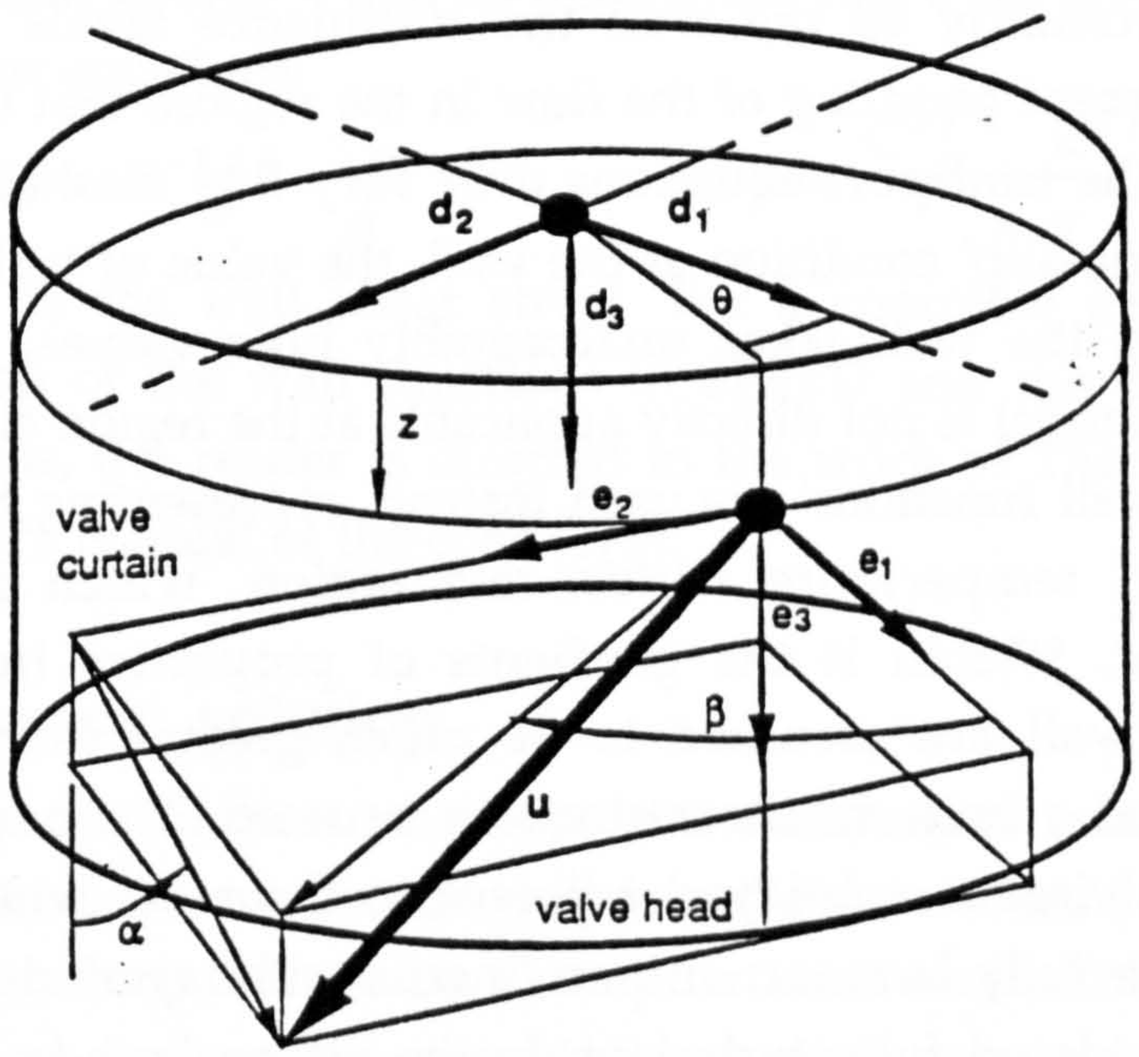


Figure 3.8 Cylindrical coordinate system of the valve curtain, and definition of the fluid velocity vector **u**, axial flow angle α, and swirl flow angle β at a point on the curtain.

Turbulence

When there is outflow through the valve, the turbulence properties on the curtain are set equal to their upwind value. In the case of inflow through the valve, the turbulence kinetic energy on the curtain k is modelled as being proportional to the radial component of the fluid velocity on the curtain u_R:

$$k(\theta,z) = \lambda_k\, u_R(\theta,z)^2 \qquad (3.18)$$

where λ_k is a user-specified turbulence intensity factor for the valve, determined by measurements to be of order: $\lambda_k \approx 10^{-2}$ (Ahmadi-Befrui [23]).

The dissipation rate ε across the valve curtain is set proportional to the valve lift Z_l :

$$\varepsilon(\theta,z) = C_\mu^{3/4} \frac{k(\theta,z)^{3/2}}{\lambda_\varepsilon Z_l} \tag{3.19}$$

where C_μ is a turbulence viscosity coefficient (=0.09) and λ_ε is a user-prescribed turbulence length-scale factor, determined by measurements to be of the order of $\lambda_\varepsilon \approx 10^{-2}$. (Values 0.045 and 0.03 have been used by Ahmadi-Befrui [23] and Zarzalis [180] respectively, for engine applications).

3.2.2.3.2 Wall boundaries

In direct proximity to the wall the turbulence levels are very low, requiring a different handling of the flow in the region, that is possible only by solution of the transport equations over very fine meshes. Furthermore, because of the 'no-slip' condition at the wall, the value of turbulence energy at the region of the wall takes unacceptably high values. For the above reasons the k-ε model is not directly applicable at the region close to the wall, and algebraic 'wall functions' are used instead, representing the distribution of velocity and temperature within this region, which is called 'wall boundary layer'. Within it the gradients of properties in the direction normal to the wall are assumed to be much greater than in directions parallel to it, and a uniform shear stress is assumed. The boundary layer is divided into a 'viscous sub-layer' adjacent to the wall, where the flow is considered to be fully laminar, and an 'inertial sub-layer' elsewhere, where the flow is considered fully turbulent. In the viscous sub-layer the velocity has a linear dependence on the distance from the wall:

$$u^+ = y^+ \tag{3.20}$$

whereas in the inertial sub-layer a logarithmic one:

$$u^+ = \frac{1}{\kappa} \ln(E \cdot y^+) \tag{3.21}$$

where $u^+ = \frac{u}{u_\tau}$ is the dimensionless velocity (u is the relative velocity to the wall, parallel to the wall and u_τ is a velocity scale), y^+ is a dimensionless distance from the wall defined as

$$y^+ = \frac{\rho C_\mu^{1/4} \kappa^{1/2}}{\mu} y \tag{3.22}$$

where C_μ is the turbulence viscosity coefficient (=0.09), μ is the molecular viscosity of the fuel, κ is the empirical von Karman constant (=0.487) and E is an empirical wall-texture coefficient (= 9.793).

Assuming that the heat flux is constant at the region near the wall, the heat flux and temperature are calculated as follows:

$$q_w = \left(\frac{\mu C_p}{P_r}\right)\left(\frac{T-T_w}{y}\right) \qquad \text{in the viscous sub-layer} \qquad (3.23)$$

$$q_w = \left(\frac{\mu C_p}{P_{r_t}}\right)\left(\frac{T-T_w}{y}\right)\left(\frac{y^+}{\frac{1}{\kappa}\ln(Ey^+) + P\frac{P_r}{P_{r_t}}}\right) \qquad \text{in the inertial sub-layer} \qquad (3.24)$$

where q_w is the heat-transfer rate through the wall, C_p is the fluid-specific heat capacity at constant pressure, P_r is the Prantl/Schmidt number of the fluid, T is the temperature, T_w is the wall temperature, y is the distance from the wall, and P_{r_t} is the turbulence Prantl/Schmidt number. Finally, the turbulence kinetic energy k and its dissipation rate ε are given in the inertial sub-layer by the equations:

$$k = C_\mu^{-1/2}\frac{\tau_w}{\rho} \qquad \varepsilon = \frac{C_\mu^{3/4}\,k^{3/2}}{\kappa\, y} \qquad (3.25)$$

where τ_w is the wall shear stress. For a complete description of the implementation of the wall functions in SPEED and the derivation of the above equations, the reader is directed to the work of Theodoropoulos [24] and to the user's manual of the code [173].

3.2.3 Numerical solution

The numerical solution of the partial differential transport equations described in the previous section, requires their transformation to algebraic equations at locations defined by a computational mesh. This process, called discretisation, as well as the computational mesh arrangement involved in SPEED, are described in the following paragraphs of this section.

3.2.3.1 Discretisation

The transport equations, expressed in general form by equation 2.4 and in detail by equations 3.1 to 3.9, are described by the finite-volume method, in which they are first integrated over the individual computational cells and over a finite time increment δt, and then approximated in terms of the cell-centred nodal values of the dependent variables. In this way, the discrete transport equation for a dependent variable ψ_P over a finite volume P of hexahedral shape, with six faces A_j, becomes:

$$\underbrace{\frac{\rho_P^n \psi_P^n V_P^n - \rho_P^o \psi_P^o V_P^o}{\delta t}}_{\text{Rate of change}} + \underbrace{\sum_{j=1}^{6} F_j \psi_P}_{\text{Convection}} - \underbrace{\sum_{j=1}^{6} \Gamma_j (\nabla\psi)_j A_j}_{\text{Diffusion}} = \underbrace{S_P^\psi}_{\text{Source}} \qquad (3.26)$$

where: the superscripts 'o' and 'n' refer to 'old' and 'new' time levels, separated by the time interval δt,

V_P is the volume of cell P,

F_j is the mass flux through face j and normal to it,

ψ_j is the average value of variable ψ at face j,

Γ_j is the diffusion coefficient,

$(\nabla\psi)_j$ is the gradient of variable ψ on face j,

A_j is the area of face j,

$S_P,^{\psi}$ is the source (or sink) of variable ψ.

The manner in which the diffusive and convective fluxes are expressed in terms of nodal values ψ is one of the key determining factors on accuracy and stability of the solution. In the diffusion term of equation 3.26, the gradient $(\nabla\psi)_j$ on face j between the cell node P and the neighbouring cell node N is calculated by linear interpolation from the gradients at the adjacent cells centres $(\nabla\psi)_P$ and $(\nabla\psi)_P$. These are calculated from the values of ψ_P and ψ_N according to the interpolation formula:

$$(\nabla\psi)_P = \frac{1}{Vp} \sum_f (1 - l_f^P)\,(\psi_P - \psi_N)\,A_j \qquad (3.27)$$

where l_f^P is a geometric weighting factor of cell P, shown in figure 3.9. For the calculation of the mass flux in the convection term of equation 3.26, SPEED provides a choice between an 'upwind', a 'hybrid', or a 'self-filtered central' differencing scheme, described in detail in [173]. The latter, locally blending central- and upwind- differencing schemes, is the most frequently used option, since it minimises numerical diffusion.

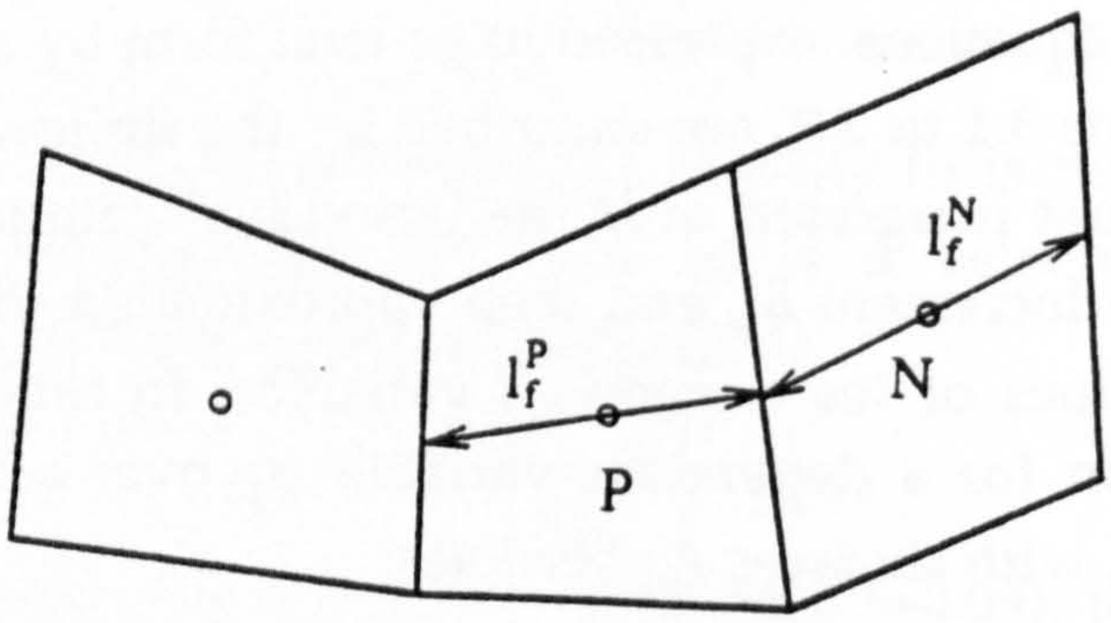


Figure 3.9 Definition of the geometric weighting factor l_f^P for cell P.

3.2.3.2 Computational mesh

The solution of the discretised transport equations requires a computational mesh arrangement, defining the locations at which the dependent variable is to be calculated. SPEED employs an unstructured mesh system, that is fitted within the moving boundary surfaces of the three-dimensional domain. The volume subdivision is by straight-edged hexahedral cells. Wedge-shape cells may exist, but they are conceptually regarded as degenerate hexahedra, formed by collapsing cell faces to lines. For the purposes of the finite-volume methodology described in the previous section, computational nodes are defined at the nominal centroids of the cells and boundary cell faces. It is at these locations that the values of the dependent variables are calculated. The generality and flexibility of the meshing features just described, endows SPEED with the following capabilities:

- Some continuity of methodology with previous codes is preserved, allowing use of meshes that were originally created for other simulation systems (e.g. EPISO, KIVA, FIRE);
- Real engine geometries may be handled;
- The mesh need only exist where it is needed, unlike structured meshes, which often contain inactive mesh regions;
- Local control of mesh density in high-shear or recirculation regions, may be achieved through local grid refinement;
- The mesh may comprise layers of single-cell thickness, that can be introduced or removed to accommodate valve and piston motion.

3.2.3.2.1 Mesh-addressing conventions

In a structured mapped mesh the method of locating a cell, determining its neighbours, and relating the mesh vertices, is an implicit indexing system, usually of the 'IJK' type, like the one described by Begleris [43]. This method cannot be applied to the unstructured, non-mapped meshes involved in SPEED, where all this information needs to be stored for each cell, in what is an explicit mesh-addressing scheme. According to this, cells and vertices are numbered and stored separately, and arrays store the six nearest neighbours, called the 'neighbour list', and eight vertices associated with each cell, called the 'connectivity list'. While the connectivity list is necessary, the neighbour list can be deduced from the connectivity list. In

SPEED both lists are stored permanently, to reduce the time overheads in creating the neighbour list, every time this is needed. The advantage for using such a connectivity system lay in that it allows for both structured and non-structured meshes and that lower overheads are required to store the mesh in computer-memory. This can be illustrated with the help of the SPEED-mesh for an in-direct-injection Diesel engine, shown figure 3.15. While codes employing structured mesh strategies would allocate memory for inactive cells around the swirl chamber and piston regions, SPEED generates and stores in computer-memory only cells within the combustion chamber.

Boundary values are defined at 'nodes' located at the face-centres bounding the hexahedral mesh, and may be considered as either zero-volume hexahedra or quadrilaterals at these locations. These boundary 'cells' are grouped into 'patches'. Each patch is labelled as a particular boundary type such as wall, inlet or outlet, symmetry plane or cyclic boundary.

3.2.3.2.2 Mesh adaption to boundary change

The motion of the valves and piston is catered for in SPEED by ascribing time-varying positions to some or all of the vertices, causing the mesh to translate or distort in any prescribed way. This is done either by user-specified routines, or by a semi-automatic mesh-motion module of the code. In both cases vertices on the moving boundaries are translated to the new boundary locations and internal vertices are displaced to adjust the mesh correspondingly, ideally while attempting to preserve certain mesh quality criteria. In cases where the mesh has a layered structure in the squish region, its topology can be changed to avoid generating cells of high aspect ratio, by addition or removal of mesh layers above the piston crown, below the cylinder head, or below the valve head, by means of a layer addition and removal module of the code. An example of mesh motion, accommodating piston and valve motion in the case of a direct-injection Diesel engine with a piston recess, is shown in figure 3.10. In figure 3.10.a, a cross section of the mesh is displayed with the piston at TDC of induction and the valve closed. In figure 3.10.b, the same cross section of the mesh is shown with the piston displaced to mid-stroke of induction, and the valve displaced close to its maximum valve lift position. Two mesh layers have been added around the valve curtain, and five mesh-layers have been added in the region below the valve base and above the piston crown. In figure 3.10.c, a top-side view of the mesh at mid-stroke is displayed, showing its unstructured topology.

3.2.3.2.3 Mesh-quality characteristics

SPEED is able to solve the flow for any unstructured mesh, as long as all cells have eight corner vertices and each cell shares a face with one and only one neighbouring cell. However, there are certain cases of meshes that are computationally inefficient; such meshes can in the worst case cause instability and hence failure; at best inaccuracy and increased calculation times. The main mesh-quality criteria are:

Cell non-orthogonality. The presence of angles smaller than 30 or larger than 150 degrees (see figure 5.17) is not recommended, since this will introduce inaccuracies in the interpolation of values from the nodes to the face-centres, and increase computational times.

Cell aspect ratio. Cell aspect ratio is defined as the ratio of the longest over the shortest edge of a cell, as shown in figure 5.18. The presence of cells with aspect ratio greater than 20 may have an adverse effect on the accuracy of results. Aspect ratio limitations are more relaxed for near-wall cells. Acceptable aspect ratios can be achieved even at TDC, where the mesh is compressed, through appropriate removal of layers.

Cell warpage. This is the angle formed by the unit area vectors of the two triangles, each face can be divided into (figure 5.19). As this angle becomes bigger, so does the approximation involved in the approximation of the centroid of the face.

Mesh distribution for optimum resolution. The mesh should adequately resolve the gradients in all dependent variables. This means that the mesh should be fine where the gradients are expected to be high.

In all cases the mesh designer must balance the above quality criteria in order to achieve a mesh which resolves the geometry of the computational domain and the flow into it, will fit into the available memory at run-time, and will not cause excessive solution times. In manual mesh generation techniques, these design decisions are left largely to the engineer. In more automated techniques, the mesh generation algorithms must generally achieve well-balanced mesh qualities.

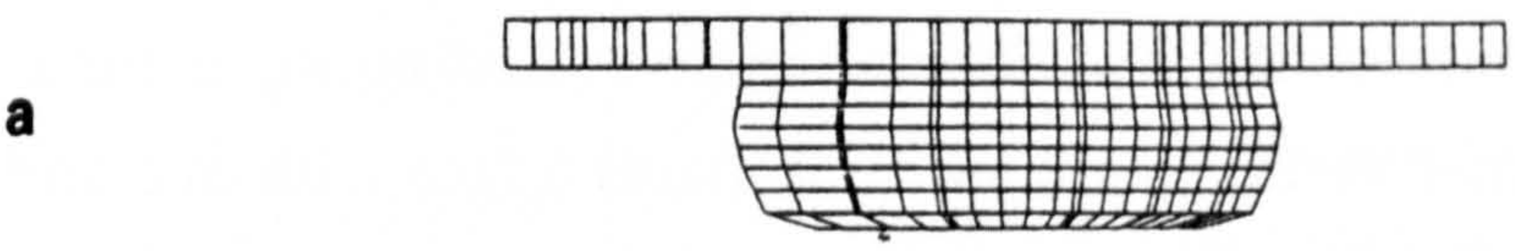


Figure 3.10 A SPEED mesh for a direct-injection Diesel engine with bowl-in-piston. The unstructured nature of the mesh is shown in figure c. The cross-sections of the mesh in figures a and b, show how piston and valve motion is accommodated by mesh adaption to boundaries and layer addition and removal.

3.2.3.2.4 Mesh generation

Until the present work, SPEED meshes were generated by separate numerical mesh-generation programs written by individual users. This was not difficult for the case of simple, non-realistic geometries, usually having polar symmetry or a bipolar structure. Such meshes can be seen in figures 3.11 and 3.12, taken from Fateh [181].

The mesh-generation system proposed in this work, provided for meshing of more complex, realistic engine geometries, exploiting the unstructured, non-orthogonal mesh features of the SPEED code. Such meshes are shown in figures 3.10, 3.13, and 3.14. Figure 3.13 shows a mesh fitted in the combustion chamber of a FIAT direct-injection Diesel engine, featuring an off-centered valve and a lipped-bowl recess in the piston. The mesh combines radial sections (regions A and B shown in figure 3.13), that are adjacent to the cylinder liner, rectangular sections, in the regions of the cylinder and valve axes (regions D in figure 3.13), and mesh refinement zones to control the mesh density between regions A and B of the radial mesh, to achieve a uniform mesh. Figure 3.14 shows a mesh fitted in the combustion chamber of a Renault in-direct injection Diesel engine, featuring a swirl chamber, a vertical glow plug protrusion that is not visible in the figure, and a piston recess.

3.2.3.3 Solution procedure.

Given an appropriate mesh and defined on it suitable initial and boundary conditions to describe the problem, SPEED solves the discretised equations (equation 3.26) for each dependent variable ψ. This is handled by the PISO solution method [182], suitably extended for non-orthogonal grids. The main features of the method are:

- An operator splitting technique to decouple the energy, pressure, and velocity in the flow equations (eqns. 3.1 to 3.5);
- combination of the mass and momentum equations to a pressure equation;
- a predictor stage in the solution procedure, which produces provisional velocity and pressure fields;

- field-dependent coefficients arising from the coupling, that are updated between a user-defined number of corrector stages;
- repetition of the above corrector sequence until solution is reached.

The above has been modified by Gosman and Begleris [43] to create the EPISO algorithm, that includes solution of the equations for global cylinder pressure, density, and energy, required for the handling of boundary motion and flow through valves (equations 3.12 to 3.16). The presence of the liquid phase, as well as auto-ignition and combustion are handled by further extensions to the EPISO algorithm, described in [173]. The main features of the overall procedure are given in the following table:

Table 3.1 The solution algorithm of the SPEED code

Solution algorithm.		
1	*Data input.*	Mesh data, initial conditions, boundary values for discrete times, engine-operation data and calculation-control data are read by the SPEED code and stored in memory.
2	*Piston and valve motion.*	Calculation of piston position for time θ; Reading of valve lift $Z_{li}(\theta)$, for time θ, from valve-lift tables for each valve i.
3	*Mesh adaption.*	Mesh adaption to the new boundary positions by means of the mesh-motion and layer addition and removal modules; Calculation of new volumes and face-areas for each cell.
4	*Valve flow calculation and global thermodynamics.*	Solution of the global continuity equations for determination of the mass-flow rate through each valve and the mean cylinder pressure, density and temperature. Setting of the gas-velocity distribution at the valve curtains according to the calculated mass-flow rate and the prescribed valve-curtain flow direction and profiles.
5	*Initial bulk adjustment.*	Adjustment of local pressures, densities, and temperatures to the global values.
6	*Spray and gas predictor.*	Solution of droplet momentum, break-up, mass, and energy equations and calculation of contributions to the gas mass, momentum, and energy. The auto-ignition model is invoked to calculate the reaction rates for the the ignition species. The full gas species and energy equations are solved to obtain the predicted values for these quantities.
7	*Gas momentum predictor step.*	Solution of the gas momentum equations using values from step 6 and the spray source term to estimate the predicted velocity.
8	*Spray corrector.*	Repetition of step 6, without the ignition calculation.
9	*Gas corrector.*	Solution of the gas pressure and momentum conservation equations, to yield the corrected velocity, pressure, density, energy, and gas species concentrations.
10	*Corrector sequence.*	Repetition of steps 8 and 9 for a user-specified number of times (usually two for non-combustion and five for combustion calculations).
11	*Global adjustment.*	Correction of the pressure, density, and temperature in the cylinder, taking in account the evaporated mass of fuel.
12	*Turbulence predictor and corrector.*	Solution of the transport equations for k and ε and correction of these values.
13	*Output.*	Output of results in a file.
14	*Time advancement.*	Advancement to the next time step and restart from step 2.

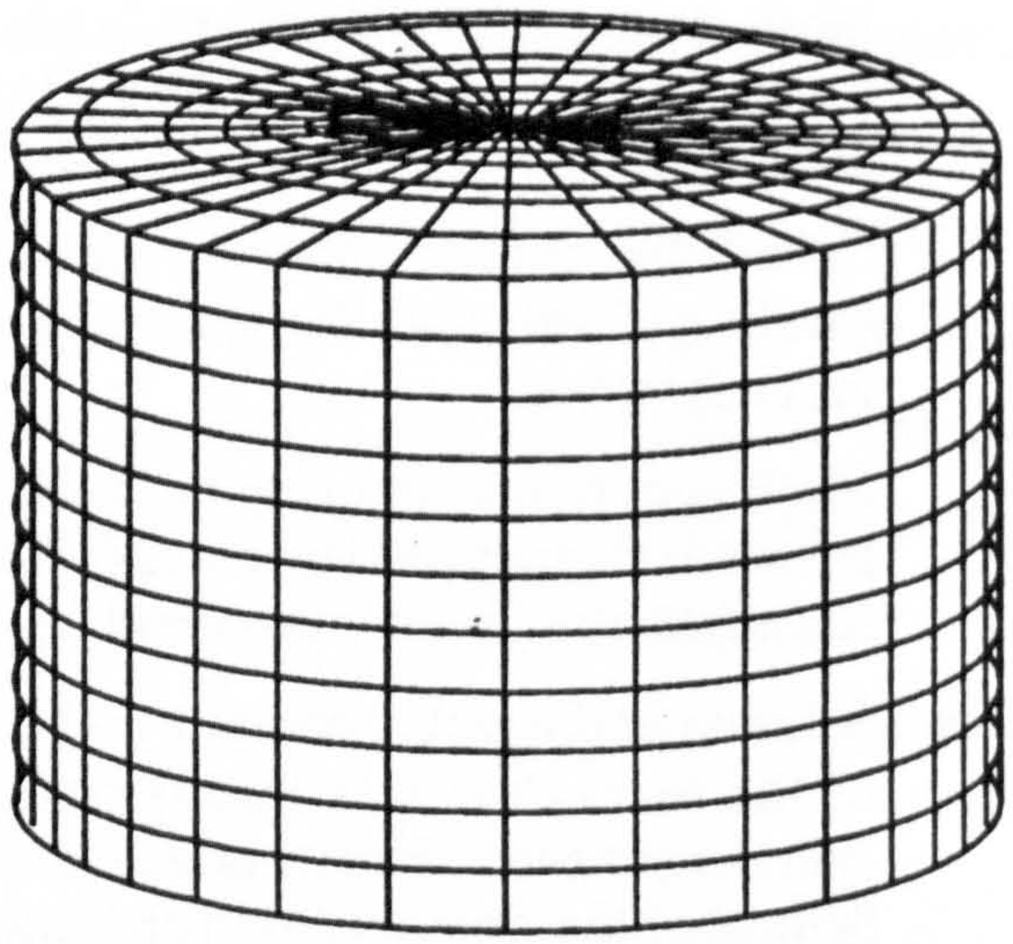


Figure 3.11 A polar mesh for an idealised disc-chamber engine, with one valve. The valve axis coincides with that of the cylinder.

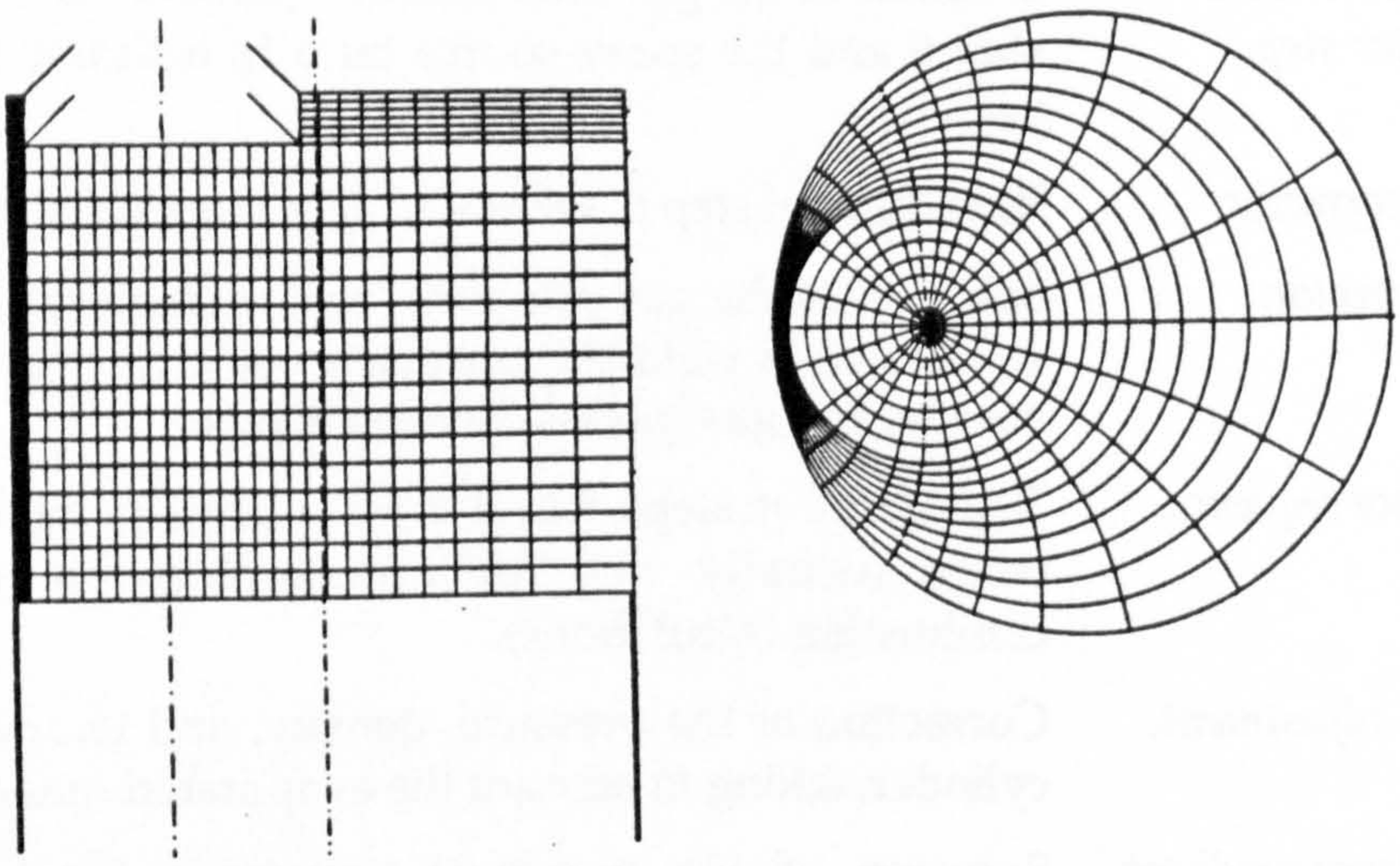


Figure 3.12 A bipolar mesh for a disc-chamber engine, featuring one off-centered valve.

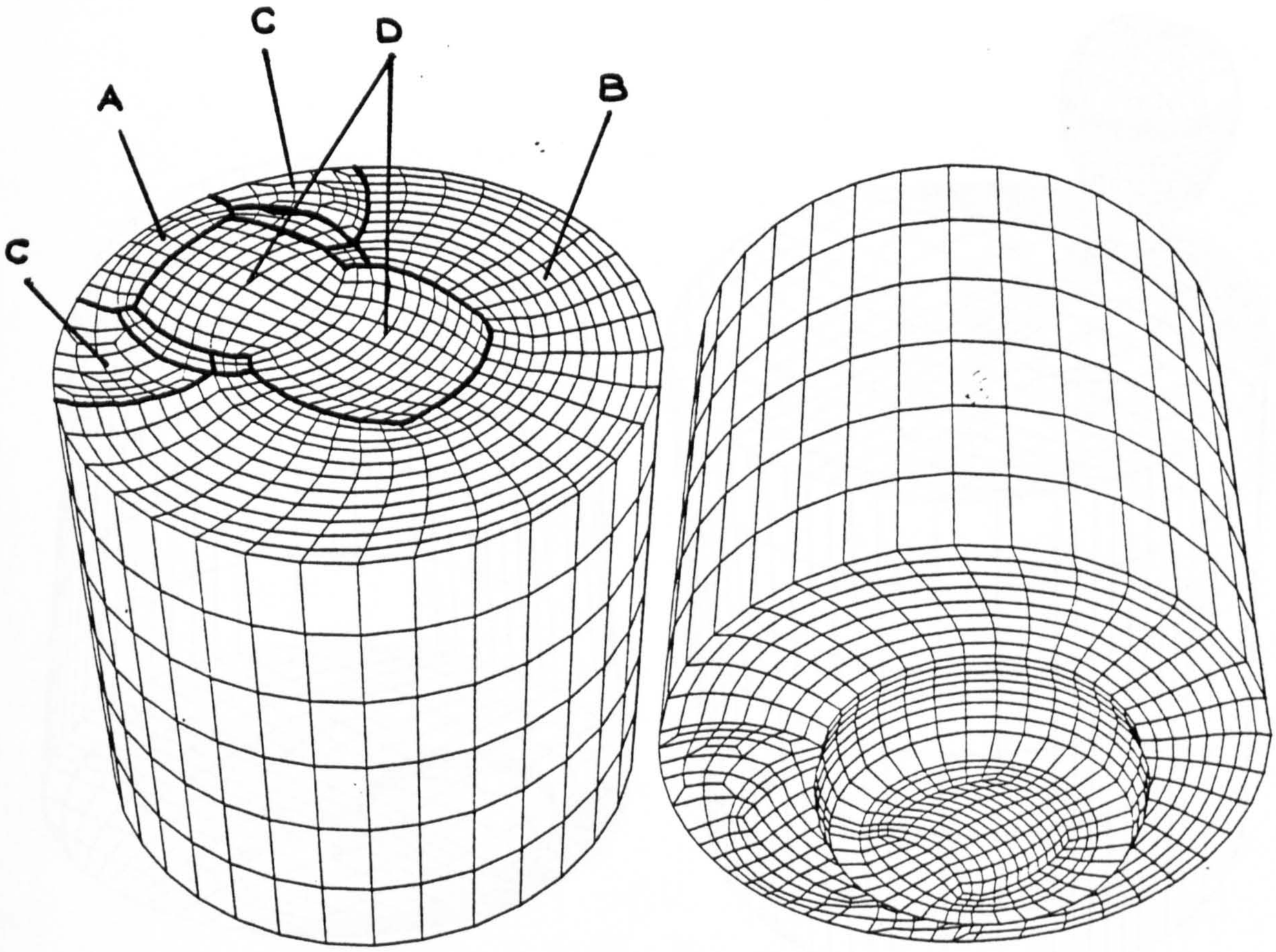


Figure 3.13 A mesh for the combustion chamber of a FIAT direct-injection Diesel engine, featuring an offset valve and a lipped bowl. The mesh features radial and rectangular regions, as well as mesh density transition regions, to control the overall mesh quality. The mesh has been generated with the preprocessing system presented in this work.

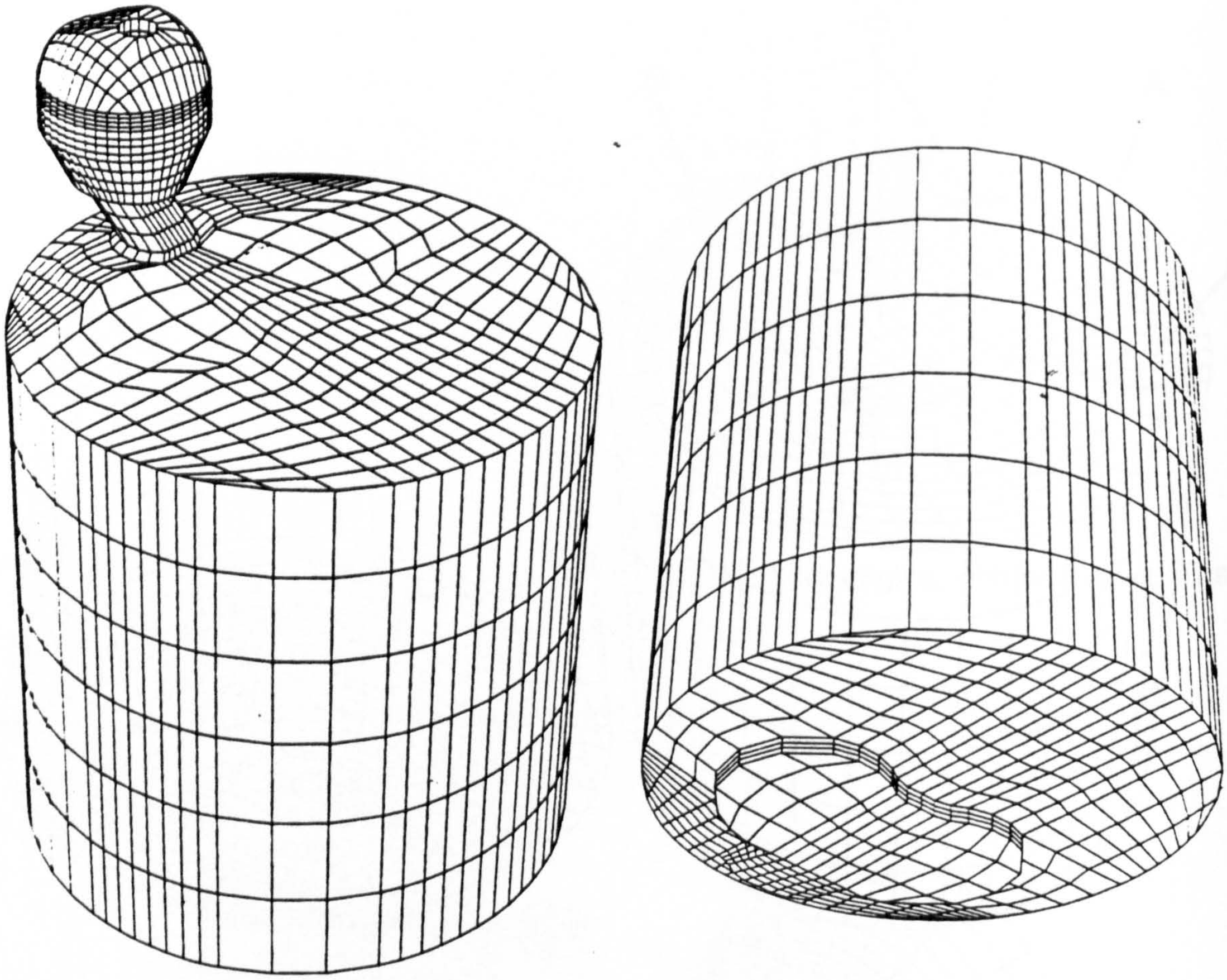


Figure 3.14 A SPEED mesh for the combustion chamber of a Renault in-direct-injection Diesel engine, featuring a swirl chamber, glow plug protrusion, and a bowl-in-piston. The mesh has been generated by the preprocessing system presented in the present work.

Chapter 4

Software tools for interaction with the CFD engine simulation process

The objective in the present chapter is to present a variety of software techniques, also variably called *tools* or *utensils*, for interface with the CFD simulation process, and show how these can be modified to accommodate the special requirements of CFD simulation of engine flow processes, with an emphasis on the SPEED code and the unstructured data organisation it encompasses. Such tools will mainly be provided in the form of algorithms, functions provided by graphics systems such as CAD systems or user interface management systems (UIMS), or ready-made modules of toolkit environments, such as AVS [120] or apE [121].

In general, tools that facilitate the task of CFD simulation of engine processes can be classified in the following categories:

- *Geometry definition*: Software means for definition of the geometrical configuration (combustion chamber geometry) in which the calculation will take place, in terms of computational surfaces and solids.
- *Mesh generation*: Techniques for fitting a computational mesh in the chamber volume.
- *Visualisation*: Numerical and graphical utensils for conversion of numerical data defined in discretised 3D space, into images, in a way that allows their comprehension, interpretation and evaluation.
- *Computer graphics*: Technologies for display of 3D data on the computer screen, in a way that enables perception of their 3D nature.
- *Graphical user interfaces*: Implementation of software means for human communication with the simulation process.
- *Software engineering*: Software design and programming principles and implementations, mainly for storing and handling simulation results as well as for efficient programming of long-lasting codes.

Recent developments, especially in the visualisation area, although not available at the time of initiation of the present research project, are presented for completeness. In the remaining paragraphs, a variety of tools will be presented, several of which have been implemented in the software systems presented in chapters 5 and 6. Their origin and details are given by reference to their creator; a few have been devised by the author and they are the ones without a reference.

4.1 Geometry definition tools

As already noted, engine simulations with the SPEED code involve the generation of three-dimensional computational meshes fitted into complex combustion chamber geometries. The lengthy process of mesh generation can be simplified by the implementation of computer techniques for construction and manipulation of surfaces that define grid boundaries. Such tools, called *solid modelers* provide means for definition of curves, smooth surfaces, that are called *patches*, and combinations thereof to form 3D entities. A variety of solid modelling techniques have been developed in the past decade, such as *constructive solid geometry (CSG)* [57], *boundary-representation (B-rep)* [56], and *analytic solid modelling (ASM)* [58].

Constructive solid geometry (CSG) representations of solids involve the use of simple primitives, such as blocks, cones and spheres, combined by means of Boolean set operators and transformations. CSG models are stored as binary trees, where nodes represent intermediate solids, which are a combination of primitives and intermediate solids lower in the tree [57]. A CSG tree for a simple combustion chamber configuration with closed valve and an offset square piston recess is shown in figure 4.1.a.

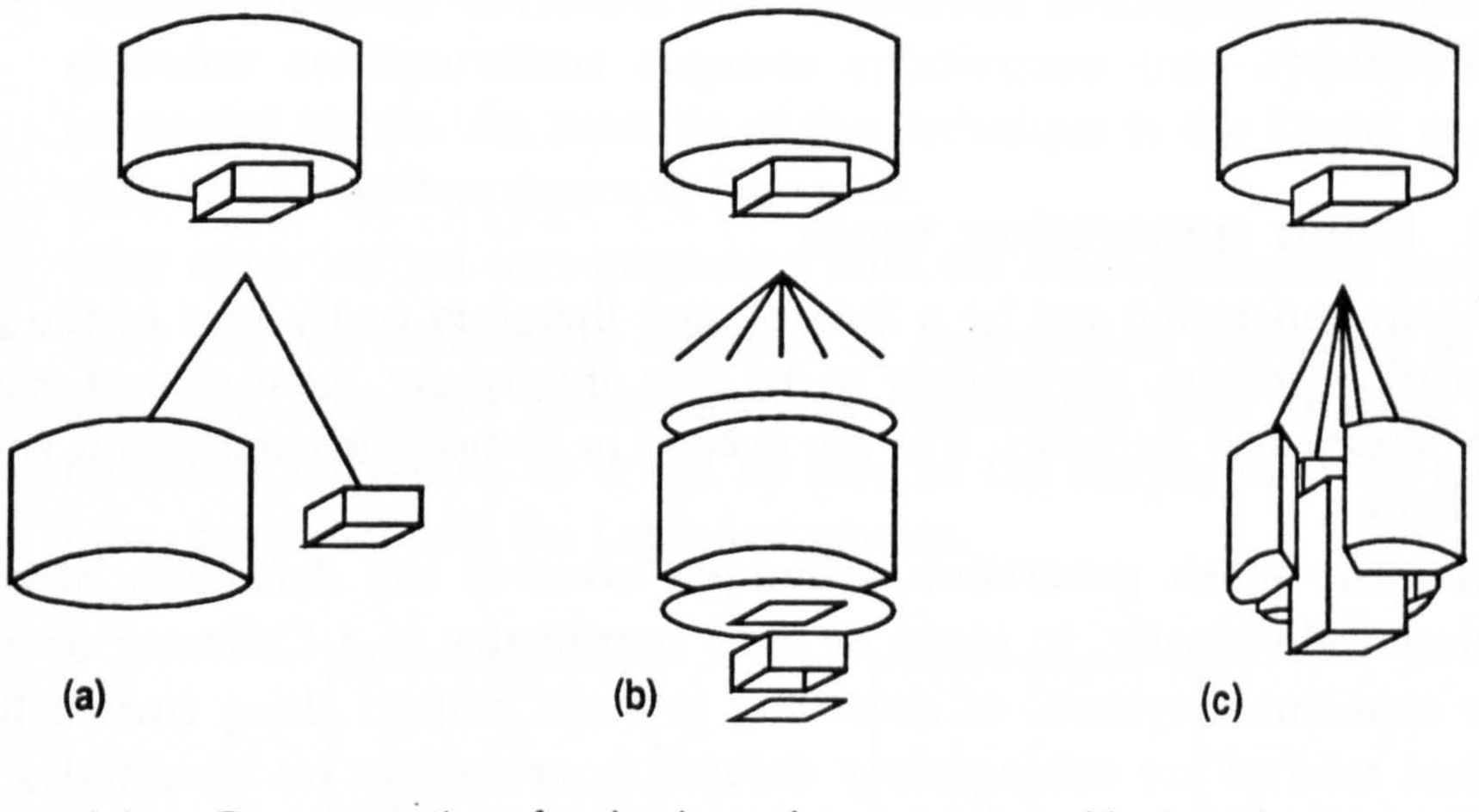


Figure 4.1. Representation of a simple engine geometry, with closed valve and offset square piston. a). CSG representation by two primitive solids, b) B-rep representation by five surfaces, c) ASM representation by five hyperpatches.

Boundary representation solid models (B-reps), [56], represent solids in terms of their smooth-surfaced bounding faces. A B-rep description for the above combustion chamber is illustrated in figure 4.1.b. Construction

operations in such systems ensure that the boundaries are closed and have an orientation associated with them. B-rep systems that accurately represent free-form surfaces of solids are based on non-uniform B-spline surfaces called nurbs [183]. Nurbs-based techniques have been implemented in the Alpha1 [61] and GEOMOD [60] solid modelling systems.

Analytic solid modelling (ASM) techniques represent solids as an assembly of non-overlapping solid components called *hyperpatches,* named so because they are three-dimensional extensions of surface patches [58]. For example, in figure 4.1.c, a simple combustion chamber model is constructed by 5 hyperpatches. Hyperpatches have in general curved faces, and hyperpatch construction operations reflect the natural nesting of curves, surfaces and solids. ASM techniques have been implemented in the PATRAN solid modelling system [6], [59]. ASM is described in more detail in the subsequent chapter, where a system for geometry representation and mesh generation for SPEED simulations is presented

Solid modelling is now a well-established field of computational geometry with several modelling systems available commercially. Most of these employ a combination of the above techniques; for example GEOMOD employs a hybrid CSG-B-rep technique, whereas PATRAN uses a hybrid ASM and CSG technique. A new approach is the incorporation of solid modelling tools in software libraries, available to developers of simulation systems as modular components thereof. Such libraries have been reviewed in section 2.2.

4.2 Mesh generation tools

Mesh generation can be a lengthy and therefore costly part of the flow simulation process, particularly in the case of irregular, unstructured meshes with hexahedral elements, that are needed to fit realistic combustion engine topologies.

A basic mesh generation technique involves the definition of mesh vertices individually, in terms of their coordinates in a Cartesian or other base coordinate system, or according to some pattern along curved lines; surface meshes are subsequently defined from vertices on boundaries and volume meshes are extruded from surface meshes. This technique is time consuming but suitable for generation of unstructured meshes; for this reason and despite the labour overheads involved, it cannot be ruled out for mesh generation in irregular topologies. Several computer codes, such as SUPERTAB [7] and PROSTAR [184] provide such tools, among the other mesh generation tools they support.

In the past fifteen years a number of techniques have evolved for the generation of meshes from boundary surface data generated from solid modelling systems, particularly in the case of simulations with the finite element method. These techniques require some sort of prior definition of the domain geometry, in terms of bounding curves and surfaces while several require the segmentation of the domain geometry in blocks that are meshed individually. Among these techniques four easily discernable groups can be formed: numerical, algebraic, mapping isoparametric, and advancing front ones.

Numerical techniques for mesh generation involve the numerical solution of partial differential equations (PDEs), often the Laplace or Poisson equation being the basic generating system. Such techniques, reviewed in section 2.2, have been implemented in computer programs such as TOMCAT [75], GRAPE [76], EPISO [40], and IGG [73]. They provide smooth meshes with controlled line distribution, and can be efficiently coupled with the numerical solution of flow simulation, in this way enabling mesh adaption to the flow or to user specified points or lines, using different control functions. However they have the following drawbacks:

- They are computationally intensive, since they require the numerical solution of PDEs.
- They are restricted to structured meshes that are transformable to rectangular grids. Therefore, their application to irregular combustion-chamber configurations requires subdivision into appropriately connected blocks. An example of this technique is the Diesel engine with swirl chamber, shown in figure 2.2.
- They allow limited user-interaction with the mesh-generation process.
- They are not sufficiently sensitive to information contained in the specified boundary node locations concerning grid spacing and boundary curvature, as it can be seen in the simple mesh of figure 4.2.a, generated with the Laplacian scheme.

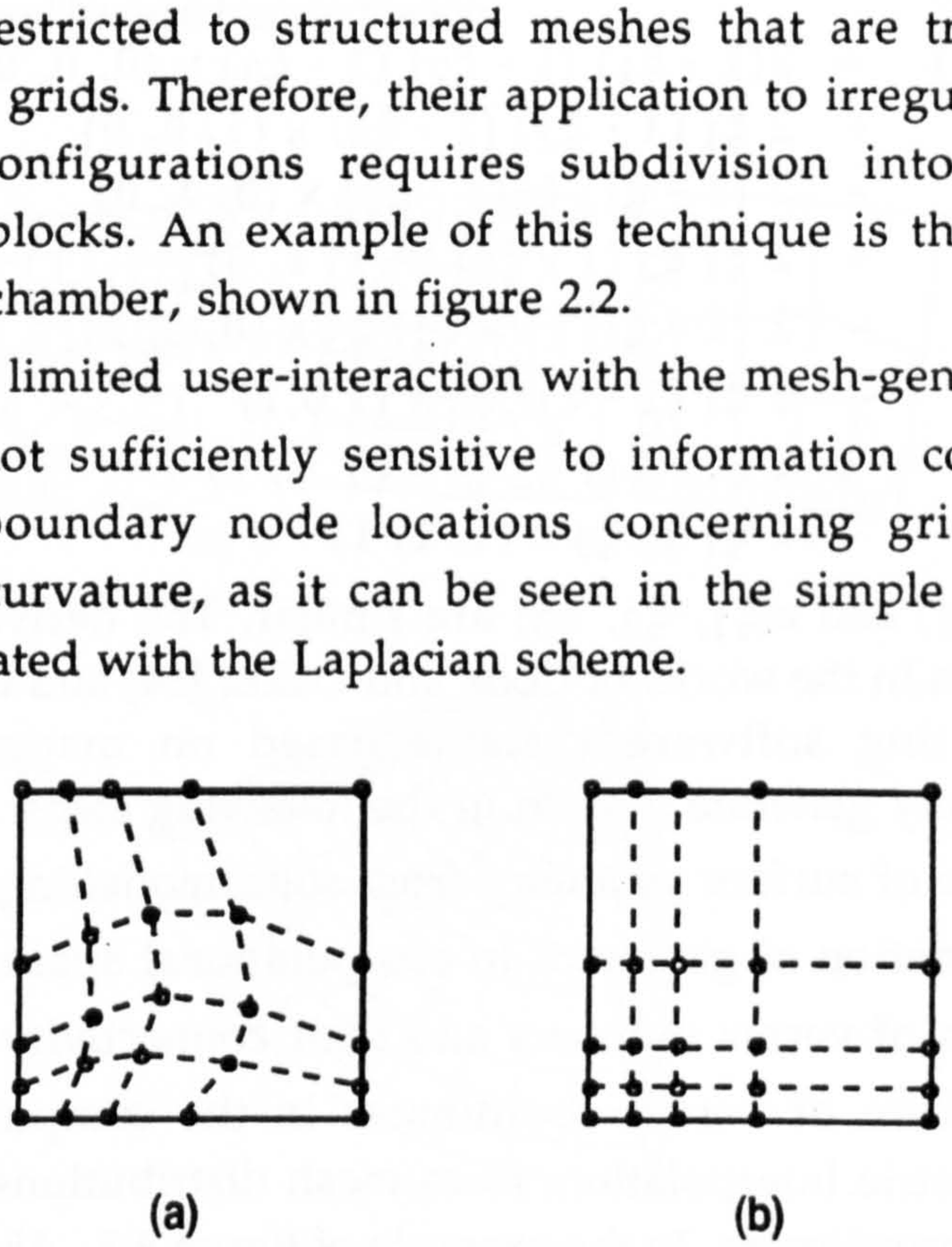


Figure 4.2. Example of effect of boundary node spacing on the location of interior nodes. (a) Numerical mesh generated with the Laplacian scheme, (b) Isoparametric grid.

Algebraic techniques are mainly based on mathematical interpolation functions. Several algebraic techniques have been reviewed in section 2.2. They are computationally efficient and, according to Haber and Abel [70], well-suited to interactive graphics implementation. Most such techniques are restricted to structured or block-structured meshes.

Mapping isoparametric techniques are a subcategory of algebraic ones, but possess some features that are very useful, therefore they are mentioned separately. They involve transformation of the physical domain, defined in the Cartesian coordinates x,y,z, to a computational domain, defined in body or normal coordinates ξ_1, ξ_2, ξ_3, normalised so that they vary from 0 to 1 (see figure 4.3). The coordinates and connectivity of the vertices are generated in the computational space, from mesh distributions along the ξ_1, ξ_2, ξ_3 body coordinates, by application of an isoparametric interpolation scheme, and transformed back to the physical domain. This transformation is defined as follows:

$$
\begin{aligned}
x(\xi_1, \xi_2, \xi_3) \;=\; & \frac{1}{2}\,[(1-\xi_2)\times(\xi_1,0,\xi_3) + \xi_2 \times (\xi_1,1,\xi_2)] \\
& + (1-\xi_1)\times(0,\xi_2,x3) + \xi_1 x\,(1,\xi_2,\xi_3) \\
& + (1-\xi_3)\times(\xi_1,\xi_2,0) + \xi_3 \times (\xi_1,\xi_2,1)] \\
& + \frac{1}{4}\,C(\xi_1,\xi_2,\xi_3)
\end{aligned}
$$

Where:

$$
\begin{aligned}
C(\xi_1, \xi_2, \xi_3) \;=\; & 2(1-\xi_1)(1-\xi_2)(1-\xi_3)\times(0,0,0) \\
& - 2\,\xi_1(1-\xi_2)(1-\xi_3)\times(1,0,0) \\
& - 2\,(1-\xi_1)\,\xi_2(1-\xi_3)\times(0,1,0) \\
& - 2\,\xi_1\,\xi_2\,(1-\xi_3)\times(1,1,0) \\
& - 2\,(1-\xi_1)(1-\xi_2)\,\xi_3\times(0,0,1) \\
& - 2\,\xi_1(1-\xi_2)\,\xi_3\times(1,0,1) \\
& - 2\,(1-\xi_1)\,\xi_2\,\xi_3\times(0,1,1) \\
& - 2\,\xi_1\,\xi_2\,\xi_3\times(1,1,1)
\end{aligned}
$$

and $y(\xi_1, \xi_2, \xi_3)$ and $z(\xi_1, \xi_2, \xi_3)$ are similar. The derivation of the above equation is given in the works of Cook and Oakes [64] and Coons [185].

Grid-generating software systems based on mapping-isoparametric techniques usually generate a mesh in the following steps:

1. Definition of surface geometry from solid modelling systems.
2. Transformation of geometry to computational space.
3. Definition of vertex numbers and their connectivity.
4. Computation of vertex coordinates in the computational space, by isoparametric interpolation, from mesh distributions along the ξ_1, ξ_2, ξ_3 body coordinates. In the example of figure 4.3, $\Delta\xi_1 = \Delta\xi_2 = \Delta\xi_3 = 4$.
5. Mapping of vertex coordinates into the physical space.

The definition of vertex connectivity in a distinct step (step 3), renders this technique suitable for unstructured meshes, since it enables implementation of mesh-refinement algorithms based on vertex connectivity. Several mesh-refinement patterns, that have been generated with mapping isoparametric techniques and are suitable for hexahedral-cell meshes, are shown figures 4.3 and 4.4, the latter taken from Cook and Oakes. As it will be shown in the following chapter, the combination of the above mapping isoparametric technique with block-structured ones provides several useful mesh grading possibilities.

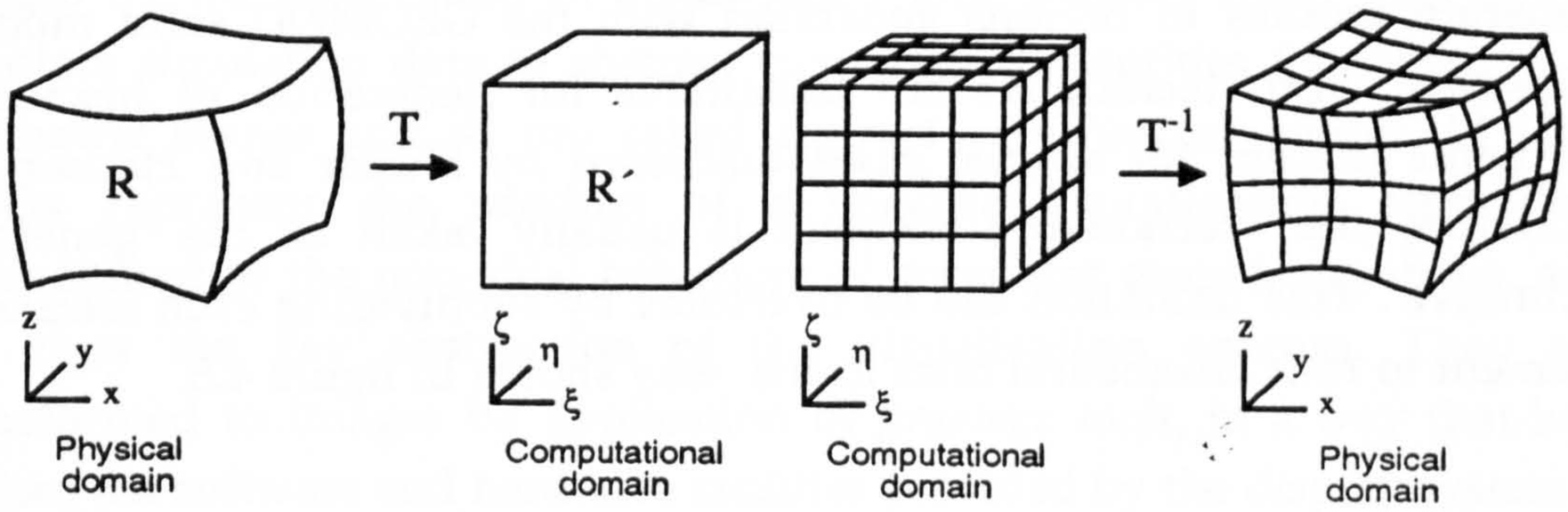


Figure 4.3 Transformation from the physical to the computational domain, at which the mesh is generated, and inverse transformation.

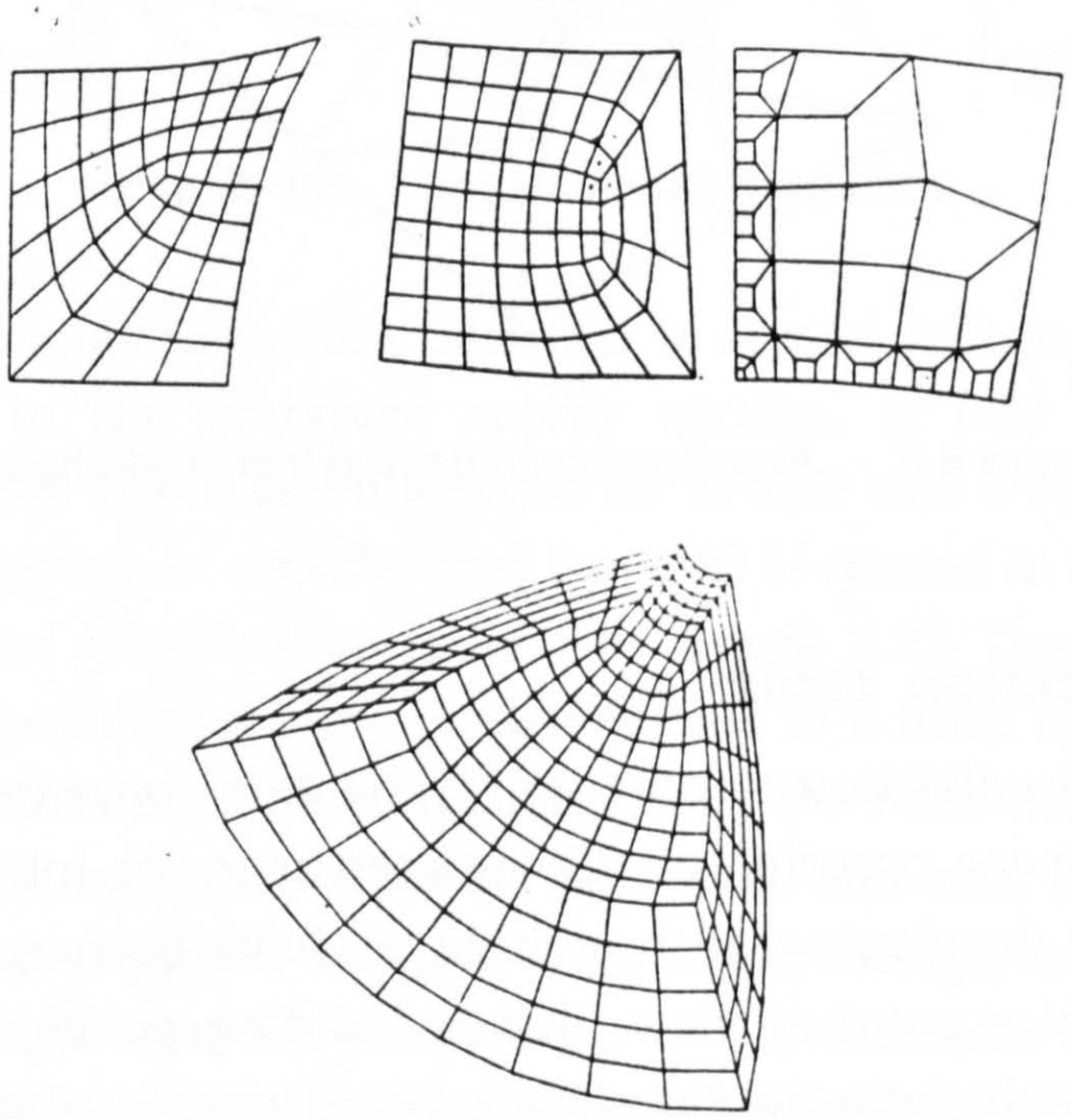


Figure 4.4 Several refinement patterns that are allowed by the mapping isoparametric mesh generation method. They allow for gradual or rapid transition of element density. Top: application in 2D, bottom: application in 3D.

Advancing front techniques have been used since 1982 for generation of unstructured meshes fitted inside irregular, three-dimensional, n-sided regions, from surface meshes defined on each of the n boundaries of these regions. Two widely acknowledged techniques of this type are 'triquamesh', described by Sluiter and Hansen [8] and implemented in the SUPERTAB code, and the advancing front technique described by Dener and Hirsch [73] implemented in the IGG code. The only input they require is an indication of the mesh density. They can be coupled with B-rep solid models, as is the case with SUPERTAB, whose triquamesh implementation can automatically generate meshes in regions generated with the GEOMOD solid modeler. However, both techniques are unsuitable for generation of hexahedral elements (except for special cases discussed by Sluiter and Hansen [8]), therefore the tetrahedral element is usually taken as the 'generation primitive'. This limitation can be overcome by subdividing each tetrahedral element to four hexahedral ones in the way shown in figure 4.5.

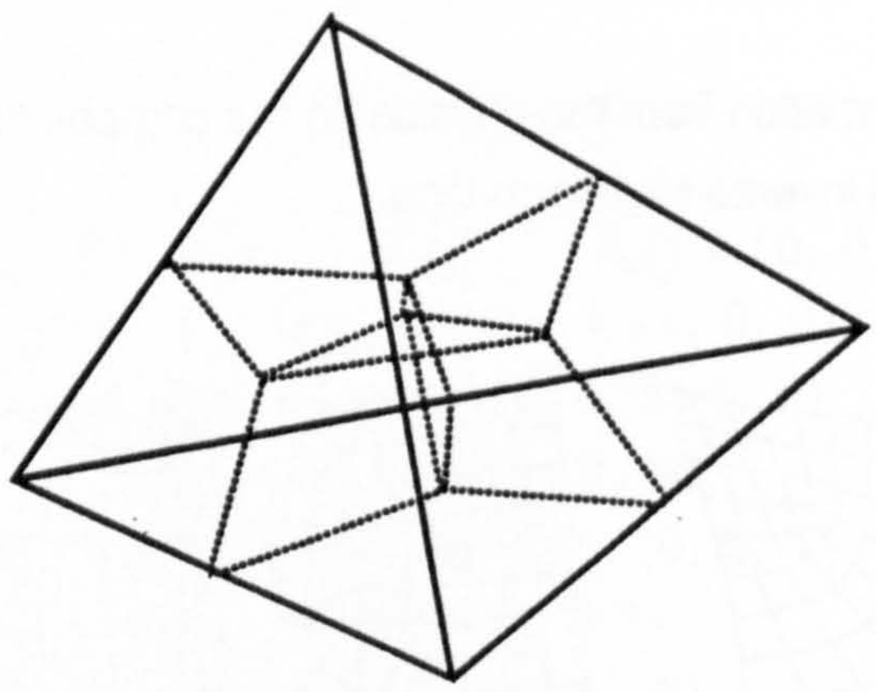


Figure 4.5. Four hexahedra derived from a tetrahedron.

4.3 Visualisation tools

In general visualisation techniques for 3D data concentrate on trying to examine phenomena occuring on 3D surfaces or in the interior of volumes. Most efforts at developing such techniques have been conceived from a computer graphics point of view, that is building points, lines, polygons, spheres etc. directly from the simulation data. However, the domain of CFD simulation is not confined to primitive structures with direct graphical analogs. Such simulations also result in scalar or vector fields defined over a 3D structure of points. Some structures, such as orthogonal, rectangular,

mapped meshes or groups of spray droplets, are analogous to graphical primitives and their translation to graphics is direct. However, for the scientist working with irregular grids, like those involved in SPEED, there is a gap between the scientist's computational domain and graphics domain. This gap is bridged by the so-called *visualisation pipeline*, a notion introduced by Haber and McNabb [4], that is inspired by the conventional graphics pipeline, on which graphics technology is based. The visualisation pipeline is the process by which simulation data are transformed into images, as shown in figure 4.6. Simulation data are converted to graphical images by the application of *visualisation* and *graphics tools*. The former translate simulation data to abstract computational entities (descriptions of geometric scenes and so on) called *abstract visualisation objects (AVOs)*. AVOs represent the product of a specific visualisation algorithm, independent of the non-ideal behaviors of computer graphics systems, and are thus the key abstraction of the visualisation process. They are transformed to images by application of *graphics tools*, in a way that best utilises the software and hardware facilities provided by the display system.



Figure 4.6. The visualisation pipeline.

Visualisation tools have been classified to several groups, some of which are presented in the literature survey section. It will best serve the presentation of the visualisation systems for SPEED that will be presented in the following chapters to consider that the AVO is created in two stages: *Data manipulation* and *graphical mapping* (see figure 4.7). Data manipulation involves the translation of raw simulation data to a form that is perceptive to visual operations. Such manipulation can be the normalisation of data to a given range of values, such as between zero and one, so that graphics processing can be applied on data with known range, or by the piston velocity, so that data can be compared with experimental data. Additional visualisation entities are obtained from the raw data at this stage, an example being temperature fields from internal energy values, or particle traces obtained from flow velocity values. Graphical mapping involves the transformation of simulation data into structures that can be displayed

graphically on the computer screen. Such operations include definition of cutting planes, isosurfaces, and shading of 3D scenes.

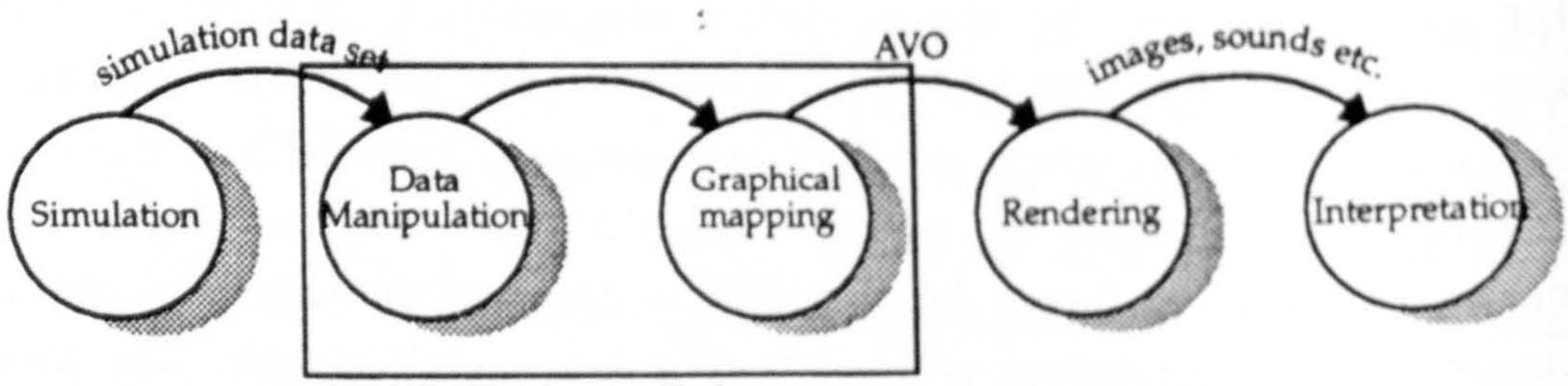


Figure 4.7. The visualisation pipeline, showing the different stages in the pipeline covered by visualisation tools.

Several visualisation tools presented in this section have been implemented in the software systems for the visualisation of SPEED results that will be presented in subsequent chapters. For the purpose of completeness as well as to point possible ways of enhancing the suggested visualisation environments for SPEED, visualisation tools that have been developed after the present project have been included. This section is organised into four parts: data manipulation tools, graphical mapping tools, scalar visualisation tools, and vector visualisation tools.

4.3.1 Data manipulation tools

Data manipulation tools are numerical tools that map SPEED simulation results data to a more useful form, from a storage, interpretation, or visualisation point of view, and before visual operations can be applied to it; For this reason they could be called numerical mapping tools or filters. They operate both on geometry and field data. Such tools include data contraction, interpolation, normalisation, filtering, and mesh rezoning.

4.3.1.1 Data contraction

The raw simulation results produced by SPEED often come in voluminous data sets and for several discrete times. It is highly desirable to reduce the size of such data in order to:

- simplify the visualisation problem, by creation of simpler, more comprehensive fields;

- focus attention on specific field ranges or regions of interest, in this way simplifying interpretation of the data;
- facilitate storage and transmission through a computer network, in a distributed processing system.

Several data contraction tools for the SPEED code, including cutting volumes, surface sampling, mesh rezoning, and thresholding, are discussed in the following paragraphs.

Cutting volumes

One data reduction technique is to visualise only a portion of the computational domain. This is achieved with the definition of surfaces or volumes, and the interpolation of the raw data on surfaces, or the exclusion from the data base of data that is not included in rectilinear volumes (figure 4.8). An algorithm to achieve this is given in appendix A.1.

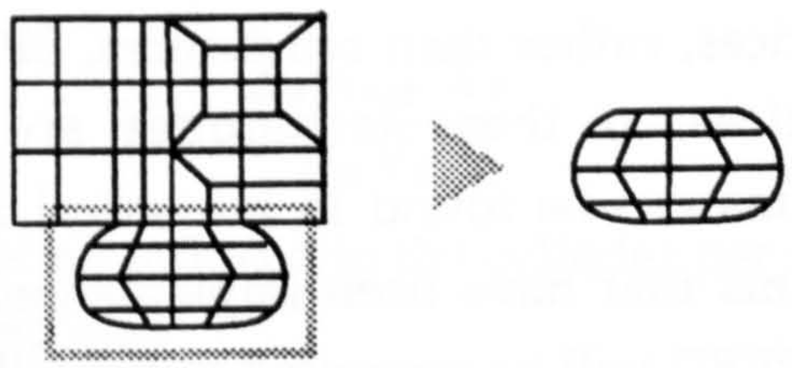


Figure 4.8 Sampling out of the data base of those data that is totally included within a rectilinear volume. The rectilinear sampling volume is represented in this 2D figure by a gray-lined rectangle.

Surface sampling

Often only data on the boundaries of the computational domain is of interest. This is the case in calculations where the SPEED code is used for the production of temperature distributions on the surfaces of the piston crown, cylinder liner, and cylinder head. A technique for creation of a surface mesh on wall boundaries, based on the indirect addressing structure of the SPEED code, is given in appendix A.2.

Thresholding

Mapping on a new mesh can be combined with exclusion of data that is outside a user specified range of values. This technique, called thresholding,

can be useful for focussing on the interesting portions of the data while at the same time removing a lot of visual mess.

Field contraction

Field contraction techniques can be used to simplify the visualisation problem. An example of such a technique is the calculation of the scalar kinetic energy density field $\frac{1}{2}\rho(u \cdot u_p)$ from the vector velocity field u and the average piston velocity u_p, or the calculation of the scalar velocity magnitude field from the vector velocity field u. In this way, we are provided with an alternative way of description of vector fields.

4.3.1.2 Interpolation to vertices

Since several visualisation software systems (e.g. PATRAN, AVS) require fields defined on vertices, rather than cell-centres, SPEED results need to be interpolated to vertices, if these techniques are to be used. Several interpolation algorithms can be found in numerical analysis textbooks. The interpolation algorithms that have been implemented in the pre- and post-processing codes of SPEED will be presented in the following chapter.

4.3.1.3 Normalisation

Data normalisation, a common data preparation operation, scales the simulation data to a given range of visualisation values, such as between zero and one. It is necessary to normalise data, regardless of the type of data, so that:

- simulation data is scaled or fitted between user selected visualisation values, that relate to the dimensions and resolution of the screen;
- further graphics processing can work on data with a known range or values;
- direct comparison with experimental data, where data are often normalised (e.g. gas velocity magnitude is often normalised by piston velocity magnitude), is possible.

Several normalisation techniques for the visualisation of droplets in injected fuel sprays have been implemented in the pre- and post-processing systems for SPEED and are discussed in subsequent chapters.

4.3.1.4 Filtering

Filtering techniques are used to extract additional information from the simulation results. For example, in engine simulations, engineers may be interested in the volumetric efficiency, the mean indicated pressure, the specific fuel consumption of the simulated engine, as well as physical integral properties of the moving gas in the cylinder, that are not directly output in the SPEED results file. Formulas for numerical calculation of the above properties from the SPEED results file are given subsequently:

Volumetric efficiency λ

The volumetric efficiency λ, only used with four stroke engines [157], is the parameter used to measure the effectiveness of an engine's induction process. Typical maximum values for λ are in the range of 80 to 90 percent [187]. It can be calculated from the mass-flow rate through the valves (calculated by SPEED) and the displacement volume with the following formula:

$$\lambda = \frac{\sum \dot{m}_{air}\ \Delta\phi}{\rho_a\, V_d} \tag{4.1}$$

where: $\dot{m}_{air}$ mass of air induced in the cylinder per timestep [kg/ °CA]
$\Delta\phi$ timestep size
ρ_a atmospheric air density ($\rho_a = 1.3\ kg/m^3$)
V_d displacement volume

Mean indicated pressure

The mean indicated pressure p_i is the average rate of work transferred from the gases in the engine cylinder to the piston during the compression and expansion strokes, divided by the displacement volume [157]. It can be deducted from the pressure in the cylinder according to the following formula:

$$p_i = 0.5 \sum_{\phi} p(\phi)\left(1 + \frac{\cos\phi}{\sqrt{\mu^2 - \sin^2\phi}}\right)\sin\phi\ \Delta\phi \tag{4.2}$$

where: ϕ crank angle
$p(\phi)$ pressure [Pa]
l connecting rod length [m]
r crank radius [m]
$\mu = \frac{l}{r}$

Specific fuel consumption b_i

The specific fuel consumption b_i, showing how efficiently an engine is using the supplied fuel to produce work, is defined as the fuel flow rate per unit power output [188]. It is calculated from the total mass and the mean indicated pressure with the form [187]:

$$b_i = \frac{Q}{p_i V_d} \tag{4.3}$$

where Q is the total mass of injected fuel [kg] and b_i is calculated in $\frac{g}{kWh}$ units. For petrol engines b_i is about 270 $\frac{g}{kWh}$ whereas it is lower than 200 $\frac{g}{kWh}$ for large Diesel engines [187].

Kinetic energy

The kinetic energy E_T of the moving gas in the combustion chamber is defined as follows:

$$E_T = \frac{1}{2} \sum_{1}^{NC} m_c \cdot |\mathbf{u}_c|^2 \tag{4.4}$$

where: m_c mass of cell
$\mathbf{u}_c$ gas velocity at cell centroid

Angular momentum

The angular momentum **L** of the moving gas in the chamber is defined as follows:

$$\mathbf{L} = \sum_{1}^{NC} m_c \cdot (\mathbf{u}_c \otimes \mathbf{r}_c) \tag{4.5}$$

where: m_c mass of cell
$\mathbf{u}_c$ gas velocity at cell centroid
$\mathbf{r}_c$ distance of cell centroid from centre of mass

Swirl number

The swirl number can be calculated in two different ways: a) assuming that the combustion chamber is a solid disc and b) with respect to the previously calculated angular momenum.

a. Assuming that the combustion chamber is a solid disk

The swirl number of the engine, assuming that the combustion chamber is a solid disk with diameter that of the cylinder and mass that of the gas is:

$$S_N = \frac{8\ |\mathbf{L}|}{m \cdot D^2 \cdot \omega_{crank}} \tag{4.6}$$

where: **L** angular momentum
m mass of gasses in the cylinder
ω_{crank} angular velocity of the crank
D piston diameter

b. With respect to the angular momentum

The moment of inertia I_L around the axis of the angular momentum is:

$$I_L = \sum_{1}^{NC} m_c \cdot |\mathbf{b}_c|^2 \tag{4.7}$$

where: m_c mass of cell

$\mathbf{b}_c$ distance of cell centroid from axis of **L**. $\mathbf{b}_c = \frac{1}{|\mathbf{L}|}\{\mathbf{r}_c \otimes \mathbf{L}\}$

The swirl number of the engine, with respect to the angular momentum vector is:

$$S_N = \frac{\omega}{\omega_{crank}} = \frac{|\mathbf{L}|}{I_L \cdot \omega_{crank}} \tag{4.8}$$

where: ω Is the effective angular velocity ($\omega = \frac{|\mathbf{L}|}{I_L}$)

4.3.1.5 Mesh rezoning

The raw simulation data is often defined by a mesh that is irregularly structured, or defined in a non-cartesian coordinate system. Mesh rezoning tools resample raw geometry data to an overlapping grid structure (see figure 4.9). This might be done to map data to a regular rectangular grid which more readily translates to a grid of pixels, i.e. a computer image, or can directly accept volume rendering techniques. Algorithms for this task are given by Gelberg [186].

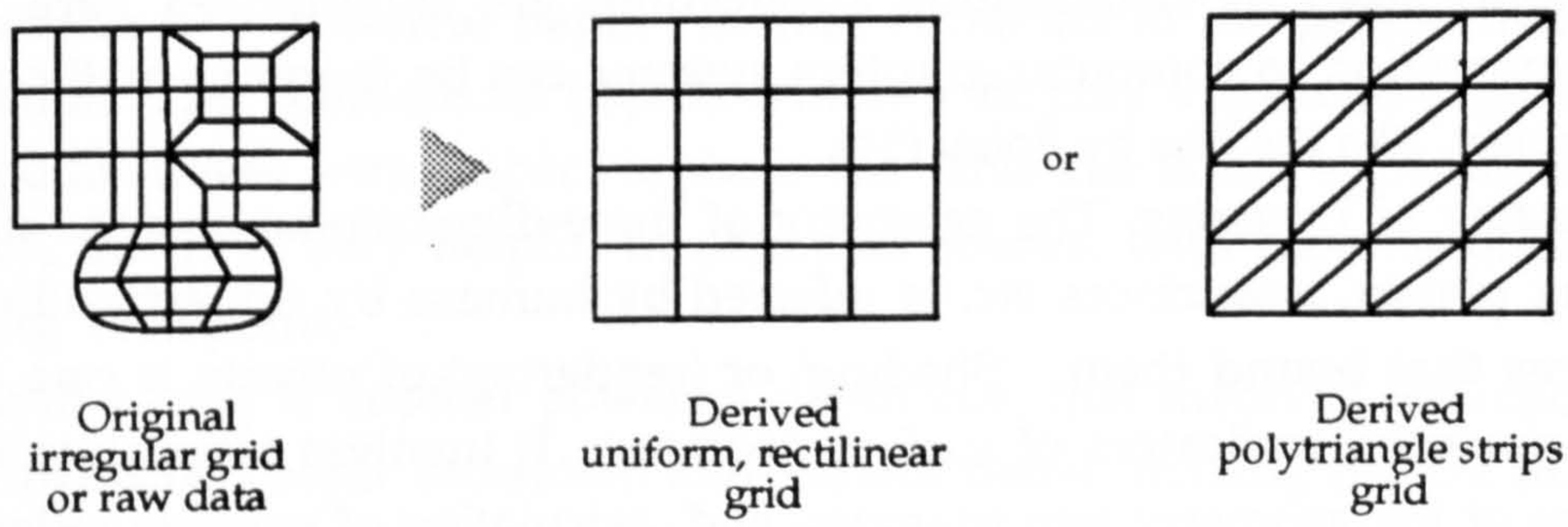


Figure 4.9 Interpolation of data from a fine irregular grid onto a uniform rectilinear or polytriangle-strips grid.

4.3.2 Graphical mapping tools

Graphical mapping tools transform the raw simulation data or the previously numerically manipulated data into structures that can be displayed on the computer screen with colour graphics. Such tools are colour mapping, rendering, cutting planes, and sampling planes.

4.3.2.1 Colour mapping

An important graphical mapping task is assigning data to a colour model. A familiar technique is to map a scalar value at each point in the domain to a table of colours, by means of a colour-transfer function. An understanding of the colour function is essential in interpreting whether a given colour indicates a high or low value of the scalar field. A step function produces images with regions of uniform colour. The human perceptual system easily detects boundaries where colour is discontinuous, so this provides an alternative method for displaying contours without explicitly computing their position.

Usually, colour maps are done by means of linear functions in the RGB (red-green-blue) colour space. This typically produces unsatisfactory colour maps, therefore it has been suggested by Haber [2] to generate perceptually linear colour maps in alternative colour spaces, such as the HSB (hue-saturation-brightness) one, described by Smith [189].

4.3.2.2 Computer graphics tools

Three-dimensional computer graphics tools provide perceptual cues for the viewer to interpret the geometry and positions of the objects involved in mesh and results visualisations scenes. Several perceptual cues that are commonly used in visualisation applications are summarised here. Their implementation in computer graphics systems can be found in textbooks on the subject, like the one by Foley [54].

Shading or rendering. The geometry of three-dimensional objects, such as cutting planes, isosurfaces etc. is inferred by humans by understanding the surfaces that bound them. 'Shading' or 'rendering' of objects is one of the most powerful indicators of surface geometry. It involves tesselation of the surface of the geometry into triangles, and calculation of variable reflectance of light as a function of surface normal vectors for each triangle, and angles of the incident light. Shading algorithms range from very fast 'individually shaded polygons' algorithms with diffuse reflections, that result to non-

smoothly shaded images, to computer intensive 'Gouraud' [193] and 'Phong' [194] algorithms with diffuse reflection, that result to smoothly shaded images by colour interpolation techniques. The latter techniques are widely used in visualisation applications, because of the smooth shading result they provide at allowable computing times in the case of state of the art graphics workstations. Secondary lighting effects, such as specular reflections, enhance understanding of an image depicting a complex geometry. However such effects are achieved by computationally intensive tools, such as extended Gouraud or Phong shading with specular reflections, or 'ray tracing' [54] and although they have been used for presentation visualisations, as it will be shown in chapter 6, they are not suitable for everyday visualisation of the voluminous data sets involved in engine simulations with SPEED, where speed of processing is very important. Very good examples of the effects of different illumination and shading algorithms on the same scene are given by Foley [54].

Occlusion, by removal of lines and surfaces that are hidden by opaque objects nearer to the vier, is one of the major perceptual cues that establishes the relative positions of objects in a scene. This is a powerful cue that allows us to rapidly infer the relative distances of objects with respect to the view direction. However the removal of hidden lines and surfaces is a fundamental problem in computer graphics and can account for a good deal of the expense of rendering. Several hardware systems for rendering have resolved this problem using additional memory, called z-buffer, whereas software systems often use an explicit depth-sort of the objects in the scene to resolve this problem.

Perspective projection provides another powerful depth cue, indicating the distance of objects from the viewer. While perspective is a useful cue, orthographic projection techniques are preferred in engineering applications, because they maintain a uniform length metric in the display.

Parallax is a powerful depth cue that is related to the perspective view projection. The positions of objects within a display appear to shift as the viewpoint moves with respect to the scene. This cue is not apparent in still images, but it is very helpful in animated scenes, with moving objects or moving viewpoint.

Depth cueing is another powerful depth cue, that involves reductions in light intensity, colour saturation and surface detail, making objects that are distant from the user to appear dimmer than closer ones. This is achieved by calculating the colour of the particle as an interpolation of the colour of a

primitive and a user specified depth-cue colour, which is typically the colour of the background.

4.3.2.3 Sampling planes

Several field visualisation techniques involve interaction with a sampling 'probe' introduced into the calculation domain. When this probe is a plane, and fields are interpolated onto it and displayed as a colour distribution or as a 'hedgehog' of arrows, then this probe is called 'sampling plane' or 'slicing plane'.

Orthogonal slicing is a technique which operates on orthogonal, structured meshes of the I,J,K type. The raw simulation data may be defined or mapped, with data manipulation tools mentioned earlier, into such meshes. An orthogonal slice is a slice of constant I, H, or K value.

Arbitrary slicing requires computing the intersection of a uniform planar grid with cells of the data set. Algorithms for the definition of slicing planes in both uniform-rectilinear and irregular meshes are given by Gelberg in [186].

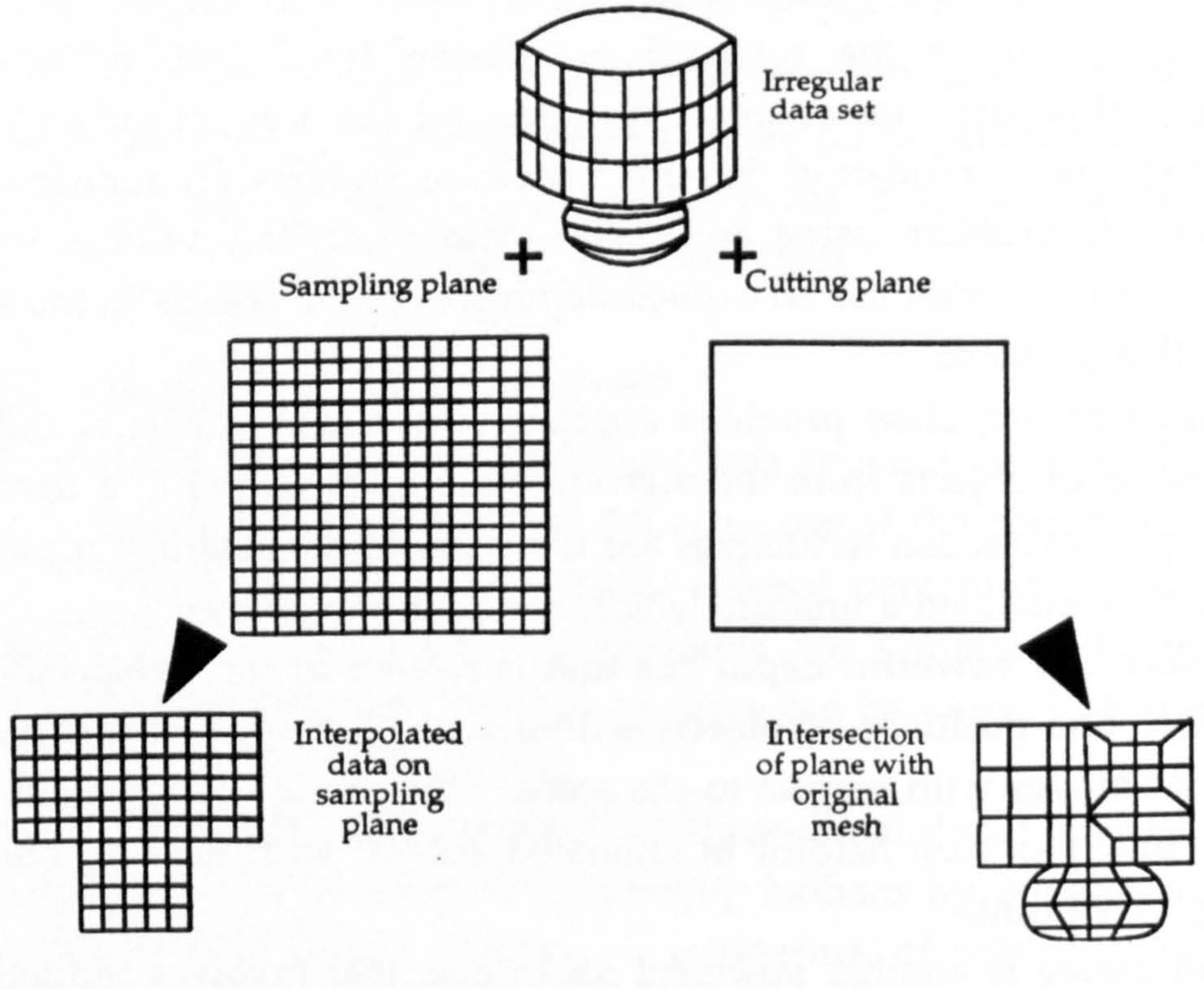


Figure 4.10 Illustration of the sampling-plane and cutting-plane idioms. a. Sampling plane. b. Cutting plane.

4.3.2.4 Cutting planes

A 'cutting plane' probe can also be used to define a geometric cut through the domain, revealing internal surfaces of information.The cutting plane approach differs from the sampling plane approach in that the original structure of the mesh is preserved. This may be deemed advantageous – it informs the viewer of the true resolution of the simulation data; or disadvantageous – vectors will be plotted irregularly if the mesh is irregular, reducing the legibility of the view.

4.3.3 Tools for visualisation of scalar fields

Scalar as well as vector fields in three-dimensional domains are basic to CFD simulation of engine processes; a variety of tools have been developed for the visualisation of such fields, ranging from 1D tools, such as point-probes, to 2D tools, such as contour plots on cutting planes, and to 3D tools, such as isosurfaces. These tools are discussed below.

4.3.3.1 One-dimensional tools

A useful technique for getting quantitative information out of scalar fields is *point probing*. That is displaying a three-dimensional cursor in a graphical representation of the data and reporting on: the position of the cursor in coordinates, cell identifier, and the interpolated value ψ at that position.

There have been several user-interface approaches to control probing operations in an interactive environment. Visual3 [127] displays a slice plane in both a 2D and a 3D view of the model and confines the probe to that plane. This allows for specific probe placement using a 2D input device, the system's mouse. AVS [120] allows not only for point, but also for what is known as neighbourhood probing; this involves the display of a sub-set of the data, that is selected by a window, by a point or by a threshold value. Once a neighbourhood has been defined, the data can be displayed for this subset (e.g. contouring, data values at nodes) [186].

Integral to probing and other techniques is determining in which cell the probe lies. An algorithm for this, that has been implemented in the pre- and post-processing system of SPEED, is given in appendix A.3. It is very fast, it is applicable on unstructured data with hexahedral cells, and it works in cases of highly distorted meshes where the nearest vertex to point P may not belong to the cell point P lies within.

4.3.3.2 Two-dimensional tools

Visualisation techniques for scalar fields defined over two-dimensional domains have long been available and are familiar to most engineers. Two-dimensional techniques for visualisation of scalar fields can be applied on three-dimensional fields by interpolation of such fields on slicing or cutting planes. One such form of visualisation, common until the last half of the 80's, is the line contour, a set of curves of constant value of a scalar field; an example is shown in figure 4.11, taken from Begleris [43], where line contours are used to visualise the swirl field in a combustion chamber. Line-contour plots require some effort on the part of the viewer to interpret the value of the scalar function at any given point. However, they do make it very easy to detect regions of high and low gradient (based on the spacing of contours) and the precise orientation of the gradient (perpendicular to the contours).

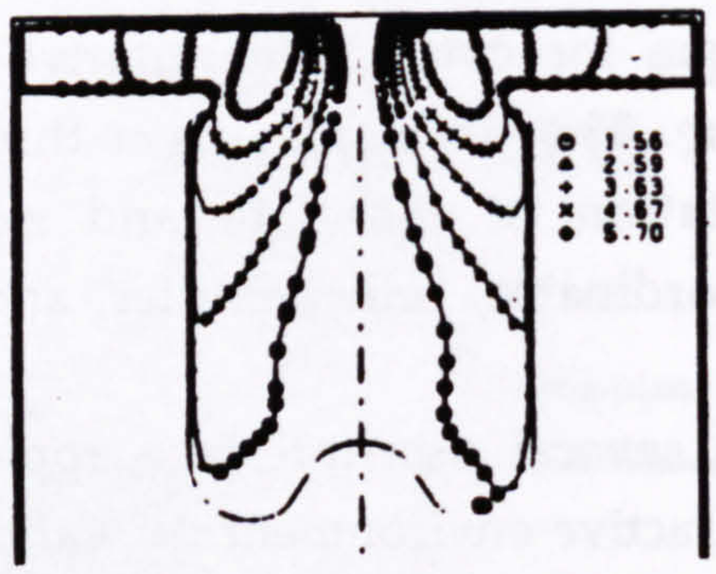


Figure 4.11 Line-contour plot of the swirl component of mean velocity (Begleris [43].

In some cases, the contours themselves are of interest, as in the case of stream lines, that are a set of computed integral curves through the velocity field that are tangential to it at a specified time. A method for calculation of stream lines in unstructured grids is given by Lohner [190]. Stream lines can be very useful in displaying regions of recirculation, that in 2D are represented by closed stream curves.

Another familiar technique, that became popular in the end of the 80's is mapping the scalar value at each point in the domain to a table of colours. This provides a type of plot that is called continuous-tone contour plot. The user can interpret such a plot by comparing it to a colour key or colour scale,

representing the colour transfer function. A step function produces images with regions of uniform colour, called filled-area contour plots or fringe plots. Figure 4.12 indicates the conceptual differences between line, filled-area, and continuous-tone contour plots. Several more examples can be found in chapter 5.

Contour plots on cutting planes comprise a frequently used form of visualisation of scalar fields that is familiar to most engineers. They feature several vantage points in that:

- they convey their information in a single view, whereas three-dimensional tools, such as iso-surfaces, contain occluded parts;
- a number of contours, corresponding to equal values of a scalar field, can be displayed in an efficient way, whereas the same result with isosurfaces would require a number of semi-transparent isosurfaces, that are often confusing.
- engineers are used to interpreting such forms of visualisation.

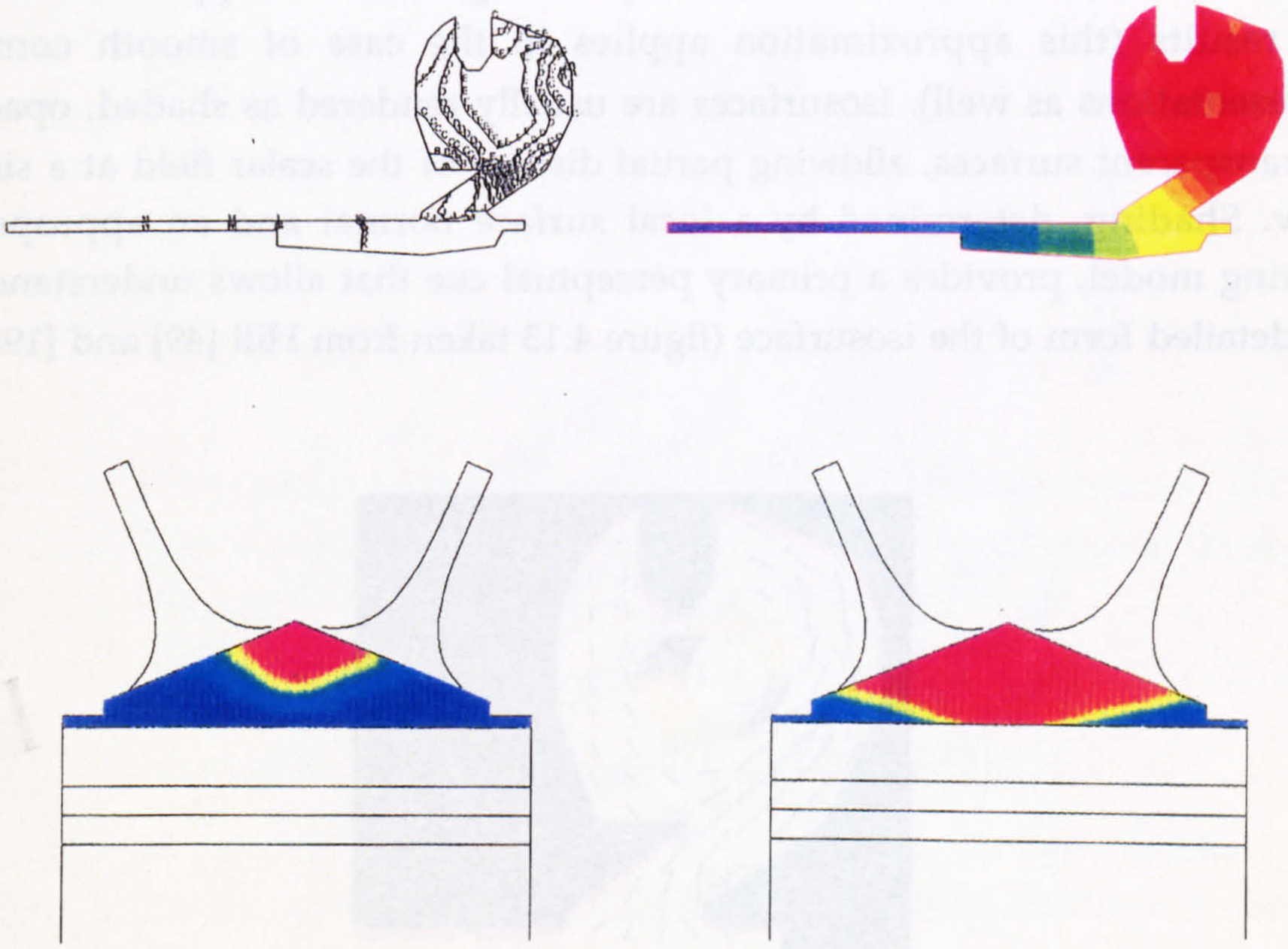


Figure 4.12 Different types of contour plots on cutting planes through the combustion chamber. Top left: line-contours and Top right: filled-area contours of pressure in an IDI engine. Bottom: continuous-tone contours of temperature during combustion; pent-roofed petrol engine.

4.3.3.3 Three-dimensional tools

The advent of super-workstations, with fast processing, memory access, and rendering capability, encouraged the development of volume-oriented utensils for the visualisation of three-dimensional scalar fields. Such utensils are isosurfaces, carpet plots, and multiple cutting planes.

Isosurfaces

An isosurface is a surface of constant, user-specified value, that comprises the three-dimensional analog of a contour curve. A technique for computing isosurfaces on structured data, provided by the 'marching cubes' algorithm of Lorenson and Cline [191], uses a lookup table based on binary (above/below threshold) values at the corners of a cell, to produce high-speed polygonal rendering of threshold surfaces. An extension of the marching cubes algorithm for unstructured data is provided by Gallagher and Nagtegaal [225]. Both techniques use local gradients to produce visually smooth isosurfaces across element boundaries, in this way adding a level of approximation to the results (this approximation applies in the case of smooth contour representations as well). Isosurfaces are usually rendered as shaded, opaque, or transparent surfaces, allowing partial display of the scalar field at a single view. Shading, determined by a local surface normal and an appropriate lighting model, provides a primary perceptual cue that allows understanding the detailed form of the isosurface (figure 4.13 taken from Hill [49] and [192]).

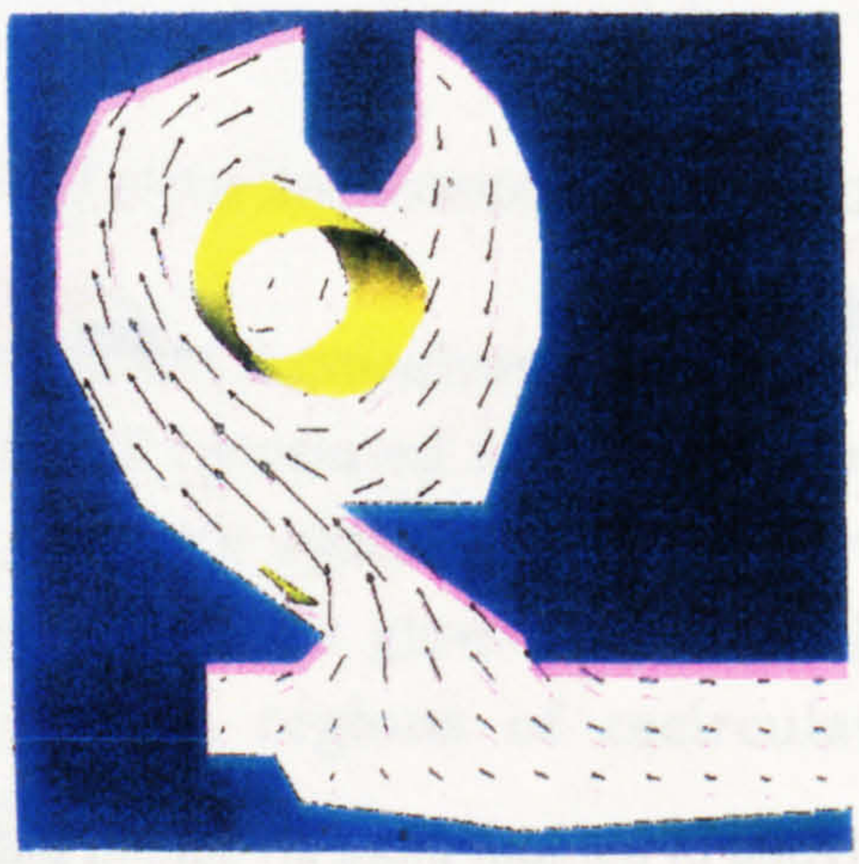


Figure 4.13 Flow-velocity vectors and pressure isosurface in the prechamber of an IDI engine [192].

Occluded parts of an isosurface can be viewed by rotation with interactive graphics techniques. Hidden parts of the engine geometry and hidden isosurfaces, when more than one scalar levels are represented as nested surfaces, can become visible making the isosurfaces semi-transparent. This comprises a very powerful means of visualisation, but it quickly becomes difficult to interpret the geometry of the individual surfaces when there are more than a few nested surfaces.

Multiple cutting planes

A very useful technique for exploring three-dimensional fields is provided by application of the cutting plane tool at several locations in the computational domain, in a fashion similar to that shown in figure 4.14. Multiple cutting planes can be perpendicular (fig. 4.14 top), parallel (fig. 4.14 bottom), or form any angle with each other.

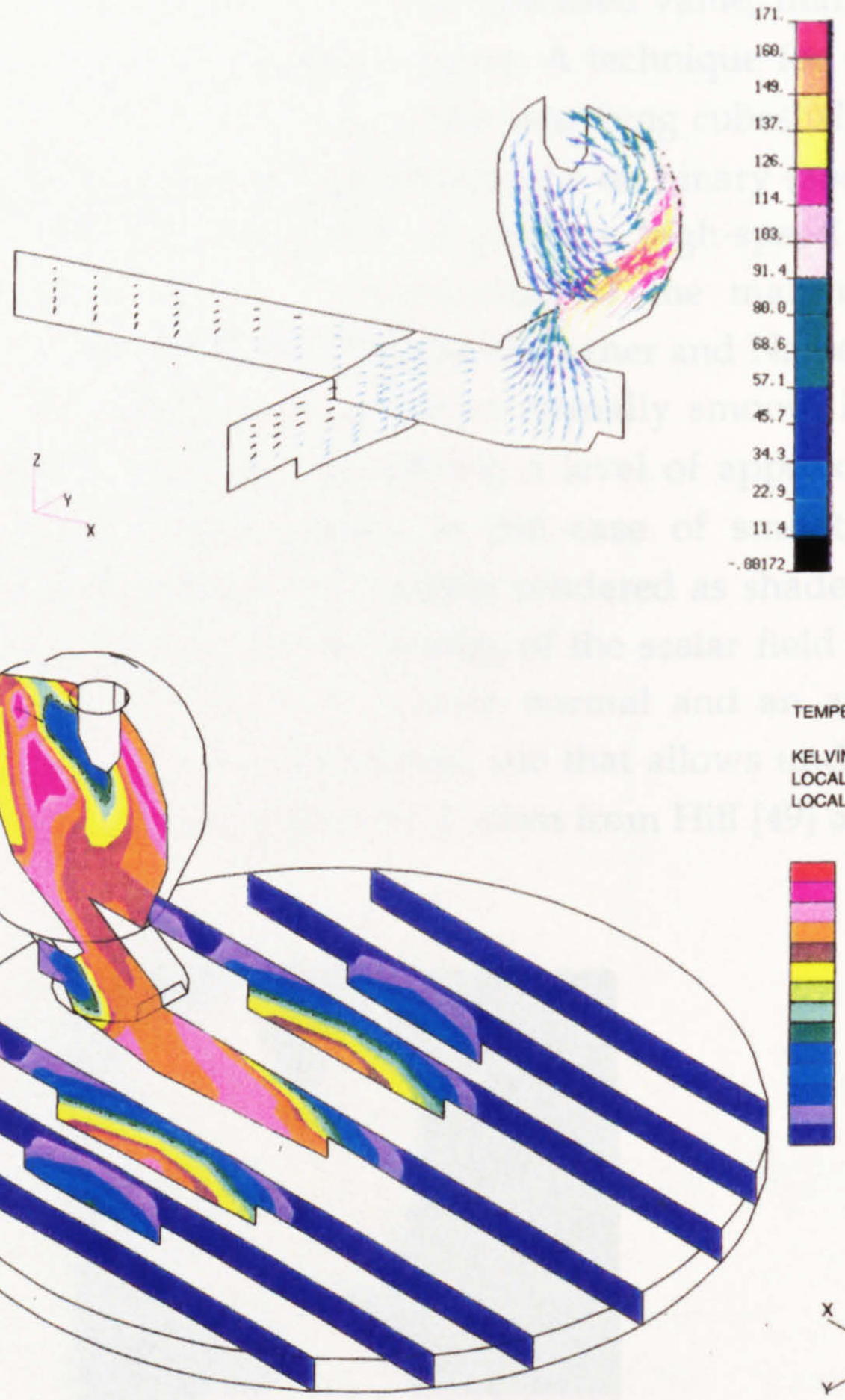


Figure 4.14 Multiple cutting planes in the combustion chamber of an IDI engine. Top: flow velocity vectors at TDC of compression. Bottom: filled-area contours of temperature during combustion.

Carpet plots

Sometimes it is useful to represent scalar fields not only by means of filled-area contour plots on cutting planes, but also by assuming that the surface of the cutting plane is distorted by a force that has direction perpendicular to it, and magnitude that is a function of the scalar field. This type of plot is called carpet plot, and it is particularly useful in the case of representations of velocity profiles on surfaces in the flow field, such as cutting planes or valve curtains. A carpet plot of the swirl component of the flow velocity on a cutting plane through the swirl-chamber of an IDI engine is given in figure 4.15.

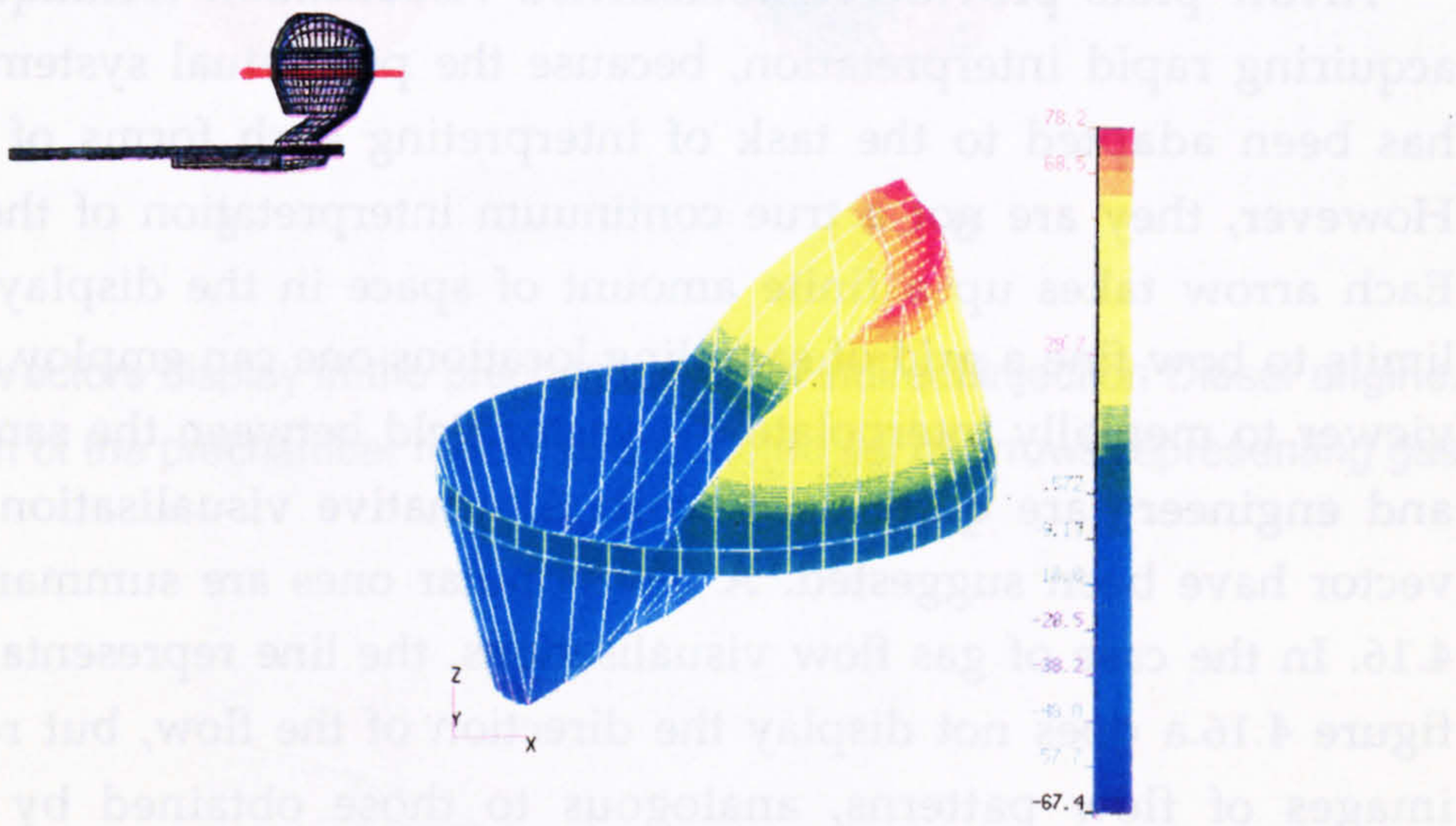


Figure 4.15 A carpet plot of the swirl-component of flow-velocity (TDC of compression) at a cutting plane through the swirl chamber of an IDI engine. The location of the cutting plane is indicated by a red line at the top left of the figure.

4.3.4 Tools for visualisation of vector fields

The task of visualising a field of three-dimensional vectors is not equivalent to visualising three-independent scalar fields, because a vector is a unified entity, with direction and magnitude. These two essential notions of a vector must be conveyed by the visualisation idiom for vectors. Display techniques for vectors can be classified in two categories:

- direct, providing pictorial representations of the local characteristics of the vector field, that is magnitude and direction, in a direct way;

- indirect, visualising the physical effect of the vector field, allowing insight into alternative aspects of its local characteristics.

4.3.4.1 Direct vector tools

The most common direct method for displaying vectors is the *arrow* plot, that is the construction of the geometric representation of an arrow, at discrete locations provided by the computational mesh or sampled from it, and with direction and length corresponding to the local characteristics of the vector field. Arrows should be 'anchored' at their centre, so that during frame-buffer animation sequences of consequent arrow plots, they do not provide the visual effect of propagation or retreat. This undesirable effect is studied more in the test cases chapter of the present work.

Arrow plots provide an established visualisation technique for vectors acquiring rapid interpretation, because the perceptual system of engineers has been adapted to the task of interpreting such forms of visualisation. However, they are not a true continuum interpretation of the vector field. Each arrow takes up a finite amount of space in the display, so there are limits to how fine a grid of sampling locations one can employ. It is up to the viewer to mentally interpolate the vector field between the sample locations, and engineers are quite used to it. Alternative visualisation idioms for a vector have been suggested. A few popular ones are summarised in figure 4.16. In the case of gas flow visualisations, the line representation shown in figure 4.16.a does not display the direction of the flow, but results to clear images of flow patterns, analogous to those obtained by experimental techniques (see also figure 3.6). The representation of fig. 4.16.e does not indicate clearly the direction and magnitude of a vector, and hence it should be avoided. The vector representations shown in figures 4.16.b, c, and d are the preferred options, because they represent both the magnitude and the direction of the flow.

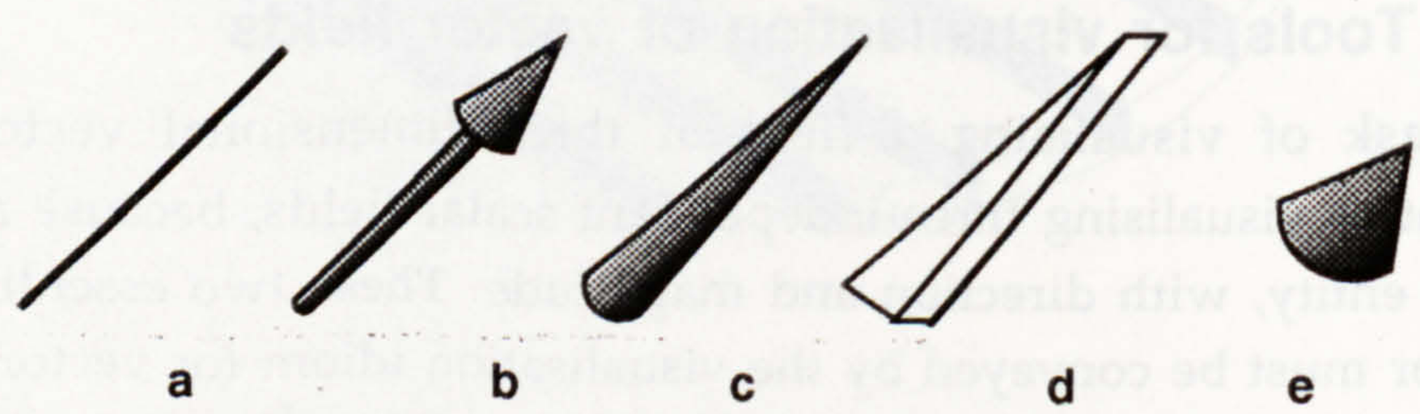
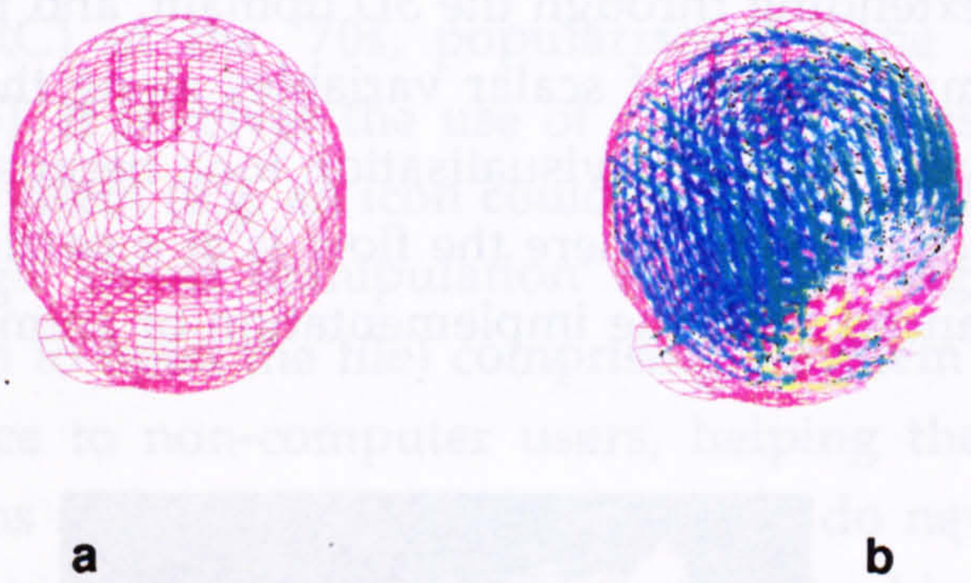


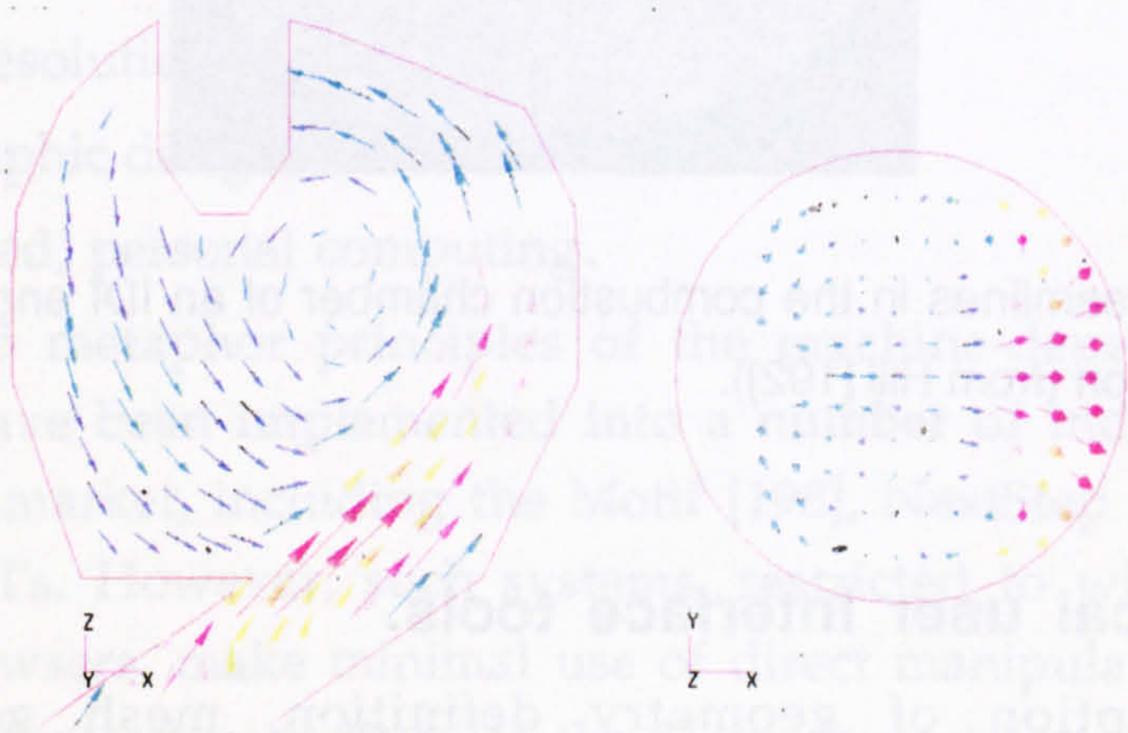
Figure 4.16 Alternative representations of an arrow, used for the visualisations of vector fields.

Arrow plots for three-dimensional domains can be difficult to interpret. As it can be seen in figure 4.17.b, it is difficult to determine the precise location of an arrow from a cluttered single image, since arrows in the foreground might occlude other arrows. Although interactive rotation of the arrow plot and depth-cueing can partly overcome the problem of occlusion, the best way to ameliorate this problem is by definition of slicing planes, in the fashion shown in figure 4.18.



a b

Figure 4.17 Vectors display in the prechamber of an indirect injection Diesel engine. a: surface mesh of the prechamber region, b: a cluttered set of arrows representing gas flow in it.



a b

Figure 4.18 Vectors display in the prechamber of the above indirect injection Diesel engine. a: cutting plane through the symmetry plane of the prechamber, b: cutting plane through the prechamber at the location indicated in figure 4.15.

4.3.4.2 Indirect vector field visualisation tools

Indirect vector field visualisation tools depict the physical effect of a vector field, allowing us to infer its local characteristics. The most common indirect visualisation tool in CFD is 'particle advection'. It involves the integration of the velocity field to obtain a set of *'stream lines'*, from the instantaneous flow field (see figure 4.19), or a set of *'particle traces'* in the case of time-dependent flow. The trajectories can be displayed as curves, ribbons or tubes extending through the 3D domain, and they can be coloured to represent the magnitudes of scalar variables along the trajectory. Particle traces provide a very powerful visualisation tool, because they depict where the flow goes, as opposed to where the flow is at a particular time step. This effect can be enhanced with the implementation of animation techniques.

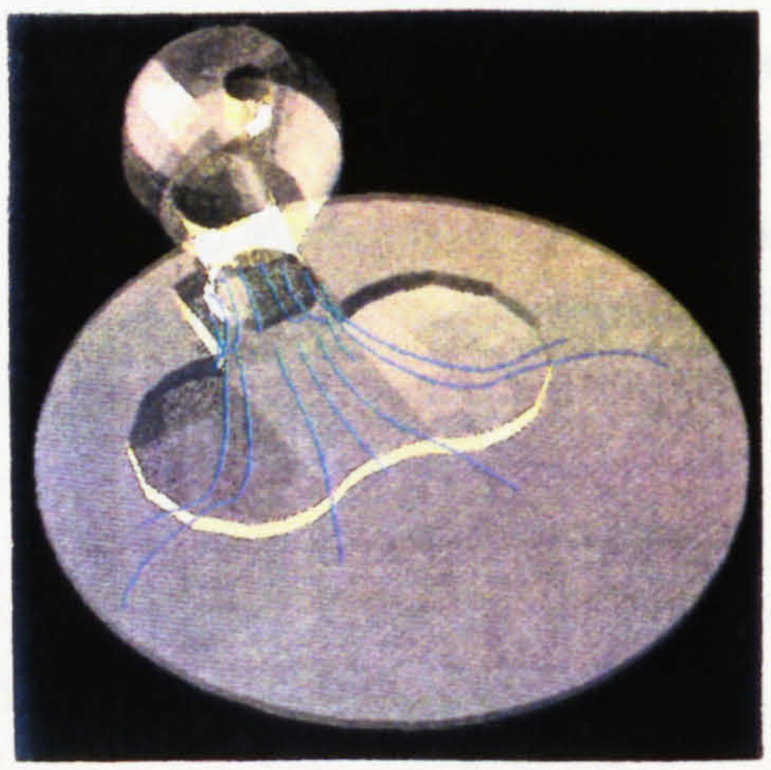


Figure 4.19 Streamlines in the combustion chamber of an IDI engine, 10° CA after TDC of compression (from Hill [192]).

4.4 Graphical user interface tools.

The description of geometry definition, mesh generation and visualisation tools has so far been focused primarily on technological issues and little reference has been made to the engine designers who will utilise them. The objective in this section is to investigate approaches to the design of user interfaces and to assess them from the user's point of view.

Human communication with computers currently relies less on alphanumerical input through the keyboard, in so-called character or *command user interfaces*, and more on graphical interface through input

devices such as the mouse, in so-called *graphical user interfaces (GUI)* accompanied by appropriate user-interface hardware. Despite the widespread use of command user interfaces, mainly with the Unix operating system, and the merits this possesses as far as complexity, completeness and machine independence is concerned, we will focus our attention to GUIs, because they provide a graphical approach to an essentially graphical problem, that of geometry and mesh definition and data visualisation. The present convention for graphical user interaction with software is the so-called *desktop metaphor*, developed initially at the Xerox Palo Alto Research Center (Xerox PARC) in the '70s, popularised by the Apple Macintosh microcomputer [195]. It involves the use of pictorial 'icons' to represent the components of the system (e.g. an icon could represent a file); these icons are manipulated through 'direct manipulation' interfaces (e.g. click the mouse button on a file-icon to open the file) comprising a system that offers a high degree of acceptance to non-computer users, helping them to predict the result of their actions and 'learn by playing' how to do new tasks [196]. The main elements of the desktop metaphor are summarised by its developers to be [197]:

1. An iconic, direct manipulation, object-oriented user interface;
2. Generic commands (e.g. 'CUT' and 'PASTE') and a high degree of consistency;
3. Interaction using a pointing device (e.g. mouse, stylus etc.);
4. A high-resolution display;
5. Good graphic design;
6. Distributed, personal computing.

The desktop metaphor principles of the machine-dependent Star and Apple GUI's have been implemented into a number of industry standards for the Unix™ market, including the Motif [198], NextStep [199] and Open Look [200] GUI's. However, such systems, restricted to windows, pop-up menus and browsers, make minimal use of direct manipulation techniques, and have transformed very little of the Unix character user interface into visual concepts. Recently Explorer offers a refined version of the Motif system which achieves more efficient use of screen space.

User interface management systems (UIMS) are available to provide tools for the development of graphical user interfaces tailored to particular applications that operate within the above industry standards. They provide the programmer with a set of ready-made interactive tools, called 'widgets', such as windows, menus, buttons, browsers etc. which can be invoked by

programming. Such systems are Interviews [201], Teleuse [202], NextStep [199], and Open Dialog [203]. A limitation of the above is that they provide a limited library of widgets, covering the most commonplace types of interaction. Adding new interaction techniques involves low-level graphics and window management programming.

GUIs have in the past decade been complemented by a number of *visual programming techniques* involving networks of icons, representing program components, organized in so-called data-flow diagrams [204], [205]. Such techniques have been applied to computer graphics programming tools, such as ConMan [206], visualisation tools such as AVS, apE, and Explorer, image processing tools such as IDF [207] and Iconicon [208], user interface construction tools [209], and object oriented programming tools [210]. Despite the wide range of applications of visual programming languages, they are not expected to replace traditional symbolic programming languages in the foreseeable future. An intermediate approach is 'program visualisation', in which a symbolic program is analysed and visuálised using visual programming tools [211].

Significant for users of visualisation systems is the development of digital video systems which are a key component for the integration of moving and still images, sound, text, and programs into 'multimedia' documents. Apart from video devices providing recording and play-back, standards are being rapidly adopted for image compression techniques for which up to 20:1 reduction in the size of still images and up to 200:1 reduction of digital movies are claimed, depending on the type and quality of the images [212]. Cheap and general devices for multimedia storage and play-back have been developed by Phillips (Compact Disk Interactive) [213], Commodore [214], and Sony [215], and new tools have been devised to aid the composition of multimedia 'documents' incorporating sound and video such as MediaView [216]. Given that engineers must process very large quantities of moving images as their use of time-varying simulations increases, the ability to generate, archive, retrieve and play back digital documentation of simulations may prove an essential element of the simulation activity.

Alternative approaches to the desktop metaphor are being studied at the 'media laboratory' of MIT and involve the integration of natural language processing and voice-control in user-interface systems [217]. Elsewhere systems that can overlay 'voice notes' onto visualisations are being developed [218] (see also the general review of techniques by [219]). Despite the efforts at MIT and elsewhere, it appears that for the foreseeable future one must learn to manipulate a computer system through the use of

whatever interface it makes available and no natural language system will provide an interface comparable in naturalness to that which one experiences when interacting with a person [220].

4.5 Software Engineering Tools

4.5.1 Programming tools

Modularisation and object-orientation are mechanisms of improving the flexibility and comprehensibility of a program while shortening its development time. *Modularity* is defined by Parnas [221] as a well-defined segmentation of a program, in a way that each task forms a separate, distinct program entity. The advantages of modularity are in:

- faster software development because modules can be developed in parallel, with little communication needed between development subgroups;
- software maintenance: system errors and deficiencies can be traced to specific modules, in this way limiting the scope of detailed error searching;
- product flexibility: a single module can be changed at a time in order to accommodate new requirements;
- comprehensibility: the system can be studied one module at a time;

Recently languages like Smalltalk [222] and C++ [223] have established a new approach to high-level programming, called *object orientation*. They enable computing techniques to be expressed as interacting abstractions, called 'objects', and support information hiding (decomposition into modules in a way that representation and processing in a module is hidden from others), inheritance (a programming mechanism which promotes software re-use), and polymorphism (the possibility to perform common operations on many distinct types of information.

4.5.2 Data base management tools

The current generation of super-workstations, super-computers, and distributed processing systems has encouraged the development of three-dimensional CFD simulation techniques and their application on large, irregular computational meshes. While such simulations are becoming increasingly popular, the visualisation of their output brings new demands for storing and handling the data for interpretation in a timely fashion. *Data*

base management systems (DBMS) provide tools for efficient and persistent data storage and retrieval, in combination with user-interface programming. In this way, they can facilitate application of simulation, visualisation, and distributed processing techniques [224]. In most cases they require the storage of the data to be in the form of hierarchical data types, in several cases in object-oriented ones, in which case they are called 'object-oriented databases'.

Chapter 5

A pre- and post-processing system for SPEED

The utilisation of the SPEED code by its various industrial users, required a fully portable pre- and post-processing software environment, providing tools for geometry modelling of real engine configurations, unstructured mesh generation, and results visualisation and interpretation. The design and implementation of such a software system, that makes use of existing CAD software in automotive companies is described in the present chapter. It can also be found in references [48] and [49] that are included at the end of this work. The presentation of the system is organised in the following sections:

- *Design goals*. Definition of the system's objectives and functionality.
- *System structure and user interface*. Description of the software architecture of the system.
- *Geometry modelling*. Description of utilisation of various solid modelling tools for definition of real engine geometries.
- *Mesh generation*. Description of the employed methodology for the generation of structured, block-structured, and unstructured meshes. Mesh design criteria are discussed alongside the provided mesh-quality checking tools.
- *Problem definition*. Description of the user interface for definition of non-mesh-related problem-setup data, such as initial and boundary conditions, calculation control parameters and engine operation parameters.
- *Visualisation*. Description of the facilities provided by the system for visualisation of SPEED results.

The applicability of the proposed system to a variety of engine applications is shown with several example meshes and visualisations at the end of this chapter as well as in chapter 7.

5.1 Design goals

The first step in developing a pre- and post-processing system for the SPEED engine simulation process, was to define its objectives and intended functionality. The design goals of the system were:

- *Interaction.* The aim was for a practical tool for everyday use by engineers, in which they would be able with interactive computer graphics techniques to define the irregular geometries of real engines, generate unstructured meshes fitted into such geometries, specify all problem parameters for the simulation, and investigate the result of the calculation, as thoroughly as possible, with visualisation tools.
- *Engine geometry modelling capability.* Since real engines have highly irregular combustion chambers, (e.g. containing swirl-chambers, offset bowl recesses, moving valves, and crevices) the system should provide facilities for construction of such geometries.
- *Suitability for unstructured mesh generation.* An important objective was for the system to be able to generate both structured and unstructured meshes, with hexahedral-shape cells, that satisfy the mesh quality criteria of SPEED as they have been specified in section 3.2.3.2.
- *Suitability for unstructured mesh visualisation.* The system should provide for visualisation of vector, scalar, and other fields defined over unstructured meshes. This should be done by implementation of the tools that were described in section 4.3.
- *Flexibility and generality.* The intent was that sophisticated users should be able to modify the package to some extend, for example to add checking features, new boundary types, and new cell types. Also the modules of the system should be suitable for use with a variety of graphics systems, such as CAD systems, PLOT3D, PROSTAR, and toolkit environments.
- The system should be *easy to learn*, and provide a user interface that although adds functionality to the system, it is perceived by the user as an extension of the existing SPEED interface.
- *Portable graphics interface.* Interactive graphics tasks should be realised without sacrifice to portability and performance.
- The system should as much as possible make use of graphics software, that already exists at the sites of the industrial users.
- It would be an advantage if the system could be integrated in the computer aided engineering (CAE) engine design process..

5.2 System structure and user interface

The next step was to decide upon the system's architecture. Critical decisions had to be made at this stage placing tight constraints on the functionality, ease of use, and portability of the system. Two distinct approaches, illustrated in figure 5.1, were available to the interface with the engine simulation process with SPEED: The *single user interface and database approach*, and the *multiple user interface and database approach*. A more detailed discourse on the architecture of the system can be found in [49].

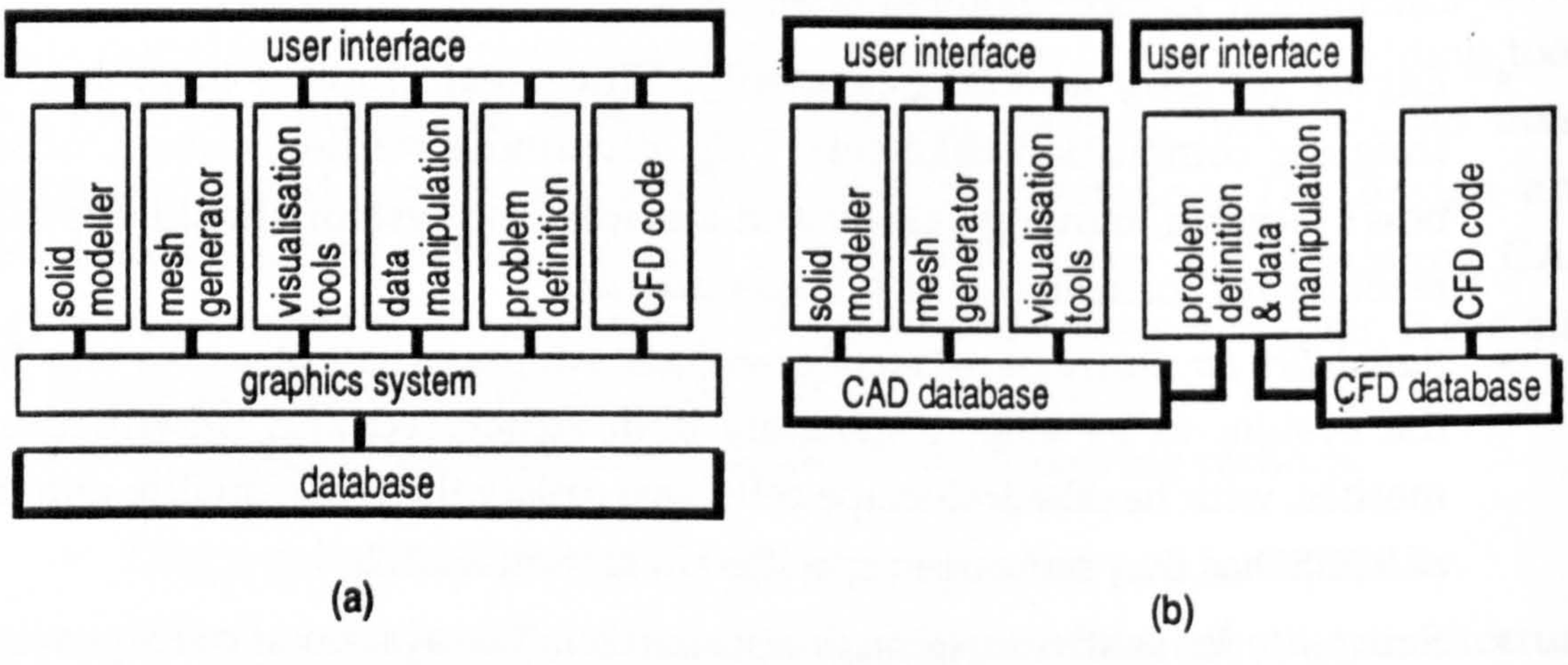


Figure 5.1. The two alternative approaches to the interface of the simulation process. (a) single U.I. and database approach achieved by a single application program based on a standard graphics system, (b) multiple U.I. and database approach, achieved by separate CAD, numerical pre- and post-processing, and CFD programs.

Single user interface and database approach

The single user interface and database approach, illustrated in figure 5.1.a involves the development of a single application program, in the style of PLOT3D [228] or PROSTAR [184], based on one of the available standard graphics systems to achieve portability. The only standard graphics system available at the time the decision had to be made was GKS, since PHIGS, X-Windows, and PEX became available at a later time. The weaknesses of this approach were:

- *Expense of development*. Design and development of such a system, implementing solid modelling, mesh generation, and visualisation tools on three-dimensional unstructured meshes, would be an expensive undertaking.

- *Loss of flexibility and generality*. Such a system would present a preset number of options; if a user wanted to plot some variable other than the ones pre-programmed, then he would have to modify the internal code.
- *Conditional portability*. GKS portability, although promised was not fully achieved, since several GKS 'standards' have emerged. Also performance would have been poor because GKS graphics system libraries were in all implementations built on top of the system graphics libraries, that addressed hardware assisted operations, such as rotation, hidden-line removal, and rendering.

Multiple user interface and database approach.

The alternative approach was to use solid modelling, mesh generation, and visualisation tools provided by CAD systems to cater for the graphical needs of the system, and write a numerical code for implementation of numerical tools, such as data manipulation, problem definition, and checking (see figure 5.1.b). The drawbacks of this approach were:

- Lack of flexibility, since this approach is limited to the graphical tools provided by the CAD systems. Until recently such systems had a monolithic architecture, mainly tailored to finite-element stress-analysis applications, that did not permit incorporation of additional functions.
- No unified user interface, since at least two user interfaces had to be employed, that of the CAD system and that of the numerical system.
- No unified data base, since the results of the CFD simulation needed to be stored in two data bases; that of the SPEED code and that of the CAD system.

The advantages of this approach were:

- Portability. Full portability is achieved throughout all hardware platforms and operating systems.
- Wide availability of CAD systems in design departments of automotive industries. This has resulted to high levels of familiarity with their user interface, within the engine design community.
- Integrated CAE approach. CAD systems are widely used in other phases of the engine design process, such as stress analysis, heat transfer through engine parts, etc. Use of CAD systems in CFD engine flow and combustion analysis, would enhance its integration with the other engine design processes.

- State-of-the-art solid-modelling and mesh-generation tools are provided by several CAD systems (e.g. GEOMOD solid modeller, triquamesh mesh-generation facility of SUPERTAB, PATRAN mesh-generation tools for unstructured meshes)
- Relative ease of application.

5.2.1 System architecture

The considerable advantages of the multiple user interface approach, based on a CAD system, made it the preferred option. The system architecture of the designed system, based on this approach is shown in figure 5.2.

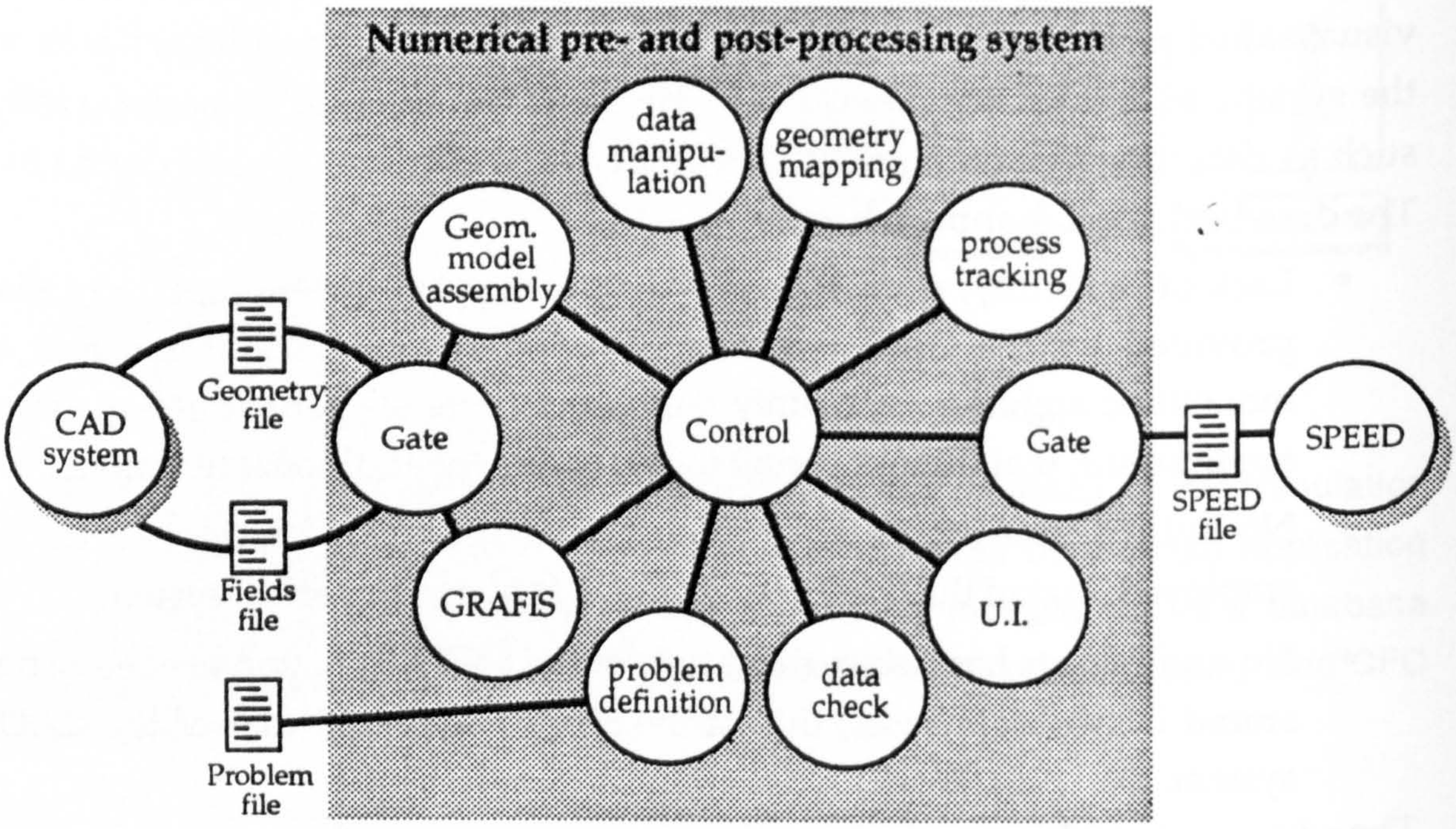


Figure 5.2 Illustration of the system for interaction with the CFD engine simulation process with SPEED.

As can be seen from figure 5.2, the numerical pre- and post-processing system is the central component of the interface with the simulation process. Other components of the system are the CAD system, and the SPEED code. The distribution of tasks between components is:

- Solid modelling, mesh generation, and rendering tasks are realised within the CAD systems environment.
- The numerical solution for gas flow and combustion is realised with the SPEED code, in most cases in batch mode and often as a separate unit of a distributed processing system.

- Numerical tasks related to data checking, definition of initial and boundary values, problem setup, visualisation data manipulation and graphical mapping are realised by the numerical pre- and post-processing system.

The numerical pre- and post-processing system has a modular software structure. At the top level of the systems architecture is the control module, that distributes system control to each of the modules. Data communication between modules is through FORTRAN COMMON blocks. SPEED and the Pre- and post-processing system share the same geometry and field COMMON blocks, allowing the option to use SPEED as a module of the system. Data communication with the SPEED and the CAD system components is through files. Data in files are read and converted to systems structures through simple modules called *gates*. In more detail:

The *geometry model assembly* module reads vertex information and element connectivity lists from system's geometry file, called the *'geometry file'* or *'neutral file'* (because it has the same format as the geometry part of PATRAN's neutral file) and creates SPEED's unstructured indirectly addressed mesh data structures. In a reverse process, during visualisation, it creates vertex and element connectivity information from the SPEED mesh structure and outputs it in the geometry file.

The *problem definition module* reads other problem specification data, such as initial and boundary conditions, calculation control parameters, and engine data, from the system's problem setup file, called *'problem file'*, and creates SPEED's field structures for the particular simulation problem.

The *data manipulation module* provides tools for mapping the raw data defined from the user or produced from the simulation to a more useful - from a storage, simulation, or visualisation point of view - form, and before simulation or visual operations can be applied to it. Such tools operate both on geometry and field data. They include mesh and field contraction, data filtering, definition of cutting volumes, surface sampling, mesh rezoning, boundary and mesh adaption to piston position, thresholding, interpolation of nodal values to vertices, and normalisation.

The *geometry mapping module* provides tools for transformation of raw simulation data or previously numerically manipulated data, into structures that can be rendered graphically by CAD systems in a manner that enhances their interpretation. Such tools are mesh rezoning, mapping field values, defined at mesh points, to a table of colours according to a colour-transfer function, cutting planes and sampling planes.

The *GRAFIS* module converts SPEED scalar, vector, and droplet field data stored in the CFD data structures of SPEED, to generic scalar and vector variables, and stores them in systems *'fields file'*.

The *data check module* checks and visualises mesh quality at all stages of the simulation. Checking options allow fast evaluation of mesh addressing, aspect ratio, skewness, or orthogonality. The allowable limits of these quality measures are set by the user.

The *process tracking module* provides tools for interactive monitoring of the simulation process. Tracking options allow monitoring geometry and field data structures, boundary values, etc.

The *user interface module* provides a command user interface to the numerical pre-and post-processing system and on-line help for all the available options of the user interface. This command U.I. together with the graphical U.I. of the CAD system, comprise the U.I. of the whole simulation process.

Gate modules are used for input and output to files. Such modules are designed so that the system can be interfaced to any CAD system, mesh generator, or visualisation codes. Currently, gates have been created for:

- PATRAN;
- SUPERTAB;
- KIVA;
- AVS;
- All versions of SPEED files;

5.2.2 Data structures

To provide flexibility and generality, the pre-and post-processing system does not deal directly with CFD variables. Instead it only processes generic scalar and vector data. When these data are dumped in the fields file for rendering, they are associated with a character string such as 'pressure', 'velocity' etc. Generic values are associated with CFD fields in the GRAFIS module, from where they communicate with the top-level control module through FORTRAN COMMON blocks.

Mesh data structures are classified in two levels. One *generic* and one *SPEED specific*. The generic level includes arrays for vertex coordinates (array *xcorn*), cell type and list of corner-vertices of a cell (called connectivity list and stored in array *icorn*), and variables for the number of elements (*nc*), and number of vertices (*nv*). They are generated by the CAD system and stored in the geometry file of the system. The SPEED specific level includes additional information on indirect addressing of the hexahedral-shape cells that comprise the irregular

mesh structure of SPEED. Indirect addressing of a SPEED mesh, involves a local orientation system for each cell, with directions -1, 1, -2, 2, -3, 3, as shown in figure 5.3, and data structures such as:

- List of six neighbour cells or boundary nodes, for each cell (array *neighb*);
- Direction of each cell, as seen from its six neighbours (array *ldthrf*);
- Total number of boundary nodes (*nb*);
- Type of boundary for each boundary;
- Number of patches-per-boundary, and nodes-per-patch;

For each boundary node (see also figure 5.3):

- Relating global boundary index (*ib_bound*);
- Adjacent internal cell index (*cbound*);
- Direction of boundary from adjacent internal cell (*dir_bound*);

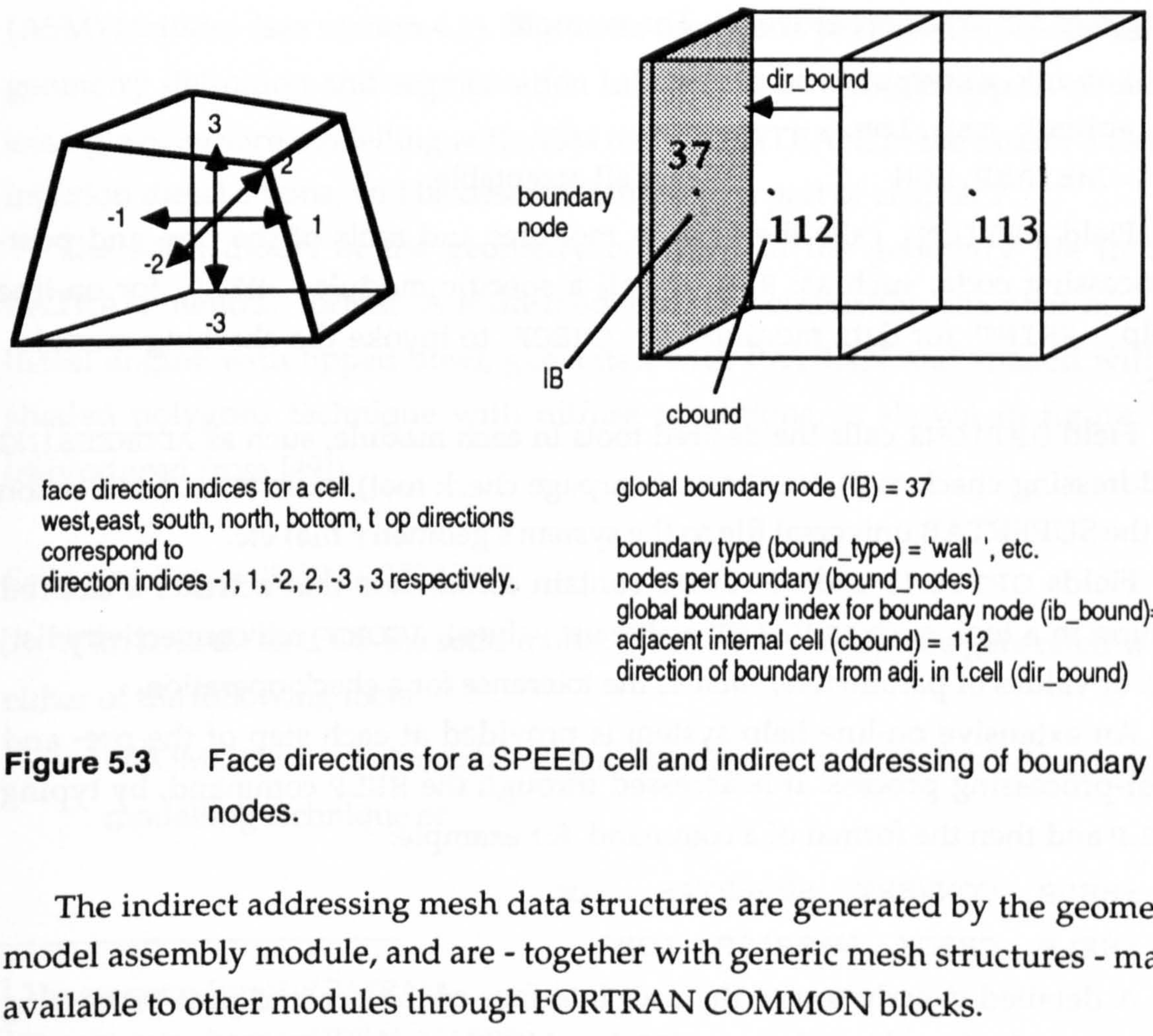


Figure 5.3 Face directions for a SPEED cell and indirect addressing of boundary nodes.

The indirect addressing mesh data structures are generated by the geometry model assembly module, and are - together with generic mesh structures - made available to other modules through FORTRAN COMMON blocks.

5.2.3 User interface

Solid modelling, mesh generation, cutting plane definition, and rendering is done within the graphical user interface of the CAD system. This features a windowing environment and a combined graphical U.I., for selection of items with a pointing device (e.g. mouse), and command U.I., for definition of numerical data (e.g. grid coordinates). Alternative use of pull-down menus is provided but not preferred for this type of application because they often clutter up the screen and are cumbersome to use.

Numerical tasks, such as definition of problem specification data, filtering, and checking, are controlled by a command user interface, within the X-Windows™ and Motif™ windowing environments. User commands have the following format:

```
>OPTION1,OPTION2,OPTION3,OPTION4
```

where fields `OPTION1,...,OPTION4` are user-specified actions. A parsing program, built into the user interface module of the pre- and post-processing system, permits flexible command format, where fields may by separated by spaces or commas, or typed in upper or lower case letters, with a minimum length of 3 characters per word. For example:

```
>check,aspect,long      or
>check,asp,lon          or
>CHE,ASP,LON            are all acceptable.
```

Field `OPTION1` calls the various modules and tools of the pre- and post-processing code, such as: `RUN` to call a specific module, `HELP` for on-line help, `PRINT` for data monitoring, `CHECK` to invoke the checking module, etc.

Field `OPTION2` calls the desired tools in each module, such as `ADDRESSING` (addressing check tool), `WARPAGE` (warpage check tool), `SUPERTAB` (conversion of the SUPERTAB universal file to the system's geometry file) etc.

Fields `OPTION3` and `OPTION4` contain commands that activate a desired option in a tool, such as `VLIFT` (valve lift values), `ICORN` (cell connectivity list) etc. or values of parameters, such as the tolerance for a check operation.

An extensive on-line help system is provided at each step of the pre- and post-processing process. It is accessed through the `HELP` command, by typing `HELP` and then the format of a command, for example:

```
>HELP, CONVERT, SUPERTAB     or
>HELP, CHECK, WARPAGE, LONG
```

A detailed description of the user interface of the pre-and post-processing system can be found in references [226] and [227].

5.3 Geometry modelling

Solid modellers are used to construct and manipulate curves, surfaces, and solids that define the combustion chamber. Surface models of the combustion chamber are either stored in the geometry file in neutral[1], universal[2], or IGES[3] format, or are further manipulated within the CAD system for mesh generation. Two solid modelling systems are supported as part of the pre-processing system: PATRAN and IDEAS/CAEDS. They both provide direct manipulation graphics tools for definition of free-form surfaces in the 3D space. User interaction is accomplished either with a pointing device (mouse or tablet), or with the alphanumeric keyboard.

Solid modelling with PATRAN

With PATRAN the solid model of the geometry can be generated either with constructive solid geometry (CSG) tools, or by means of analytic solid modelling (ASM) facilities (see section 4.1). The latter method is preferred, as it combines geometry definition and segmentation into blocks in a single step. A detailed example of surface modelling with ASM tools in PATRAN, in the case of a direct injection diesel engine, will be described in the first part of chapter 7.

The solid model of the geometry is stored in the geometry file in the PATRAN 'neutral' format. A solid model of the combustion chamber of a DI diesel engine with lipped bowl, generated with PATRAN and shaded with a shaded polygons technique with diffuse reflections, is shown in figure 5.4 (reproduced from [49]).

Solid modelling with IDEAS/CAEDS

With IDEAS/CAEDS the solid model of the geometry can be generated with either of the following tools:

- GEOMOD, a solid modeler that employs a hybrid CSG-B-rep solid modelling technique or

---

[1] The geometry format of PATRAN.

[2] The geometry format of IDEAS/CAEDS.

[3] An application-independent format for storage of CAD geometry data.

- SUPERTAB, a solid modeler that employs basic curve and surface construction tools.

The solid model of the geometry is stored in the IDEAS/CAEDS universal file format. The solid model of the combustion chamber of a 4 valve pent-roofed petrol engine, shown in figure 5.5 was generated with CSG techniques provided by GEOMOD.

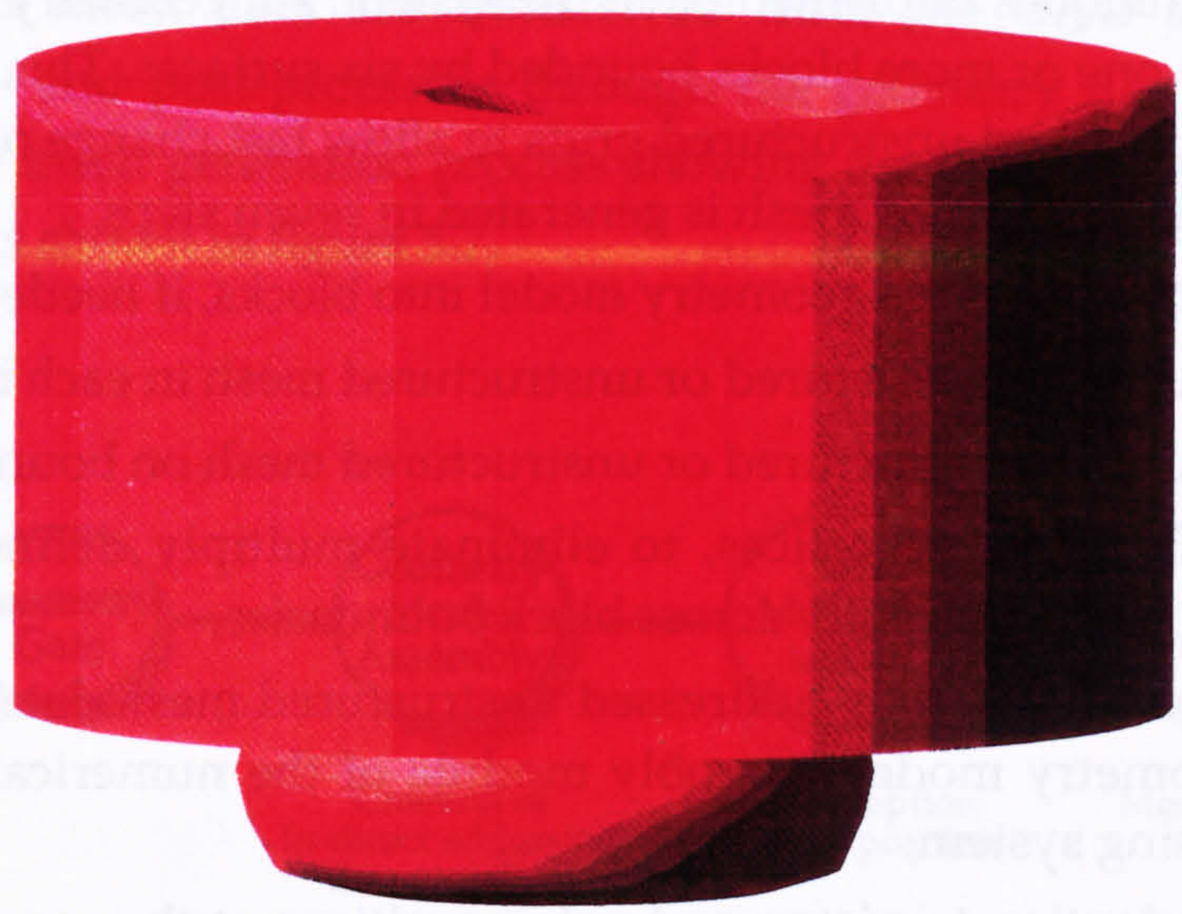


Figure 5.4 Solid model for the combustion chamber of a DI Diesel engine with bowl-in-piston and open intake valve. The model is generated at mid-stroke of induction.

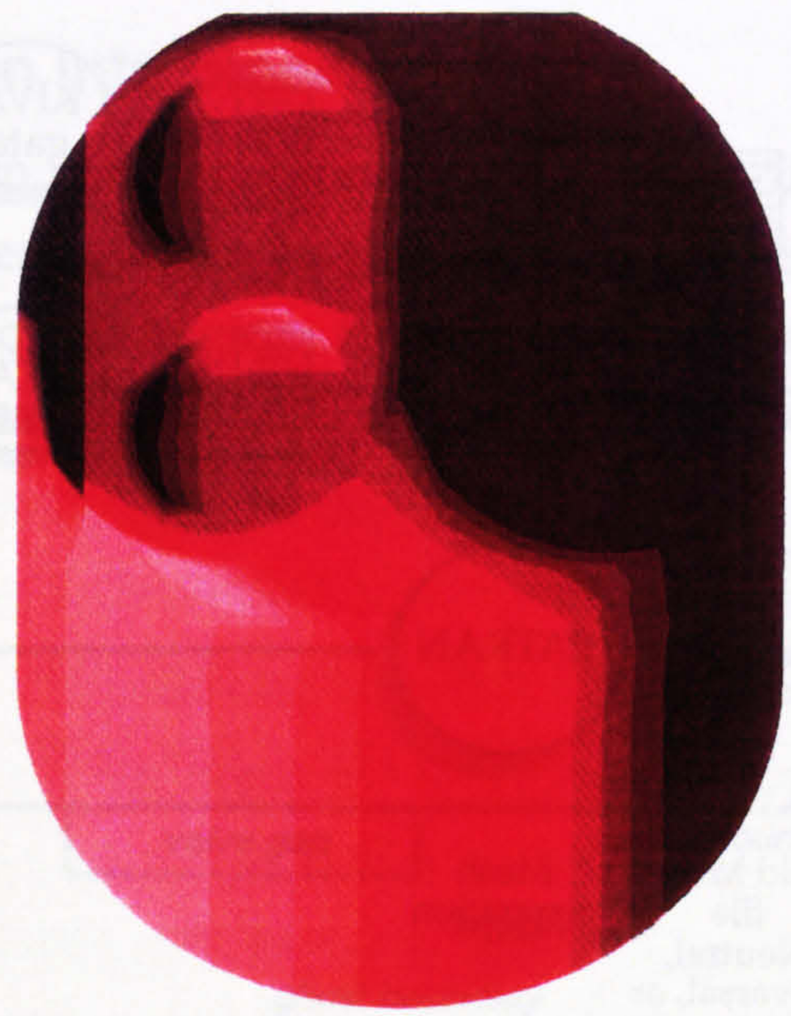


Figure 5.5 Solid model for the combustion chamber of a pent-roofed petrol engine with four valves, of which two are shown open. The model was generated with the CSG facilities of GEOMOD.

5.4 Mesh generation

The CAD surface model of the geometry is taken as the basis for all grid generation techniques implemented in the system. All geometry models must be constructed in one or more blocks bounded by six surfaces. This notation is used both in structured and unstructured grids to allow fast change of mesh topology during mesh generation. A mesh is generated in seven steps:

1. Segmentation of the geometry model into blocks, if needed;
2 Construction of structured or unstructured mesh in each block;
3. Construction of structured or unstructured mesh on boundaries;
4. Equivalencing of vertices, to eliminate multiply defined vertices and ensure mesh continuity across block boundaries;
5. Creation of indirectly-addressed unstructured mesh data structures with the geometry model assembly module of the numerical pre- and post-processing system;
6. Mesh adaption to piston and valve positions at the required time in the piston cycle;
7. Checking of mesh quality.

Steps 1 to 3 are performed with mesh generation tools provided by CAD or other systems such as KIVA or PROSTAR. The data flow diagram of the geometry modelling and mesh generation process is illustrated in figure 5.6.

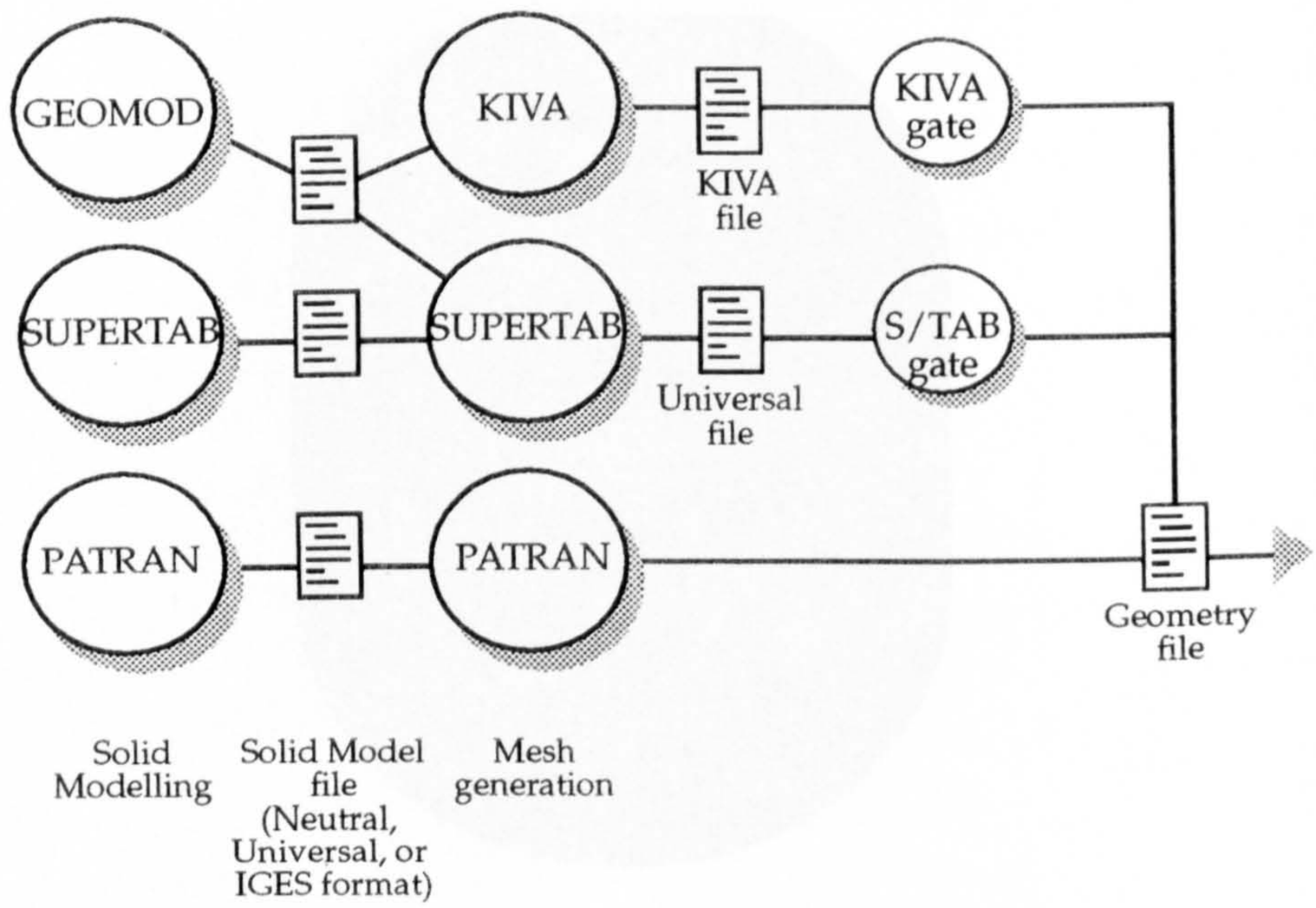


Figure 5.6 Data flow diagram of the solid modelling and mesh generation process, with GEOMOD, PATRAN, or KIVA and the numerical pre- and post-processing system.

Steps 4 to 7 are realised by means of the geometry model assembly, data manipulation and data check modules of the numerical pre-and post-processor. Geometry and mesh data are transferred from the CAD systems to the numerical pre- and post-processor through the geometry file. The data flow diagram of the mesh generation process showing the function of the numerical pre-processor is shown in figure 5.7.

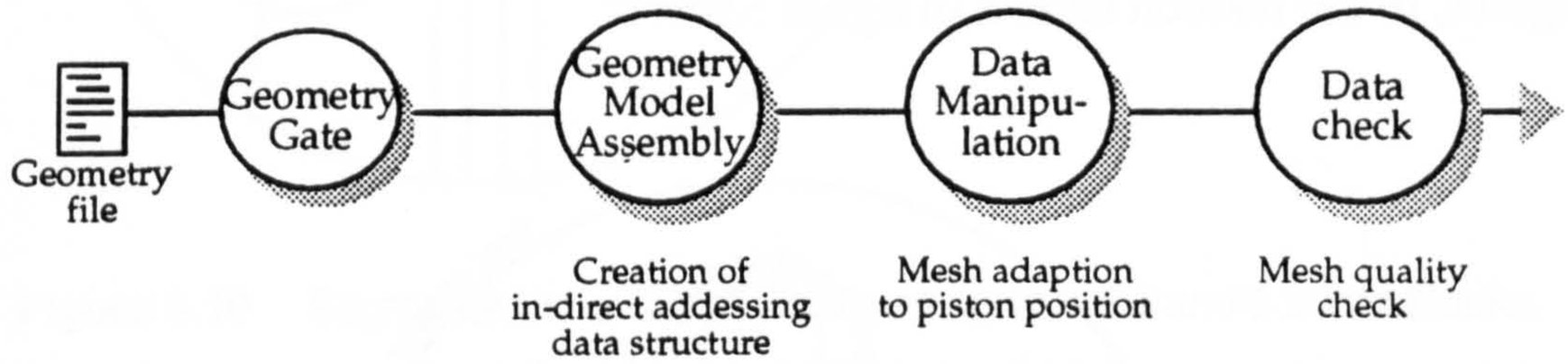


Figure 5.7 Data flow diagram of the tasks realised with the numerical pre-processor. Geometry and mesh data is make available to the pre-processor via the 'geometry file'.

The seven-step mesh-generation process is described in the following paragraphs. A detailed presentation of the mesh generation process with the suggested system can be found in chapter 7 and in references [49] and [226].

5.4.1 Segmentation into blocks

From a mesh design and generation point of view, the combustion chamber of a Diesel or petrol engine can in the general case usually be split in three regions, namely the squish region, the piston-recess region, and – when one exists – the head-recess region, as it is illustrated in figure 5.8.

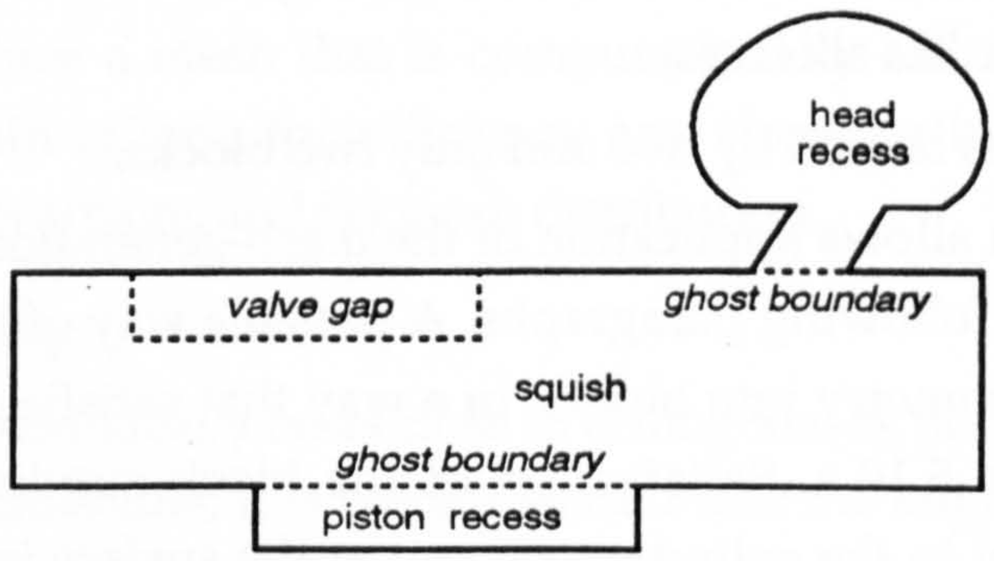


Figure 5.8 Separation of the combustion chamber in squish, piston, and head-recess regions in the case of an IDI Diesel engine.

These regions are topologically separated by 'ghost boundaries', that are used solely to facilitate mesh-manipulation operations such as mesh adaption to piston and valve motion, layer addition and removal, etc. and have no effect on the flow and combustion simulation.

The squish region is defined by the piston crown, the piston-bowl-throat ghost boundary (if any), the cylinder head, the valve heads, and the prechamber-throat ghost boundary (see figure 5.8). Viewed from the top, it can be further divided in sub-regions by the liner contour, the valve contours, the piston bowl contour, if any, and the prechamber throat contour in the case of IDI engines, in the fashion shown in figure 5.9.

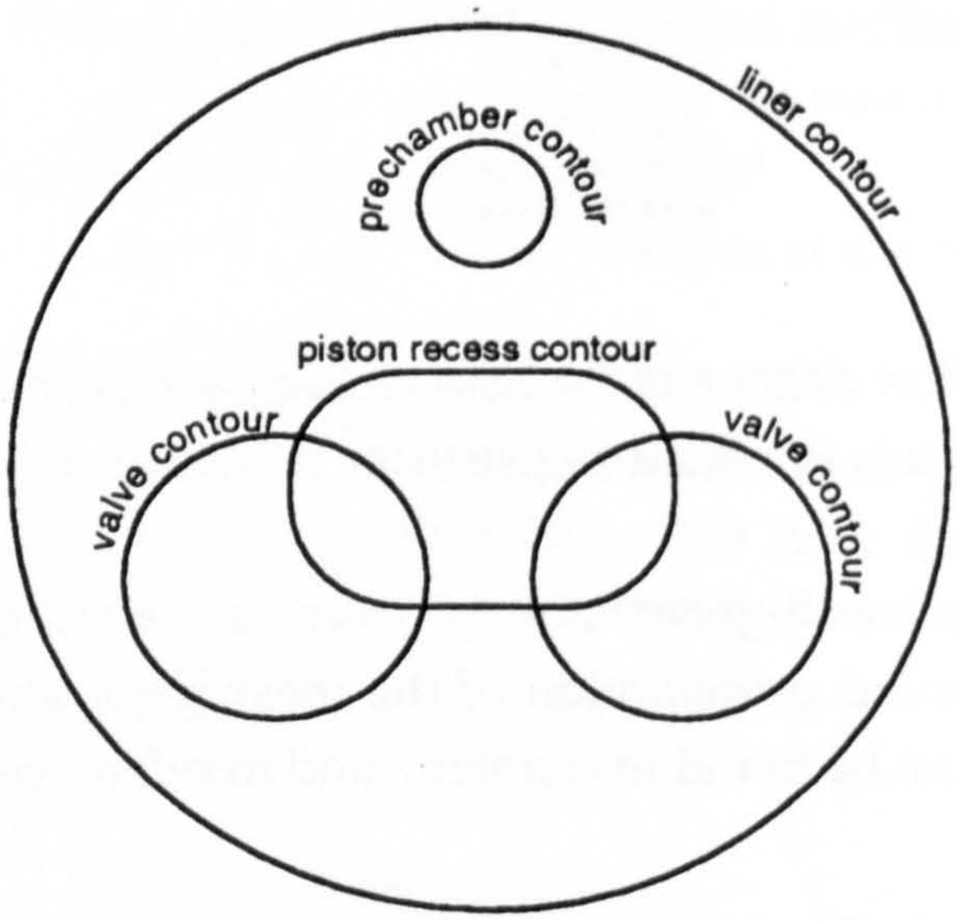


Figure 5.9 Segmentation of the squish region of the combustion chamber of an idealised IDI Diesel engine by the contours of the liner, the valves, the piston recess, and the prechamber throat.

Subregions are further subdivided into blocks in such a way that:

- each block has six faces,
- each face is shared by two and only two blocks;

This subdivision allows application of the mesh-generation utensils that will be described in the following paragraphs. A possible way of segmenting the above squish-region geometry into blocks, in a way that satisfies the above criteria, is shown in figure 5.10.a. Satisfaction of the block subdivision criteria at the direction parallel to the cylinder axis and at the surface joining the squish and bowl regions, requires that the latter is segmented in a similar fashion, as it is shown in figure 5.10.b.

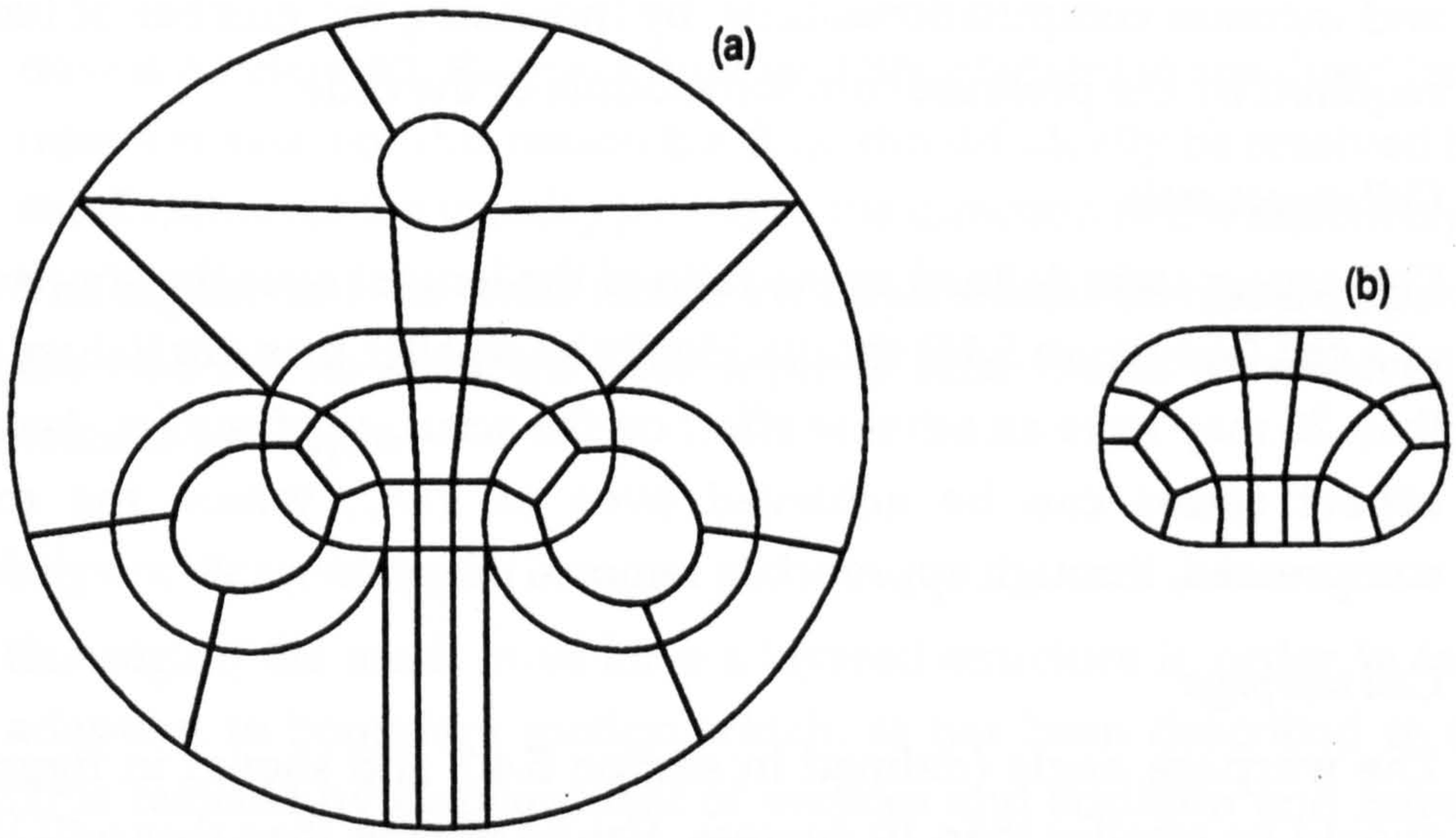


Figure 5.10 Segmentation of the geometry of figures 5.8 and 5.9 into blocks. a) Squish region, b) bowl region.

5.4.2 Mesh design and construction in each block

After the construction of a surface model of the calculation domain and its segmentation into blocks, a mesh of hexahedral-shape cells is fitted in each block. Mesh generation is based on user-specified vertex distributions on a block's boundary faces. Cells and vertices are subsequently generated in each block by means of isoparametric mapping techniques provided by the CAD system, that have been described in chapter 4 and are documented in [64] and [6]. This technique is suitable for generation of both structured and unstructured meshes and their application in individual blocks, within a multiblock structure, releases a great number of meshing possibilities and allows application of fitting strategies that result in uniform meshes, like the one shown in figure 5.12.

Prior to specification of the vertex distributions on the face of a block, it is recommended that the user designs the mesh, that is decides which mesh-fitting strategies will produce a mesh that is computationally efficient for a SPEED simulation. The main criteria for efficiency are: i) cell orthogonality, ii) cell aspect ratio, iii) cell warpage, and iv) mesh distribution.

i. *Cell orthogonality*

The mesh should ideally be as close to orthogonal as possible. Since this is not always achievable, it is recommended that the angles formed by the intersecting edges of a cell are in the range from 30 to 150 degrees (see figure 5.17) The presence of angles smaller than 30 or larger than 150 degrees is not recommended, since this will reduce the quality of results

and increase computational time, by increasing the number of iterations required by the pressure solution module of the code.

ii. *Cell aspect ratio*

Cell aspect ratio defined as the ratio of the longest over the shortest edge of a cell (see figure 5.18) should ideally be smaller than 20. Values greater than 20 may have an adverse effect on the accuracy of results. Acceptable aspect ratios can be achieved even at TDC, where the mesh is compressed, through appropriate removal of layers.

iii. *Cell warpage*

The warpage angle (defined in section 5.4.7 and shown in figure 5.19) should be smaller than 10 degrees. Values greater than that will have an adverse effect on the accuracy of results.

iv. *Mesh distribution for optimum resolution*

The mesh should adequately resolve the gradients in all dependent variables. This means that the mesh should be fine where the gradients are expected to be high. Such regions in the flow are:

- Inflow jets, where ideally the mesh should be resolved to the jet direction (see figure 5.11).

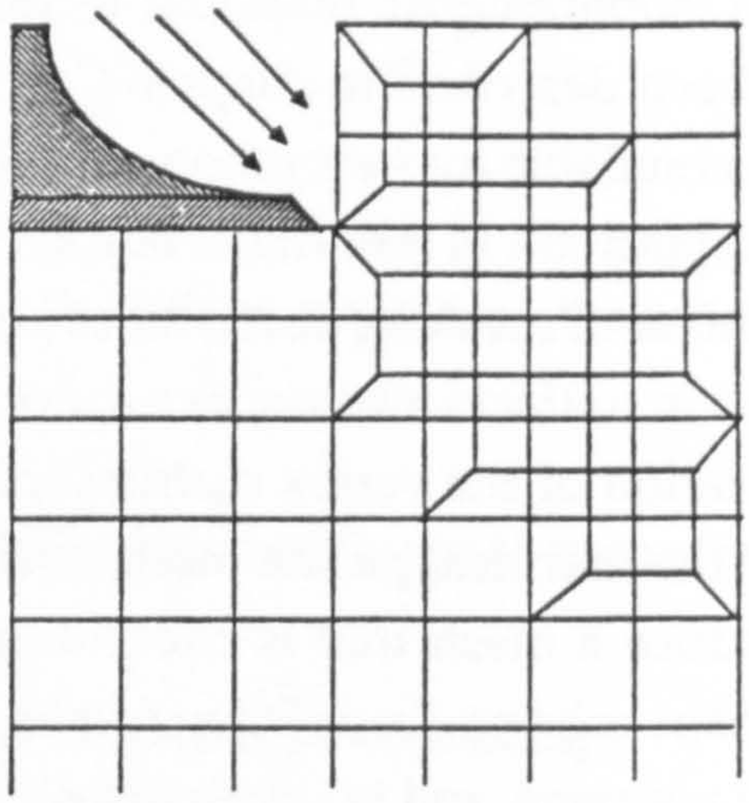


Figure 5.11 Local resolution of the intake jet, by means of mesh refinement.

- The region near the bowl lip when squish takes place, where again the mesh should be resolved in the direction of the jet.
- The region where fuel is injected, where, because the droplets input momentum into the flow, a velocity jet is created, and also, because the

flow is accelerated, there is a high velocity gradient in the direction of the injection axis. For this reason the flow should ideally be resolved both in the direction of the velocity jet and in the direction of the injection axis.

- The region near the walls where the mesh should be resolved parallel to the wall but no more than one layer of cells should be resolved in the viscous sub-layer.

Mesh design in the squish region

In this region the mesh must have a layered structure in order to facilitate mesh adaption to boundary motion which, as has been described in section 3.2.3.2.2, is realised by displacement of vertices and addition and removal of mesh layers in the squish region. Because of its layered structure it can be said that the mesh in this region is quasi-three-dimensional. Hence its design can be done in the two dimensions, say at a top view of the mesh. The mesh should be generated in each block of this region in such a way that:

- Where blocks join, the mesh cells must share common faces, so that mesh continuity is ensured;
- When the valves open or close, by means of removal or addition of layers below the valve base, the mesh in the other regions is unaffected.

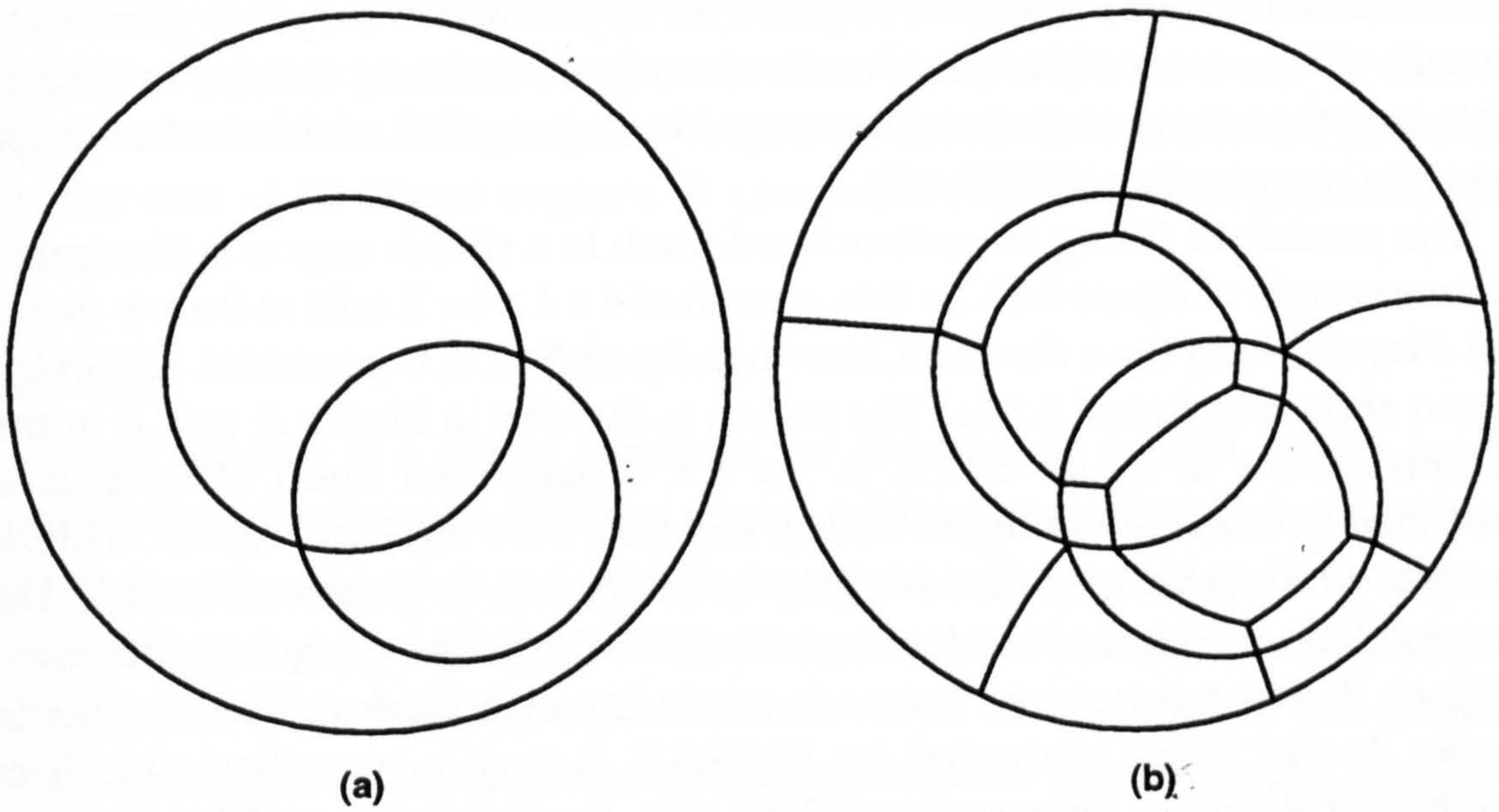
Mesh design in two-dimensions is illustrated in the example shown in figure 5.12. In this example, the squish region of the combustion chamber of the DI Diesel engine shown in figure 5.4 has been segmented in sub-regions by the liner, valve, and piston-bowl-throat contours (see figure 5.12.a). Subsequently it has been further subdivided into blocks (see figure 5.12.b), to achieve the mesh shown in figure 5.12.c, that has been generated for optimum resulution. As can be seen in figure 5.12.c, the radial mesh of size n=11 in zone (A), is reduced to the size n=2 radial mesh of zone (B) by application of mesh refinement patterns in zones (C). A polar mesh could have been used in the region of the axes of the valve and the bowl, but this would have introduced cells of very small angles and volumes, which are unnecessary and undesirable. This has been avoided by use of a topologically rectangular mesh, that merges with the polar one by use of transition points, shown in zone (D) of the mesh. The mesh in the region defined by the valve contour has been designed in such a way that when the valve opens, by removal of layers below the valve base, the remaining mesh is unaffected.

Mesh design in the head or piston recess regions

In these regions the mesh need not have a layered structure, since no layer addition or removal takes place. Depending on the recess configuration, a mesh should be fitted that best satisfies the mesh distribution for optimum resolution criteria mentioned earlier.

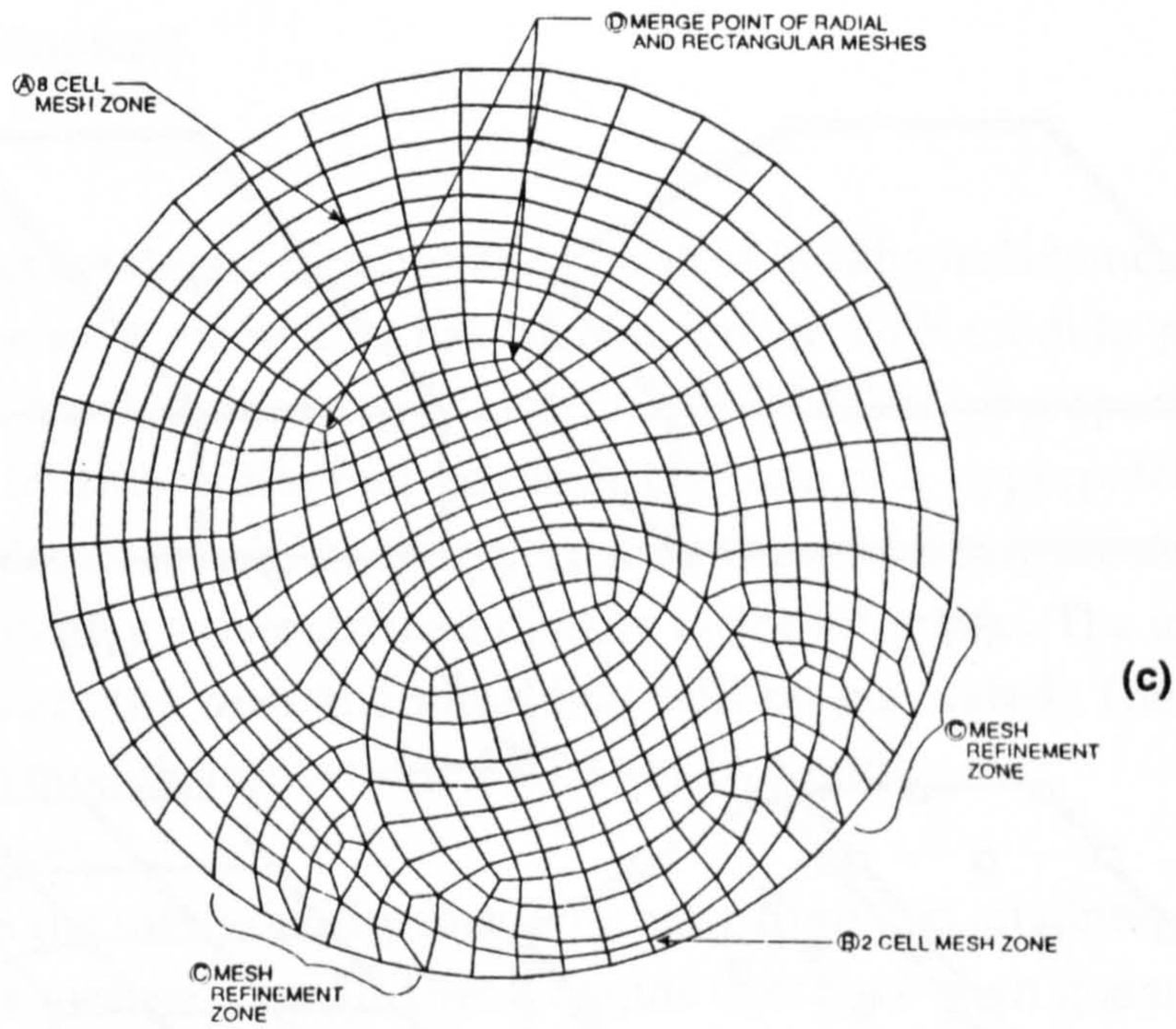
Examples

Several examples of meshes that have been generated with the above system are given in section 5.7 and in chapter 7.



(a)

(b)



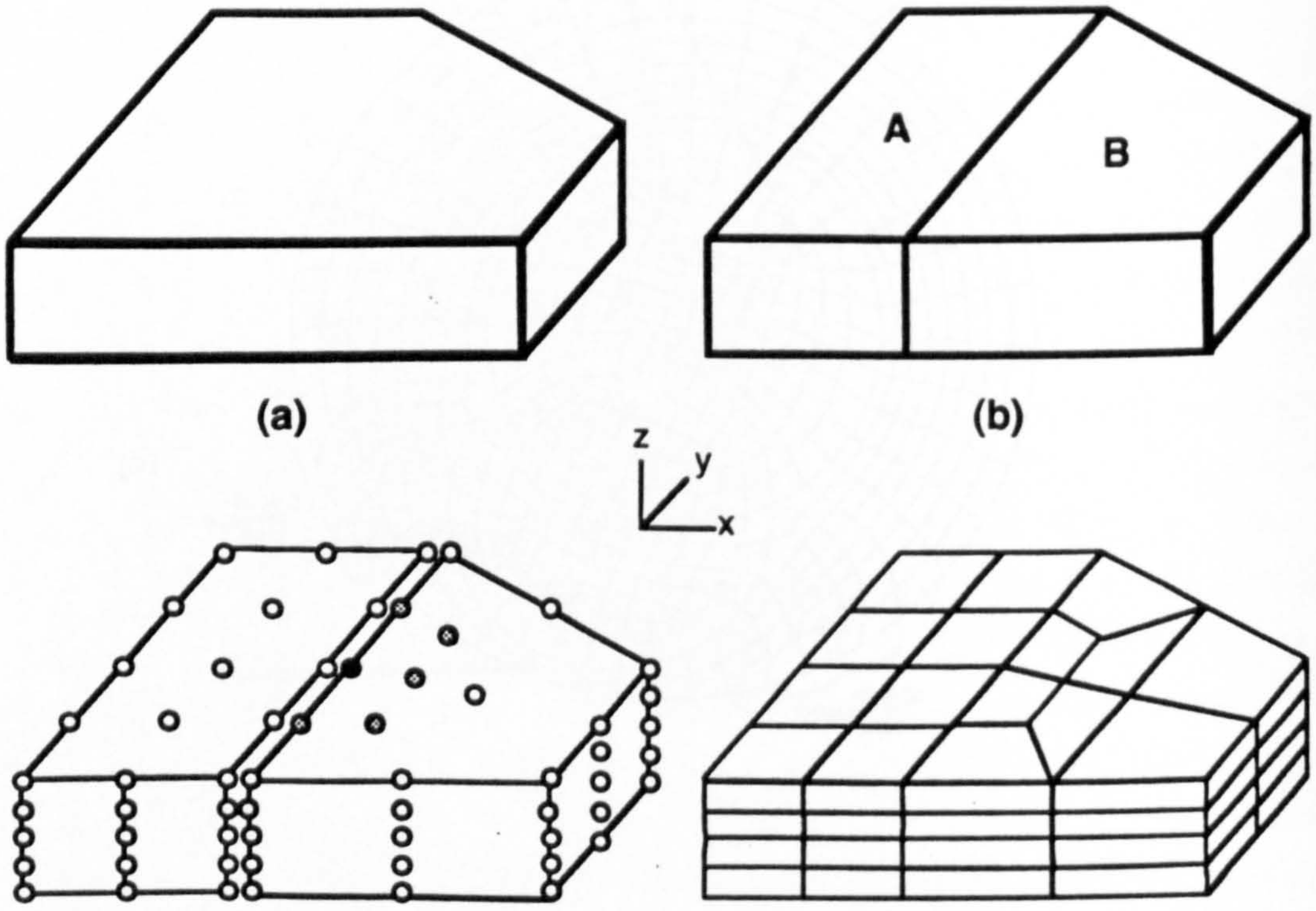
(c)

Figure 5.12 Mesh generation in the squish region of a DI Diesel engine. a) top view of contours of cylinder liner, valve and bowl lip, (b) segmentation into blocks (c) generation of a uniform unstructured mesh.

5.4.2.1 Mesh generation with PATRAN

When the geometry is described by means of PATRAN's analytic solid modelling tools, then it is generated as a set of 3D entities called 'hyperpatches'. When this partition follows the block-subdivision rules given earlier, these can be considered the blocks that are required for application of the mesh-generation utensils. Given a set of blocks, a mesh of vertices is initially created in each of them, and then these vertices are connected to make a mesh of hexahedral-shape elements representing SPEED cells.

The process of fitting an unstructured mesh in a simple region is illustrated in the example of figure 5.13. In this, a mesh of 4 x 4 x 4 x 2 cells at the x-y plane, and 4 layers along the z direction, shown in figure 5.13.d, is generated within the region shown in figure 5.13.a. The region is divided in blocks A and B in the fashion shown in figure 5.13.b. A 3 x 5 x 5 structured mesh of vertices is generated in block A (see figure 5.13.c), and a 3 x 3 x 5 mesh of vertices in block B. A set of 5 vertices, filled with gray in the figure, is generated around the vertex filled with black, in order to generate the 5 vertices along the y direction of block B, that are required for mesh continuity with block A. The vertices in blocks A and B are connected by PATRAN is such a way that a mesh of hexahedral elements, representing SPEED cells, is generated in each block.



(a) (b) (c) (d)

Figure 5.13 Generation with PATRAN of the mesh shown in (d) in the region shown in (a). (b) subdivision into blocks, (c) definition of vertices in each block.

5.4.2.1.1 A semi-automatic mesh-generation program

PATRAN provides a programming language, called PATRAN command language or PCL for short, that can be used for utilisation of CAD tools from within a programming environment. The pre-processing system is complemented by a PCL program for geometry definition and mesh generation in the case of DI Diesel engines. It comprises two modules: 'geometry' and 'mesh'.

Geometry

This module is a set of functions for creation of a solid model for the combustion chamber of a DI Diesel engine, with two valves and a bowl recess. It caters for valve chamfers and valve indentations in the piston, and a bowl recess with a lipped entry (see figure 5.33). The user needs to input the location and dimensions of the inlet valve, the exhaust valve, the piston, and the cylinder. The structure of this module and a detailed list of the required input is given in appendix A.5. Example solid models that have been generated with this facility can be found in chapter 7.

Mesh

This module generates an isoparametric mesh of hexahedral elements in each hyperpatch of the solid model that has been generated with module 'geometry'. To do so it calls mesh-generating commands of PATRAN, by specifying the number of cells in each of the four edges of the base of a hyperpatch and the number of cells along the height of a hyperpatch. These numbers are determined for each hyperpatch by a user-defined density factor `enumber`. The greater the value of `enumber`, the finer the mesh that will be generated. The user can execute the mesh module from within PATRAN, by typing:

```
MESH(enumber)
```

where *enumber* is the mesh-density factor, ranging from 1 to any integer greater than 1. Examples meshes for a DI Diesel engine that have been generated with module 'mesh' can be seen in figure 5.33. They have both been generated from the same solid model. The one on the left has been generated by setting enumber=1, whereas the one on the right by setting enumber = 3. A more detailed description of the process – and a mesh for a variation of the above design – can be found in chapter 7. An outline of the mesh module can be found in appendix A.6.

5.4.2.2 Mesh generation with SUPERTAB

When generating a mesh with SUPERTAB, a GEOMOD-generated surface model of the geometry needs to be partitioned into blocks, called 'volumes'. Meshes of hexahedral 'bricks', representing SPEED cells, are fitted in each block by means of the 'mapped geometry generation' tool, in the case of structured meshes, or by 'free geometry generation' facilities otherwise. In the latter case, a layer of bricks is generated by explicit specification of their corner vertices, and duplicated along user-specified directions to fill a block. This process is illustrated in the example of figure 5.14.

Figure 5.14 Mesh generation with SUPERTAB. (a) Region to be meshed. (b) Subdivision in blocks (c) Generation of a structured mesh in block A and definition of vertices on a face of block B (d) Duplication of vertices in B (e) Creation of bricks for the first layer of B. (f) Generation of the remaining layers.

In this, a mesh of 4 x 4 x 4 x 2 cells at the x-y plane, and 4 layers along the z direction, shown in figure 5.14.f, is generated within the region shown in figure 5.14.a. The region is divided in blocks A and B in the fashion shown in figure 5.14.b. The 2 x 4 x 2 x 4 structured mesh of bricks with four layers in the z direction is generated in block A by means of a mapped mesh generation utensil (see figure 5.14.c). A 5 x 3 x 2 x 3 mesh of vertices is generated on the bottom face of block B (see figure 5.14.c) by determination of the number of vertices along each edge of the face, and repeated four times in the z direction in the fashion shown in figure 5.14.d. A layer of bricks is generated by specification of the vertex-connectivity list for each (see figure 5.14.e). The layer of bricks already defined is duplicated three times in the z direction, to create the remaining layers of the mesh (figure 5.14.f).

5.4.3 Definition of boundaries

A boundary node in SPEED is located at the centre of the face of a cell that is adjacent to the boundary (see figure 5.3). The thickness of the node is zero and it has no metric information associated with it. As noted earlier, three types of boundaries are allowed: wall, valve curtain, and ghost. Typically walls are divided into several wall-regions which are really part of the same wall.

The simulation of SPEED boundary nodes is done with quadrilateral elements, created on the faces of the cells that are adjacent to the boundary. CAD systems provide two means for the creation of such elements:

- individual selection of the boundary face of each cell;
- specification of the face of an already meshed block that is adjacent to a boundary

The type of boundary – wall, valve curtain, or ghost – and the boundary region is indicated by appropriate specification of a flag associating a quadrilateral element with a boundary. The appropriate flag number for each type of boundary is shown in table 5.1. For example a quadrilateral element denoting a boundary node on wall region 1 should have flag 11, on wall region 2 should have flag 12 etc. Figure 5.15 illustrates with different colour various boundaries and regions defined in the case of a combustion chamber model for a four-valve pent-roofed petrol engine. Wall regions corresponding to the cylinder head are coloured light red and blue, to the cylinder liner are coloured red, and to the valve bases light red and gray. The valve-curtain regions are coloured blue. A set of boundary quadrilateral elements in the case of an IDI Diesel engine is shown in figure 5.16. In this, ghost boundary elements separating the squish from the bowl and prechamber regions are filled with gray.

SPEED boundary type	Configuration number
Wall patch	11 to 19
Valve curtain	21 to 29
Ghost boundary	61 to 69

Table 5.1 Flag numbers for elements, associating them with SPEED boundaries.

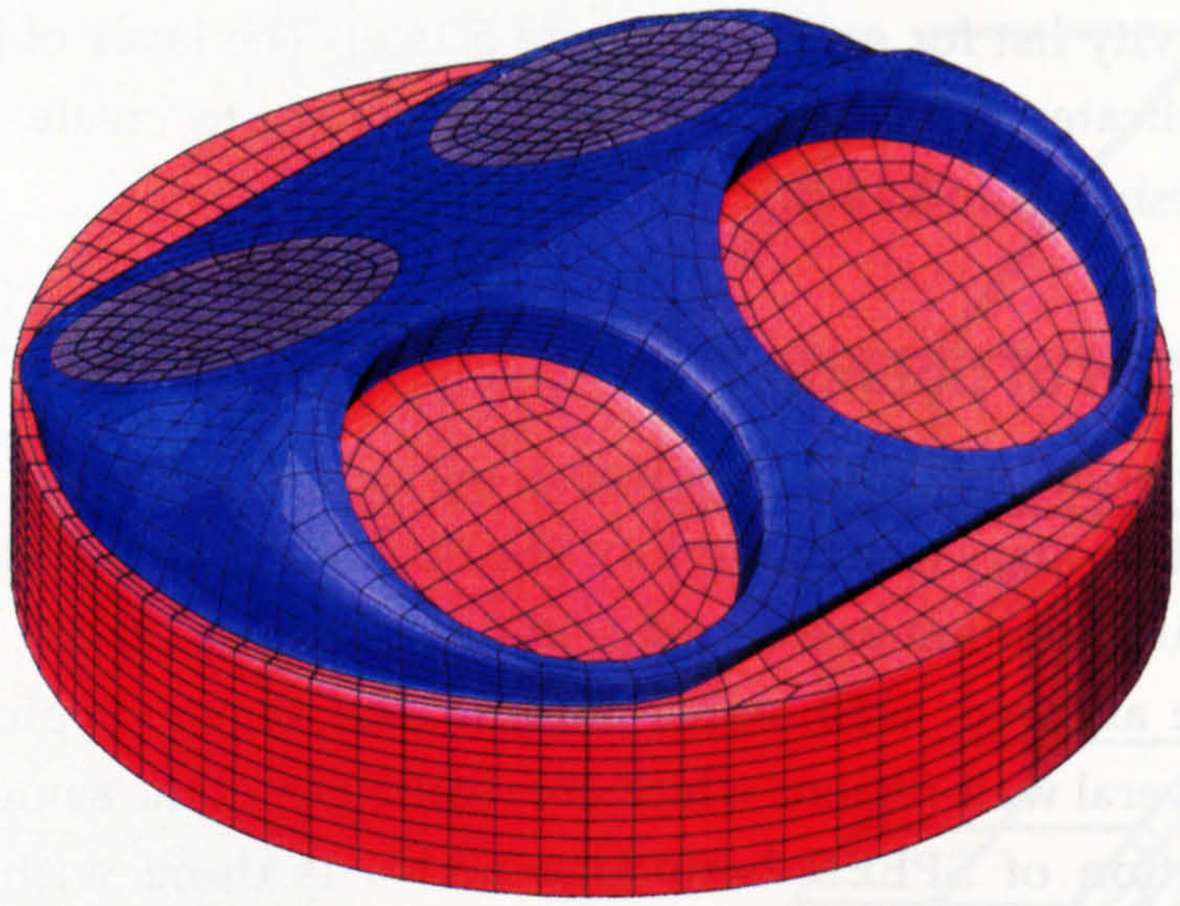


Figure 5.15 Boundary regions in a mesh for a pent roofed petrol engine chamber.

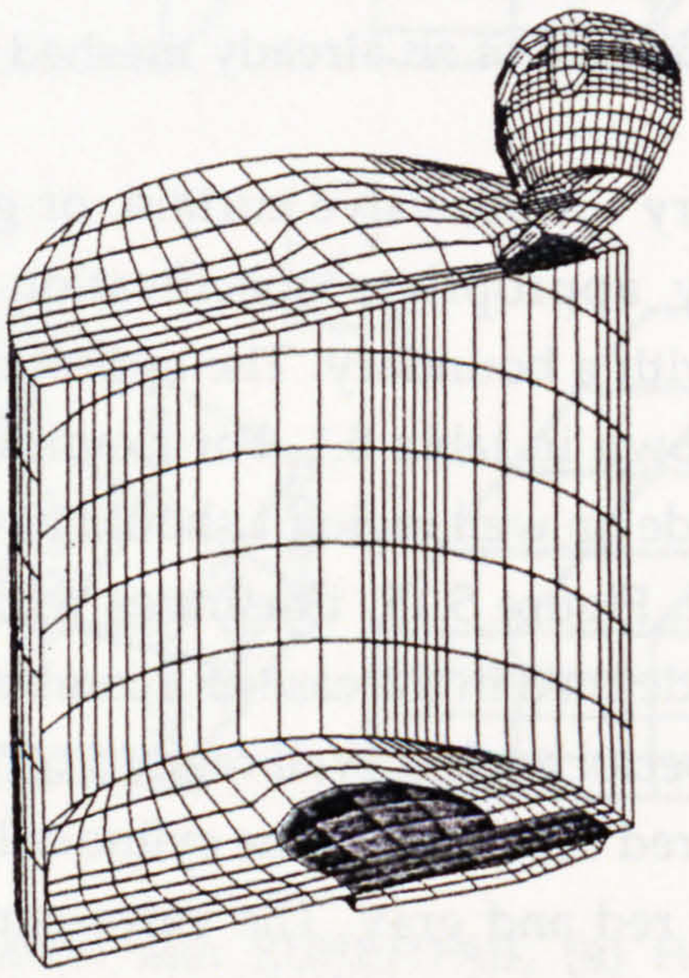


Figure 5.16 Boundary quadrilateral elements in the case of an IDI Diesel engine, with vertical glow plug. Ghost boundaries are filled gray . Only half the geometry is shown.

5.4.4 Mesh continuity across block boundaries

Due to independent mesh generation in each block all vertices on the faces of blocks have been defined twice, once for each block. In this step, double-defined vertices are replaced by a single vertex in this way ensuring continuity across block faces. This process is followed by compaction of the vertex numbering, to ensure that there are no gaps in the numbering of vertices in the geometry file, and update of the cell and boundary-quad connectivity lists (list of corner vertices for each cell or boundary quad) so that they correspond to the new set of vertices. The coordinates of each vertex in the set, the cell-connectivity lists, the boundary flags, and the boundary connectivity lists comprise the mesh data that is stored in the geometry file of the system.

5.4.5 Geometry model assembly

The geometry model assembly module (see figure 5.2) reads mesh data from the geometry file, and creates SPEED's unstructured indirectly-addressed mesh data structures for internal cells and boundaries that have been described in section 5.2.2. This is done in two steps. The first is to create a vertex-connectivity list for each vertex, storing all the cells that are adjacent to a vertex. The second is to create the indirect-addressing information for each cell, that is find for each cell the neighbouring cell at each face and the direction of the neighbouring cell. An algorithm for this, that is based on SPEED's data structures and is implemented in this module, is given is appendix A.4. It is applicable to any unstructured mesh with cells and faces of any shape, as long as there is only one neighbour to a cell at a face. Its implementation in the geometry model assembly module, so that it is applicable to hexahedral cells with quadrilateral faces, is done with the following settings:

number of vertices at cell = 8
number of faces at cell = 6
number of vertices at face = 4

5.4.6 Data manipulation

Geometry data produced so far are converted to a more useful, from a simulation point of view, form by application of manipulation tools of the pre- and post-processing system. These tools are:

- Translation of the boundary corresponding to the piston surface, to its position at the time in the engine cycle the calculations are to commence;
- Translation of the boundaries corresponding to the valve bases, to their positions at the time in the engine cycle the calculations are to commence;
- Mesh contraction or expansion in the squish region, to adapt the mesh to the positions of the piston, valve, and other boundaries.

5.4.7 Mesh quality checking

Mesh quality checking options are provided by the data check module of the pre- and post-processing system (see figure 5.2) to check the suitability of a mesh for SPEED simulation. Checking options allow fast evaluation of mesh addressing, distortion, skewness, and aspect ratio.

Mesh addressing

The check addressing tool of the data check module is used to check the indirect-addressing data structures related to internal cells, wall patches, and valve boundaries, as well as to check for mesh discontinuities at cell faces, by checking the vertex list for multiply defined vertices. Multiply-defined vertices are detected and the one with the smaller index is kept.

Distortion

Mesh distortion is checked by calculation of the diversion from orthogonality for each cell. It involves calculation of the angle f (see figure 5.17) formed by all joining faces of each cell and comparison with a user-set tolerance.

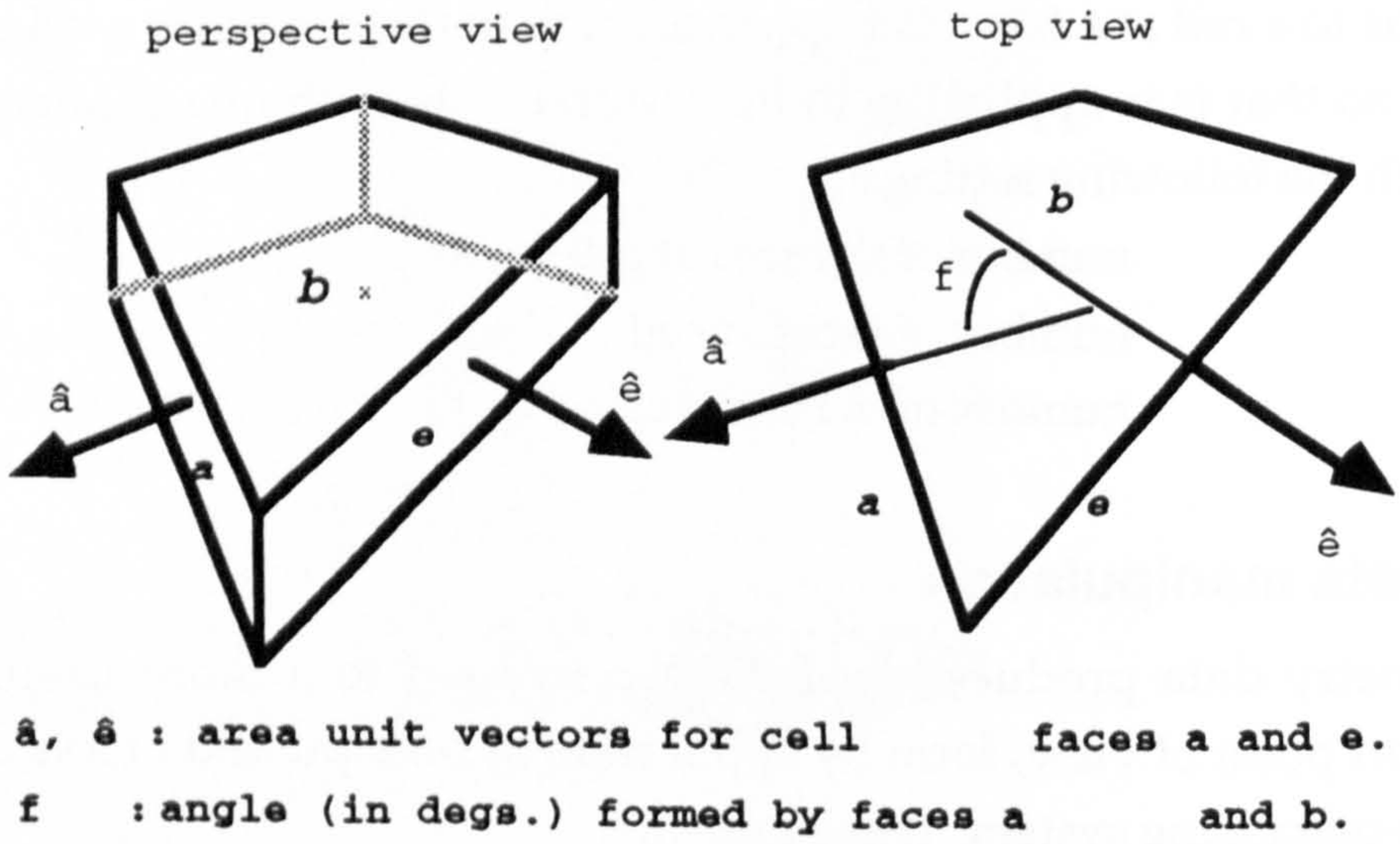


Figure 5.17 Check for deviation from orthogonality. Angle f must be smaller than a user-specified value.

Aspect ratio of cells

Mesh distortion is also checked by calculation of the aspect ratio of each cell in the mesh. The aspect ratio of a cell is taken as the maximum distance between the opposing face centres divided by the minimum distance (see figure 5.18), with the formula:

$$AR = \frac{\max(|EW|, |NS|, |BT|)}{\min(|EW|, |NS|, |BT|)} \tag{5.1}$$

The aspect ratio of each cell is checked against a user-specified tolerance.

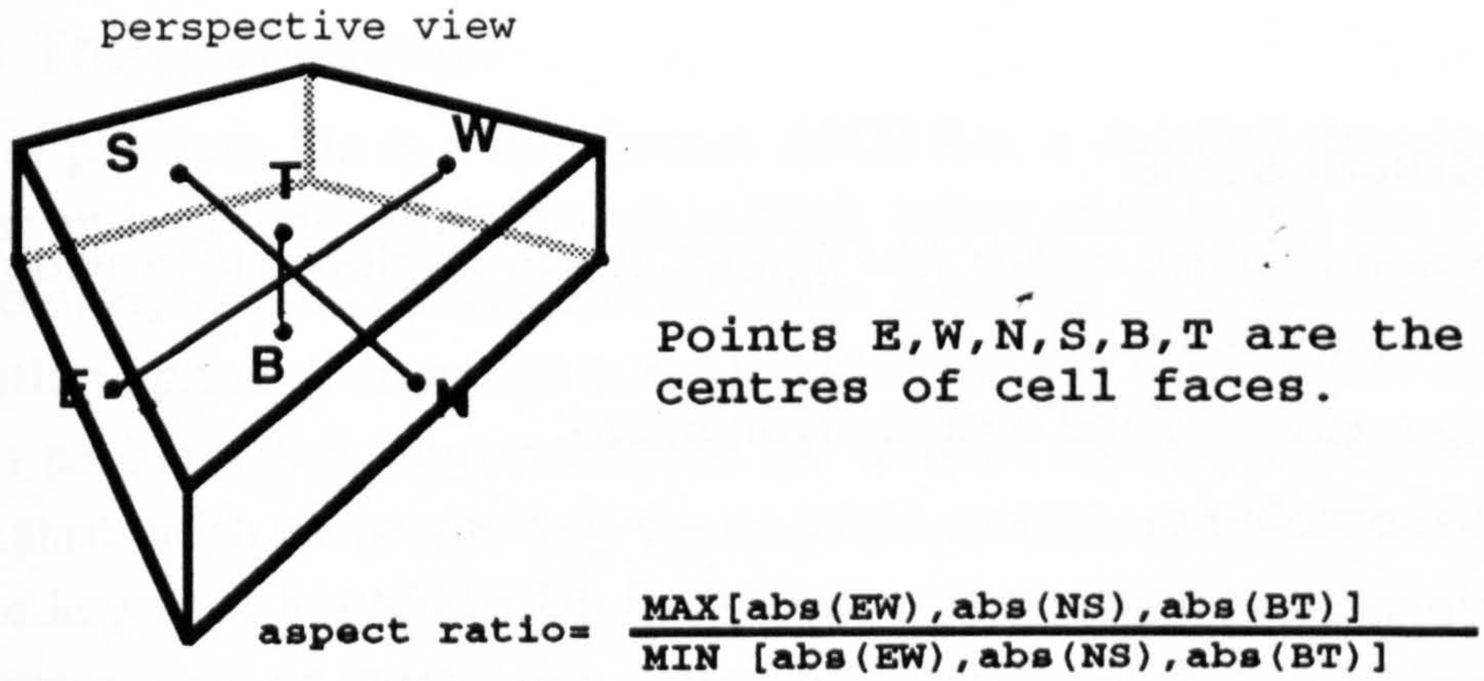


Figure 5.18 Method of calculation of the aspect ratio of a cell.

Skewness

Mesh skewness is checked by calculation of cells with warped faces. This involves subdivision of each cell face in triangles – in the fashion shown in figure 5.19 – and calculation of a dimensionless warpage number W with the formula:

$$W = 1 - \hat{a}1 \cdot \hat{a}2 \qquad (0.0 < W < 1.0) \tag{5.2}$$

where â1, â2 are the area unit vectors of each triangle. Since a cell face can be divided in two triangles in two ways, there are two sets of area unit vectors â1 and â2, therefore the dot product in formula (5.2) is taken to be the smaller of the two. The calculated warpage number for each cell face is compared against a user-specified tolerance.

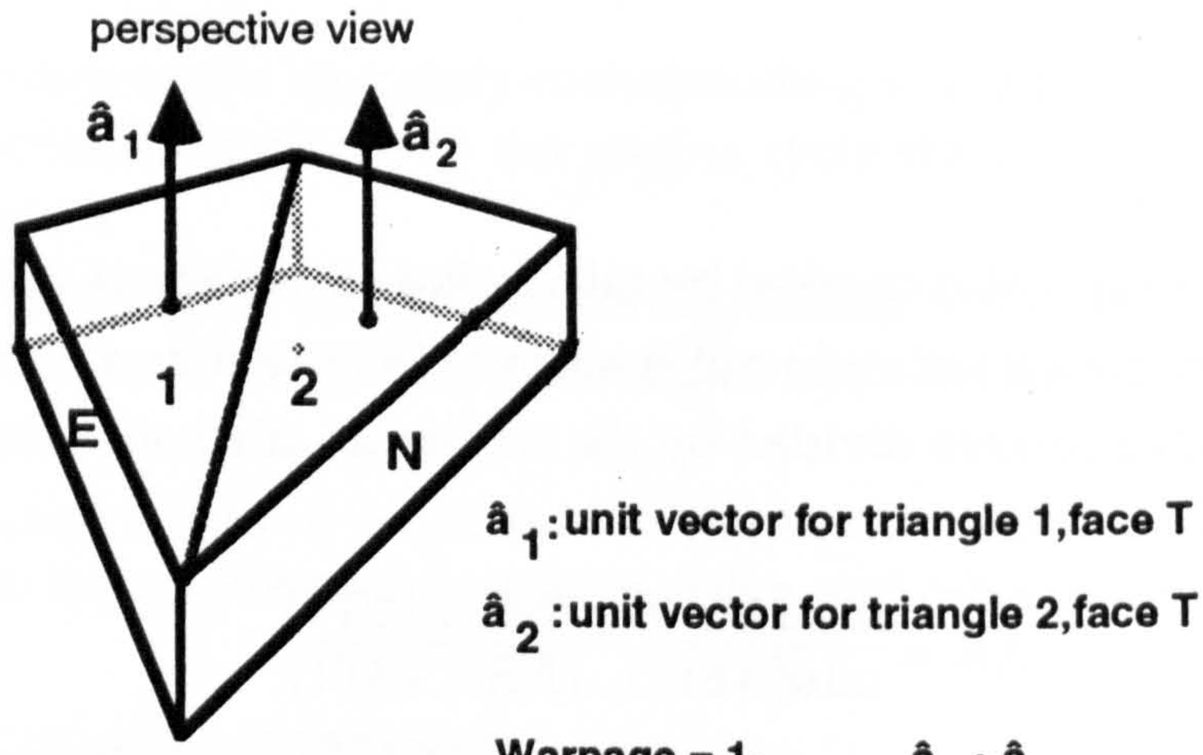


Figure 5.19 Definition of cell face warpage.

5.5 Problem setup

The *problem definition module* (see figure 5.2) was developed to provide a user interface for specification of:

- engine geometry and operation conditions,
- initial conditions,
- boundary conditions,
- parameters for control of the calculation,
- the type of fuel,
- injection data,
- combustion model parameters,
- parameters for control of the layer addition and removal module.

Such data, called 'problem-setup data' throughout this work, is input by the user in a file called 'problem file' (see figure 5.20). The contents of this file are described in section 5.5.1. Alternative means, involving definition of problem-setup data within CAD systems, are provided; however, they are not preferred because the design of such systems does not allow for customisation of their graphics processes and user-interface menus.

The interaction between the problem definition module and other modules of the preprocessing system is illustrated in figure 5.20. The pre-processing process ends with creation of the SPEED file that contains mesh and problem setup information for simulation of gas flow, fuel injection, and combustion with SPEED.

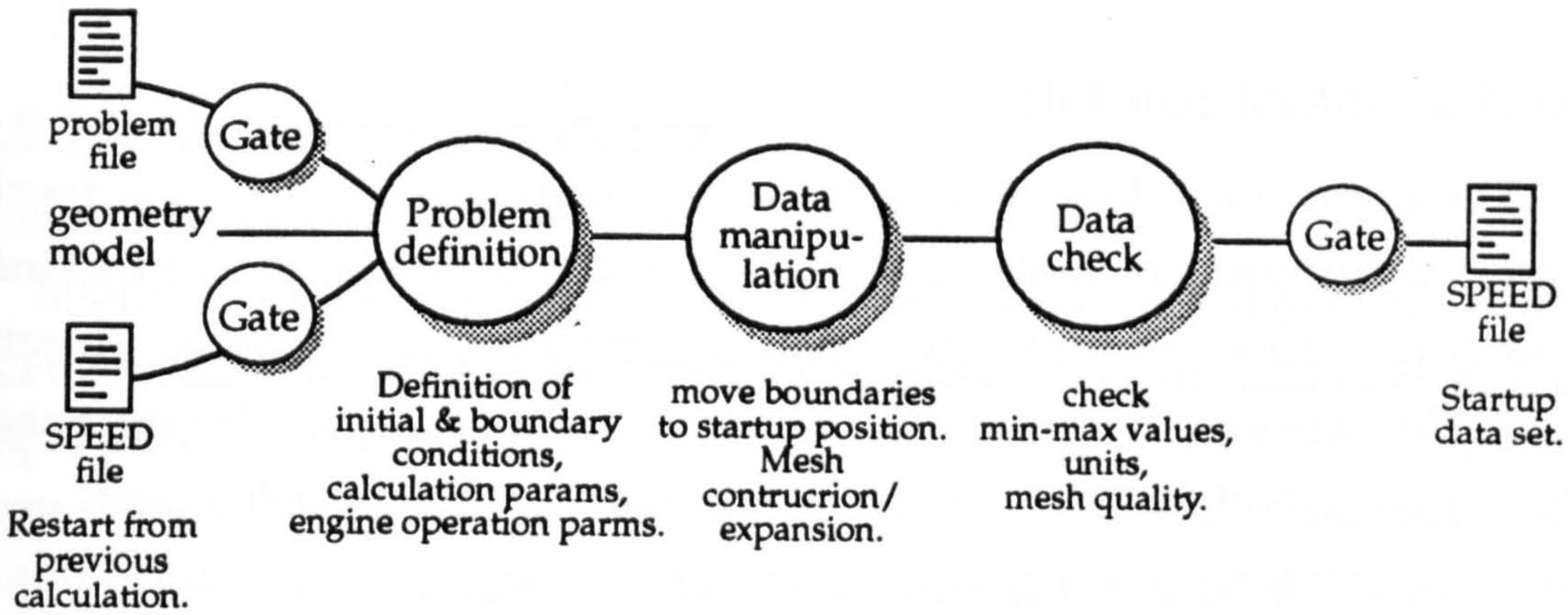


Figure 5.20 Data flow diagram of the problem definition process.

5.5.1 The problem file

The problem file is a free-format ASCII file; a detailed description of its format and contents can be found in [226]. Information in this file is organised into blocks, called 'option blocks', each referring to a particular problem definition option. If an option is not found in the file, default data are assigned by the problem definition module to the relevant data structures. Each option block starts with a keyword (e.g. the manifold condition block must be preceded by the keyword MANIFOLD). Each block contains one or more records, and each record one or more fields. Comment lines can appear anywhere between blocks. An example problem file for the specification of problem-setup data for simulation of the direct injection Diesel engine described in chapter 7, is given in appendix B.

5.5.1.1 Engine operation conditions

The engine geometry and operation conditions parameters, provided by the manufacturers, are input by the user in the problem file. The system provides for definition of:

- The engine speed, in r.p.m.,
- The length of the connecting rod,
- The stroke, and
- The valve lift profile for each valve as a function of time $Z_l(t)$. This is provided by the manufacturer of the engine, usually as a function of camshaft angle. In this case, it must be converted to crankshaft angle timing, since this is taken as the time base for input by the system.

The prescription of injector data is described in a separate section.

5.5.1.2 Initial conditions

These are the values of velocity **u**, thermodynamic pressure p, specific energy e, turbulence kinetic energy k, rate of dissipation of turbulence kinetic energy ε, and the unburnt fuel vapour concentration Y_{fu} described in section 3.2.2. They are specified for each cell in the calculation domain for the stage in the engine cycle the SPEED calculation is to commence. Their values are:

- available from earlier calculations with the SPEED code;
- available from experimental measurements; or
- estimated.

In the first case, initial conditions are available in the SPEED file (see figure 5.20) and are directly supplied to the preprocessing system. In the second case, such values, available in the form of data sets or tables, often normalised by some engine feature (e.g. velocity normalised by piston velocity), are converted to the format of or typed in the problem file. In the third case, global values for the whole chamber are calculated by means of one-dimensional tools for simulation of gas flow and combustion in engine chambers or estimated by the user.

The main options provided by the system for definition of these values in the problem file are:

- input of a separate value for each cell;
- input by a single value for the whole chamber;
- input by an algebraic function.

An example of the last option is the definition of a solid-body swirl, defined as:

$$\mathbf{u} = \Omega \wedge \mathbf{r} \tag{5.3}$$

where: **r** is the distance of the cell centre from a user-specified swirl axis.

Ω is the z-component of the angular velocity of the gas in the chamber (in rad/sec), defined as:

$$\Omega = S_N E_{speed} \tag{5.4}$$

where: S_N is the swirl number, characteristic of the geometry of the engine

E_{speed} is the engine speed (rad/sec)

5.5.1.3 Boundary conditions

Boundary conditions on inlet or outlet flow boundaries

SPEED provides for flow to enter or exit the computational domain through inlet and exhaust valve areas called valve curtains. Their definition and the requirements by the SPEED code for description of the flow through them have been discussed earlier in section 3.2.2.3.1. These requirements are met by the preprocessing system by prescription, in the problem file, of the following data:

- The discharge coefficient $C_D(Z_l)$ (effective flow area/valve curtain area), defined in section 3.2.2.3.1. It is determined experimentally as a function of the valve lift and input in the problem file as a set of valve lift and corresponding discharge coefficient pairs for each valve.
- The dimensionless radial velocity $u_r(\theta,z)$, axial flow angle $\alpha(\theta,z)$, and swirl flow angle $\beta(\theta,z)$ distributions, expressed as functions of position in the local cylindrical coordinate system of each valve, shown in figure 3.8. These values are interpolated to the boundary nodes of the curtain, where the fluid-velocity vector $u(\theta,z)$ is calculated by equation 3.17.
- The pressure and temperature in the manifold. These values are either determined from measurements or calculated with other simulation codes (e.g. STAR-CD) and are input into the problem file as a set of crank angle, pressure, and temperature triplets.
- The experimentally determined or estimated turbulence intensity λ_k and length-scale λ_ε factors for each valve curtain. These are used for the calculation of the k and ε on the valve curtain through equations 3.18 and 3.19.

Boundary conditions on walls

The physical boundaries of the computational domain have, during mesh generation, been split into several wall regions The system allows for prescription of the wall region type, velocity and heat-transfer status. The type of each wall region is specified by a character string (e.g. 'PISTON', 'LINER', 'HEAD'). Its motion status is specified by associating each wall with a wall-motion flag to be:

- stationary,
- moving with the speed of the piston, or
- moving with the speed of a valve.

Finally, by setting a heat-transfer flag, the wall is defined to be:

- adiabatic or

- heat-conducting

In the latter case, the pre-processing system allows for definition of a uniform or non-uniform surface temperature distribution.

5.5.1.4 Calculation control data

The system provides for specification of:

- The time, in crankshaft-angle values, in the stage in the engine cycle at which the calculations in SPEED are to commence.
- The type of the SPEED run. The user is provided with a choice of:
  - *Meshonly run*, to move the mesh without performing flow calculation.
  - *Global thermodynamics run*, to run the SPEED code treating the mesh as one single control volume for each timestep. This calculates the average cylinder pressure, density, temperature, and inlet mass flux (if any).
  - *Cold flow run*, to setup the code to perform a simulation of gas flow under motoring conditions.
  - *Diesel combustion run*, to setup the code to perform gas flow, fuel injection, autoignition, and combustion simulation.
- Piso algorithm parameters for pressure convergence, described in section 3.2.3.3 and [173].
- The k-ε turbulence model parameters C_1, C_2, C_3, and C_μ, described in section 2.1.1.1 and [173].

5.5.1.5 Fuel injection parameters

The system provides for specification of the injection system and selection of the spray related model that will be used. This is done by input in the problem file of the following data:

- The location of the injector. If the injector has more than one holes this is the location of each hole.
- The diameter of each injector hole.
- The direction of injection for each hole.
- The start and end of injection (in degrees of crank angle).
- The temperature of the injected fuel.
- The injection flow rate profile. This is the volume of fuel that is injected within a time period in the engine cycle. It is available from the engine manufacturer and is input for each injector hole as a set of crank angle and injected volume pairs.

- The number of parcels that are injected per timestep per injector hole (see section 3.2.2.2). Unlike the above items, which are given by the injector's manufacturer, this is estimated by the user. As it will be shown in chapter 7, this number may have an effect on the calculation of the peak combustion pressure.
- Selection of the nozzle flow, atomisation, and spray-wall interaction model to be used and parameters thereof. These are described in section 3.2.2.2 and in reference [173].

5.5.1.6 Fuel and combustion model parameters

The system provides for specification of the type of fuel and parameters of the Magnussen Diesel combustion model. Since only single-component fuels are supported in SPEED, the type of fuel is specified by entry of the following parameters:

- Constants m and n in hydrocarbon fuel C_mH_{2n+2}.
- Initial air/fuel mass ratio.
- CO/CO_2 mass ratio.
- Enthalpy of combustion at standard conditions (T=298K) giving products liquid water and carbon dioxide gas.
- The reaction acceleration parameter of the shell ignition model for the particular fuel. It represents the ratio of the fuel breakdown reaction rate to that of RON90 fuel (see reference [173]). This value is estimated by the user. It is known that the heavier the fuel, the faster the breakdown rate, with n-dodecane fuel having a value of 145.0.

The user does not need to input the above data when using n-heptane or n-dodecane fuel, because they are predefined in the system. In that case these fuels are defined as follows:

	n-heptane	n-dodecane
chemical type	C_7H_{16}	$C_{12}H_{26}$
initial air/fuel mass ratio	1.0	1.0
CO/CO_2 mass ratio	20.0	20.0
enthalpy of combustion	4.8169E+9 J/Kmole	7.0E+9 J/Kmole
reaction acceleration parameter	95.0	145.0

The parameters of the Shell ignition and Magnussen combustion models, described in section 3.2.2.2, are specified by entry of the following values:

- Constants A and B of the Magnussen combustion model, described in section 3.2.2.2.
- The transition temperature from ignition to combustion. This value is estimated by the user.

5.5.1.7 Layer addition and removal control parameters

The layer addition and removal module of the SPEED code, described in section 3.2.3.2.2, is controlled by the following parameters, that are input in the problem file:

- A user-estimated time range within which layer addition above the piston or below the valve base is allowed.
- A user-estimated time range within which layer removal above the piston or below the valve base is allowed.
- Distortion values that trigger layer addition or removal action. These values are estimated by the user for each mesh. Distortion is defined as the product of the minimum, average, and maximum cell aspect ratio of all the cells adjacent to the piston or valve base. A detailed description of the layer addition and removal module of the SPEED code and its input requirements can be found in reference [173].

5.6 Visualisation

CAD systems are used for visualisation of SPEED results, that are stored in SPEED's results file. They are complemented by data manipulation and visualisation mapping tools (see section 4.3) provided by the system. A module, 'GRAFIS', converts visualisation data that have been manipulated and mapped to geometric entities, into CAD geometry and field data structures. This data is stored in the geometry and fields file of the system. An overview of the visualisation process of the pre- and post-processing system is illustrated in the data flow diagram of figure 5.21.

The CAD systems currently supported for visualisation are PATRAN and SUPERTAB. The system can also handle data in the IGES format, in this way enabling portability to other CAD systems. The facilities provided for visualisation of SPEED results are summarised in the following list. They are discussed in detail in section 4.3.

Data manipulation tools:

- *Cutting volumes*: Results can be presented on a portion of the mesh, in this way reducing data storage and computation times (see fig. 4.8).

- *Field contraction*: When possible the kinetic energy or particle traces can be visualised, instead of the three-dimensional gas velocity.
- *Surface sampling*: Temperature contours can be displayed on heat-conducting walls. The radial velocity of the gas through the valve curtain or its kinetic energy can be visualised on valve curtains.
- *Filtering*: The volumetric efficiency, the mean indicated pressure, and the specific fuel consumption of the engine are calculated from the simulation results. The calculation of these quantities has been discussed in section 4.3.1.4.
- *Normalisation*: Gas velocity can be normalised by the piston velocity. Also results can be linearly, exponentially, or logarithmically fitted between user-specified visualisation values.

Visualisation mapping and rendering tools:

- *Colour mapping* by definition of a colour-transfer function.
- *Point probing*: This is a useful tool for getting quantitative information out of mainly scalar fields. Both PATRAN and SUPERTAB allow for specific probe placement using a 2D input device, normally the mouse.
- Definition of *cutting planes* through the computaional domain and interpolation of scalar and vector fields on them. Multiple cutting planes are also possible (see fig. 4.14).
- Display of *line, filled-area,* and *continuous-tone contour plots* on cutting planes are used for visualisation of scalar fields (see figs. 4.12 and 4.14).
- *Arrow plots* on slicing planes through the flow are used for visualisation of the gas velocity (see figs. 4.13, 4.14, and 4.18). Arrows can be 'anchored' at their centre, tip, or base, the former being the preferred option.
- *Carpet plots* (see fig. 4.15) on cutting planes can be used for visualisation of scalar fields.
- *Isosurfaces* , corresponding to user-specified threshold values ,can be used for visualisation of scalar fields (see fig. 4.13).

Results interpretation with the above visualisation tools is supported by means of the following computer graphics functions that are provided:

- Definition of light sources and shading.
- Semitransparent display of entities.

- Perspective.

The above utensils have been discussed in detail in section 4.3 of the previous chapter. Several example visualisations are given there, as well as in chapter 7. Two techniques that were presented in section 4.3 and are not supported here are particle traces and spheres representing fuel droplets. The latter is particularly useful for the visualisation of fuel sprays, in the case of Diesel engine simulations. This limitation, which is due to the fact that such entities can not be rendered in CAD systems, was overcome by implementation of additional visualisation systems, presented in the following paragraph.

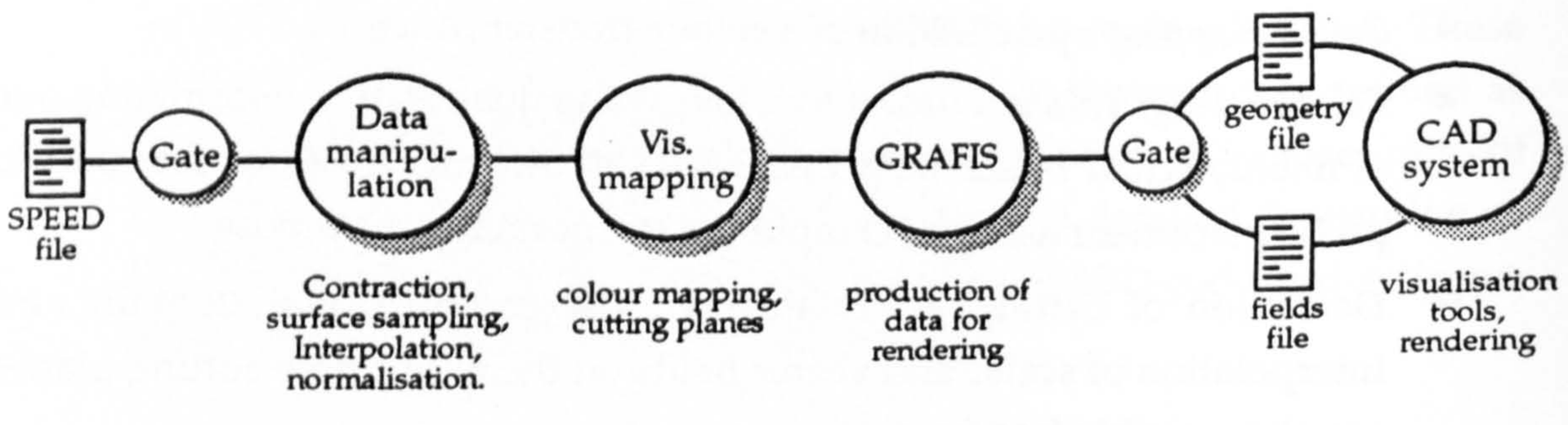


Figure 5.21 The visualisation pipeline of the pre- and post-processing system.

5.6.2 Alternative visualisation approaches

The inability of the above system to efficiently visualise fuel-sprays and particle traces was overcome by customisation of two visualisation systems: PROSTAR and AVS. This was done by development of additional gates (see figures 5.2 and 5.22) to convert previously manipulated SPEED results to the file formats of the above packages.

Spray droplets were visualised in PROSTAR as a set of spheres with diameter that of an injected fuel droplet and colour according to its size or temperature. The visualisation with PROSTAR of spray droplets, alongside the gas flow and temperature in a DI Diesel engine combustion chamber (shown in figure 5.29) is shown in figure 5.30.

Particle traces were visualised in AVS as curved lines in the fashion shown in figure 4.19. The customisation of AVS and the relative merits of the system are discussed in reference [49].

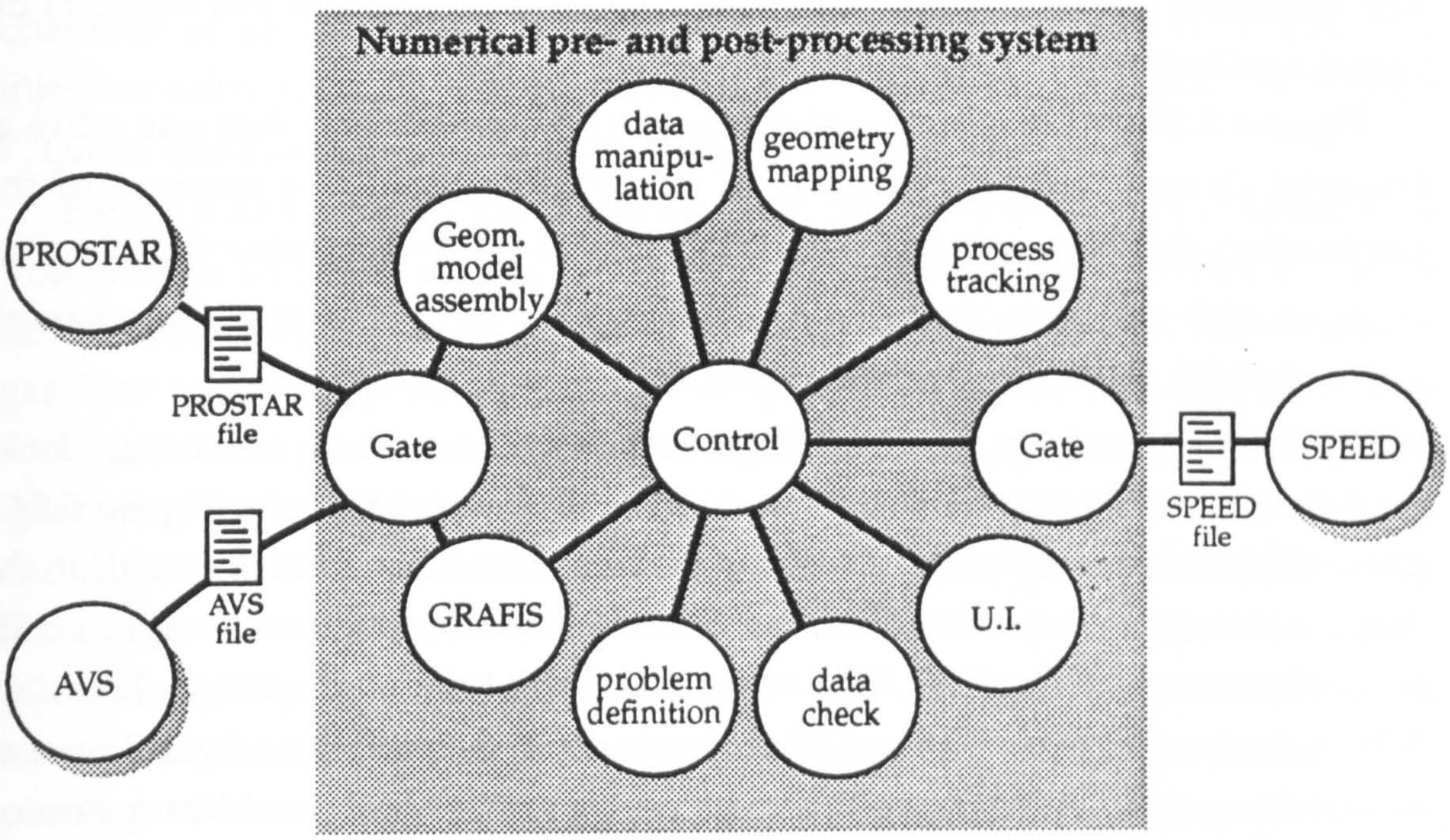


Figure 5.22 Data flow diagram of the interface of the pre-and post-processing system for SPEED and the PROSTAR and AVS visualisation packages.

5.7 Examples

Several examples of meshes and SPEED-results visualisations that have been generated with the above system have been shown earlier in this chapter as well as in chapter 4. These examples were taken from both petrol and Diesel engine simulations. One petrol engine simulation is for the pent-roofed, four-valve combustion chamber, a solid model of which is given in figure 5.5. The mesh for this is illustrated in figure 5.15. Continuous-tone contour plots of temperature during combustion are shown in figure 4.12. The objective in this section is to present several examples of meshes and visualisations of SPEED results; They are chosen to show the applicability of the proposed system in a variety of engine applications, with features such as:

- flat pistons;
- piston recesses of various designs;
- structured, block-structured, and unstructured meshes;

- definition of – single or multiple – cutting planes and presentation of scalar and vector fields on them;
- DI or IDI designs;
- Model or production engines;
- Meshing of valve-chamfers.

The emphasis in this section, as has been the case throughout this work, is on Diesel applications.

Figures 5.23 to 5.28 correspond to a model DI Diesel engine that was setup at Imperial College for the validation of the SPEED code. Three versions of the combustion chamber were studied, each one with a different piston design:

1. A disk-chamber. Two unstructured meshes for this, with different mesh density, are illustrated in figure 5.23.
2. A re-entrant lipped bowl. Figures 5.24.a and b show a coarse, block-structured mesh in elevation and top views respectively. Figure 5.24.c illustrates a top view of a finer version. An unstructured mesh for the same geometry, with peak mesh size 140K cells, is given in figure 5.25. Instances of the flow field at vertical and horizontal cutting planes are shown in figures 5.26 and 27 respectively. The simulation of gas flow was done with SPEED at the fine mesh of figure 5.25. For more information on this run the user is directed to references [46], [47], and [48]; the latter is included at the end of this work.
3. Square bowl. Elevation, side, and top views of the mesh are illustrated in figure 5.28.a, b, and c respectively. A top view of a finer mesh is shown in figure 5.28.d. SPEED simulations and visualisations of gas flow in this chamber can be found in reference [24].

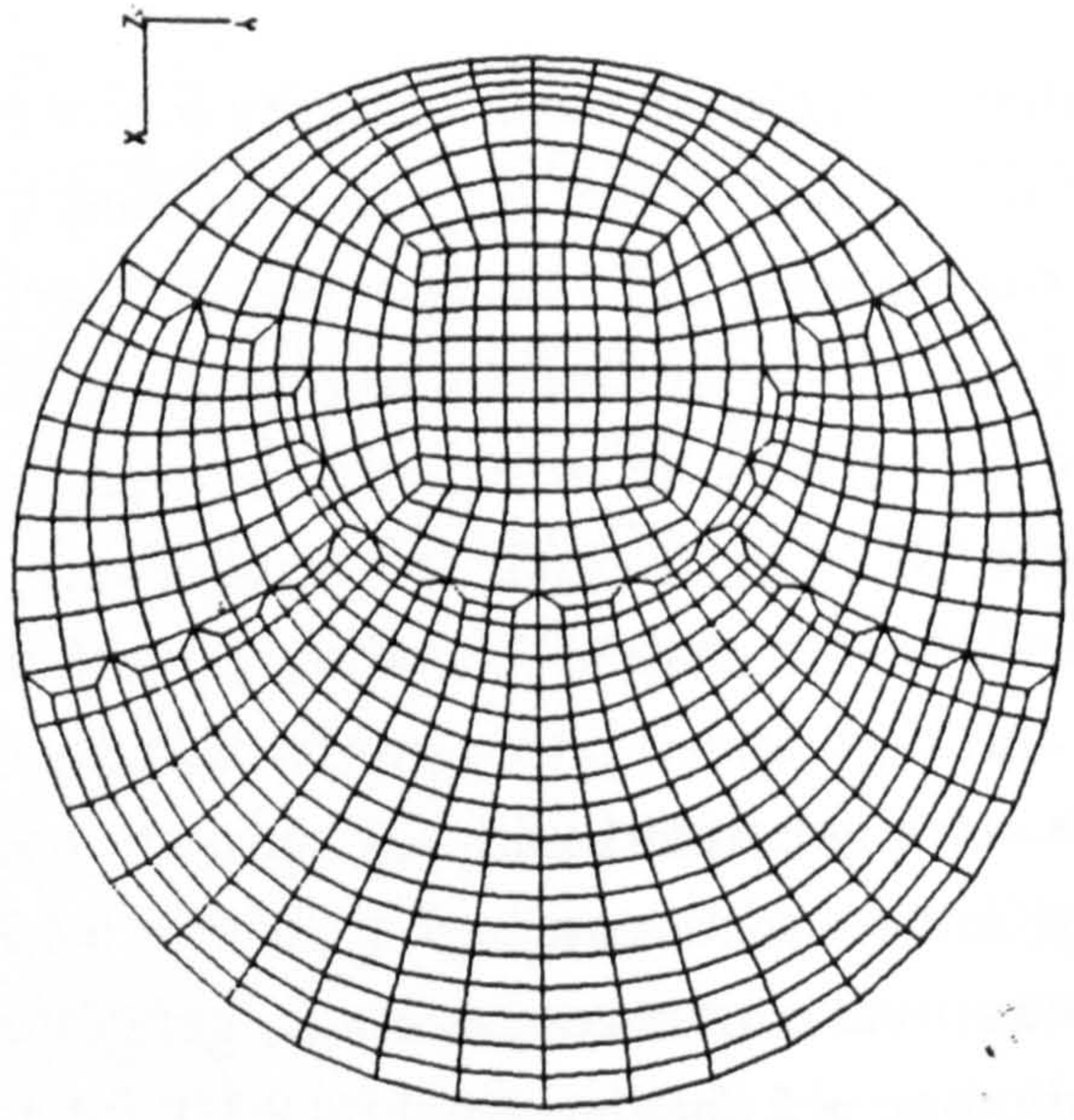
Figure 5.29 illustrates an unstructured mesh for the combustion chamber of the FIAT, CRF 1930, DI Diesel engine, with bowl in piston. The solid model for this engine has been shown in figure 5.4. The structure of blocks is illustrated in figure 5.12. Figures 5.29.a and b show side views of the chamber, the latter illustrating the structure of the mesh at the piston crown and bowl regions. An instance of the mesh at 60° CA ATDC with the inlet valve open and two layers added around the valve curtain is given in figure 3.10.c. The simulation and visualisation of gas flow under motoring conditions can be found in reference [46]. Diesel fuel injection and combustion calculations with SPEED have been studied in reference [170]. Fuel sprays at 2° CA BTDC of compression piston position, visualised as droplets of injected-fuel parcels (see section 3.2.2.2), are shown in figure 5.30.a. The temperature field, indicating that auto-ignition has taken place, is visualised by means of contour plots on cutting planes through

the flow, shown in figure 5.30.b. Arrow plots of gas velocity are given in figure 5.30.c.

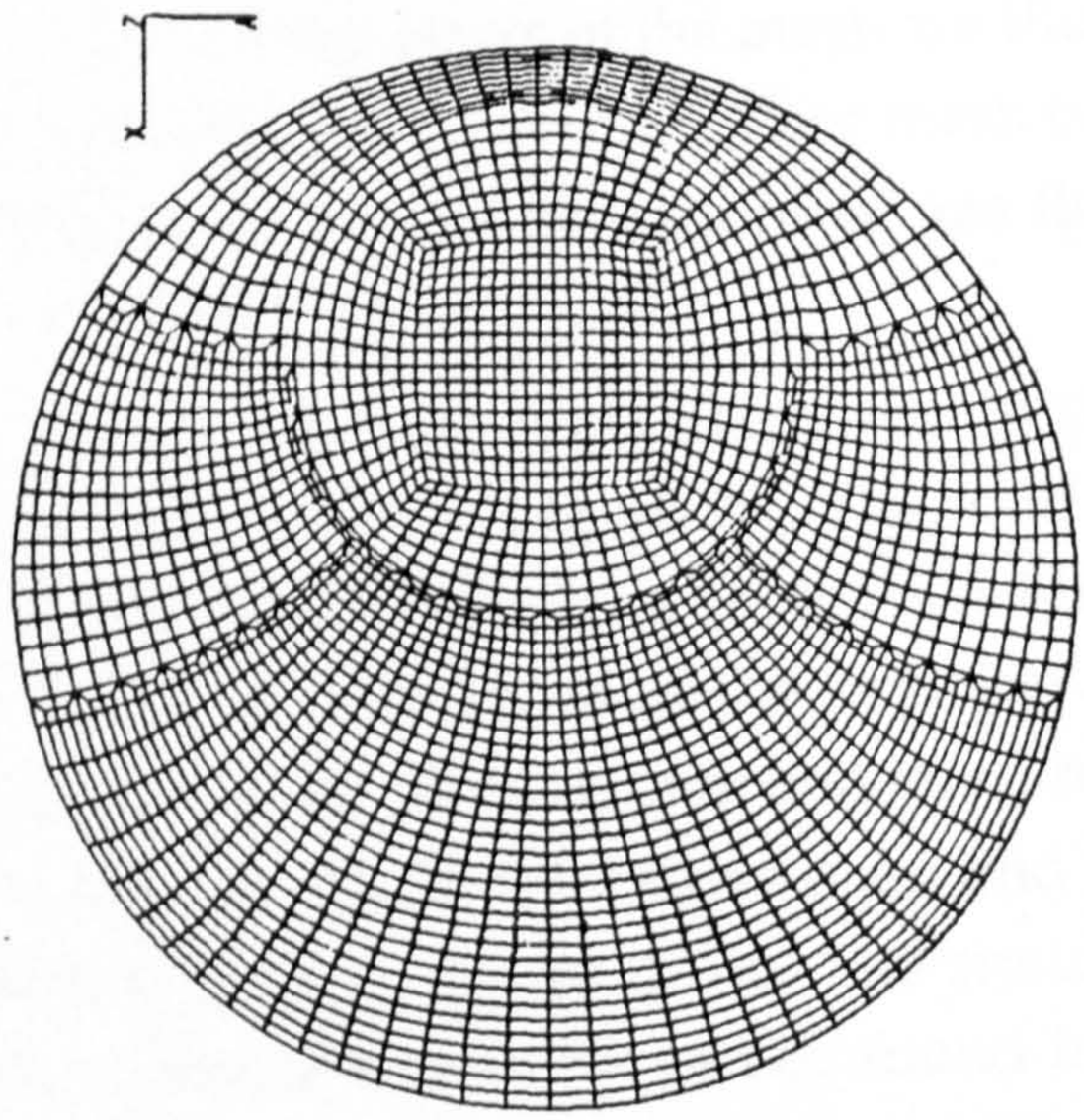
The block-structured mesh of figure 5.31 was generated for the combustion chamber of a model DI Diesel engine, with rectangular bowl, shown in figure 5.31.b.

The block-structured mesh of figure 5.32 was generated for the combustion chamber of an IDI Diesel engine provided by Renault. Gas flow and fuel injection calculations and visualisations in this chamber are presented in chapter 7. Diesel combustion studies can be found in [170].

Figure 5.33 shows an unstructured mesh for the combustion chamber of a Volkswagen DI Diesel engine. The bowl of this engine has been modified in the fashion described in chapter 7, to allow optical access (for LDA measurements of gas flow and high-speed photographs of sprays) in the chamber. The geometry and mesh were generated with the semi-automatic mesh-generation programs 'geometry' and 'mesh' described in section 5.4.2.1.1. Figure 5.28.a shows a top and elevation view of a coarse mesh, generated by setting the mesh density factor enumber = 1 (see section 5.4.2.1.1 for definition). The mesh in figure 5.28.b, featuring 21000 cells at TDC and 130000 cells at BDC piston positions, was generated by setting enumber = 3. Both meshes were generated at TDC piston position. A step-by-step presentation of the mesh-generation process for this and other variations of this engine are presented in chapter 7. Fuel injection and combustion studies, as well as visualisations thereof, are also presented there.

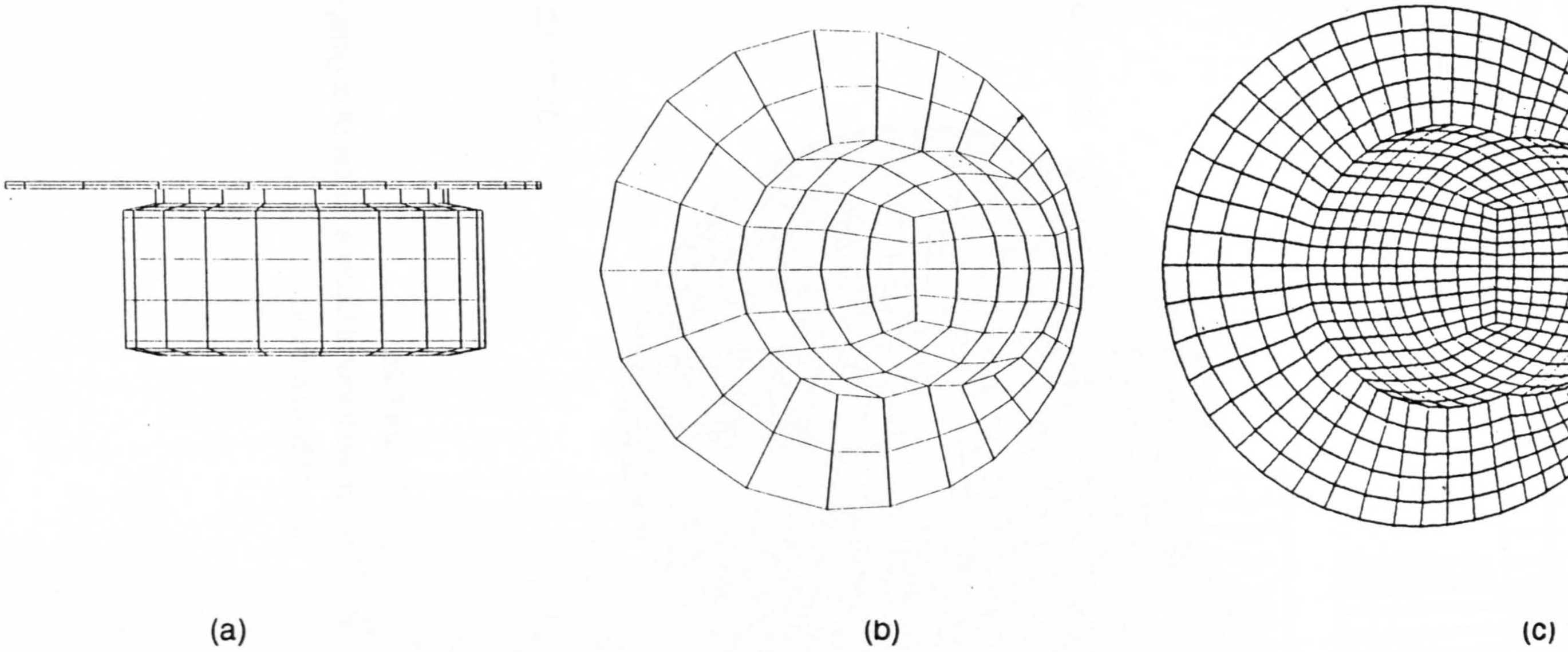


(a)



(b)

Figure 5.23 Top view of a mesh for the combustion chamber of the IC DI Diesel engine, with flat piston. (a) Coarse mesh, (b) finer mesh.



(a) (b) (c)

Figure 5.24

Computational mesh for the combustion chamber of the IC DI Diesel engine, with re-entrant lipped bowl. (a) elevation view of coarse mesh, (b) top view of coarse mesh, (c) top view of finer mesh.

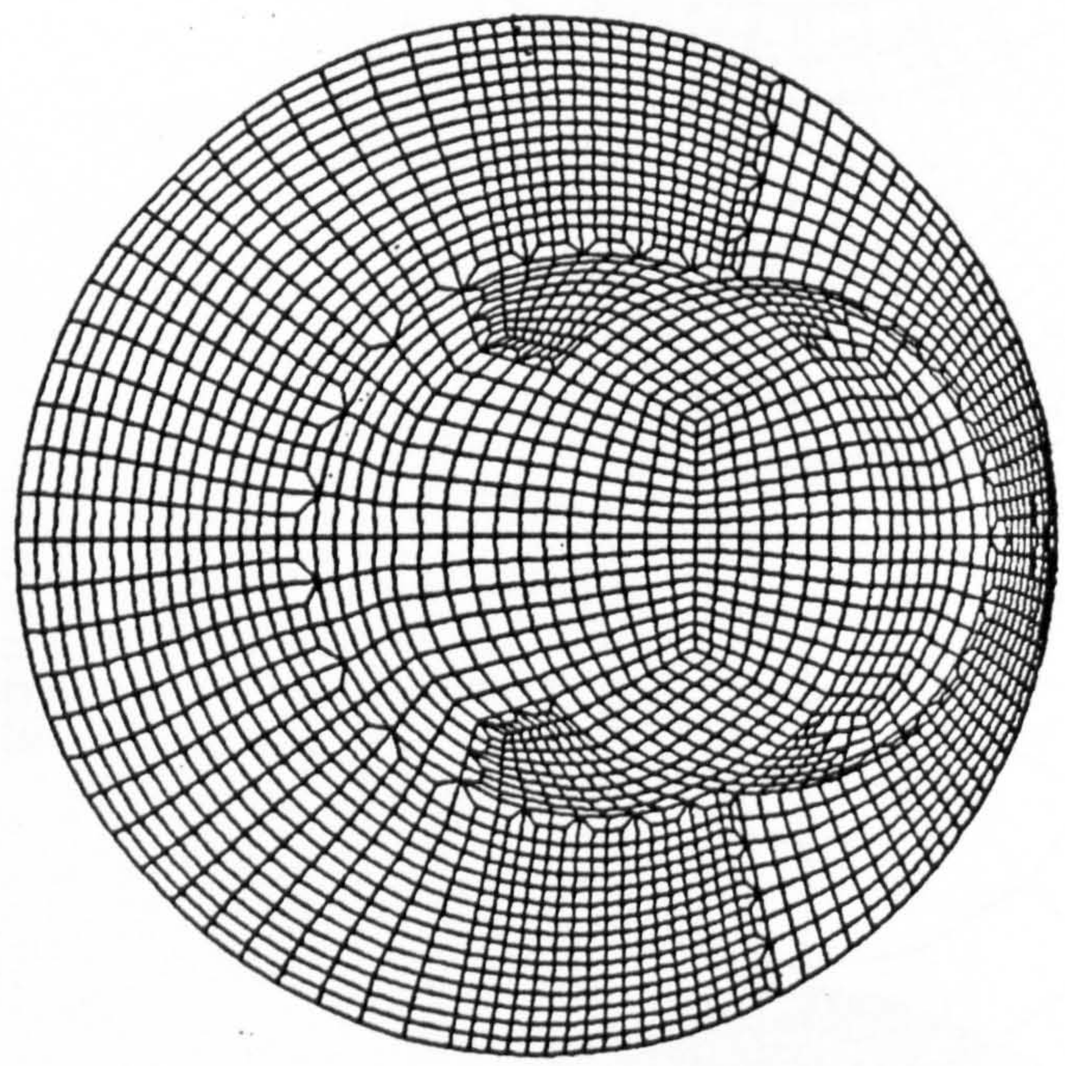
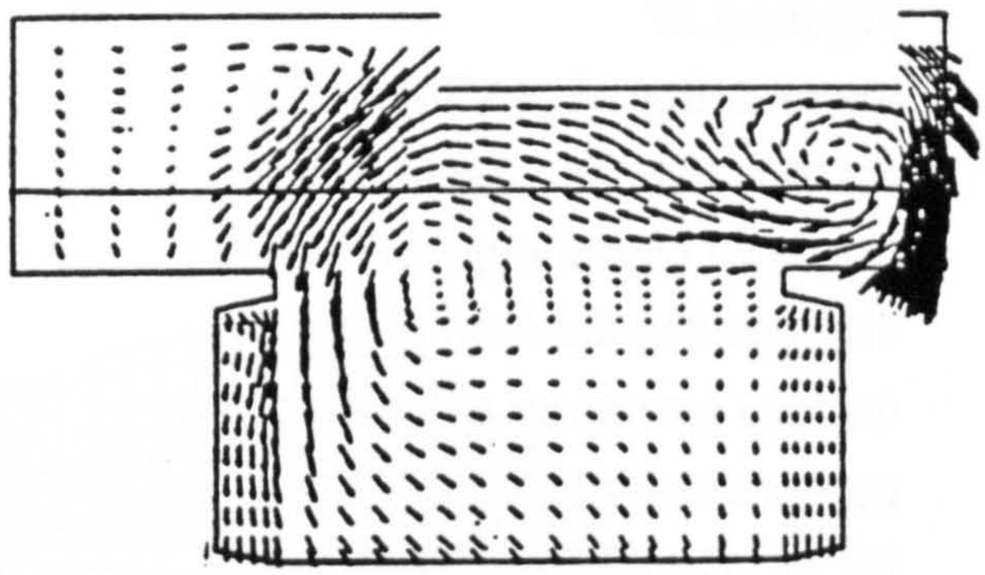


Figure 5.25

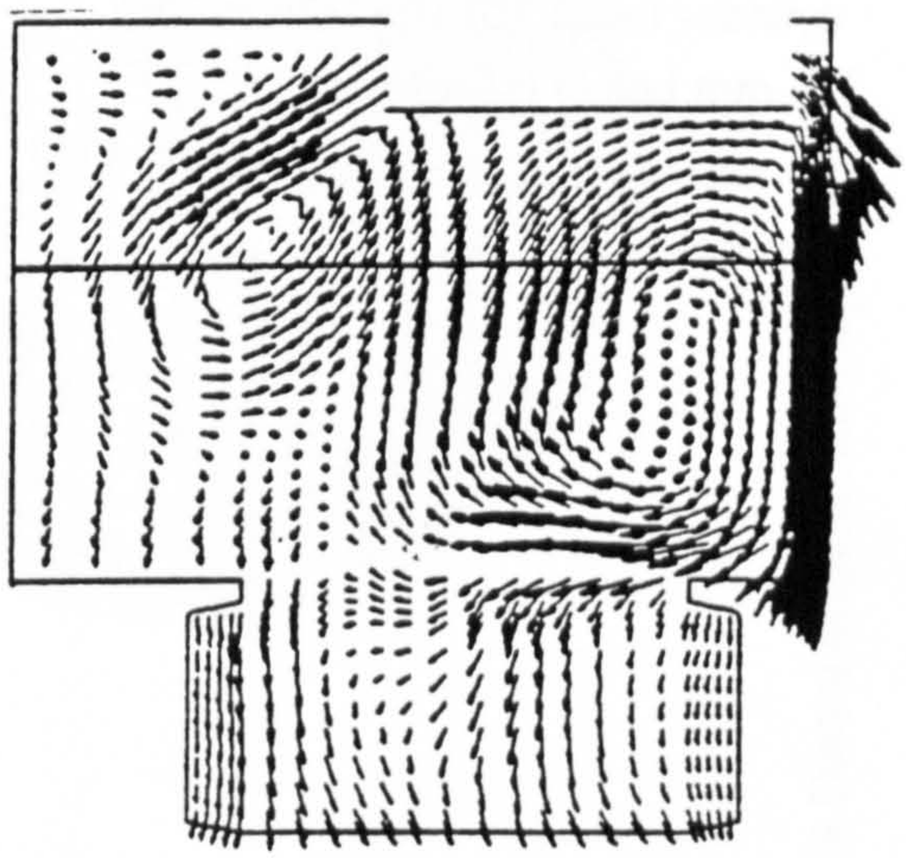
Top view of an unstructured mesh for the Imperial College model DI engine, with re-entrant bowl recess.

Figure 5.26
Visualisation of gas flow in the combustion chamber of the Imperial College model DI Diesel engine. Flow is visualised in the form of gas velocity arrows on a cutting plane through the cylinder, bowl, and inlet valve axes.

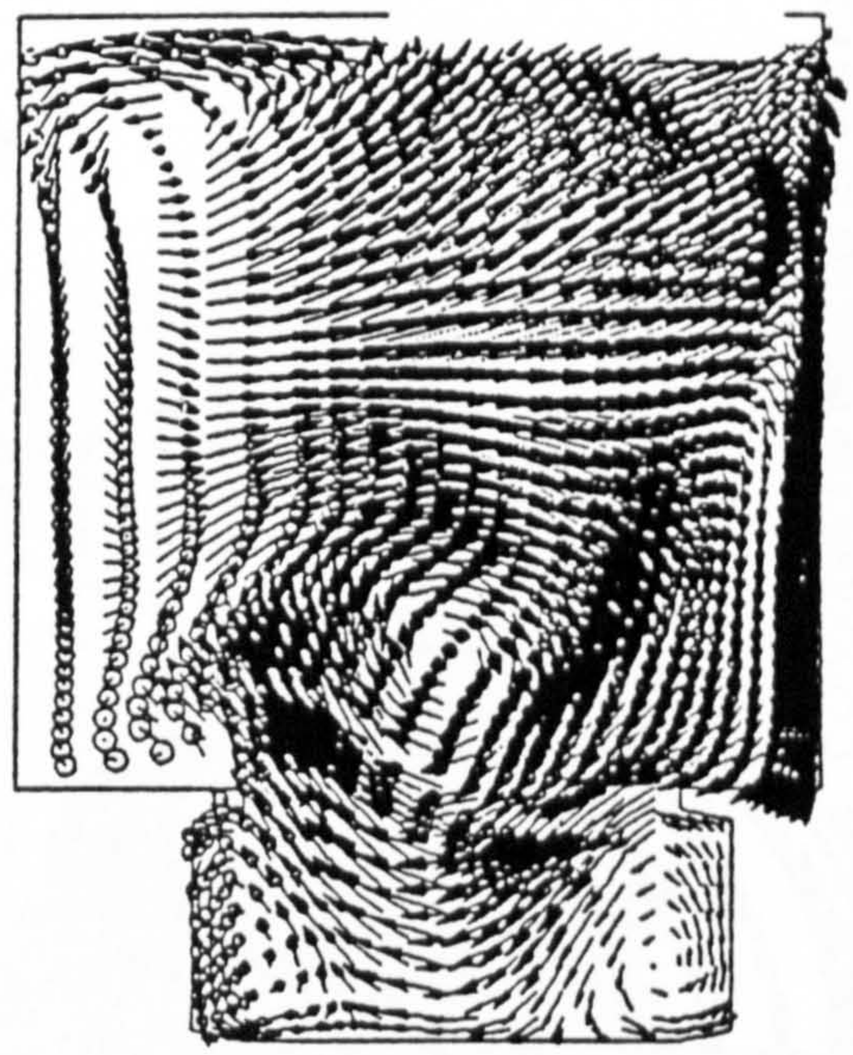
(a) 57° CA ATDC



(b) 108° CA ATDC

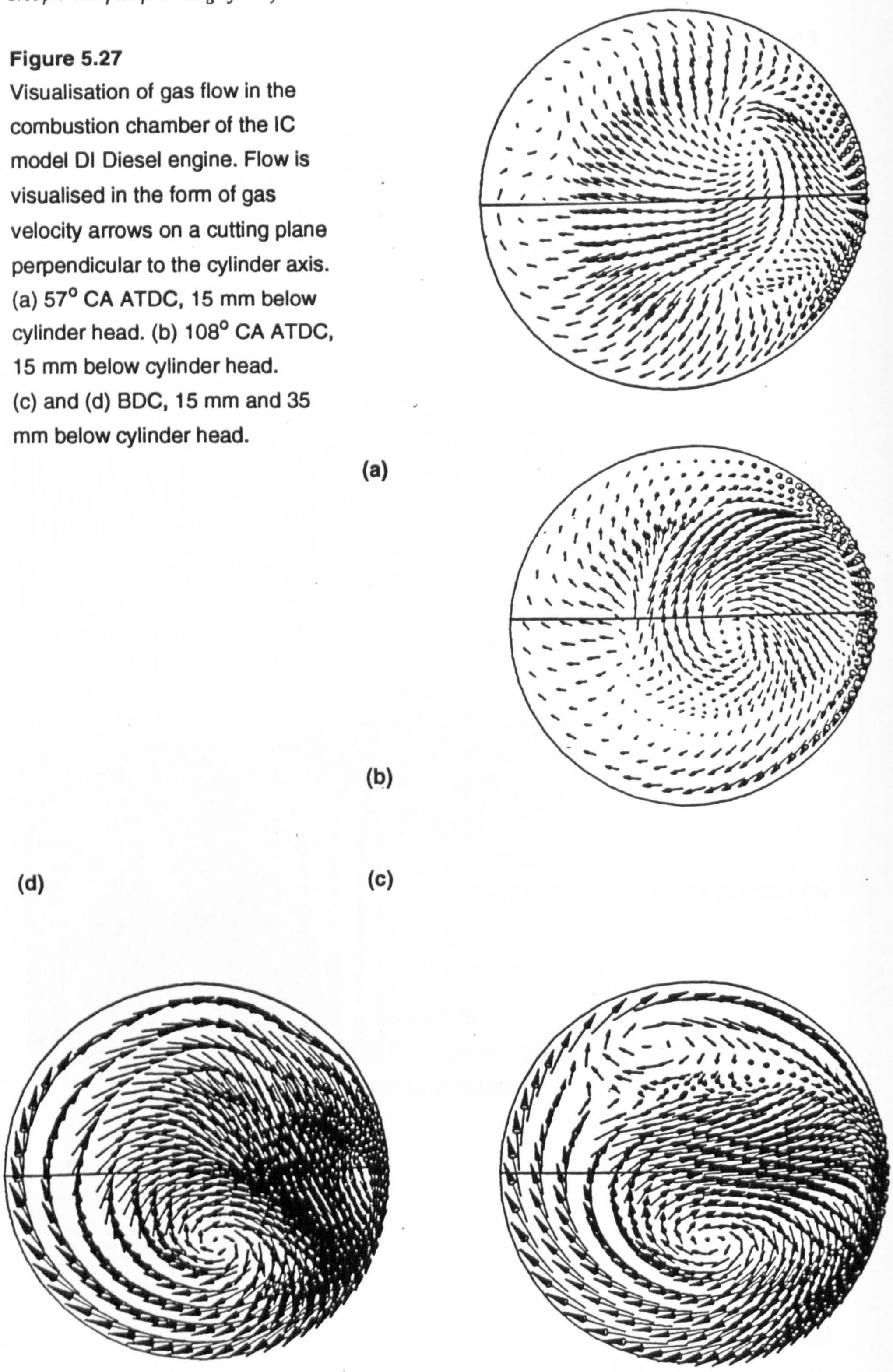


(c) 180° CA ATDC



(d) 340° CA ATDC

Figure 5.27
Visualisation of gas flow in the combustion chamber of the IC model DI Diesel engine. Flow is visualised in the form of gas velocity arrows on a cutting plane perpendicular to the cylinder axis. (a) 57° CA ATDC, 15 mm below cylinder head. (b) 108° CA ATDC, 15 mm below cylinder head. (c) and (d) BDC, 15 mm and 35 mm below cylinder head.



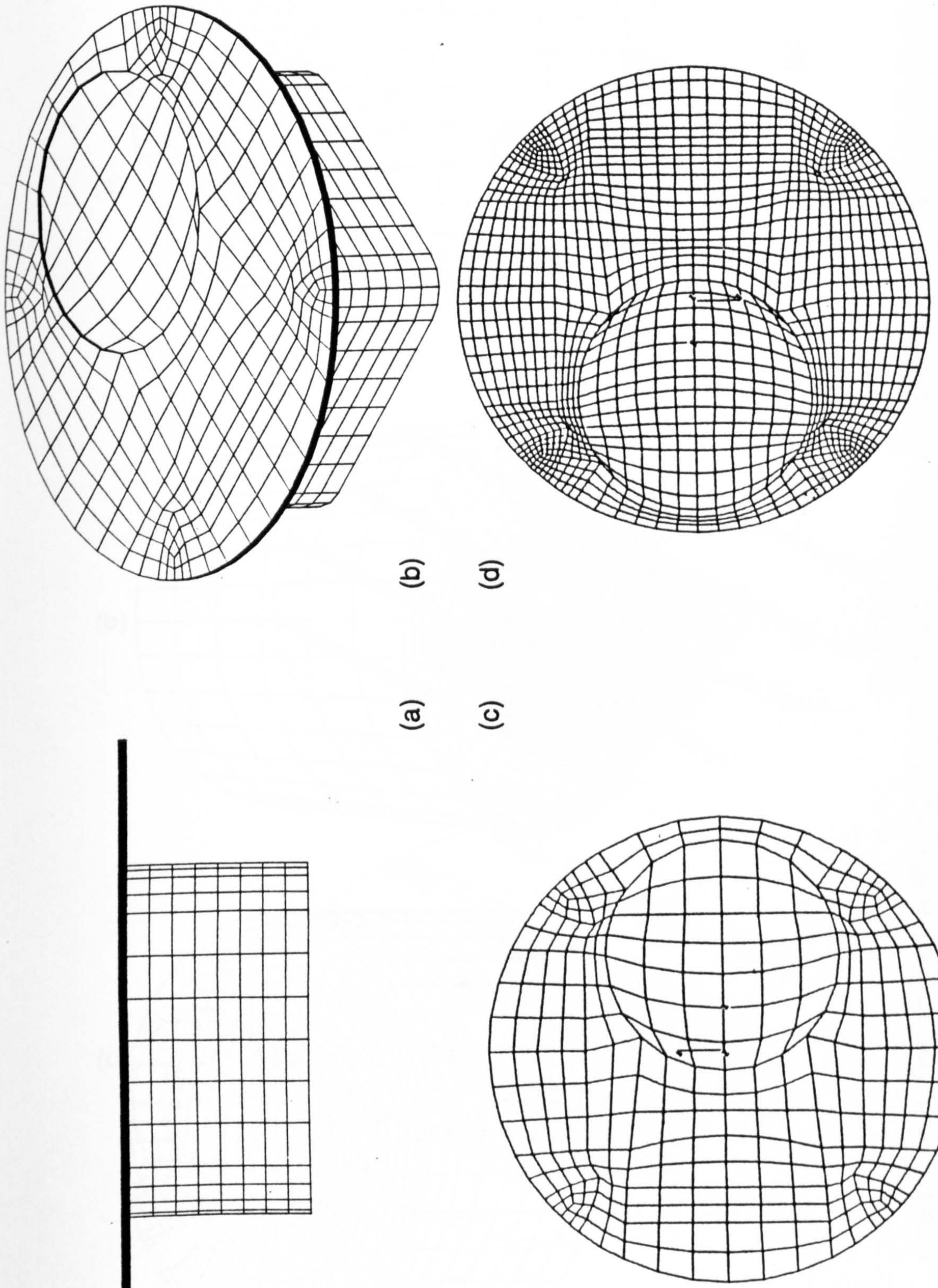
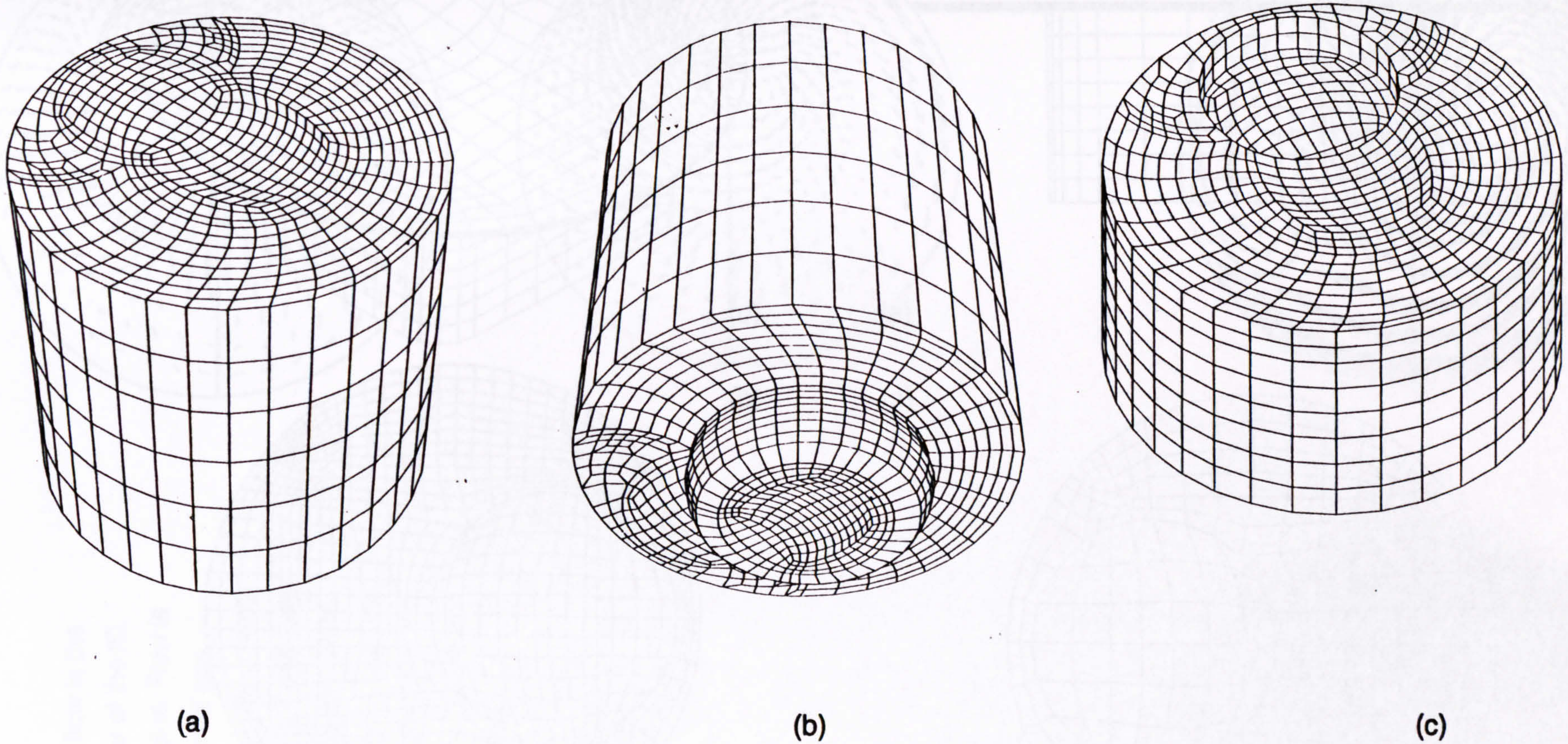


Figure 5.28 Computational meshes for the IC DI Diesel engine, with square bowl.



(a) (b) (c)

Figure 5.29

Computational mesh for the combustion chamber of the FIAT CRF 1930 DI Diesel engine, with bowl in piston.
(a) and (b) side views, (c) side view at 60 degrees CA ATDC, with inlet valve open.

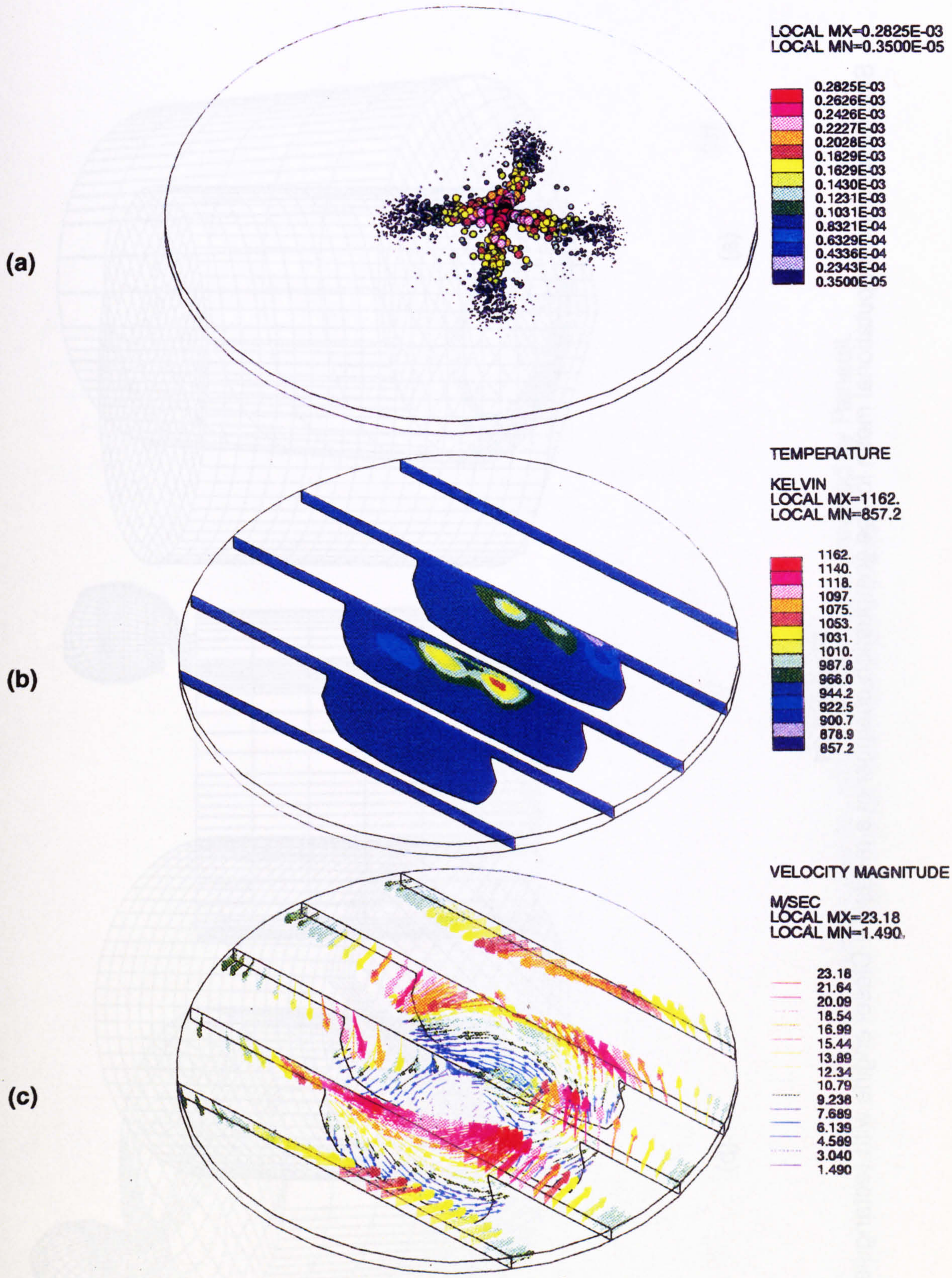
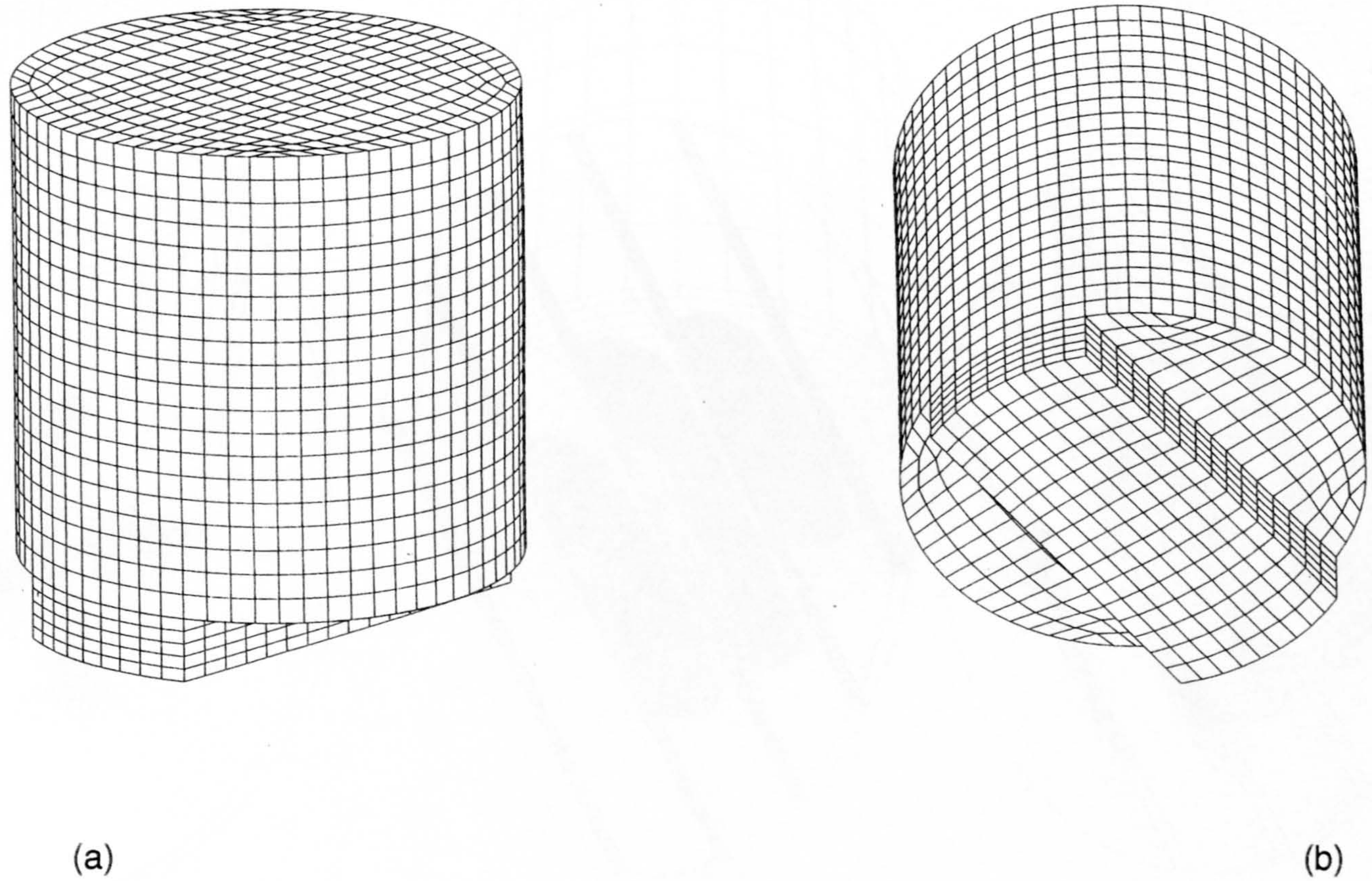


Figure 5.30

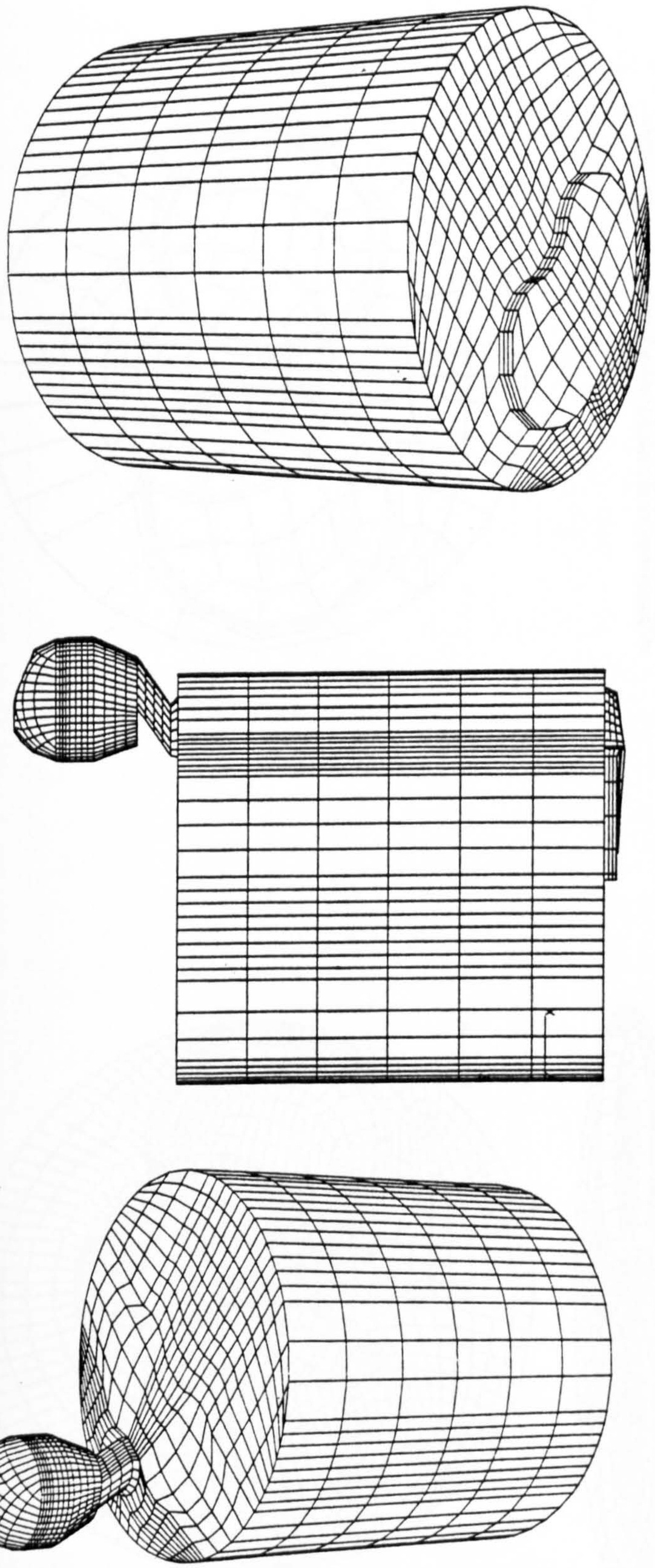
Visualisations of spray droplets, temperature, and gas flow in the combustion chamber of the FIAT CRF 1930 DI Diesel engine.



(a) (b)

Figure 5.31

Side views of a computational mesh for the combustion chamber of a model DI Diesel engine with rectangular bowl.



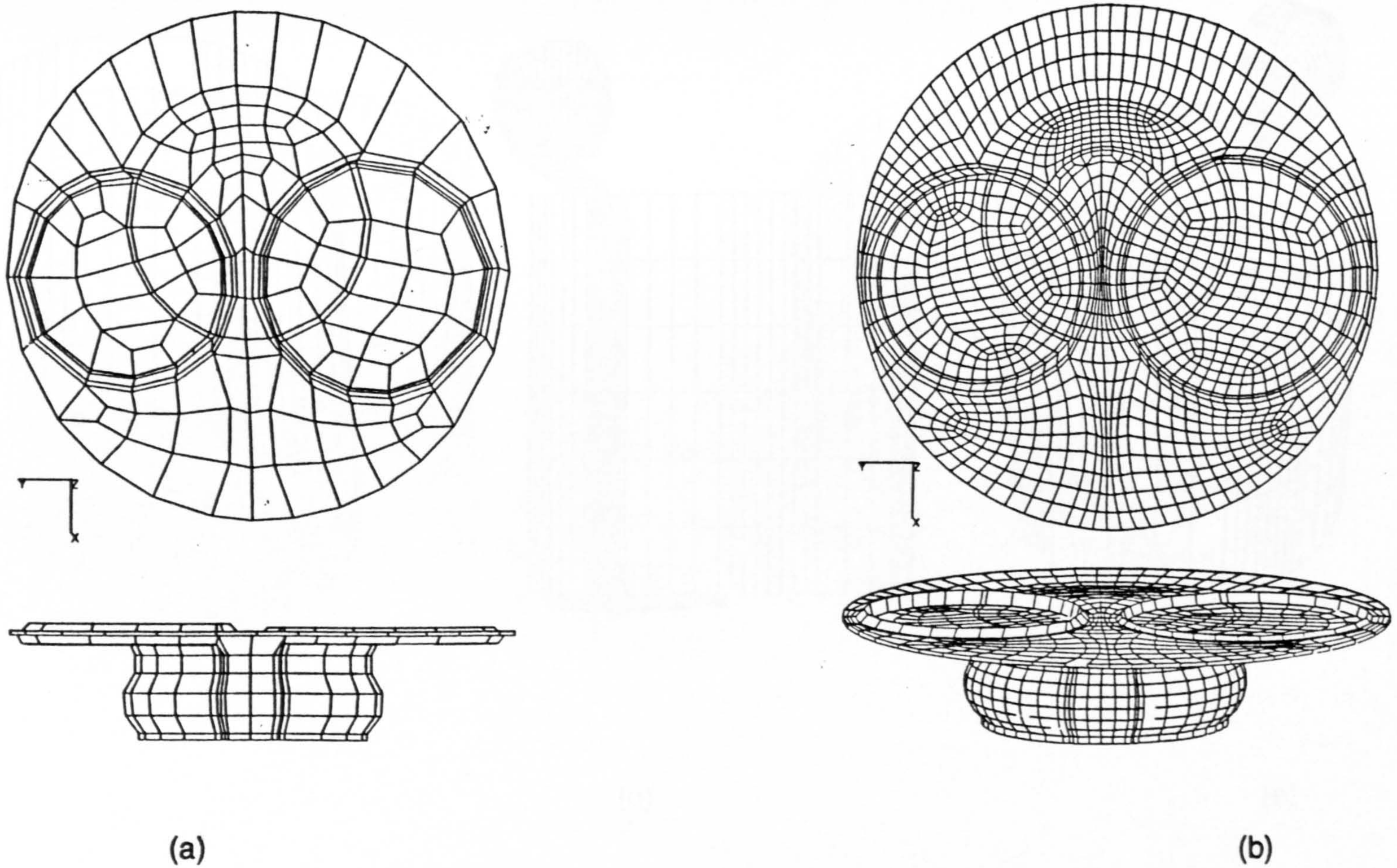
(c)

(b)

(a)

Figure 5.32

Computational mesh for an IDI Diesel engine provided by Renault.
(a) and (c) side views, (b) elevation.



(a) (b)

Figure 5.33

Computational meshes for a Volkswagen DI Diesel engine, with flattened bowl for optical access.
(a) Top: top view, bottom: elevation, (b) top: top view, bottom: side view.

Blank

In

Original

Chapter 6

Dynamic visualisation of SPEED results

The CAD system approach presented in the previous chapter has provided a working solution to the visualisation problem. However, CAD systems were until recently[1] designed for static visualisation of stress analysis data. No provision for visualisation of other type of data, such as specific CFD information, was made. An example is their inability to visualise particle traces, spray droplets, cutting planes moving through the flow, or animation. A demand has been identified in the engine community in general – and with researchers conducting engine simulations with SPEED in particular – for new tools, more functional and intuitive, that enhance interpretation of time-dependent, three-dimensional CFD data. These demands and alternative visualisation approaches are studied in this chapter, especially with respect to animation.

In this chapter requirements for the dynamic visualisation of the phenomena modelled by the SPEED code in the combustion chamber of reciprocating internal combustion engines are discussed in section 6.1. The possibility of meeting these requirements has arisen with the advent of high-performance 3D graphical workstations, combined with specially developed animation technology. Software and hardware systems for animation are discussed in sections 6.2 and 6.3, mainly from the point of view of their suitability for the visualisation of engine processes as modelled by SPEED, and cost effectiveness. Finally, a system for the dynamic visualisation of SPEED results, named AERAS (Animation Environment for Results Assessment with SPEED), that has been designed and implemented by the author at Imperial College, is presented. This system satisfies the above requirements for visualisation and enhanced comprehension of SPEED data sets, and is built around a commercial animation software named Intelligent Light 3DV.

6.1 Advanced visualisation requirements in SPEED

The result of a simulation with the SPEED code is a set of time-varying physical fields, defined over a computational domain that may contain up to one

---

[1] The wide application of CFD simulations in several engineering disciplines has recently resulted in the provision for CFD visualisation features in CAD systems. Such an example is PATRAN version 3, providing CFD visualisation tools, and an open system architecture.

million grid points for each of a large number of timesteps (typical temporal resolution is $0.1^{o} - 1^{o}$ CA). In this section, a brief description of the time-varying nature of geometry and field data generated by simulation with the SPEED code is followed by a discussion on how visual metaphors and cues that are involved in the visualisation of such fields can be more functional when animated in space.

SPEED imposes no restrictions on the shape of the calculation domain. This is the combustion chamber, defined by the moving surface of the piston, the cylinder liner and head, the moving valves and, in the case of IDI Diesel engines, the pre-chamber (figure 6.1). The shape of the combustion chamber is made more irregular by protrusions such as injectors and glow plugs, recesses on the piston surface or crevices where there are joining parts.

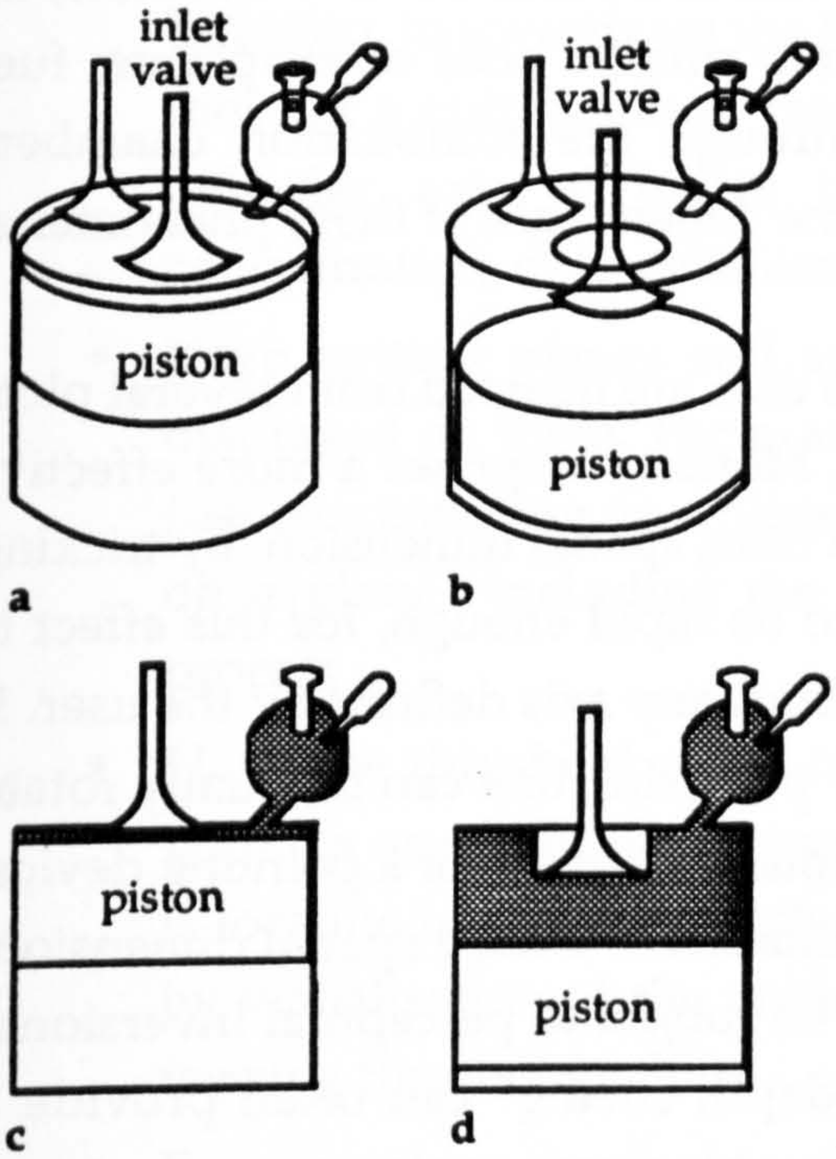


Figure 6.1
Sketch of computational domain in the case of an IDI Diesel engine.
a) Side view at TDC of induction. b) side view at mid-stroke induction c) Elevation view at TDC of induction.
d) Elevation view at mid-stroke induction.
In figures c and d, the computational domain is highlighted in gray.

The raw data from the simulation consist of scalar and vector fields, together with the computational meshes on which they are defined. Fuel droplet data is included when simulation of fuel injection has taken place. Field and droplet data are calculated for a number of user-defined timesteps (of size 1^{o} CA or less). They are output into SPEED's results file at user-specified intervals. For example, the user may have set up a simulation from 180^{o} through to 450^{o} CA ATDC with fuel injection taking place from 350^{o} to 380^{o} CA ATDC with a simulation timestep = 0.5^{o} CA. Also the user may have set up the code to output field data from 180^{o} to 450^{o} CA, for every 10^{o} degrees of crank angle, and

droplet data from 350° to 380° CA - the duration of injection - for every 0.5° CA. The times for which simulation data are output in SPEED's results file comprise the simulation time-scale.

At the visualisation stage the raw data from the SPEED simulation is converted into images depicting the changing geometry (moving piston and valves) and the physics in a manner the engine designer can interpret. Until recently engineers had to produce static images, in the form of VDU plots or hardcopies. The information conveyed by visualisations is dramatically amplified if they are animated in space and time, for the following reasons:

- The geometry, air/fuel flow and fuel injection related data is time-dependent. The piston and valves move in a reciprocating way; vortices are formed by the flow, to disappear later and give place to new ones; fuel is injected into the combustion chamber, and the fuel droplets, as they propagate hit the wall, break into smaller ones or evaporate; fuel ignites and a flame propagates through the combustion chamber. Animated visualisations reveal the time dependence of these phenomena, and enhance their interpretation.
- With static plots the 3D aspects of the data are inferred from several plots showing different views of the data. Motion comprises a more effective means for providing the illusion of a third spatial dimension, by tricking the eye by parallax[2]. Motion needs to be rapid enough, for this effect to take place. The data can be rotated about any axis defined by the user. If the graphics hardware is sufficiently powerful, one can manually rotate the data about an axis with a continuous movement of a pointing device, such as mouse, dial etc. Despite the illusion of a third spatial dimension, the viewer of rotating data may still be subject to perceptual inversions[3] during the rotation, in which case depth cueing[4] can often provide a means of eliminating this effect.
- In several cases, the surface of geometric entities, such as isosurfaces, are complicated, e.g. by being wrinkled. Light sources with the illuminations

---

[2] Parallax: apparent displacement of object, caused by actual change of point of observation.

[3] This phenomenon occurs when one stares long enough at a wire-frame representation of a 3D object, such as a 'Necker' cube or the surface cells of a mesh. Eventually the orientation of the wire frame will appear to invert spontaneously.

[4] Depth cueing: a computer graphics technique to present the depth of an object by the intensity of the image; Parts of objects that are intended to appear farther from the viewer are displayed at lower intensity.

and shadows they create on such surfaces, in combination with rotation, provide an effective means for providing an illusion of a three-dimensional surface.

- The flow field in the combustion chamber typically involves a large range of scales. Camera movements such as zooming in and out or tracking (moving the camera on a predefined track/path), allow the viewer to examine closer small-scale features of the flow.

A set of visualisation tools have been discussed in chapters 4 and 5. However, these tools were applied for a particular time step at a defined location. In a dynamic visualisation environment such tools can be expanded to enhance interpretation in the following ways:

- Move cutting planes through the computational domain for a certain timestep, to investigate the flow in various areas of interest. For example moving a cutting plane from the bowl to the cylinder head along the cylinder axis to locate the position of maximum unburnt fuel concentration, or to visualise the displacement of the swirl centre (if any).
- Keep cutting planes still and have field values for several timesteps displayed on them. For example display for every 0.5° CA the unburnt fuel concentration on a plane including the spray axis, or the temperature on a plane including the cylinder axis to visualise the combustion process.
- Have the threshold value of an isosurface (see section 4.3.3.3) ranging within a user-specified range of values for a certain timestep. An example is location of regions of unburnt fuel after the completion of combustion, by visualisation of the unburnt flow concentrations as an isosurface of varying threshold value.
- For a constant threshold value, see isosurface formation with field data corresponding to a user-specified range of timesteps. One example is visualisation of flame propagation during combustion where the flame is visualised as an isosurface with a constant threshold temperature and the temperature field ranges from ignition to the end of combustion.
- Flow is commonly visualised in the form of velocity vectors. Although this is often adequate, it is less satisfactory for unsteady, time varying flows as those produced by SPEED simulations. A useful alternative is particle advection, where the evolution of a particle system seeded into and moving with the time-varying flow is calculated and visualised with animation. Animated particle advection provides the analyst with better

insight into the flow patterns and conveys where the flow goes, as opposed to what the flow looks like at a particular instant.

- Fuel injection is a time advancing phenomenon, where new droplets of fuel are injected every time step, whereas existing droplets are carried away by their momentum and the flow in the domain, break into smaller droplets and evaporate. It is inadequate and often confusing to observe such a time-dependent phenomenon by plotting snapshots of it. Phenomena such as droplet-droplet or droplet-wall collision get obscured. Animated visualisations of fuel sprays provide a more powerful visualisation tool, providing better insight in the droplet injection, motion, atomization, evaporation and collision processes.

Having justified the need for 3D animation and discussed the limitations of the CAD-based system presented in chapter 5, the next objective is to investigate which alternative software and hardware systems best provide for fulfillment of the above requirements.

6.2 Software systems for dynamic visualisation

At the design stage of the AERAS animation system several alternatives to CAD systems were considered. These can be grouped in:

1. Single application programs, based on graphics libraries;
2. Toolkit environments;
3. Animation systems.

1. *Single application programs, based on graphics libraries.*

This option is expensive to produce and maintain and imposes portability constraints. The development of a system based on OPEN GL[5] would reduce development times; however, portability is at present ensured only between Silicon Graphics and Sun workstations, in the latter case at extra cost. An attractive alternative is expansion of existing dedicated visualisation systems such as PLOT3D [119], or GAS [122].

2. *Toolkit environments*

As said in section 2.3.2, these provide for dynamic visualisation tools, mainly of the type of isosurfaces and moving cutting planes. They are also easy to customise for use with SPEED. However, they do not provide for solid

[5] OPEN GL is a computer graphics standard developed by Silicon Graphics in 1993.

modelling facilities which are very useful for the representation of engine parts such as the injector, the piston, etc. Also their animation capabilities are limited to live video (see section 2.4.1) and moving cutting planes. Finally, when the present research started these could only handle structured meshes.

3. *Animation systems*

These are computer graphics software systems for the production of computer animation. They provide high-quality rendering tools, solid modelling utensils, animation control languages (see section 2.4.2), and facilities for live video and recording on tape, that will be discussed in the following paragraph. However, since they are oriented towards mainly media applications, they do not provide for visualisation tools. There are several such systems in the market, the most popular ones being ALIAS [229], Wavefront [229], and Intelligent Light reviewed in [230], [231], [232] and applied to CFD visualisations at UTRC [233]. Although such systems are machine-dependent and oriented to the media market, they provide an attractive means for the animation of SPEED data sets, because of their powerful rendering and animation features. For these reasons we decided to build the AERAS system for the animation of SPEED results on top of the Intelligent Light animation system. In the remaining part of this chapter a discussion on the hardware aspects of computer animation is followed by the presentation of the AERAS system.

6.3 Hardware-imposed approaches to dynamic visualisation

Dynamic visualisation is a time-consuming operation since it involves CPU-intensive processes such as floating point calculations, transformations in the 3D space, and application of shading algorithms on large data sets. For this reason there is a high performance requirement for the computational, graphical and data storage and transfer system components of the computer. The impact on animation of new technologies, such as graphics accelerators and frame buffer capture boards, live-video boards, graphics-oriented workstations, and high-speed networks, has been highlighted in section 2.4. The objective in this section is to present animation techniques that make use of these new technologies and enable the dynamic visualisation of SPEED results. These techniques include:

- Interactive live video viewing on the workstation;
- Live video with frame-buffer capture boards;
- Distributed visualisation;
- Recording and playback on tape, videodisk or 16-mm film.

Any of the above techniques can be used for the recording of animation sequences on tape or film, but the emphasis in the first three is on controlling and viewing the animation on the workstation screen, whereas the last one is on recording the animation on magnetic or digital media and viewing it by playing back the recording.

The AERAS system is designed for efficient recording on tape but, where the appropriate hardware is available, it can be used for live video (to preview the animation) or as part of a distributed visualisation system (to reduce rendering times).

Interactive video viewing on the workstation

Interactive video is a means of dynamic visualisation, in which the user has interactive control of the viewing position and the animation sequence. Today's technology allows interactive live video only in the case of simple scenes, with low image resolution and simple rendering models applied to small data sets[6]. The most common type of interactive video is one in which the light source and camera are fixed and the object rotates. Another type of live video animation frequently used in volume visualisation is one in which the geometry data, the light source, and the camera are fixed and a cutting plane moves across the data or the opacity of subsets of the data change (see [98] and [102] for examples). Such techniques are standard features of toolkit systems, such as AVS or VISUAL 3.

The interactive video capability is lost if the scenes are complex or the lighting, shading, or viewing requirements are high. In this case,non-interactive live video sequences can be achieved only by employment of frame-buffer capture boards or distributed visualisation systems.

Live video with frame-buffer capture boards

Workstations and simulation systems that approach the level of performance required for interactive video, such as the 'power Iris 4D/240 GTX' or 'Challenge' by Silicon Graphics [147], are not cheap and have been discussed in Chapter 2. In most cases the hardware components that can become available

---

[6] This is because live video normally involves calculation of images of resolution 1024 x 1024 pixels x 24 colour planes, requiring several thousand Gouraud shaded polygons, with hidden surfaces removed, antialiasing, texturing, transparency, and advanced lighting models. These calculations must be done at a rate of more than 10 frames per second. A sufficiently high rate to avoid flicker and provide satisfactory motion for understanding dynamics, is considered to be 15 to 20 frames per second

within research budget limitations are not powerful enough to handle rendering of complicated scenes at a sufficiently high rate. In such cases, sets of images can be calculated, stored in a frame buffer of sufficient memory capacity, and played back in a flip-book fashion. This animation technique, called *flip-book animation, cine sequencing* or *non-interactive live video,* necessarily means loss of interactivity, because the calculation of frames for the animation is a time consuming process, and loss of spatial and colour resolution, because of frame buffer storage limitations.

Since 1989 a number of workstation manufacturers and vendors have offered inboard peripherals that support cine sequencing in a window of the workstation screen. These boards are summarized in table 6.4 at the end of this chapter. The live video boards that are currently commercially available do not provide means for control of a videotape recorder, and present the scene in a small window on the screen (typically of a resolution of 640x512 pixels). Furthermore software interfaces often have to be written for the integration of the boards capabilities into the visualisation application. Frame-buffer capture boards are mainly used by dedicated postprocessing software, written for the requirements of a particular board.

Distributed visualisation

Workstations that approach the level of performance required for interactive video may not be easily available, but most research centres have access to supercomputers that exceed this level of performance. Such computers can be combined with workstations for production of animation in distributed visualisation systems. An increasingly popular approach is to generate images on a supercomputer at a rate of 30 frames per second[7], and transmit these images to the workstation for animation viewing. Such systems have been reviewed in section 2.5.1.

Recording and playback on tape

Systems that provide the performance required for live video are at present very expensive or limited in scene complexity, spatial and colour resolution. Despite the increase in display creation speeds and the availability of frame-buffer capture boards, many scenes are still too complex to view with adequate

---

[7] Images can be transmitted through a high-speed channel capable of a 800 Mbits/s bandwidth, allowing 1024 x 1024 pixels x 24 colour planes x 30 fps, or as a video signal, in which case the workstation must have a device that can digitize a composite video signal in real time (that is 30 fps) such as the Parallax board.

rates of motion on workstations. In such cases, the high-quality rendering of complex images, recording and playback on tape, 16-mm film or videodisc provides an attractive alternative. This alternative is offered mainly by animation software systems, such as Wavefront, ALIAS or Intelligent Light. Such systems, apart from providing tools for high-quality rendering of complex scenes and control of their recording on video, also provide tools for cine-sequencing poorly-rendered images, for previewing of the animation.

Recording onto videotape, film or optical disk provides a method of archiving and distributing high-resolution animation scenes that is inexpensive, efficient in terms of scene complexity and rendering techniques, and readily available. One drawback of this technique is that animation production is a batch process and interaction with the animation is lost. But in the case of engine simulation with SPEED and given the current computer speeds, results are produced in batch mode anyway, and hence visualisations can be rendered and recorded frame-by-frame on videotape or film in batch mode as well. This is the technique chosen for the AERAS system for the animation of SPEED results.

Of all the recording media the videotape is the cheapest, is easily portable and can easily be played back. The problem with using videotape is that getting the graphics images from the workstation screen onto videotape can be a complicated process. This is due to the incompatibilities between the video signals from the high-resolution workstation displays and the video signal standards to which videotape recording equipment conform to. These incompatibilities can be bridged by means of appropriate peripheral equipment.

6.4 Bridging the gap between workstations and video equipment

Getting the graphics images from the workstation screen onto videotape is not a straightforward process, due to the incompatibilities between the video signals from the high-resolution workstation displays and the video signal standards to which videotape recording equipment conform to. Detailed information on video technology can be found in [234] and [235]. In this section the objective is to show how such incompatibilities were bridged by the AERAS animation system.

6.4.1 Incompatibilities

The major incompatibilities between workstation and video equipment are summarised in table 6.1. These are:

Image scanning

In cinema (8-mm, 16-mm or 36-mm) the picture repetition rate is 24 frames per second; due to the high illumination levels used, flicker would still be apparent, so each frame is projected twice for equal lengths of time, doubling the projection rate to 48. The same problem relating to image flicker must be taken into consideration with video display systems.

Unlike cinema projection, the picture in video displays is not generated by the video controller as a simultaneous whole, and hence the technique of double projection to reduce the actual picture frequency rate and still avoid flicker is not possible. Instead the picture is constantly refreshed in what is called an 'interlaced' or 'non-interlaced' way. In television display systems the image is scanned in an interlaced way, in computer monitors in a non-interlaced way. This is a major source of incompatibility between television and computer display systems.

In both interlaced and non-interlaced systems the picture is 'traversed' or 'scanned' in a similar manner to the way in which one reads a book: one line at a time and at the end of each line a new line is started at the left hand side until the bottom of the picture is reached, when a new picture is started. Each traverse is known as a 'field'. The peculiarity of interlaced displays is that all odd-numbered scan lines are displayed in the first traverse, and all even-numbered ones are displayed in the second. Hence it requires two fields to produce a complete picture (a frame) with interlaced displays, as opposed to one field required with non-interlaced ones.

Refresh rate

Another source of incompatibility is in the refresh rate of interlaced video displays and non-interlaced computer displays. In non-interlaced displays the refresh rate is 50 Hz, that means that it takes 1/50th of a second to refresh a frame. In interlaced displays the refresh rate is 25 Hz, and the effect of interlacing is to produce a picture whose effective refresh rate is 50 instead of 25 Hz. This technique works as long as adjacent scan lines display similar information; an image consisting of horizontal lines on alternating scan lines would flicker badly.

Image resolution

For video displays this is 625 lines in the case of the European PAL/SECAM or 525 lines in the cases of the North American NTSC system. The image resolution of computer displays depends on the manufacturer and is usually higher than 1024 lines. As a result there is an important loss of resolution during transfer from computer to video.

Picture aspect-ratio

Another incompatibility arising between workstation and video display systems is that of picture aspect-ratio, that is picture width / picture height. Standard workstation displays have an aspect ratio of 5:4, whereas their video counterparts have 4:3. The aspect ratio of the computer generated image must be maintained when converted for video output, in order to avoid image distortion.

Type of signal

Another source of incompatibility is the various types of video signals that workstation and video equipment conform to. The video signal that is output from the video controller can be in one of four forms, RGB, composite video, component video and Y/C. In the case of RGB (red, green, blue) signal, separate cables carry the red, green and blue signals to control the three electron guns of a CRT, and another cable carries the synchronization to signal the start of vertical and horizontal retrace. The standards used for voltages, wave shapes, and synchronisation timings in the case of the RGB signal are presented in detail in [234]. Frequently, the synchronisation timings are included on the same cable as the green signal, in which case the signals are called 'composite video signals'. Typically workstation display systems output component or RGB video signals

PAL video is the signal format used for video monitors and videotape equipment in the U.K. and the majority of European countries. Colour, intensity, and synchronisation information is combined into a signal with a bandwidth of about 5MHz, broadcast as 625 scan lines, in two fields of 312.5 lines each. Other video signals are SECAM with 625 lines, NTSC with 525 lines, and the emerging HDTV with 1000 lines. For the purpose of the present research the PAL format which is the broadcasting standard in operation in the United Kingdom, has been utilised.

The types of video signals which are output by a workstation, a camera, or an encoder and need to be matched with the type of signal required by the video format of the VTR are summarised in table 6.2.

6.4.2 Bridging the incompatibilities and recording onto a VTR

'Inboard' – that is loaded on the backplane of the computer) or 'outboard' (that is external to the computer – peripherals can be used for conversion of computer generated animated sequences to video compatible ones. A functional diagram of the conversion process used in the case of AERAS is shown in figure 6.2.

Pictures for the animation are rendered, and frames are stored, on the *frame buffer* of the workstation. The frame buffer is either part of the system's memory or is an inboard or outboard peripheral of the workstation.

The composite video signal output from the frame buffer is converted to a PAL signal for input to the VTR, by means of appropriate equipment called *PAL encoder*. It is difficult to maintain the signal quality exhibited in RGB or component systems, once these are encoded into a composite format. In order to convey all of the video information for the three primaries (red green blue), the PAL encoder calculates the amount of light or 'luminance' information from the RGB information using a weighted average technique. Then combines the colour information, or 'chrominance', with the luminance by using modulation techniques. Finally includes horizontal and vertical synchronisation signals that are used by the destination for the syncronisation of the composite video signal.

The Composite PAL video signal output from the PAL encoder, is directed to a device called the *video animation controller* that for each frame rewinds the tape on the *video tape recorder* to the appropriate position and records a frame on it.

A workstation monitor and keyboard are used for user interface with the whole process, and a video monitor is used for viewing the contents of the videotape.

6.4.2.1 Choosing the recording format

Care must be taken when choosing the format of the video tape, since a variety of formats, each offering relative advantages, are commercially available. The criteria the selected format must satisfy are the following:

- High picture quality (high output signal).
- Minimum 'dropout'. Dropout is a loss of picture signal resulting in a flash across the screen. What is called a floating dropout, causes momentary loss of signal during record only. A particle of dust or a loose oxide can be responsible for this. For this reason precautions of reducing dust and smoking near recording equipment and tape stocks should be taken.

- High copyability. The quality of the picture signal should not be affected by copying from one recorder to another.
- Editing facilities, such as vision and sound editing, pre-roll for editing, search control, and digital counter.
- Low price. Also ease of availability of playback facilities for presentation purposes.
- Low abrasion (long life) of the tape stock.
- Long-term storage stability.

The most popular recording formats are U-matic (low band, high band and high band SP), VHS and SVHS. The main features of the different recording formats are summarized in table 6.3. We decided to use the U-matic SP format with the AERAS system, because it is the best format to date and it is established as the standard in the broadcasting industry.

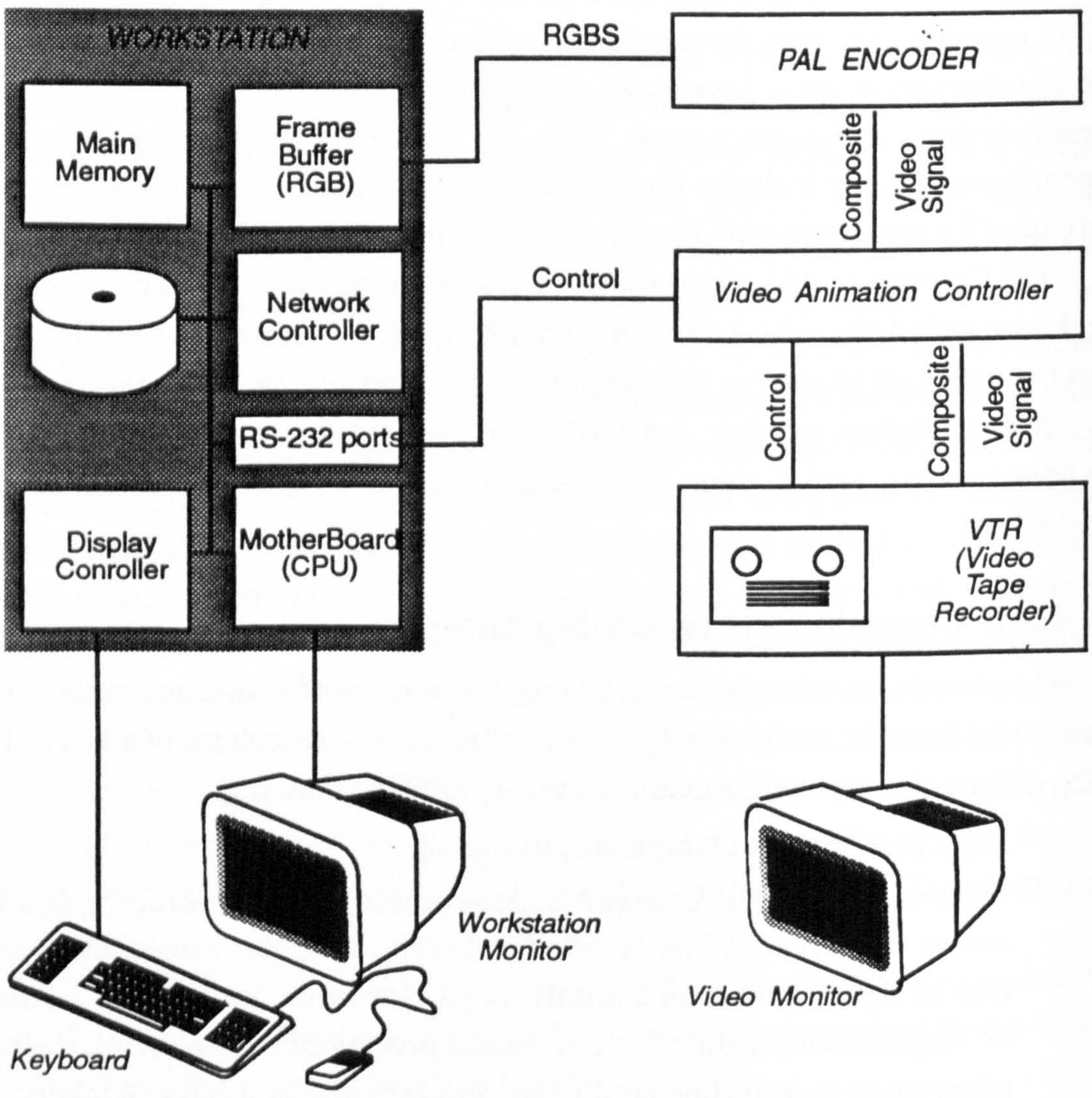


Figure 6.2 Functional diagram of the encoding and the VTR control process.

6.5 An animation system for SPEED

One of the leading computer animation companies, Intelligent Light, agreed to make available to us their animation software system, called Intelligent Light 3DV or IL for short, in order to investigate its suitability for visualisation of SPEED results. As an animation system IL is an attractive option because:

- It provides an environment for the description and animation, of 3D computer models;
- It provides high-quality rendering algorithms;
- It provides an animation control language and software facilities for VTR control;
- It provides an interactive, event driven, Graphical User Interface;
- It can handle large data sets with considerable efficiency;
- It is suitable for the generation of presentation animations;
- The only hardware expansion to our existing computing system that IL requires is a frame buffer capture board.

The objective in this section is to present the software design and implementation of the AERAS (Animation Environment for Results Assessment with SPEED) that has been developed by the author at Imperial College. This is built around the IL suite of codes and utilises the animation tools provided by IL in a way that serves the visualisation needs in SPEED. These tools are summarised in section 6.5.1.

The software design and implementation of AERAS is presented in section 6.5.2. This is done in the stages suggested by Freeman and Wasserman [236]. These stages are:

- *Design goals* , where the requirements for the design are specified;
- *System architecture,* where the architecture of the AERAS system is introduced, the software components it comprises are named, and the function of each component is demonstrated;
- *Detailed design*, where methodology, algorithms, and visualisation tools employed in each software component of the AERAS system are presented in detail;
- *Implementation*, of the above in a software and hardware system.

6.5.1 Description of IL

IL is an animation software system, that provides means for scripting animation *scenes*, previewing them on the computer screen, and recording them onto an output device. The elements of a scene are visualisation data, light

sources, and the camera. Animation with IL is the process of giving motion to data, and specifying changes in lighting or camera position.

IL requires an object-oriented organization of the visualisation data. Data is organized in hierarchical data structures called *objects;* objects reference other objects and *atoms,* the simplest type of drawable data structures, as elements in a tree-structure organization. Atoms are data sets of points or vertices and connectivity lists of points to polygons and can contain colour and face normal information for each point and/or polygon. Each atom is stored as a file that can be edited or viewed in the IL system. Objects are higher level data structures that reference atoms or other objects. The referenced data may have geometric transformation actions applied, as well as attributive actions applied, such as colour, specularity or transparency. Each object is stored as a file that can be edited with a text editor or viewed in the IL system .

Overview of the animation process and software architecture

There are four steps in producing computer animation with IL: *modelling* the objects to appear in the animation; creating an *animation script* to control the motion of the objects; *rendering* the animation frame by frame; and *recording* the animation on film or video. (see figure 6.3)

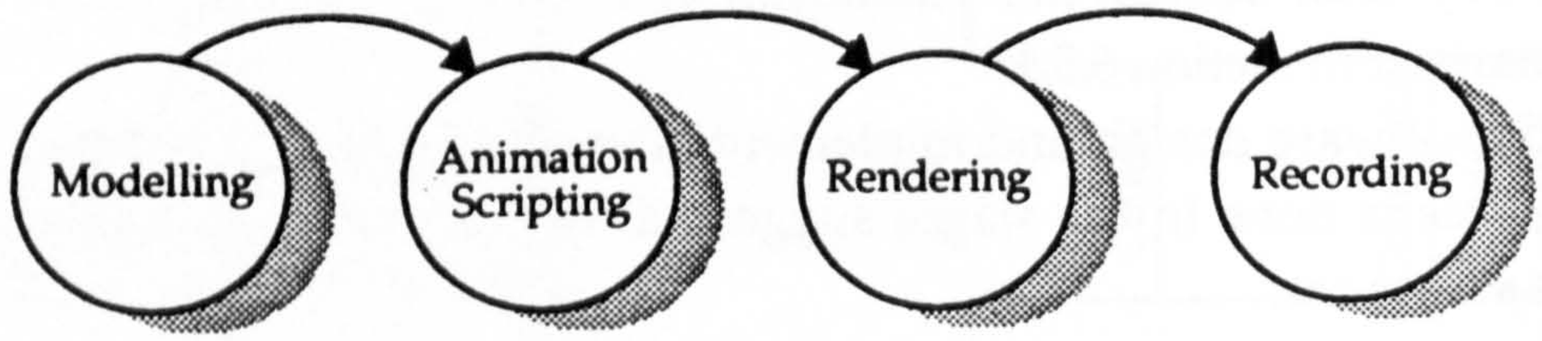


Figure 6.3 The stages of the animation process, with IL

Each step is performed by a separate software unit in what is a modular environment for animation production. Each module has its own user interface, and a control module, named *LAUNCHER,* is used for calling each module. The main modules are: a solid modeller which they have chosen to name the Friendly Editor, abbreviated *FRED;* a viewing and rendering module named *GVIEW;* and a playback or recording module named *LAYOFF* (see figure 6.4).

The modelling task is accomplished in FRED. This is a powerful and user-friendly tool of the analytic solid modelling type (see section 4.1) and is used for the definition or modification of atoms and objects with interactive graphical methods. The screen layout of the user interface of FRED can be seen in figure

6.5.a. Atoms and objects are stored in text files, that can be further modified by means of the text editor.

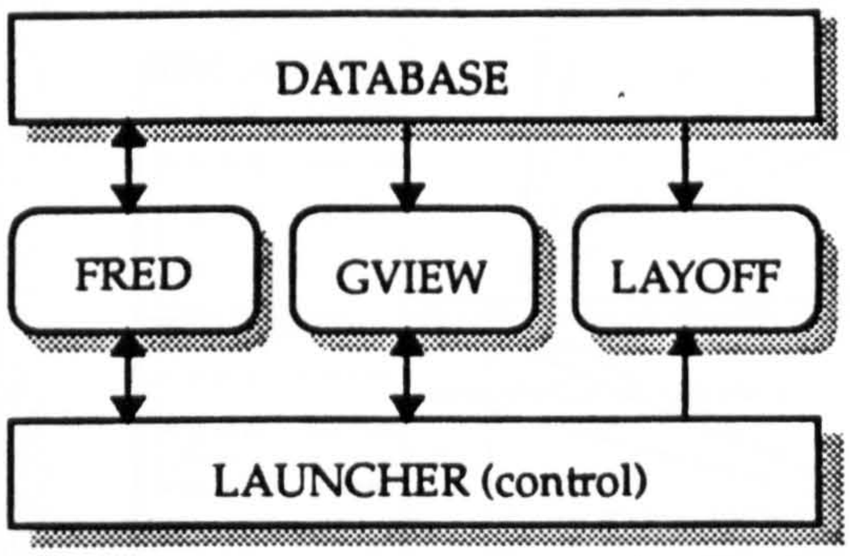


Figure 6.4

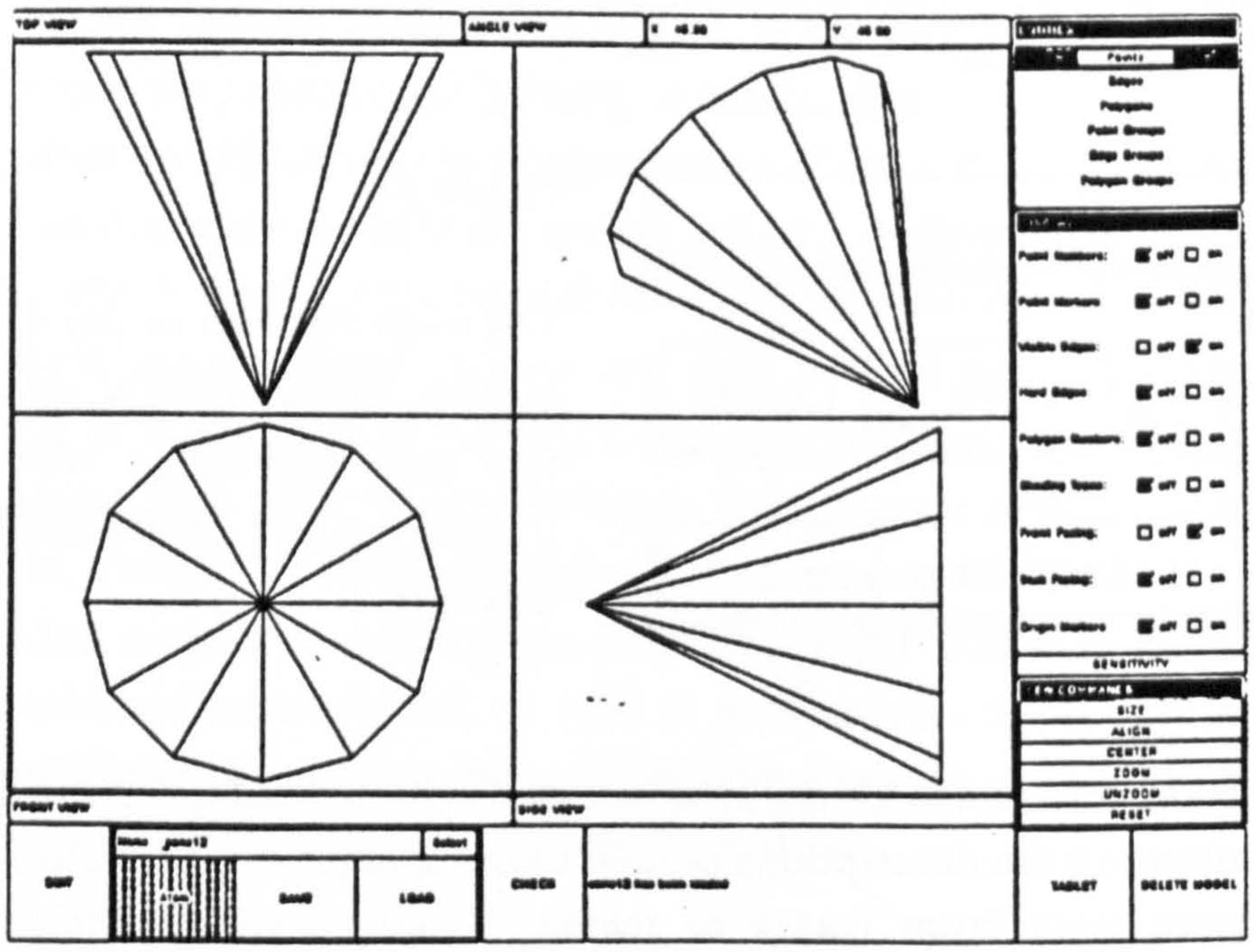
The main modules of the IL animation system, FRED, GVIEW, and LAYOFF and their interaction with the user control module LAUNCHER and the database.

Animation scenes can be defined interactively or through text files, called *scripts*, containing the descriptions of how camera, objects, object attributes, and light sources vary from frame to frame. IL automatically calculates the inbetween values of the motion of objects, so that these move smoothly when the entire sequence is viewed, and several inbetweening functions are provided. An animation language of the linear-list notation type (see section 2.4.2) is provided for animation scripting. Each event in the animation is described by a starting and ending point and an event that is to take place. The animation can be of three types:

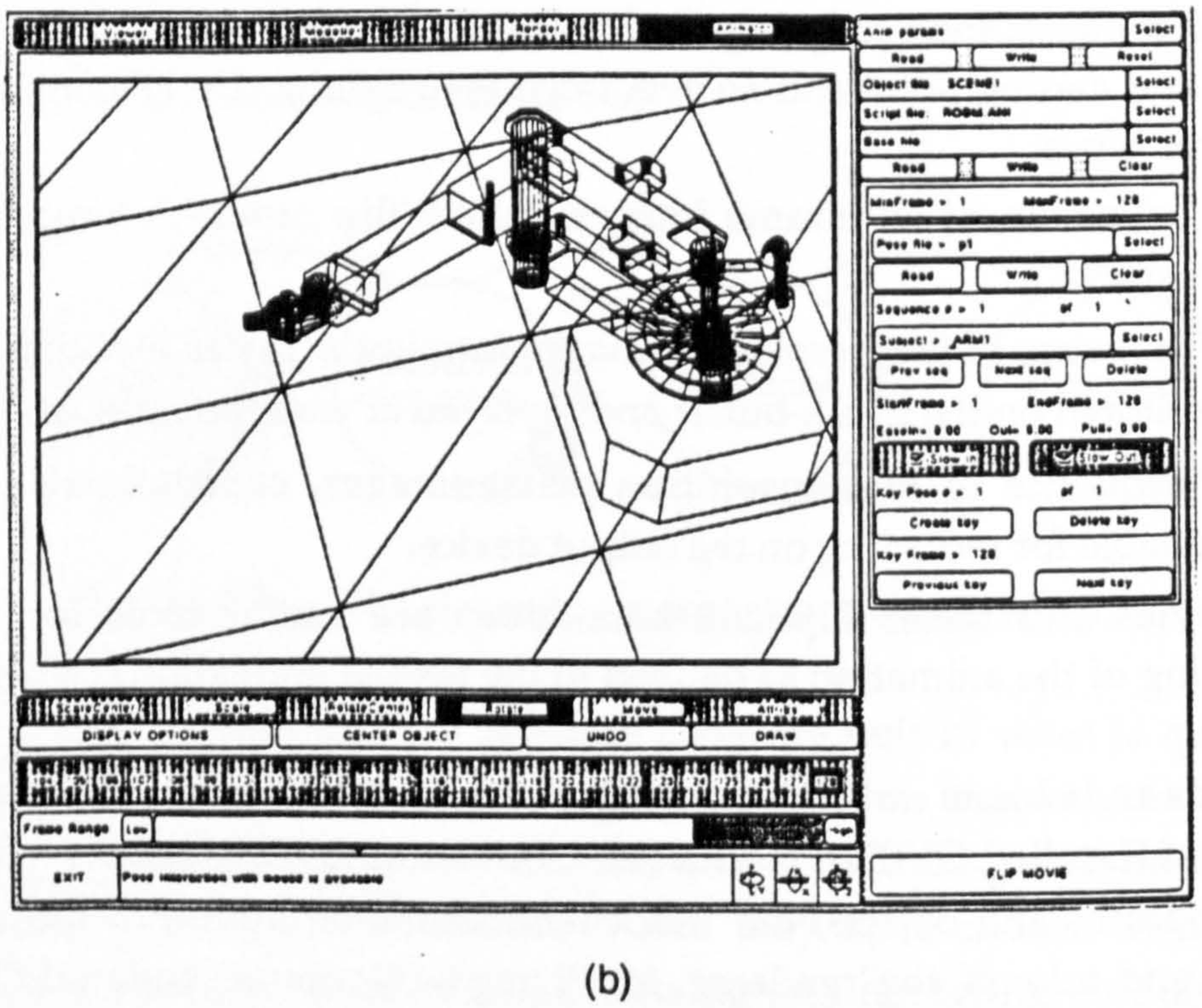
1. a collection of wireframe images, called 'flip movie', stored in virtual memory;
2. a sequence of low-resolution images forming a 'raster flipbook' that can be stored on the frame-buffer and replayed at variable rates; and
3. a sequence of high-resolution raster images, called 'finals', that are suitable for recording on the output device.

Flip movies and raster flipbook animations are useful tools for real-time previewing of the animation as defined in the textual animation scripts, prior to recording.

Atoms and objects can be displayed on the screen by means of the rendering module of IL called GVIEW. Useful sub-modules of GVIEW are the 'viewer', for viewing atoms and objects, the 'modeller', for the definition of light sources, textures and colours, the 'renderer', for Phong or Gouraud rendered finals, and the 'animator' for frame-by-frame production of flip movies or raster flipbook animations (see figure 6.6). The user interface of GVIEW is illustrated in figure 6.5.b; in the figure the animator sub-module is activated for the flip-movie animation of a robot arm.

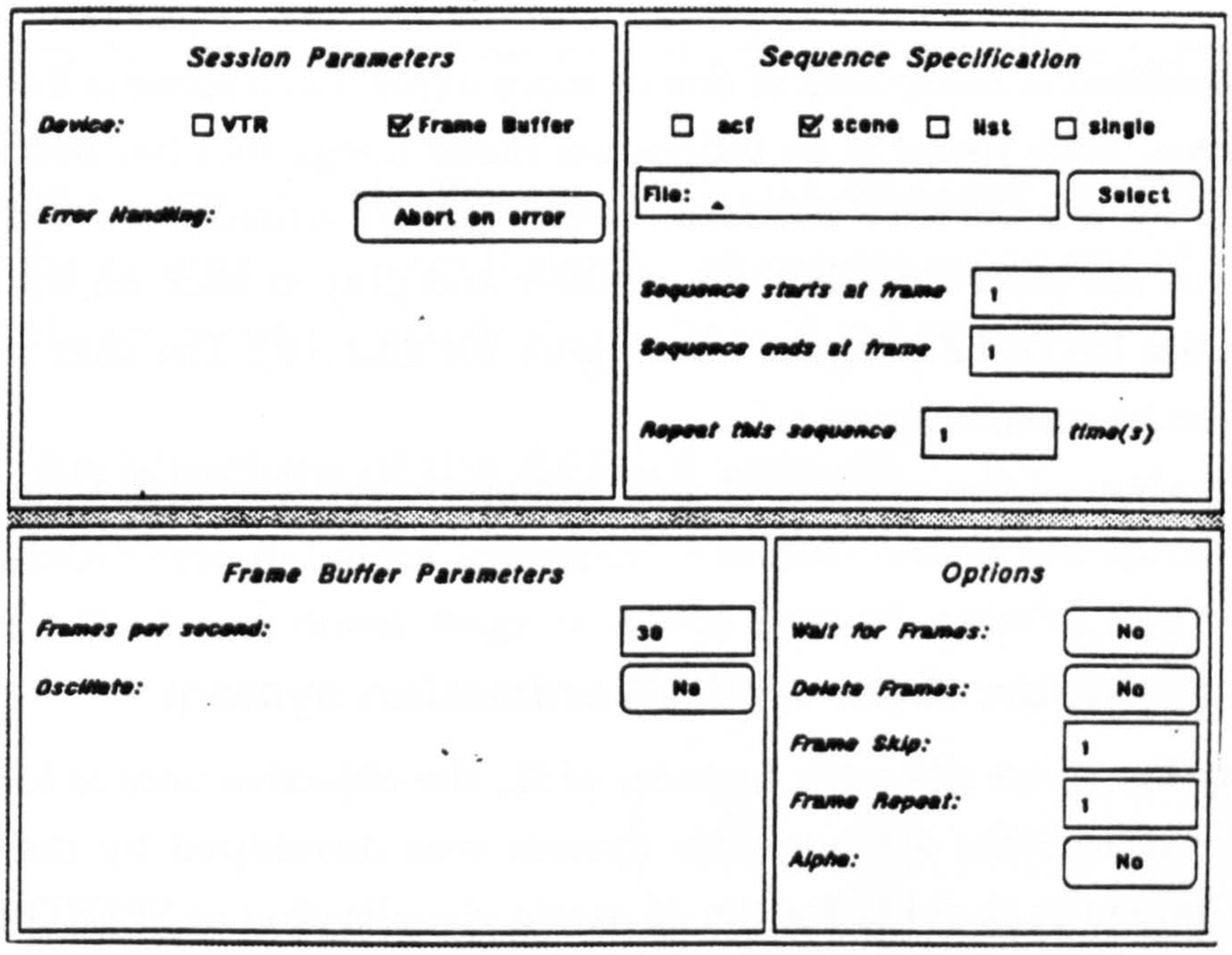


(a)



(b)

Figure 6.5 User interface of the intelligent light animation system.
(a) Solid modeller FRED (b) renderer GVIEW



(c)

Figure 6.5 (continued) User interface of the intelligent light animation system.
(c) Animation recording module LAYOFF

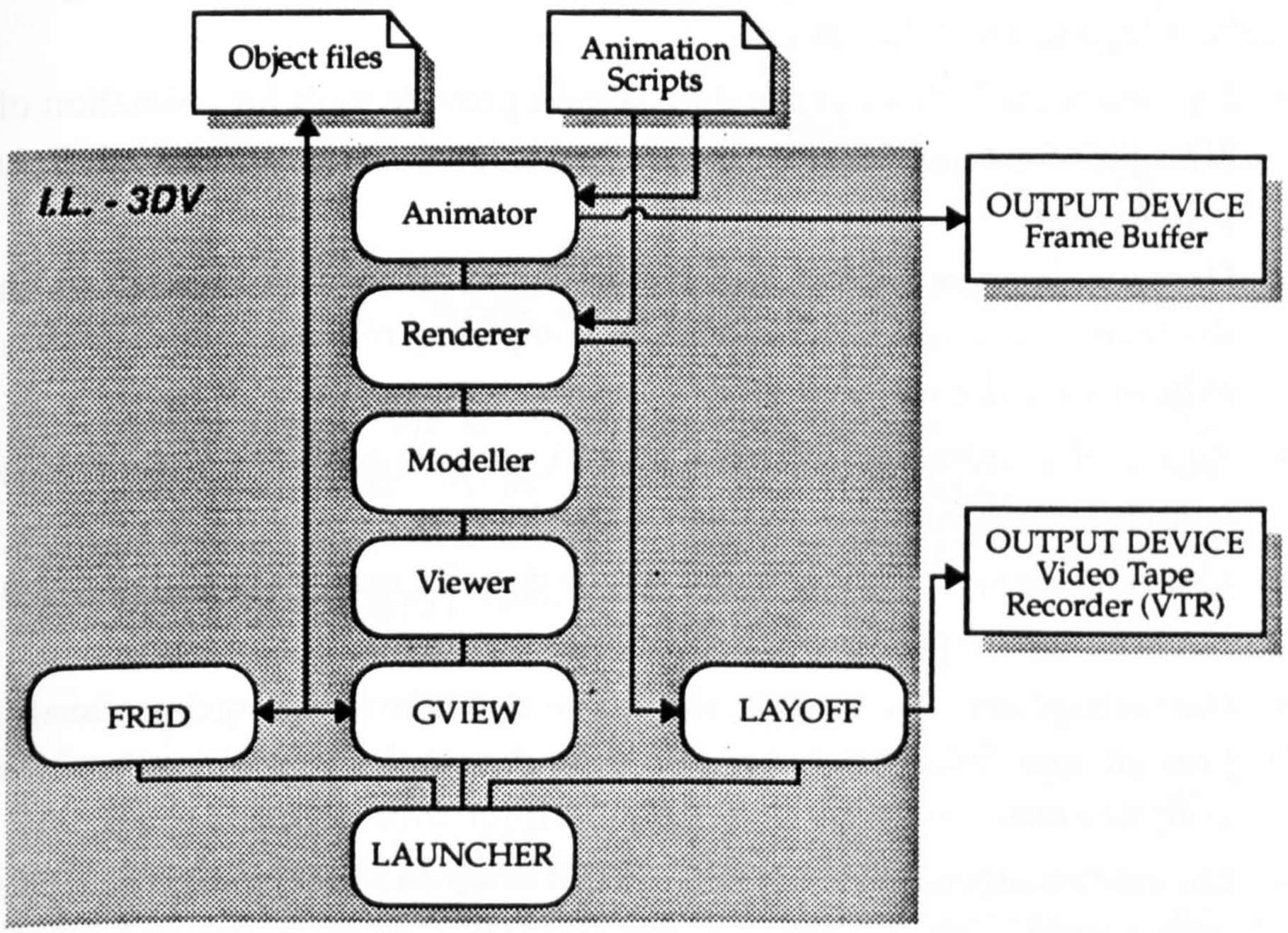


Figure 6.6 Overview of the modelling and animation processes with IL

The animation is composed of one or more *scenes*. Each scene is a sequence of related *frames*. Each frame is an individual raster image that has been rendered and stored by the GVIEW module. Frames are recorded onto videotape or 16mm file or are stored on the frame-buffer and played back on the screen by the use of the LAYOFF program (see figure 6.4 and 6.6). The user interface of LAYOFF can be seen in figure 6.6.c.

An overview of the modelling and animation process with IL is shown in figure 6.6.

6.5.2 Description of the AERAS animation system

Having described the main features of IL, the objective here is to introduce the AERAS animation system. This system was developed by the author at Imperial College to provide for the dynamic visualisation of SPEED results. Its capabilities are demonstrated in the example of Chapter 7 and in the video that is submitted with this thesis.

6.5.2.1 Design goals.

The design of AERAS was drawn upon the dynamic visualisation requirements for visualisation of SPEED data sets, that have been discussed earlier, and the tools provided by the IL animation system. The following list gives the design goals of the system:

- *Dynamic visualisation*. The system should provide tools for animation of 3D computer models of engine parts and visualisation of flow phenomena.
- *Unstructured geometry handling*. No limitations should be imposed as far the types of meshes and the shape or complexity of the engine geometry to be modelled is concerned.
- *Advanced graphics*. The system should provide tools for high-quality rendering, inbetweening, text handling, etc.
- *User interface* with the system should be done by means of an interactive, event-driven graphical user interface.
- *User interaction*. The system should be suitable for the generation of presentation animations, as well as be a useful visualisation tool for everyday use.
- *Database management*. The system should be able to handle large data sets with considerable efficiency. It is envisaged that animations of data – such as velocity vectors or spray droplets – that are defined over a

computational mesh of one million grid points, will be common in the future.

- *Hardware requirements*. The system should efficiently use the capabilities of the existing hardware.It should not require expansion of the existing hardware or such demands should be kept to a minimum.

6.5.2.2 Architecture of the AERAS system.

The AERAS system for the animation of SPEED results is a set of codes built around the IL. A functional diagram of the system's architecture is shown in figure 6.7.

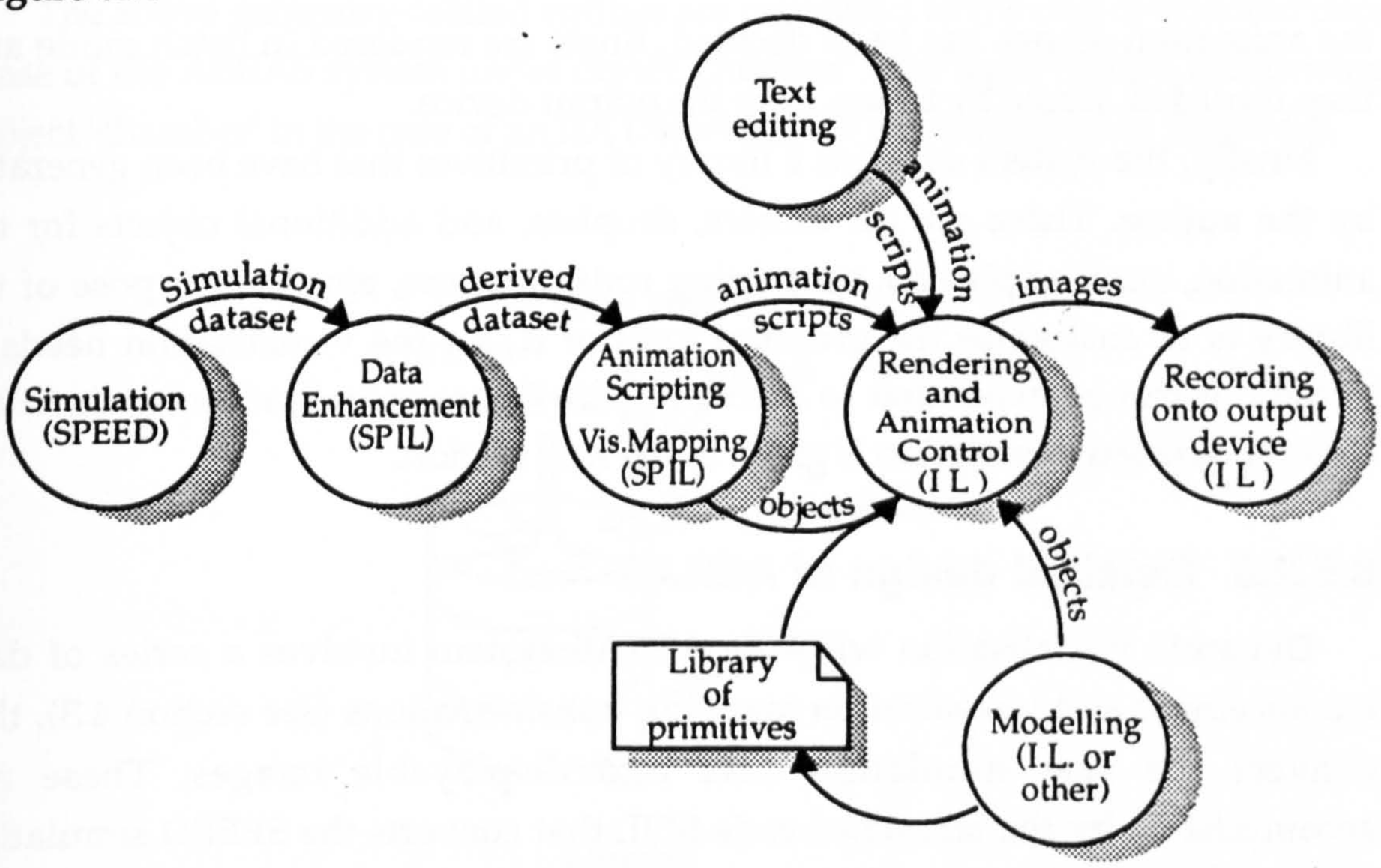


Figure 6.7 Functional diagram of the architecture of AERAS.

The simulation data set is produced by SPEED. It is not dynamically visualised as it is produced - see tracking, steering, and postprocessing in section 2.5 - but instead is stored in SPEED-format data sets for later-time visualisation. Thus a postprocessing approach is adopted.

A translation code, named SPIL (SPeed Intelligent Light translator), was developed to convert the raw simulation data into IL object - actors - for the animation. This operation enhances the raw data with operations such as time and space interpolation, application of filtering or smoothing operations, or derivation of new field quantities. SPIL subsequently transforms the output from the data enhancement stage into a geometric description consisting of objects that can be fed into the IL system for rendering and display. An example

of this is the definition of a cutting plane or the extraction of an isosurface from a 3D field. A detailed description of SPIL is found in the the following section.

Animation scripts containing descriptions of the motion of objects and specifications of changes in lighting and camera position are, when possible, generated by the translator code SPIL. Otherwise, they can be created by means of a text editor.

Images for the animation are produced by the rendering and animation control component of the system, provided by IL. Since rendering finals is inherently time-consuming, animation scenes defined by objects and animation scripts are previewed in real-time in wire-frame mode. Once the final form of the animation scenes has been decided, finals are rendered in batch mode and then recorded, frame by frame, onto the output device.

Finally, the system contains a library of primitives that have been generated by the author. These are 3D arrows, droplets, and additional objects for the animation, such as pistons, connecting rods, injectors, etc. The purpose of the library is to customise the graphics tools of IL for the visualisation needs of SPEED. It can be expanded to accommodate future visualisation needs in the code. A detailed description is given in the next section.

6.5.2.3 Detailed design of AERAS

Dynamic visualisation with the AERAS system involves a series of data enhancement and visualisation mapping transformations (see section 4.3), that convert the raw simulation data into displayable images. These are accomplished by the translator code SPIL that converts the SPEED simulation data set to IL displayable objects – actors – for the animation.

The main operations of SPIL and AERAS together with additional operations such as animation previewing and generation of titles are discussed in detail in the remaining part of this section:

6.5.2.3.1 Creation of geometry

The translator code SPIL extracts mesh information from the SPEED database, and creates:

- A set of vertices on which field values will be interpolated.
- A surface model of the combustion chamber. This consists of the cylinder-head, the valve-heads, the liner and the piston crown

The surface model is stored in the form of IL objects.

Objects that are not included in the SPEED data set but increase the comprehension of the animation scene can be selected from the AERAS primitives library. In its present form this contains the following primitives:

'piston';

'liner';

'glow plug';

'injector';

'arrow' and

'droplet'.

The above geometry-related entities are organised in the object-oriented data base of the AERAS system under object 'chamber'. The basic objects comprising object 'chamber' in the case of an IDI Diesel engine are displayed in figure 6.8.

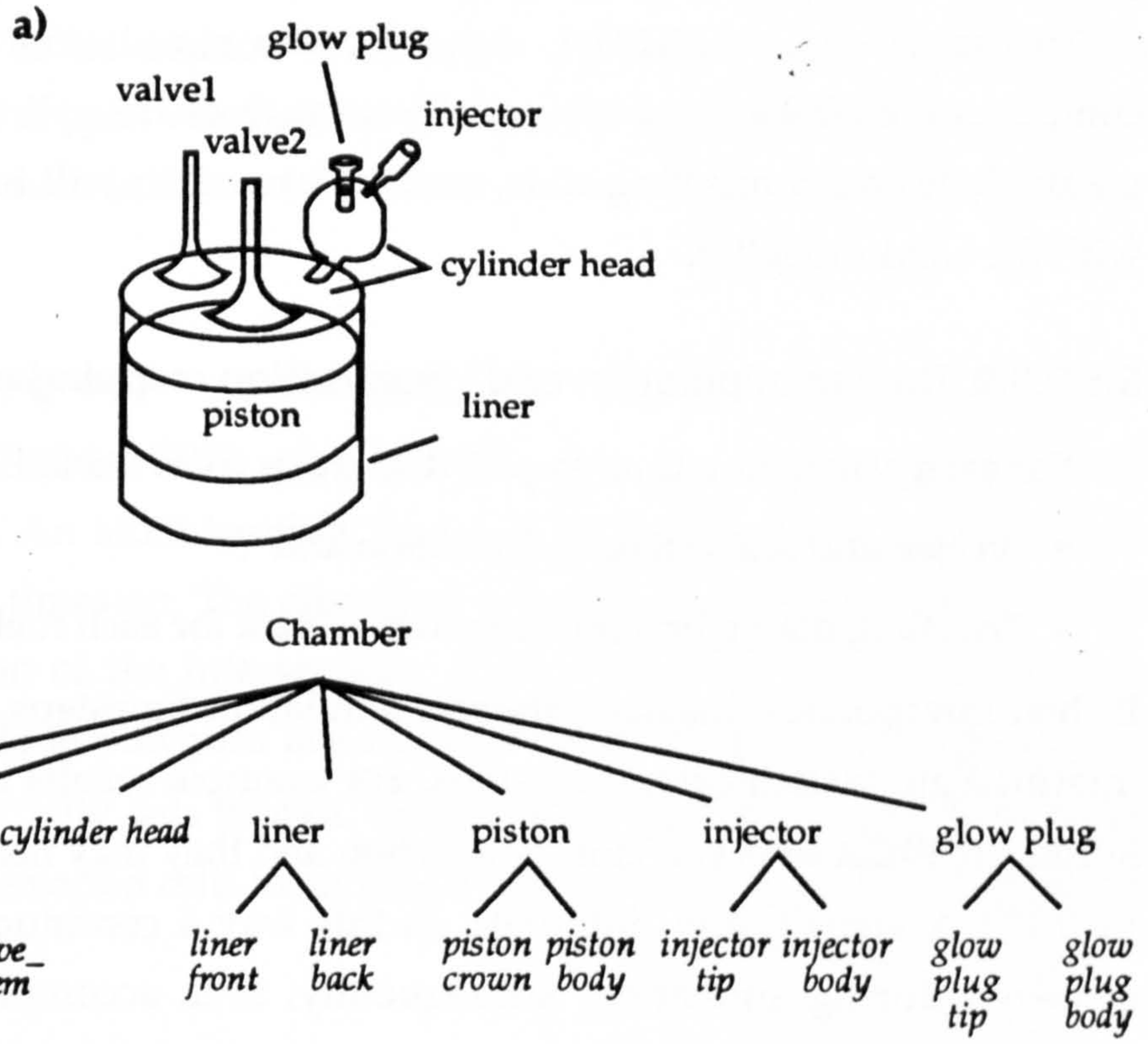


Figure 6.8 **a)** Object 'chamber' and the elements - objects - it is constructed with. **b)** The organisation of those elements into the hierarchical structure of object 'chamber'. Objects are in plain, whereas atoms in italic typeface.

Object 'chamber' has a hierarchical structure comprising other objects and atoms, with atoms at the bottom of the hierarchy. It is constructed from objects 'valve1' and 'valve2', 'liner', 'piston', 'injector', and 'glow plug' and atoms 'cylinder head', 'valve base 1', 'valve base 2', 'valve stem', 'liner front', 'liner back', 'piston crown', 'piston body', 'injector tip', 'injector body', 'glow plug tip', 'glow plug body'. Atoms 'valve base 1', 'valve base 2', 'liner front', 'liner back', 'piston crown', 'injector tip', and 'glow plug tip' are generated by the translator program from the SPEED data set. Atoms 'valve stem', 'piston body', 'injector body' and 'glow plug body', are generated by the solid modeller. In figure 6.8.b, the hierarchical structure of object 'chamber' is displayed where objects are represented by plain typeface whereas atoms are represented by italic typeface. The rendered object 'chamber' in the case of a Renault IDI Diesel engine with glow plug is illustrated in figure 6.9.

In cases where SPEED is used for the simulation of more than one combustion chambers, additional objects such as the cylinder-head block, the crank shaft and connecting rods, and the cam shaft will have to be generated with the solid modeller.

6.5.2.3.2 Data manipulation and visualisation mapping operations

For each calculation time step SPIL extracts from the SPEED data set:

- vector and scalar field information and
- location, diameter, and temperature data for each fuel droplet.

It then interpolates linearly between simulation timesteps to produce data for intermediate visualisation timesteps. For example results may be output from SPEED at 1° CA intervals during injection and they may need to be interpolated to 0.1° CA visualisation intervals. In this way a continuous motion effect is achieved during animation. Subsequently, SPIL accomplishes the following visualisation operations:

Creation of droplet data

SPIL generates a visualisation data set of droplets, where each droplet is represented by an atom, and an object 'spray' comprising the set of droplets at the end of a timestep, is generated for each timestep. Each droplet atom references a primitive sphere scaled according to its diameter and coloured according to its temperature or diameter. The primitive sphere (see figure 6.10.a)

used for the generation of atom droplets has been generated using the solid modeller and is stored in the AERAS primitives library.

SPIL is unable to calculate inbetween droplet locations between timesteps. This is because the droplets in the SPEED data set, although calculated by adopting a Langrangian approach, are stored as a set of droplets for each timestep, in a way that each droplet does not remember the droplet it originated from[8]. Moreover, due to droplet break-up,each droplet may correspond to more than one droplet in the subsequent timestep. Thus, SPIL does not know which droplet in the previous timestep to select as the parent droplet of one in the current timestep and so cannot activate its inbetweening algorithms without distortion of the information stored in the SPEED data set. To ensure smooth animation of sprays during visualisation with AERAS, droplet data have to be stored for timesteps smaller than 0.5 degrees of crank angle. The algorithm implemented in SPIL for creation of the object 'spray' is provided in appendix A.7. An example of droplet visualisation in the prechamber of the engine shown in figure 6.9 can be seen in figure 6.11. and in the video tape submitted with the present thesis.

Definition of cutting planes

The translator code SPIL can handle arbitrary slicing through the unstructured data. An atom 'cutting plane', containing cutting plane data, is generated for each timestep. The algorithm suggested by Gelberg [186] is used for the computation of the intersections of cells with the cutting plane. The resulting cutting plane polygons are stored in atom 'cutting plane' as a list of vertices and connectivity lists thereof. Scalar and vector values are interpolated from vertices of intersected cells to vertices of the corresponding polygons of the cutting plane.

Creation of contour and vector plots on cutting planes

Continuous tone fringe plots of scalar fields are generated by assigning a colour value, corresponding to the value of the scalar, to each vertex on a cutting plane.

---

8 This problem could be overcome by increasing the database size of SPEED.

In order to generate vector plots of the flow field, SPIL creates an object called 'flow' comprising atom arrows that are located at the centroid of each cell, sized according to the magnitude of the flow at that location, and coloured according to either the flow magnitude or the value of a scalar (e.g. pressure) at that location. Arrows are based on 3D primitives stored in the AERAS primitives library. Such a primitive can be seen in figure 6.10.b. The algorithm implemented in SPIL for the calculation of object 'flow' containing velocity vectors on a cutting plane through the data is given in appendix A.8. An example vector plot on a cutting plane through the axis of the cylinder in the combustion chamber of figure 6.9 is shown in figure 6.12 and in the video tape submitted with this thesis.

Creation of isosurface data

SPIL generates isosurface information for a user-specified threshold scalar value. An atom 'isosurface' is generated for each timestep, comprising a set of vertices and connectivity lists of vertices. The 'marching cube' algorithm suggested by Gallagher and Nagtegaal [225] is used for their production.

The production of multiple isosurfaces for a range of threshold values is possible. In this case different colour and opacity levels can be assigned to the isosurface polygons of an isosurface, in order to facilitate comprehension of the visualisation and allow visibility of hidden isosurfaces.

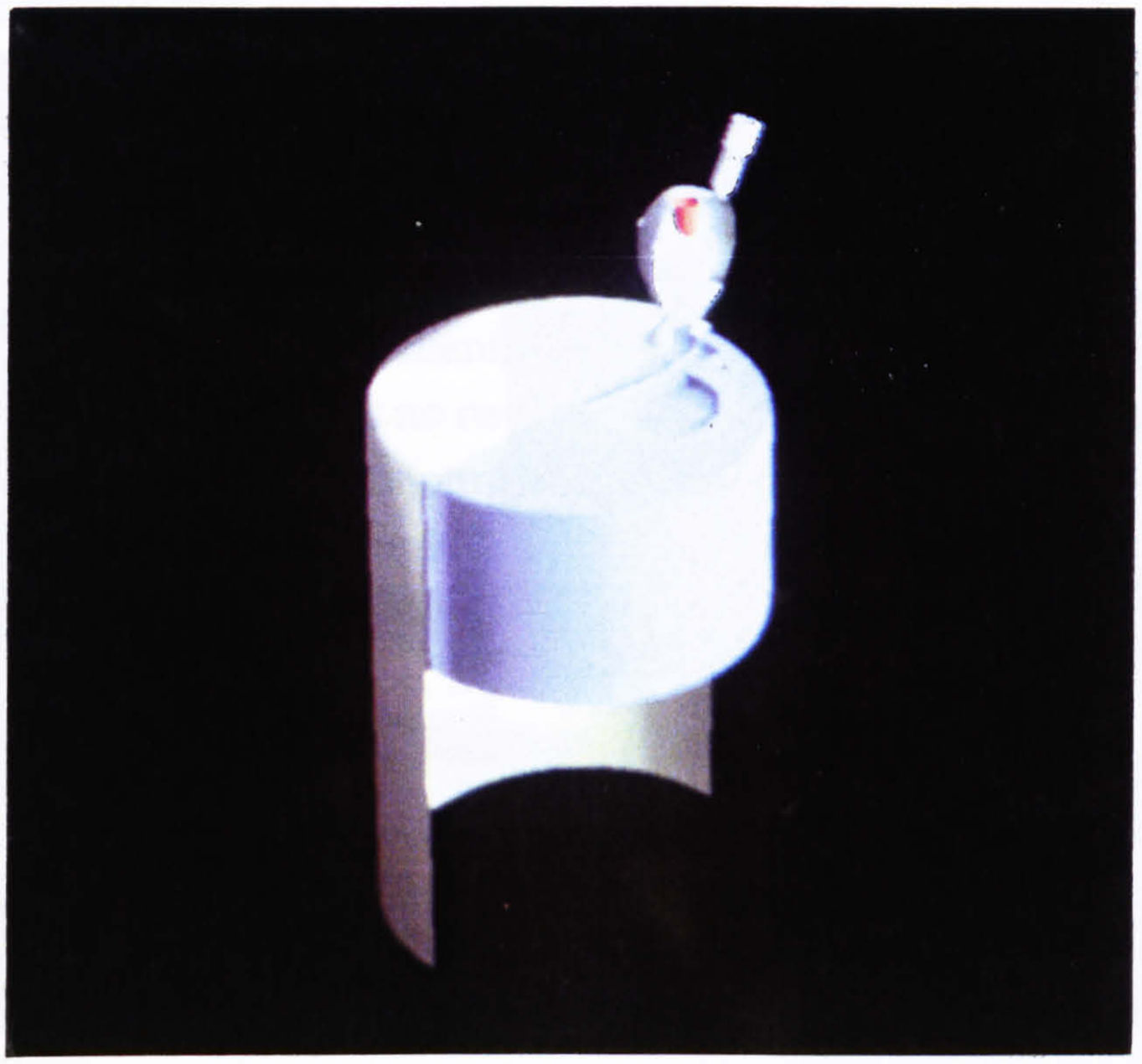
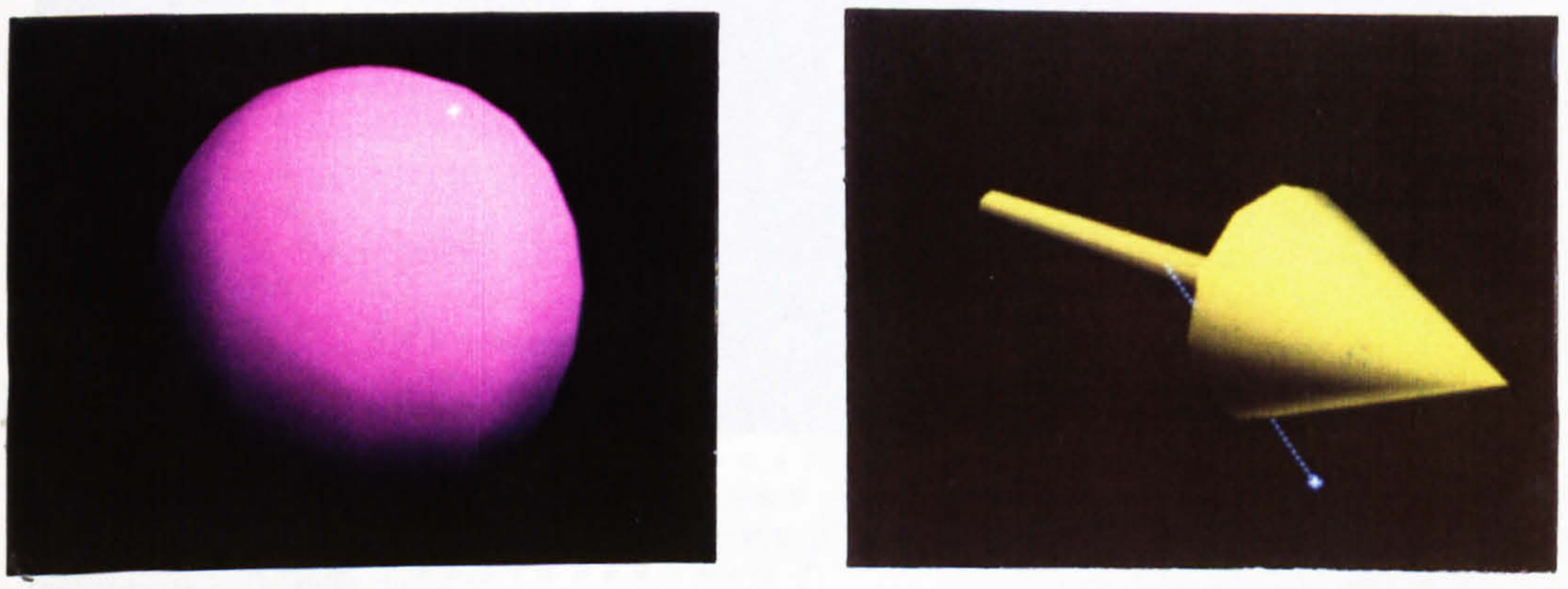


Figure 6.9 AERAS-generated geometry for the combustion chamber of an IDI Diesel engine with glow plug, provided by Renault. The injector-tip is shown in white, the glow plug in red and the piston in white colour. Only half the liner and cylinder head is shown.



(a) **(b)**

Figure 6.10 (a) Primitive sphere (b) primitive arrow.

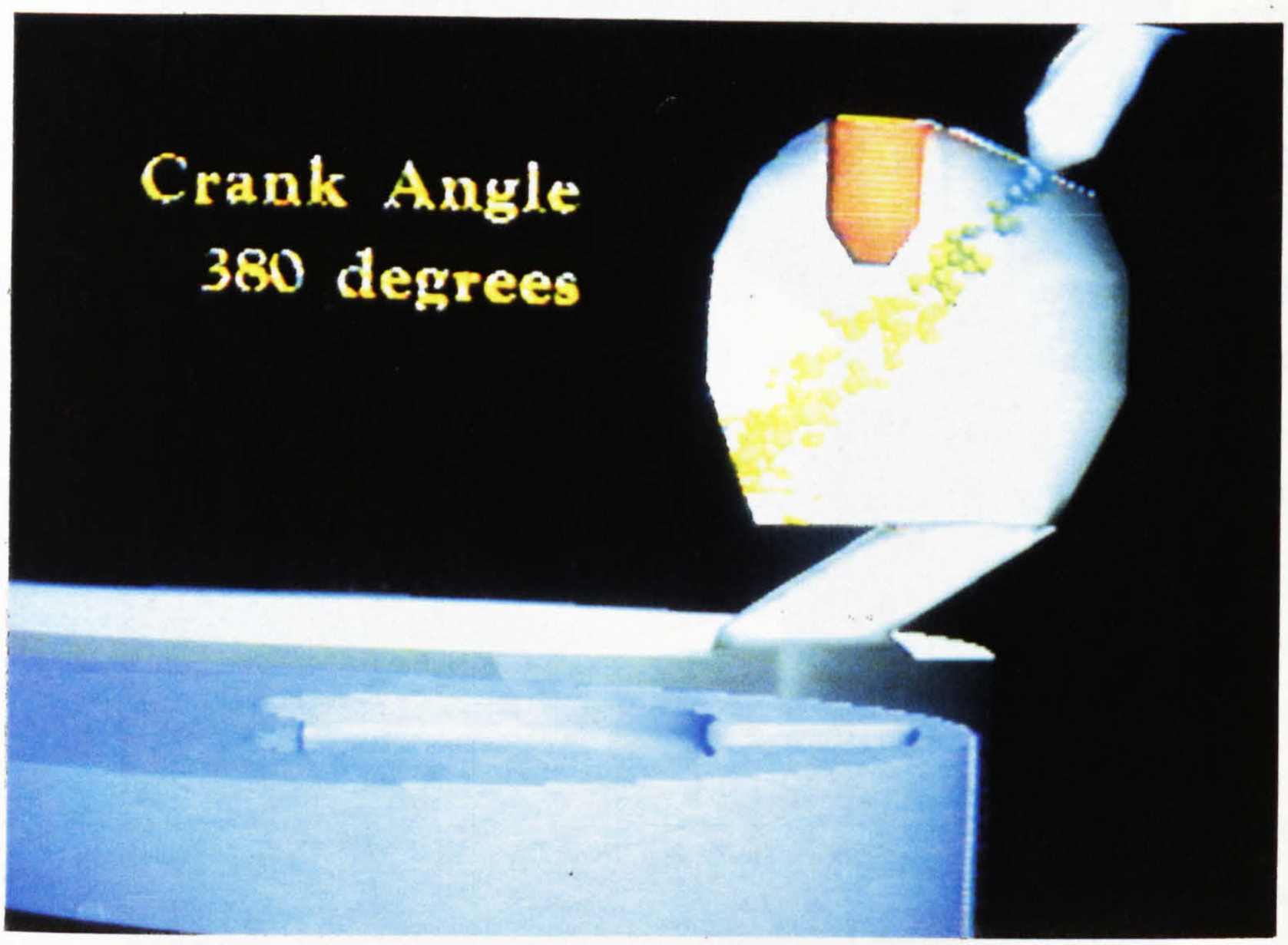


Figure 6.11 Renault IDI Diesel engine. Visualisation of fuel spray in the prechamber with AERAS.

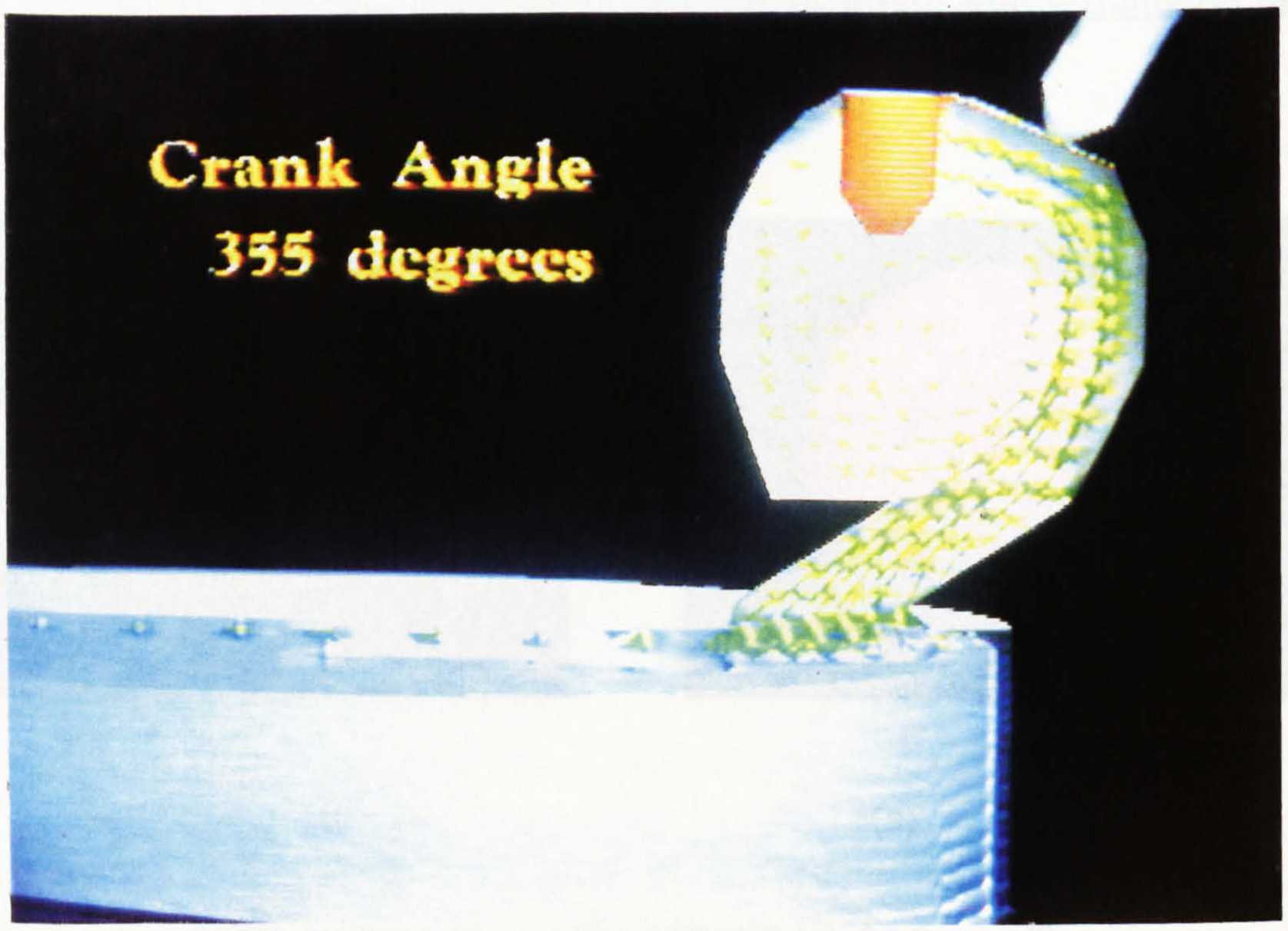


Figure 6.12 Renault IDI Diesel engine. Visualisation of flow field with AERAS.

Creation of particle trace data.

An efficient algorithm, not only from a speed of calculation but also from a data storage point of view, has been designed for use in the translator code SPIL. The algorithm models the evolution of a particle system seeded in the transient flow field within the combustion chamber, as it is modelled by SPEED.

Each particle is a flow element, and hence, although is is computationally identical to a droplet, there is no need for gas-particle interaction. The algorithm is designed for accuracy, economy and is timestep- and mesh-accurate. Shear flows and near boundary conditions are catered for. The algorithm works on structured and unstructured meshes, and with minor modifications can be used with meshes of any cell shape. Memory management and data structures are handled. The algorithm is designed in such a way that it can be used either as part of the SPEED code, or as part of the postprocessing environment. The algorithm is presented in appendix A.9.

The initial state of the particle system can be defined by the user in a variety of ways. In what follows we mention a few that can be useful in the specific case of engine flow:

- Input of a single particle through the mouse;
- Definition of a line segment and a number of particles equally spaced on it (figure 6.13.a);
- Generation of an isoparametric grid of particles on a user-defined rectangle (figure 6.13.b);
- Generation of a set of particles isoparametrically located in a cylindrical coordinate system defined at the valve curtain (figure 6.13.c).

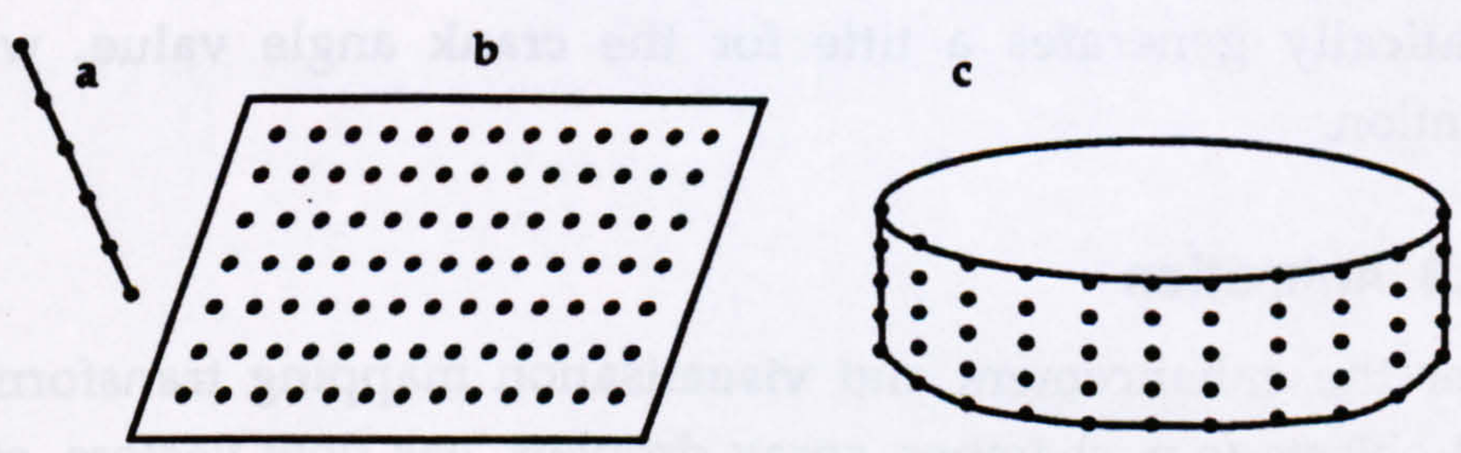


Figure 6.13 Options for definition of the initial status of the particle system.

An object 'particle' is created for each particle. Also, an animation script, containing the particle locations at discrete timesteps, is created for each object particle.

Comprehension of particle trajectories visualisations can be enhanced by the use of the following rendering techniques:

- *Motion blur:* Since the particles are moving, we can compute a short line representing the path of the particle during the frame being rendered and then render this line segment into the final bitmap. This will give the effect of motion blur.
- *Depth cueing:* Particles that are distant from the user can appear dimmer than closer ones. In colour graphics systems this can be achieved by calculating the colour of the particle as an interpolation of the colour of a primitive and a user specified depth-cue colour, which is typically the colour of the background.

Data filtering

SPIL scans the simulation data set, for all timesteps, for minima and maxima Ψ_{min} and Ψ_{max} of user specified simulation values Ψ. Ψ is either a field defined at each cell centroid or a property of each droplet in the droplet population, such as the value of pressure at the centroid of a cell or the diameter of a droplet. Then SPIL fits linearly or exponentially the simulation values Ψ to visualisation values V, ranging between user specified minimum and maximum values, V_{min} and V_{max}. V represents the way in which that value is visualised on the screen, be it colour, droplet size, or arrow length.

Titles generation

The translator is capable of generating user-specified titles, with location and text varying during the course of the animation. Each title is an object; its location and motion is handled by SPIL generated animation scripts. SPIL automatically generates a title for the crank angle value, without user intervention.

6.5.2.3.3 Animation

After the enhancement and visualisation mapping transformations, the created objects (e.g. chamber, spray droplets, gas flow vectors, etc.) must be organized in scenes for which light sources, camera and objects, as well as their motion must be described. An overview of the animation process is illustrated in figure 6.14. The motion of the elements in a scene is described by means of the animation language of IL in animation script files. The features of this language

have been described earlier in section 6.5.1. The scripts are generated by SPIL and can be modified by means of the text editor. An example animation script that has been generated by SPIL can be seen in appendix C.

Rendering and animation control

Real-time previewing of the animation, as defined by the animation scripts, is achieved by storing in virtual memory a collection of wireframe images, for what is called 'flip-movie' animation. Flip-movies are generated by the sub-module ANIMATOR of GVIEW (see figure 6.6). Once we are satisfied with the motion of objects and the camera, key frames of the animation are rendered to check the effect of the light sources, and features like transparency, textures etc.

The finals for the animation are rendered frame-by-frame and stored in virtual memory. This is accomplished by the RENDERER sub-module of module GVIEW (see figure 6.6). Rendering is a time-consuming process and is done in batch mode. One bitmap file is generated for each frame. 25 frames are generated for one second of animation.

A mapping between the visualisation time and the animation time has to be generated to control the variable speed of the animation. Inbetween frames are interpolated between the key frames of the visualisation data set, to match the frame update rate of the system.

6.5.2.3.4 Recording onto the output device

Finals are recorded onto videotape or film by using the LAYOFF module of IL (see figures 6.6 and 6.5.c). The interconnection of the hardware components of the system is discussed in the following section. Recording finals onto video is a lengthy process, thus it is accomplished in batch mode.

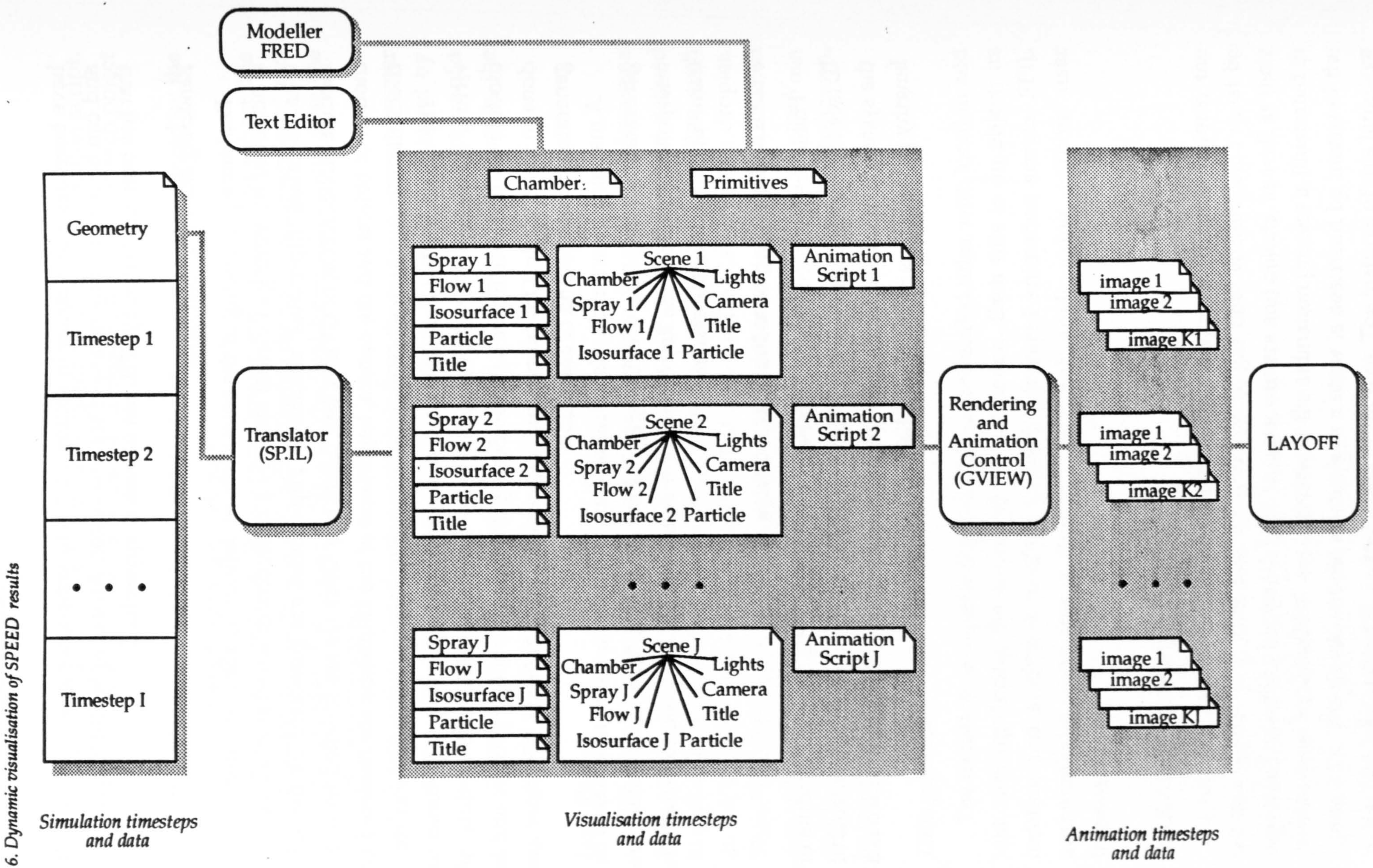


Figure 6.14 Detailed functional diagram of the animation process

6.5.2.4 Implementation of the AERAS system

The translator SPIL is a modular code written in the C programming language except for the input/output modules of the code, which were written in FORTRAN 77 for compatibility with the SPEED code. Example applications of AERAS in the case of a Renault IDI Diesel engine with vertical glow plug are shown in figures 6.9 to 6.12. Figure 6.9 shows the geometry model for the combustion chamber. It contains objects liner, head (only half of them is shown in the figure), prechamber, glow plug, injector, and piston. Three light sources are defined. The simulation data set was produced by the SPEED code. Figure 6.11 shows the visualisation of spray in the prechamber at 380° CA ATDC. Fuel droplets are represented as a set of spheres. Droplet diameter values are exponentially fitted between min and max visualisation values. Droplet temperatures are linearly fitted on a colour scale from blue to yellow. Figure 6.12 shows a set of arrows representing flow velocities on a user-defined cutting plane. The animation of simulation results from the above simulation is studied in Chapter 7 and shown in the video tape submitted with this thesis.

Hardware implementation

The Apollo DN10000 workstation is the main computing and storage component of the system. The display processor memory of the computer is expanded by means of a frame buffer capture card that is installed into the backplane of the workstation. Dedicated hardware is used for the RGBS computer signal to composite video signal conversion and control of the video output unit. Three monitors are used for user interaction and viewing of the frames that are rendered and laid off onto tape. The inter-connection diagram of the workstation and the video output systems is shown in figure 6.15. The animation workbench can be seen in figure 6.16. The components of this system are:

Apollo DN10000

The Apollo DN10000 workstation is the heart of the animation system. This unit contains the system's CPU, the main memory, the display controller, the hard disk, backup facilities such as a cartridge drive and an optical disk drive, a floating point co-processor, and networking hardware.
The implemented configuration has 3 processors and the capability for automated distributed processing throughout the Apollo network (for details see reference [237]).

TRUE-VISION AT Vista card. The TRUEVISION AT Vista frame buffer was installed into the backplane of the workstation, in the XT compatible slot of the backplane.

Workstation Monitors A 1280x1024, 19-inch, non-interlaced colour display, serves mainly as part of the user interface to the software animation system.
In addition an RGB/NTSC colour monitor is used as part of the full colour subsystem, to view the contents of the frame buffer.

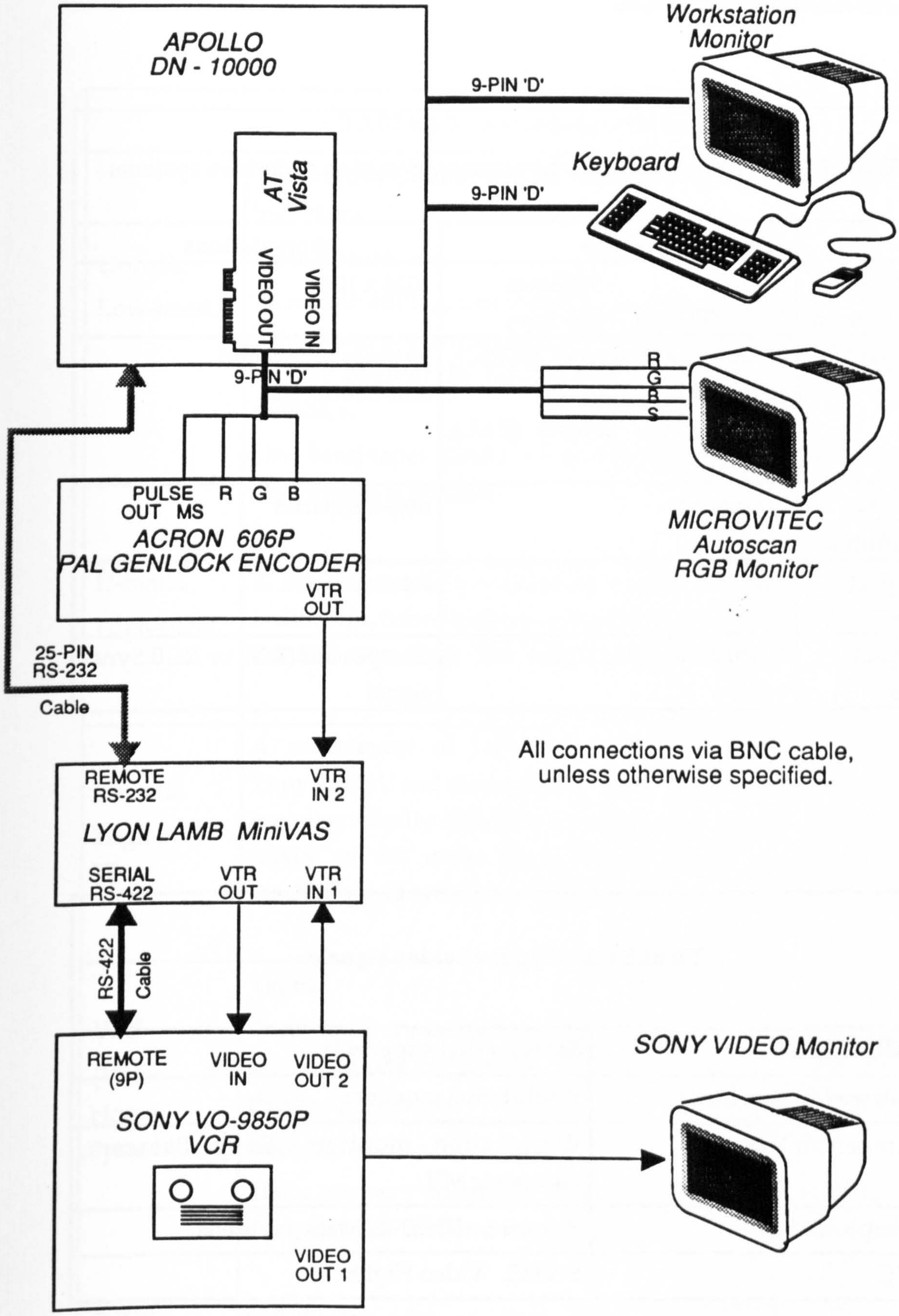
Input devices User input is accomplished via the keyboard and mouse. The keyboard is used for input of alphanumeric data, whereas the mouse serves as the input device in the desktop user-interface metaphor of IL, with windows, icons and pull-down menus.

PAL Encoder: ACRON A PAL encoder by ACRON [238] is used to encode the RGB output from the frame-buffer into a composite video signal which may be recorded into videotape.

Animation Controller: LYON-LAMB MiniVas The LYON-LAMB MiniVas, a microprocessor-based VTR controller for accurate frame-by-frame recording and frame grabbing, is used as the animation control unit of the system (see [239], [240]). It uses a proprietary 'vertical internal frame code' (VIFC) to identify the exact position of the video tape and to position the tape in accordance with command sequences. It interfaces the workstation via an RS-232 serial interface. The MiniVas has two VTR cable interfaces. A serial (RS-422) interface is used for interfacing to a number of editing VTRs. A parallel interface is used for interfacing to the SONY VO-9850P VTR.

Video Tape Recorder The SONY VO-9850P video tape recorder is used for video recording in the U-matic, high-band, SP format. It permits frame-by-frame editing and time coding. Recording and playback with the high-band, SP series cassette tapes provides a sharp and clear picture with minimal colour blurring [241].

Video Monitor A SONY 24" colour monitor is used as the video monitor.



All connections via BNC cable, unless otherwise specified.

Figure 6.15 Inter-connection diagram of the workstation and video output systems.

TABLE 6.1 Incompatibilities between workstation and video systems

	Video	Workstations
Image resolution	NTSC 525 lines PAL/SECAM 625 lines	1024 x 1024
Repetition rate	30 times per second (NTSC), interlaced 25 times per second (PAL), interlaced	50 times per second non-interlaced
Image scanning	interlaced	non-interlaced
Aspect ratio	4:3	5:4
Signal type	Composite video signal	Component (RGB or RGB Sync) signal

TABLE 6.2 Types of video signals.

Video signal	Systems that support it
RGB (red, green, blue)	Workstation monitors
Component Video	Workstation monitors, Sony Betacam, Panasonic MII.
Composite Video.	Betamax, VHS, U-Matic, Video8
Y/C	S-VHS, Video High8

TABLE 6.3 Recording formats.

U-matic Low-band.	The original U-matic format. Introduced by SONY in 1970. 3/4 inch tapes. Main features are: Assemble editing, insert editing of vision and sound, pre-roll for editing, preview facility, review facility, search control, and digital counter in hours, minutes, seconds, and frame numbers. Low band tapes should not be used for high band recordings. The reverse is possible.
U-matic High-band	A modification of low-band for broadcast use. The tape stock is different, hence high band tapes should not be used for high band SP recordings. The reverse is possible.
U-matic High Band SP	A refinement of high-band. This format offers better 'copyability', and increase in picture detail. High-band tapes are magnetically different, so a hole is to be found above the safety 'red tab' under the cassette. This hole is sensed by SP machines, and switches the electronics to the correct mode.
VHS (Video Home System)	**Tapes:** Such tapes are being used for editing, and because of this, manufacturers are providing 'Broadcast Quality' tapes to meet these requirements. **Recorders:** There are very advanced VHS recorders, especially the HQ ones, with Hi-Fi sound feature, etc.
SVHS	SVHS offers editing facilities similar to U-matic. As in U-matic high-band SP, there is an extra identification hole on the tape to distinguish between normal VHS and SVHS.

TABLE 6.4 Frame-buffer capture boards.

Inboard		Outboard	
Machine dependent	**Machine independent**	**Machine dependent**	**Machine independent**
DVI (Tektronix) LVD (Silicon Graphics) SunVideo (SUN) IndigoVideo (Sil. Graphics)	TARGA TARGA+		RGB VIEW 2000

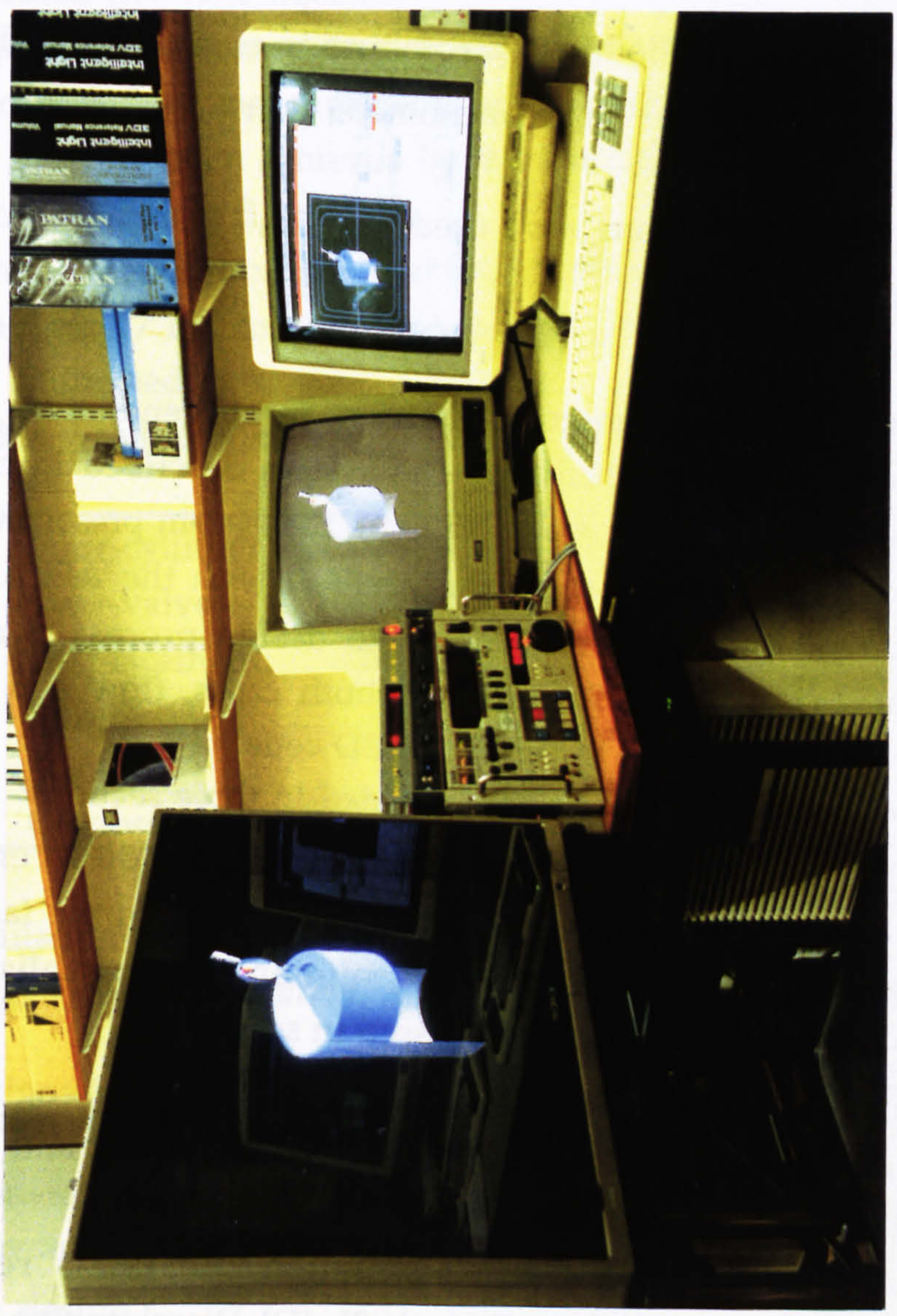


Figure 6.16 The animation workbench.
From left to right: Video monitor, animation controller, MiniVas frame grabber, VTR frame-buffer monitor, workstation monitor.

Chapter 7

Testcases

The objective in the present chapter is to demonstrate how the pre- and post-processing tools that were presented in Chapters 5 and 6 can be applied in the :

- simulation of realistic internal combustion engines with SPEED;
- interpretation of the main phenomena that occur in the combustion chamber thereof.

Two engine configurations were used for this:

1. A DI Diesel engine;
2. An IDI Diesel engine.

In the former case, geometry definition, mesh generation, and results visualisation was accomplished with the system described in chapter 5. In the latter case, pre-processing was done with the system presented in Chapter 5, whereas dynamic visualisation was realised with the AERAS animation system presented in Chapter 6. The mesh design process is discussed in length in the case of the DI engine. In both cases, flow and combustion simulations were performed with the SPEED code.

The information in this chapter is divided in three sections:

1. Simulation of the VW IDEA 1.9L DI transparent Diesel engine. In this the piston has been modified for optical access in the fashion shown in figure 7.1. Mesh generation, solution, and visualisation in the chamber of this engine are discussed in section 7.1.
2. Simulation of the VW IDEA 1.9l DI series Diesel engine. This is a production engine of Volkswagen AG. Simulation and visualisation in its combustion chamber is discussed in section 7.2.
3. Simulation and dynamic visualisation of SPEED results in the combustion chamber of a Renault 2L IDI Diesel engine. This is done in section 7.3.

Data on the above engines was kindly provided by Volkswagen AG and Renault. The aim of their study is to show the suitability of the systems proposed in chapters 5 and 6 for geometry definition, mesh generation, and visualisation of gas flow, fuel injection, and combustion related phenomena that take place in their combustion chambers.

7.1 Simulation of the VW IDEA 1.9L DI transparent Diesel engine

The objective in this section is to present the simulation of gas flow and combustion in a high-speed, four-cylinder 1.9 l. direct injection Diesel engine, with a swirl port, bowl-in-piston combustion chamber and a centrally located five-hole injector.

The engine has been modified to allow optical access to the combustion chamber, so that flow measurements can be taken by application of LDA techniques. This has been achieved by flattening the lower part of the piston bowl, as it can be seen in figure 7.1.a.

A schematic diagram of the combustion chamber at TDC of compression is shown in figure 7.1.b. In this, the topology of the cylinder head and valve regions are delineated.

The engine was provided by VOLKSWAGEN AG. Its characteristics are:

Four-cylinder in-line engine.	
Two valves.	
Displacement:	1.9 l.
Bore:	79.5 mm
Stroke:	95.5 mm
Compression ratio:	20.5

The simulation of flow and combustion in the engine was accomplished in three stages: *Mesh generation*, *Analysis*, and *Visualisation*.

7.1.1 Mesh generation

Mesh generation was accomplished with the PATRAN CAD system and the preprocessing system presented in Chapter 5, in four stages:

1. mesh design,
2. geometry definition,
3. mesh generation, and
4. mesh translation and checking.

7.1.1.1 Mesh design

Prior to geometry definition and mesh generation, the mesh topology had to be designed, to ensure that:

- The geometry features were modelled as accurately as possible. This includes the valve chamfers and the valve recesses on the piston (see the schematic diagram of the combustion chamber in figure 7.1.b).
- The mesh had a layered topology in the squish region despite the fact that the valve base enters the valve recess on the piston at TDC of compression. This would enable the application of the layer addition and removal procedures.
- The mesh satisfied the quality criteria for cell orthogonality, aspect ratio and warpage that are required by SPEED.
- The mesh topology enabled application of PATRAN techniques for geometry definition and mesh generation.
- The mesh was relatively simple to generate and modify.

According to the above criteria, the following decisions had to be made about the mesh, concerning its topology and the mesh generation process.

Design of the valve-head region

When either valve is closed, the mesh topology of the region should be the one in figure 7.2.a. However, this topology does not satisfy the requirement for layered-blocks arrangement in the squish region (see the description of the layer procedure in section 3.2.3.2.2). The mesh shown in figure 7.2.c does, but it features very distorted cells at the region of the valve curtain, where the velocity gradient is high. The meshes shown in figures 7.2.b and d approximate the valve chamfer region allowing for a layered-mesh topology without very distorted cells. From these the latter was used because it accurately represents the location and area of the valve curtain when the layer module follows the opening of the valve by the removal of a layer below the valve base, as it can be seen in figure 7.2.e.

Mesh topology at TDC of compression

At TDC of compression both valves are closed and intrude into the corresponding piston recesses, as is delineated in figure 7.1.b. Given the requirement by the layer procedure of SPEED for a layered mesh topology in the squish region, one layer of mesh blocks was needed for the generation of one or more layers of cells at TDC. The mesh-block topology employed in the region of valve, cylinder head, and piston recess is delineated in figure 7.3.a. In this, thick lines illustrate block-lines whereas the regions filled in gray represent the combustion chamber boundaries. The mesh-block topology employed in the bowl region is shown in figures 7.3.b and c. The lips of the recesses (bowl, valves) and the bowl bulge are represented by curved block-

faces. This ensures better approximation of the chamber geometry as the density of the mesh increases (e.g. compare the mesh arrangements illustrated in figures 7.3.b and c). This, in turn, ensures preservation of the correct volumes and compression ratio.

Domain decomposition into blocks in the squish region

According to the methodology of the preprocessing system the calculation domain must be segmented into blocks prior to mesh generation. In this case, this was accomplished with PATRAN. The topology of the blocks at the radial plane was determined by the contours of the chamber geometry illustrated in figure 7.4. In this, the big circle represents the contour of the liner whereas the set of contours in the middle, the bowl lip. The set of four circles for each valve is imposed by the topology of the valve chamfers and the valve recess on the piston. Given the geometry contours, the domain was segmented into blocks in the fashion shown in figure 7.5. In this, the mesh-block topology is viewed along the cylinder axis, from the cylinder head to the piston. This block topology provides for combination of rectangular meshes in blocks I,J,K,LM,N,O and radial elsewhere. It also allows control of mesh density in blocks A and E through refinement patterns in blocks B, D, F, and H. The final mesh can be seen in figure 7.10.

7.1.1.2 Geometry definition

In this stage a solid model of the computational domain was generated and segmented into blocks in the fashion shown in figure 7.5 by utilisation of solid modelling tools provided by PATRAN.

The solid model of the geometry was generated at TDC of compression. This was accomplished by means of the semi-automatic geometry-definition program 'GEOMETRY' that has been described in section 5.4.2.1.1. In this, the user has to input the dimensions of the chamber (bore, valve diameters, bowl dimensions etc.) and the TDC clearance. The geometry – segmented into mesh-blocks – is shown in figure 7.6. Gouraud-shaded solid models of the geometry can be seen in figure 7.7.

7.1.1.3 Mesh generation.

Given the solid model of the computational domain segmented in blocks, a mesh was generated as a set of vertices and computational cells of hexahedral shape defined in the domain of each block. This was accomplished by means of the semi-automatic mesh-generation program

'MESH' program that has been described in section 5.4.2.1.1. The mesh density was controlled by an integer mesh-density factor. This ranges from 1 (giving a coarse mesh of 8900 cells at BDC) to any integer greater than 1 (giving a mesh of higher density). The following meshes were generated by setting the value of the mesh-density factor from 1 to 3 :

Name	cells at TDC	cells at BDC	mesh-density factor
Coarse	800	8900	1
Medium	4500	25400	2
Fine	21000	135000	3

The 'coarse', 'medium', and 'fine' meshes at TDC of compression can be seen in figures 7.8, 7.9, and 7.10 respectively.

Boundary numbers

The mesh-generation process was completed by definition of quadrilateral elements on the boundaries of the computational domain, denoting boundary faces. The application of the valve and piston motion routines of SPEED required classification of these boundaries into two valve curtain, one ghost, and six wall regions. These are illustrated in the schematic diagram of figure 7.11. They are:

1. Cylinder head;
2. Cylinder liner;
3. Piston crown;
4. Bowl;
5. Inlet valve head;
6. Exhaust valve head.

The ghost boundary is defined in the bowl-throat region as can be seen in figure 7.11.

7.1.1.4 Mesh translation and checking

Once a mesh of hexahedral cells and quadrilateral elements, denoting boundary faces, had been generated and stored in the format of the neutral file, it was translated to the unstructured-mesh format of SPEED. This was realised by means of the numerical pre-processor presented in chapter 5. The mesh was subsequently checked for suitability for SPEED calculations. More specifically it was checked for addressing, cell non-orthogonality, face-warpage, and aspect ratio. The session file of the mesh translation and check processes is given in appendix D.

7.1.2 Setup of initial and boundary conditions

Prior to the calculation with SPEED, initial and boundary conditions for the simulation and engine operating conditions had to be specified. The following information was input into the pre-processing system's problem file (for a summary of initial and boundary conditions see also table 7.1):

- The engine speed (2000 rpm).
- The stroke and connecting rod length (95.5 mm and 144.0 mm respectively).
- The valve lift profile. This was input as a set of crank-shaft angle, in degrees CA, and valve lift pairs, in mm, for each valve. The valve lift diagram can be seen in [247].
- The pressure in the inlet and exhaust manifolds was set to 1.0 bar. The air temperature thereof was set to 300 K.
- The injector location and the hole geometry parameters. The diameter of the injector hole was set to 0.21 mm, the nozzle length to diameter ratio was set to 5.38 and the nozzle discharge coefficient was set to 0.7. The injection angles were deducted from technical drawings of the injector. A diagram of the injection directions can be seen in figure 7.19.
- The injected fuel was set to n-heptane with heat of combustion d_h = 4.897E9 J/kmol.
- The injection rate profiles were taken from measurements (Hentschel [242]) and input in SPEED by linear interpolation between 6 specified values in the fashion shown in figure 7.12. More specifically:

Injected quantity per shot:	8mg of fuel n-heptane
Start of injection:	352.4° CA ATDC
End of injection:	359.9° CA ATDC

  The temperature of the injected fuel was set to 373 K. The number of parcel droplets injected through each of the 5 injector holes during one timestep were set to 8. For 2000 rpm and a timestep of 0.1° CA this is equal to 964000 parcels per second.
- The parameters of the Magnussen combustion model (see section 3.2.2.2) were set to A=25 and B=0.5.
- A uniform temperature was estimated for selected combustion chamber surfaces, with values given in the following table:

Wall 1: cylinder head	350 K

Wall 2: cylinder liner	360 K
Wall 3: piston crown	390 K
Wall 4: bowl	400 K
Wall 5: inlet valve head	340 K
Wall 6: exhaust valve head	390 K

- Valve curtain profiles and discharge coefficients were taken from measurements on the valve curtains (Arcoumanis [243]). They were converted to axial and azimuthal flow angles and radial velocity profiles (see section 3.2.2.3.1), and input into the problem file.

The problem setup file can be seen in appendix B. The main engine operation conditions are summarised in the following table.

Table 7.1

Main engine operating conditions for the simulation of gas flow and combustion in the VW IDEA 1.9L DI transparent Diesel engine.

Compression ratio	20.5
Stroke	95.5 mm
Connecting rod length	144.0 mm
Engine speed	2000 rpm
Pressure in manifold	1.0 bar (for both manifolds)
Air temperature in manifold	300 K (for both manifolds)
Intake flow	Air enters the cylinder with initial angular momentum generated by a helical port. Measurements of gas velocities at the valve curtain were taken from Arcoumanis [243].
Fuel	n-heptane (heat of combustion d_h = 4.897E9 J/kmol)
Injector	5 hole injector provided by Bosch
Diameter of injector hole	0.21 mm
Nozzle length/diameter	5.38
Nozzle discharge coefficient	0.7
Temperature of injected fuel	373 K
Injected parcels per second	964000 for each hole
Injected quantity per shot	8 mg
Start of injection	352.4° CA ATDC
End of injection	359.9° CA ATDC

7.1.3 Simulation

The setting up of the SPEED code was completed with the definition of the layer addition and removal times for the mesh layer procedure.

Mesh motion

A set of FORTRAN routines were written to adapt the mesh to the motion of the piston and valves. The main principles of the routines include:

- calculation of the displacement of the piston and valve within the computational timestep
- motion of the vertices on the above boundaries to their new positions,
- call of the layer module of the SPEED code to perform layer addition or removal actions if mesh distortion – as a result of the previous step – renders it necessary (see also sections 3.2.3.2.2 and 3.2.3.2.3)
- creation of columns (or computational 'linked-lists') of vertices starting from the cylinder head and ending at the piston.
- redistribution of the vertices of each column so that they are all distributed in equal distances.

The above procedure ensured mesh adaption to the moving valves and piston in conjunction with the layer module of the SPEED code and without further user intervention. Views of the mesh at 90° and 180° CA ATDC can be seen in figures 7.13 and 7.14.

Flow and combustion simulations

A first analysis for motored conditions was made for piston speed 2000 rpm. Then, restarting from gas flow results at 350° CA ATDC, the injection and combustion processes were calculated. The following models of the SPEED code were used for the description of injection, mixing, ignition, and combustion (see section 3.2.2.2 and the user's manual of the SPEED code [173] for description and references thereof):

- Fuel injection and spray formation:
  - 'MPI' nozzle-flow cavitation model;
  - 'Gosman-Huh' atomisation model;
  - 'Reitz Diwakar' breakup model;
  - Droplet collision model;
  - 'MPI' droplet-wall interaction model;
- Modified 'Shell' ignition model;
- Modified 'Magnussen' combustion model.

7.1.4 Interpretation and visualisation of results

To interpret the injection, mixing, and combustion results the following visualisations were made:

- x-y plots of averaged properties in the cylinder vrs. time;
- velocity vector plots;
- unburnt fuel concentration contour plots;
- temperature contour plots;
- spray droplet plots.

The data for visualisation was produced by the SPEED code in the form of global (i.e. averaged in the whole chamber) and local (i.e. for each cell) results files, as well as fuel-monitoring and spray-monitoring files.

An overview of the gas motion, injection, and combustion processes in the cylinder is obtained by studying the flow, temperature, fuel concentration, and spray plots at the same time. This has to be done on a large number of plots, but to keep this thesis in a sensible format only some examples of typical visualisations are given.

Gas flow in the cylinder

Swirl is created by bringing the intake flow into the cylinder with an initial angular momentum. Intake-generated swirl persists through the compression and combustion processes and resembles solid-body rotation within the bowl towards the end of the compression stroke. As the piston approaches the TDC of compression position, at 340° CA ATDC, this rotational motion is substantially modified due to squish-generated air motion towards the bowl.

In more detail, during early induction the intake flow forms two main counter-rotating vortices in the cylinder. As can be seen in the schematic diagram of figure 7.15, one vortex is close to the cylinder head rotating in the clockwise direction, with its axis perpendicular to the cylinder axis. The other vortex is close to the piston head, rotating in the anti-clockwise direction, with its axis at an angle to the cylinder axis. The flow field during the induction stroke on an axial cutting plane along the bowl axis can be seen in figure 7.16.

During the compression stroke, the anti-clockwise-rotating vortex dominates the flow, resembling solid-body swirl, with the swirl-centre moving towards the bowl axis. The flow during the late phase of compression is shown on axial cutting planes in figure 7.17. While axial velocities are dominated by the piston velocity at 270° and 290° CA ATDC,

this is no longer the case at 324° CA ATDC. At 350° CA ATDC squish is evident. Velocities of the order of 30 m/s are developed in the regions of the valve indentations and above the bowl lip.

The radial cutting sections in figure 7.18 reveal further information about the location and strength of squish and swirl shortly before injection (at 352.0° CA ATDC). The velocity field presented in this figure is the initial flow field for the injection and mixing processes and therefore of importance for the combustion process.

Injection and combustion

Fuel is injected at 7.6° CA BTDC, directly into the bowl region, in the form of five sprays. The direction of these can be seen in figure 7.19.

At 358.4° CA ATDC the fuel sprays approach the wall of the bowl without reaching it. The droplets at the tips of the sprays have been slightly deflected by the swirl. Near the bowl walls the fuel vapour has been blown around by the swirl (see figures 7.26 and 7.28). The first ignition takes place at the tip of each spray and near the injector. In more detail:

- As can be seen in figure 7.22 no droplet hits the wall. The spray penetration is increasing by 20% from 355.4° CA to 358.4° CA; after this it decays. The peak droplet velocities are of the order of 145 m/s.
- Appreciative evaporation of fuel is evident from about 1.6° CA after the start of injection. Until then evaporation is negligible (see figure 7.21). A study of the unburnt fuel vapour distribution indicates that the highest vapour concentration is at the spray tip close to the wall, were the fuel is accumulating. A second centre of evaporation is near the injector tip. Distributions of the unburnt fuel vapour for 355.4° CA and 358.4° CA are given in cutting sections through the bowl axis in figure 7.23.
- The first fuel is burnt around 4.2° CA after the start of injection (see figure 7.21).
- The large gradient in the curve of the burnt fuel at 357.2° CA ATDC (see figure 7.21) indicates the start of combustion. At 357.2° CA ATDC a temperature rise of up to 500 K along the sprays is visible (see figure 7.24). A number of hot centres is present. These are located at the spray tips and in the area of the injector. The temperature in the following stage rises, spreading from those locations along the sprays. Temperature distributions at 357.2° CA and 358.4° CA ATDC are shown in figure 7.24.

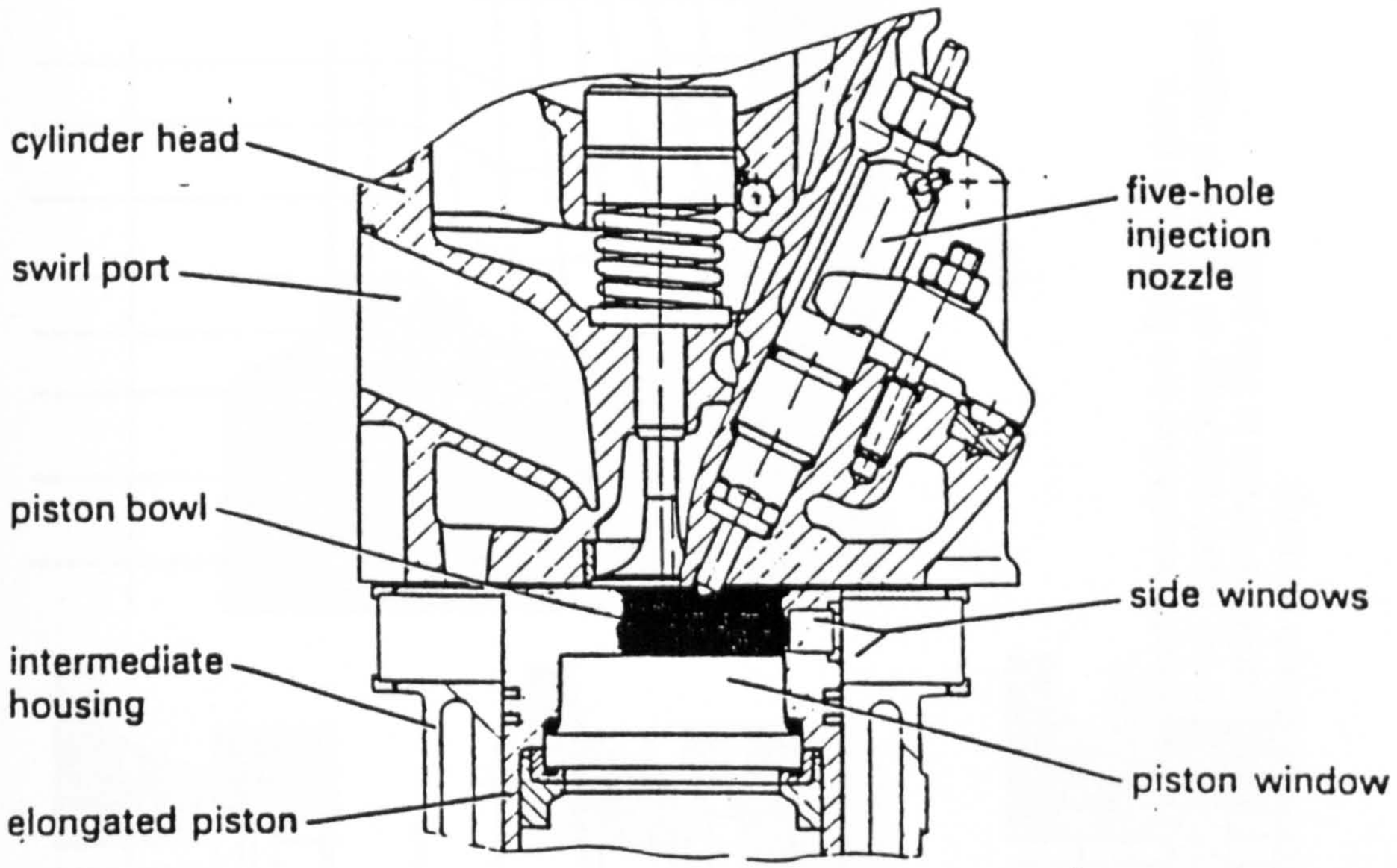
- The injection delay – defined as the time (in degrees CA) elapsed from the start of injection till the curve of the average pressure in the chamber deviates from the one obtained under motoring conditions – is 4.2° CA.
- Velocity vector fields from 357.2 CA onwards indicate a local acceleration of the flow by the sprays and the beginning of combustion (see figure 7.29).

Flame propagation along each spray is rapid. As can be seen in figures 7.23 and 7.24, 1.2° CA (0.1 ms) after auto-ignition, at 358.4° CA ATDC, the flame has propagated along the body of each spray. At TDC the top part of the bowl is filled with flame. This can be seen in figure 7.25, where temperature contours of the mixture are displayed on various radial cutting planes in the squish and bowl regions. This rapid burning is due to the presence of stoichiometric mixture of air and fuel vapour in the top part of the bowl (see figure 7.26). The flame does not propagate to the bottom of the bowl before 370° CA (see figures 7.27 and 7.28).

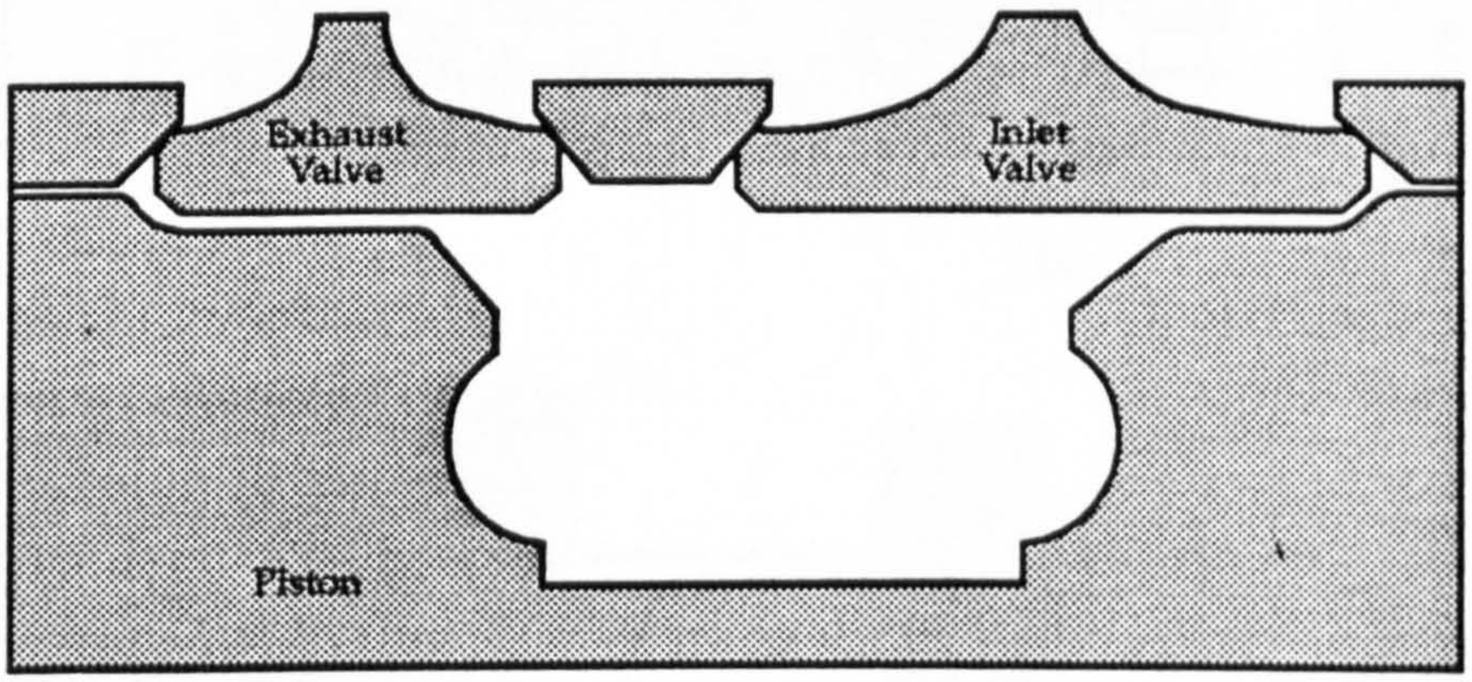
Shortly after TDC the flame spreads out over the piston crown towards the cylinder wall due to combustion-produced gas expansion and the reverse-squish flow. This motion of the mixture is evident as early as 1° CA ATDC of compression with peak velocities of the order of 44 m/s (see figure 7.29). Due to this, fuel vapour is blown into the squish region, as can be seen in figure 7.28.

As the piston speed increases the mixture is blown to the piston crown and the bottom of the bowl (see figures 7.29 and 7.31). Because of this and the combustion-produced gas expansion and reverse-squish mentioned earlier, at 380° CA burning mainly takes place in the squish region, near the walls of the bowl, and at the bottom thereof (see figure 7.30).

There is a gradual diminution of the burning regions, starting from the cylinder head (see figure 7.32) as the remaining fuel vapour is blown to the piston crown and bowl bottom (see figure 7.33). Burning regions persist till 70° CA ATDC of compression (see figure 7.32). After the end of combustion, anti-clockwise swirl resembling solid-body rotation dominates the flow in the region between the bowl and the cylinder head and within the bowl (see figure 7.34).



(a)



(b)

Figure 7.1 (a) Elevation view of the VW IDEA 1.9L DI transparent Diesel engine with a cutting plane along the axis of the bowl. The bowl base has been flattened, to facilitate optical access for LDA measurements. (b) Schematic diagram of the combustion chamber.

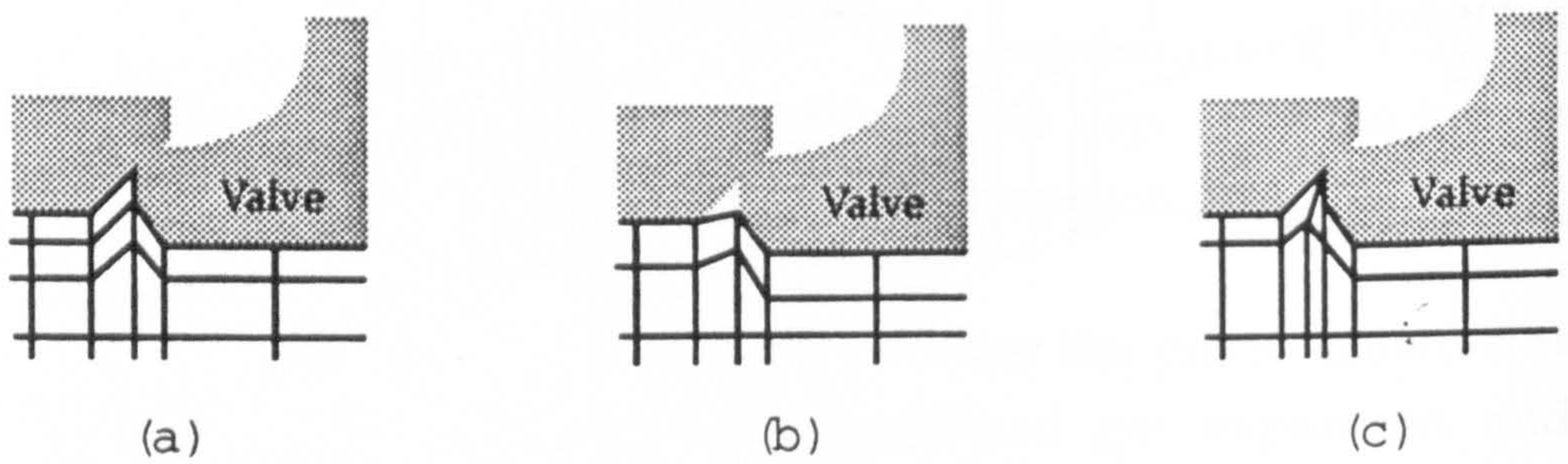


Figure 7.2 Mesh design at the valve region.
Four different approaches to meshing the valve chamfer are shown in (a) to (d).
The design of figure (d) accurately represents the valve curtain location and area,
as it can be seen in (e).

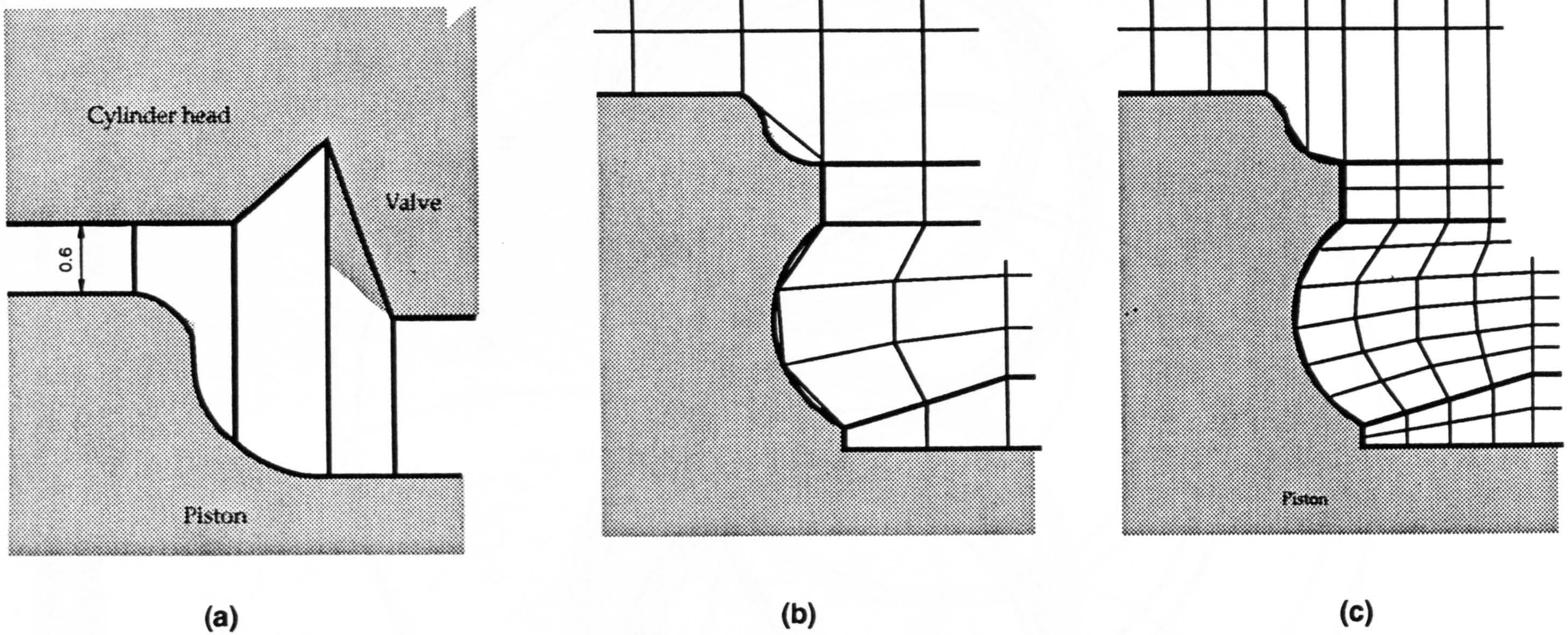


Figure 7.3 Mesh topology at TDC of compression.
Thick and thin lines delineate block-boundaries and mesh-lines respectively. Physical boundaries are filled with gray.
(a) Mesh-block topology at the cylinder head, valve and piston crown regions. (b) and (c) Block topology at the bowl region.
The lips of the recesses and the bowl bulge are represented by curved surfaces (lines in this plot),
therefore approximation of the geometry increases with mesh density.

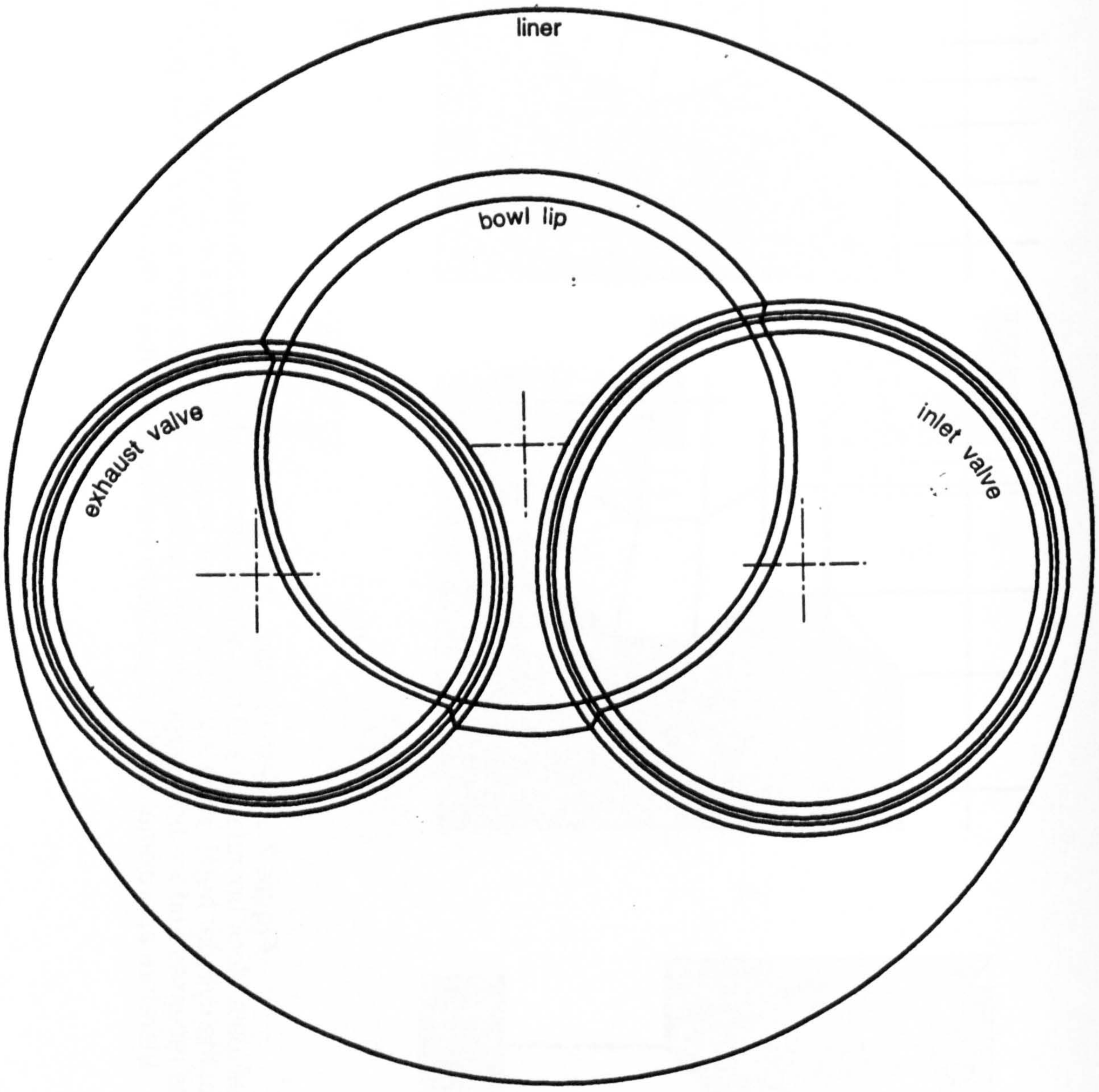


Figure 7.4 Geometry contours in the squish region. View along the cylinder axis, from the cylinder head to the bowl.

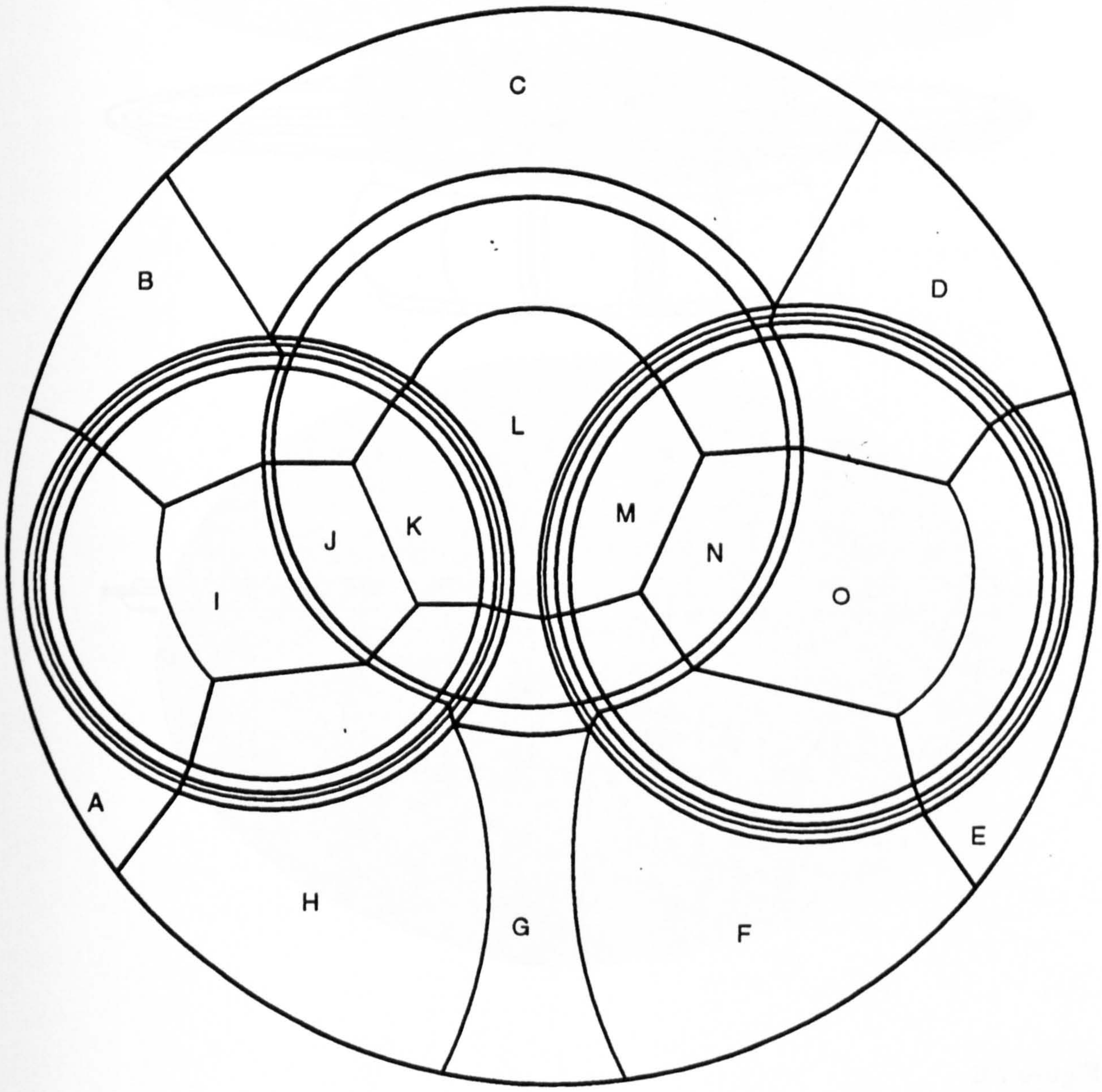


Figure 7.5

Decomposition of the computational domain in blocks. View along the cylinder axis, from the cylinder head to the crankshaft. Contours of blocks are delineated with thick continuous lines.

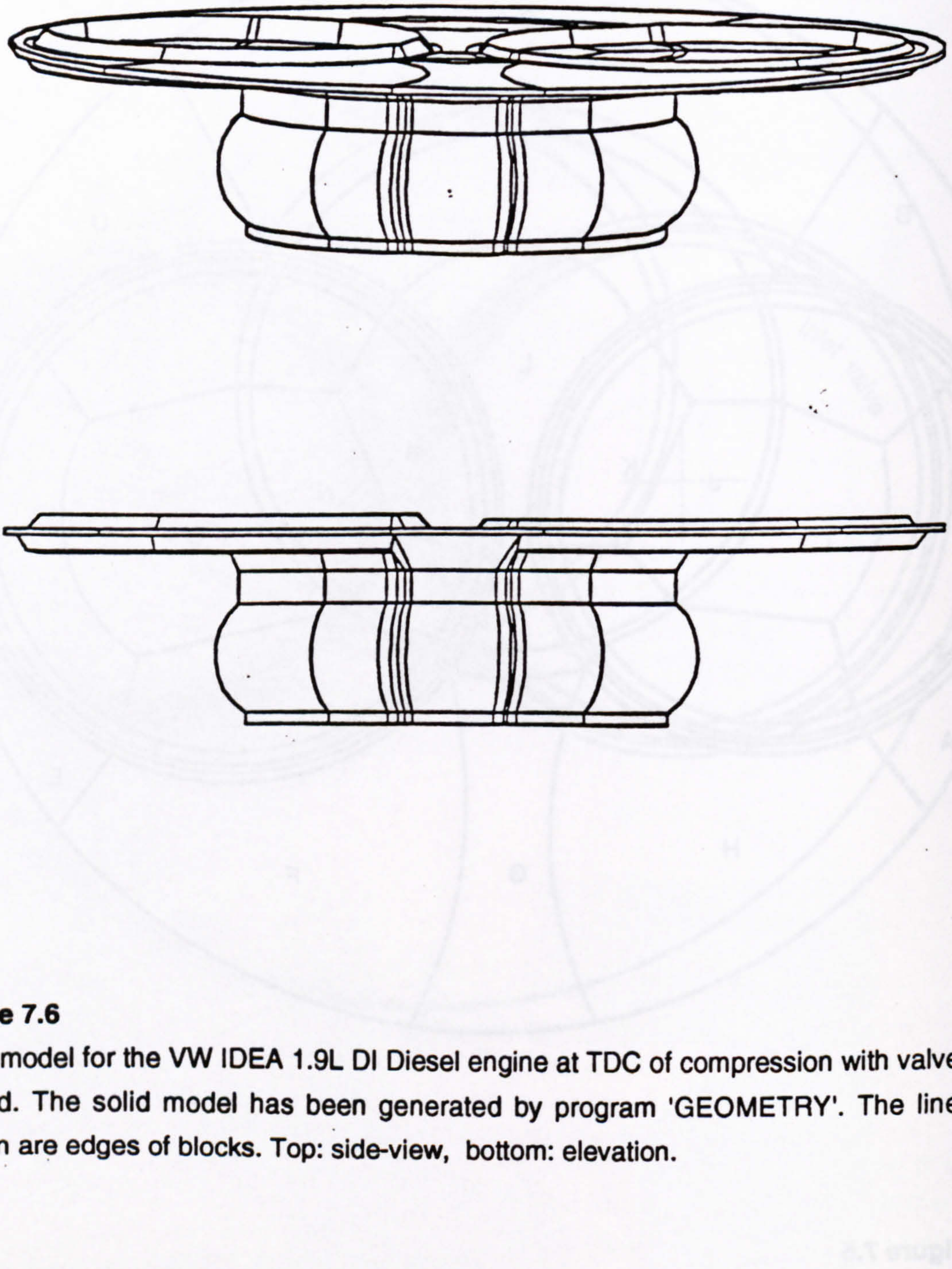


Figure 7.6

Solid model for the VW IDEA 1.9L DI Diesel engine at TDC of compression with valves closed. The solid model has been generated by program 'GEOMETRY'. The lines shown are edges of blocks. Top: side-view, bottom: elevation.

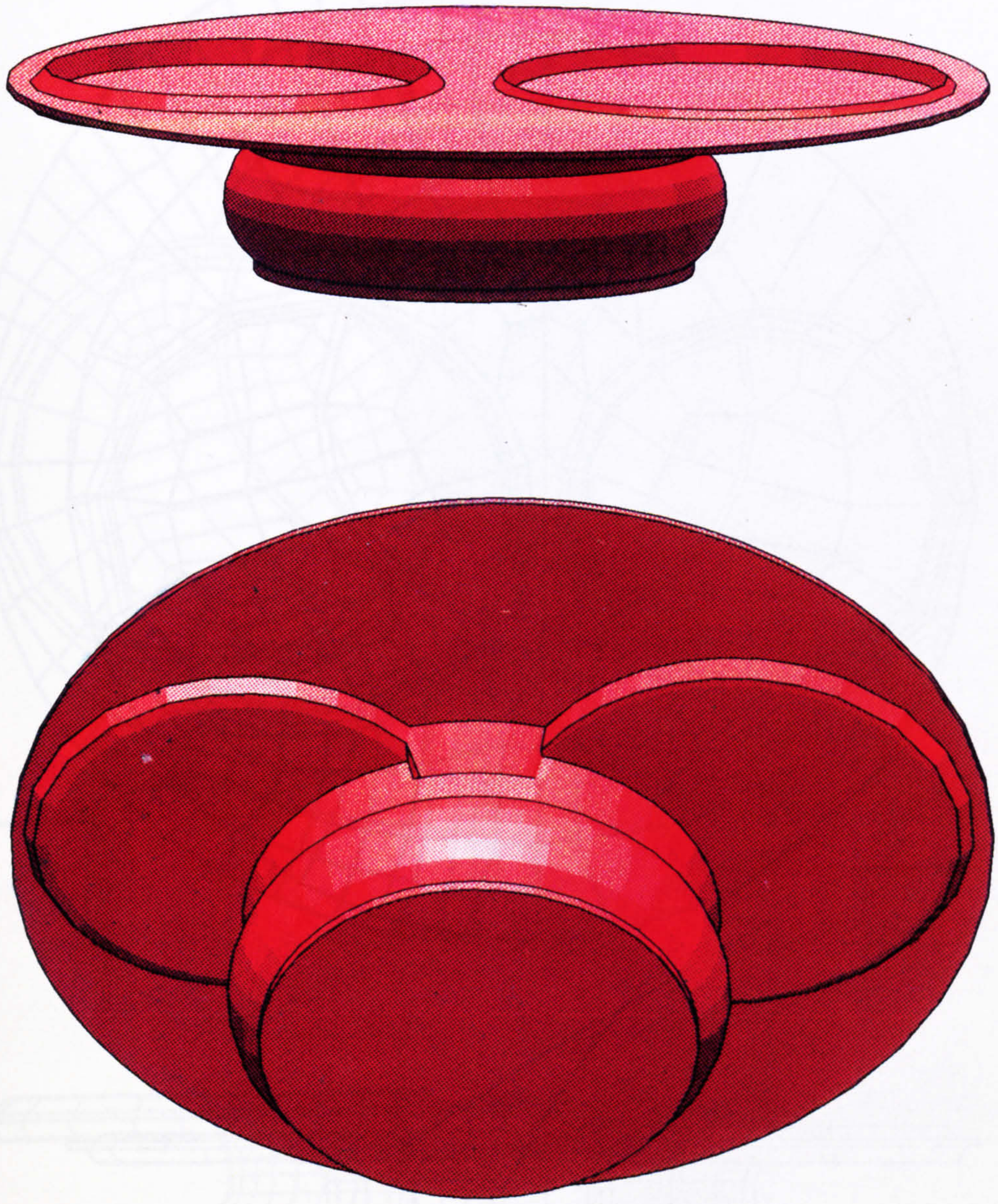


Figure 7.7
Gouraud-shaded solid model for the VW IDEA 1.9L DI Diesel engine at TDC of compression with valves closed. Top: side view showing the valve chamfers on the cylinder head. Bottom: side view showing the bowl and valve recesses at the piston.

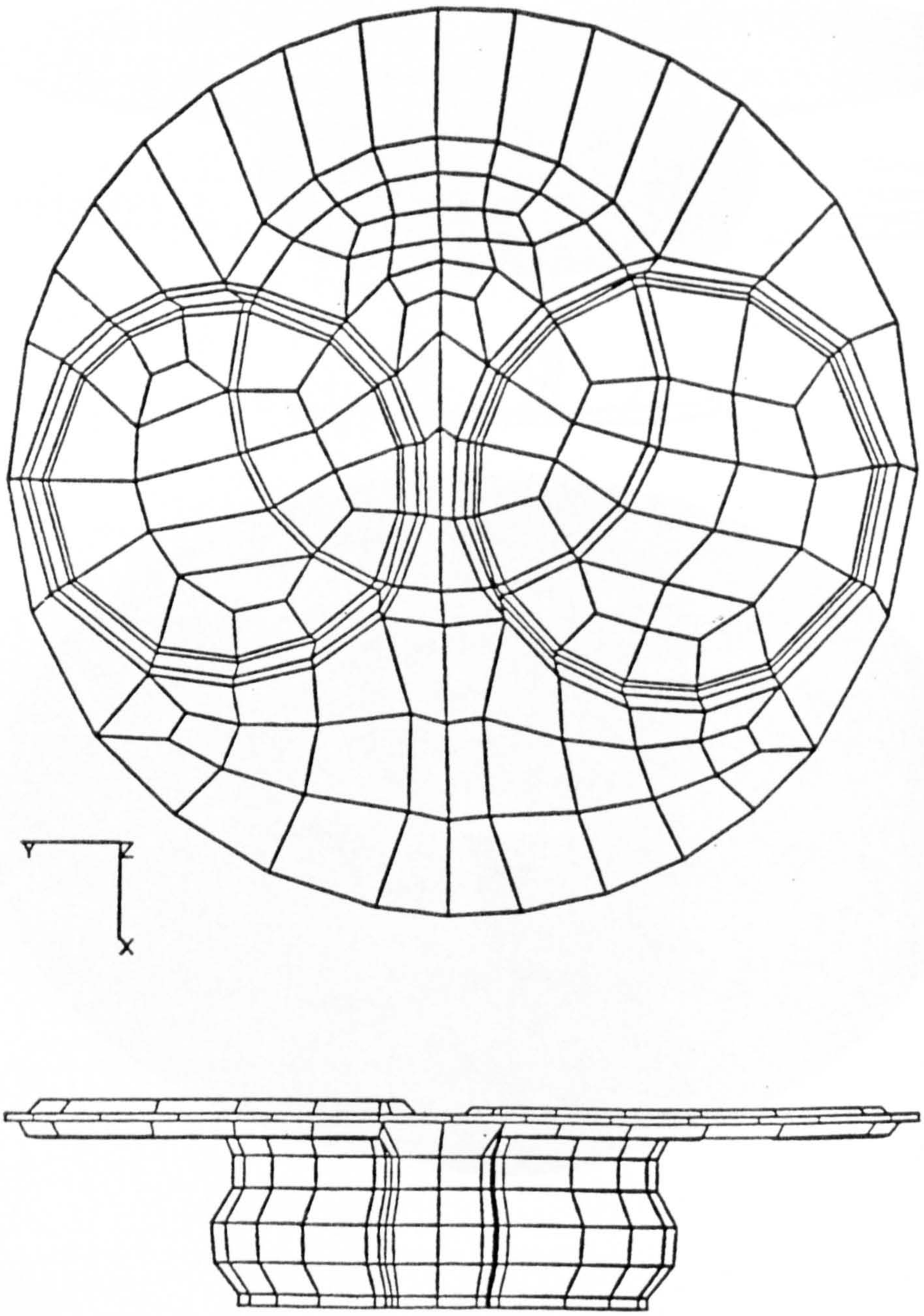


Figure 7.8 Coarse mesh for the VW IDEA 1.9L DI Diesel engine. Generated with program 'mesh' by setting enumber = 1. Top: top view. Bottom: elevation.

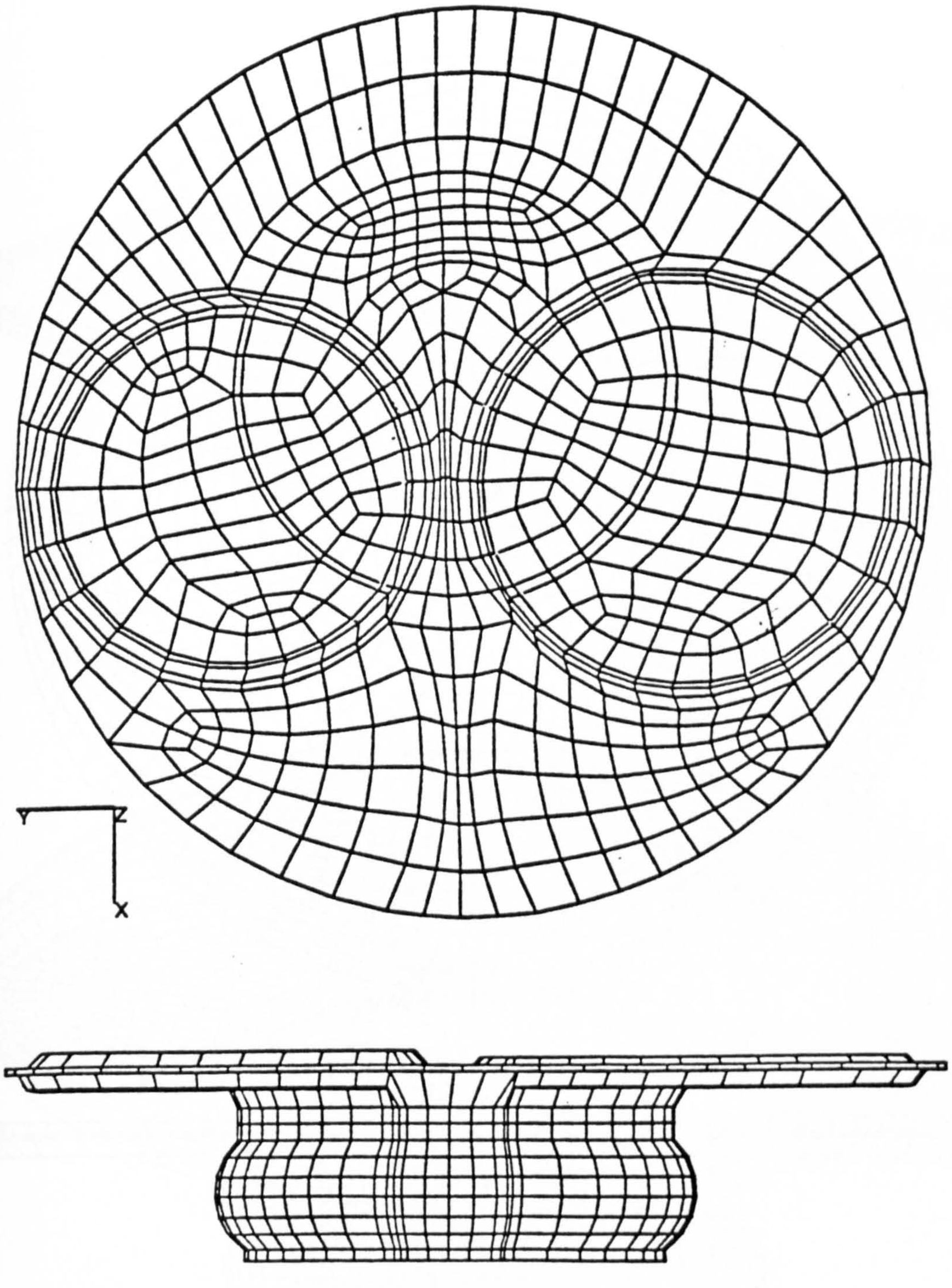


Figure 7.9 'Medium' mesh for the VW IDEA 1.9L DI Diesel engine. Generated with program 'mesh' by setting enumber = 2. Top: top view. Bottom: elevation.

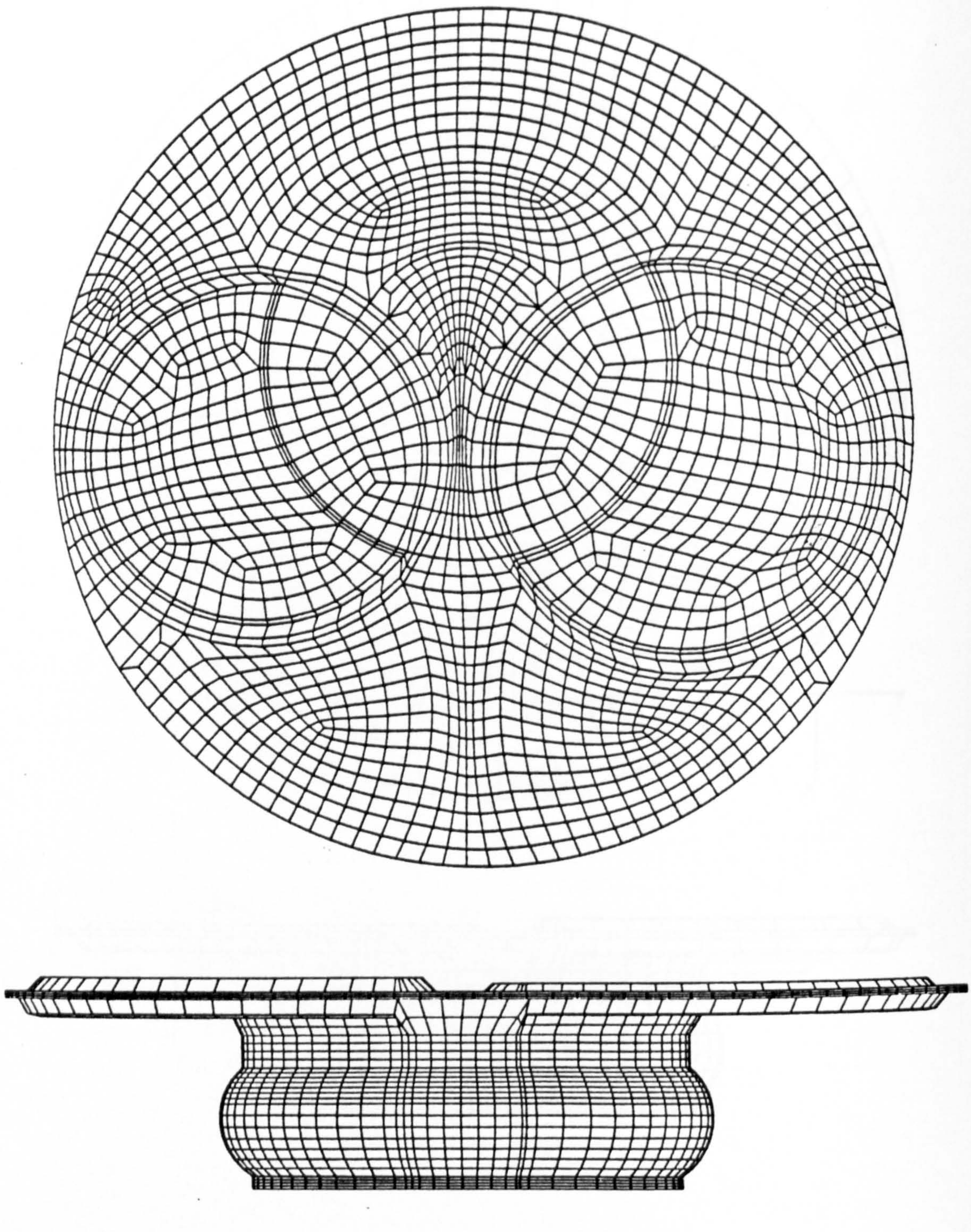
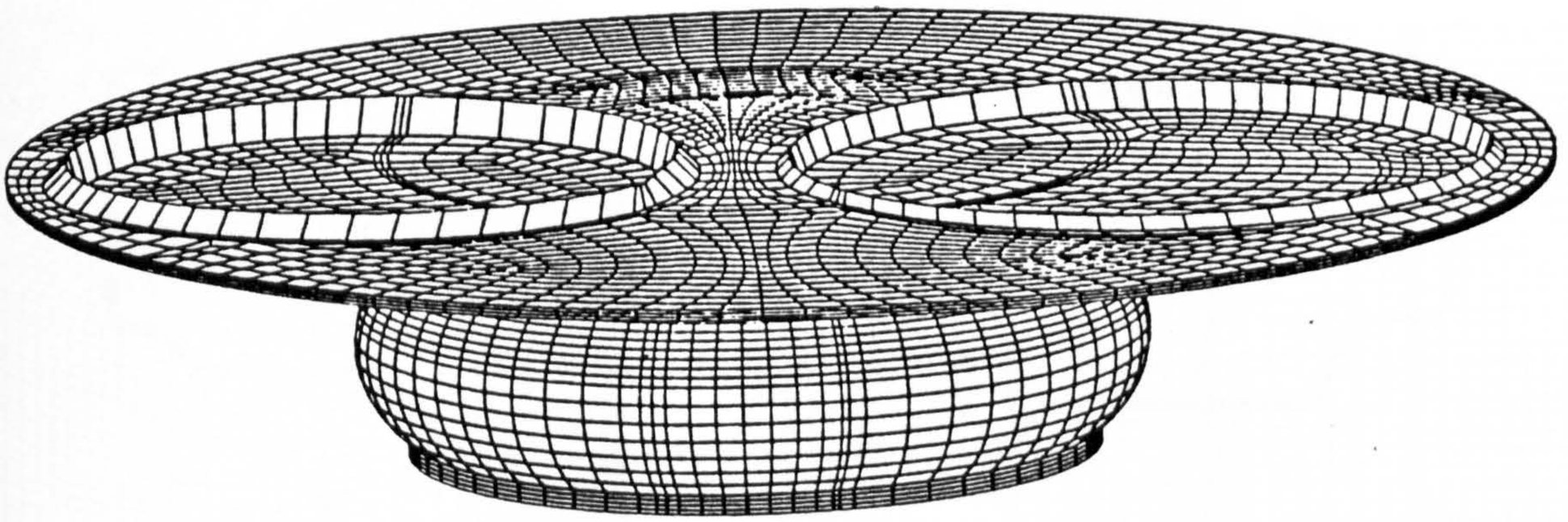


Figure 7.10 Fine mesh for the VW IDEA 1.9L DI Diesel engine. Generated with program 'mesh' by setting enumber = 3. Top: top view. Bottom: elevation.



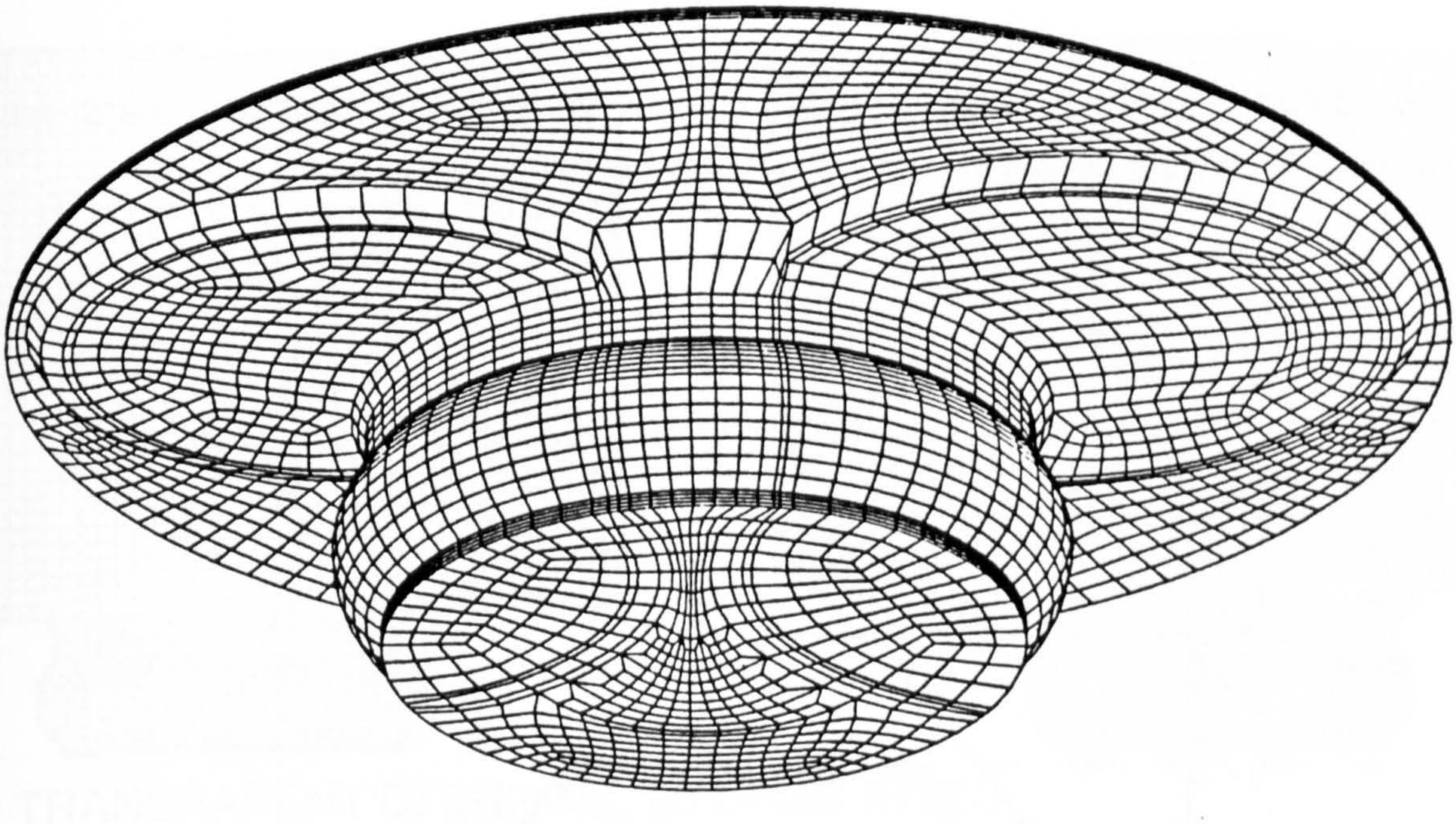


Figure 7.10 (continued) Fine mesh for the VW IDEA 1.9L DI Diesel engine. Generated with program 'mesh' by setting enumber = 3. Top: side view showing mesh topology in the cylinder head. Bottom: side view showing the topology of the mesh in the piston region.

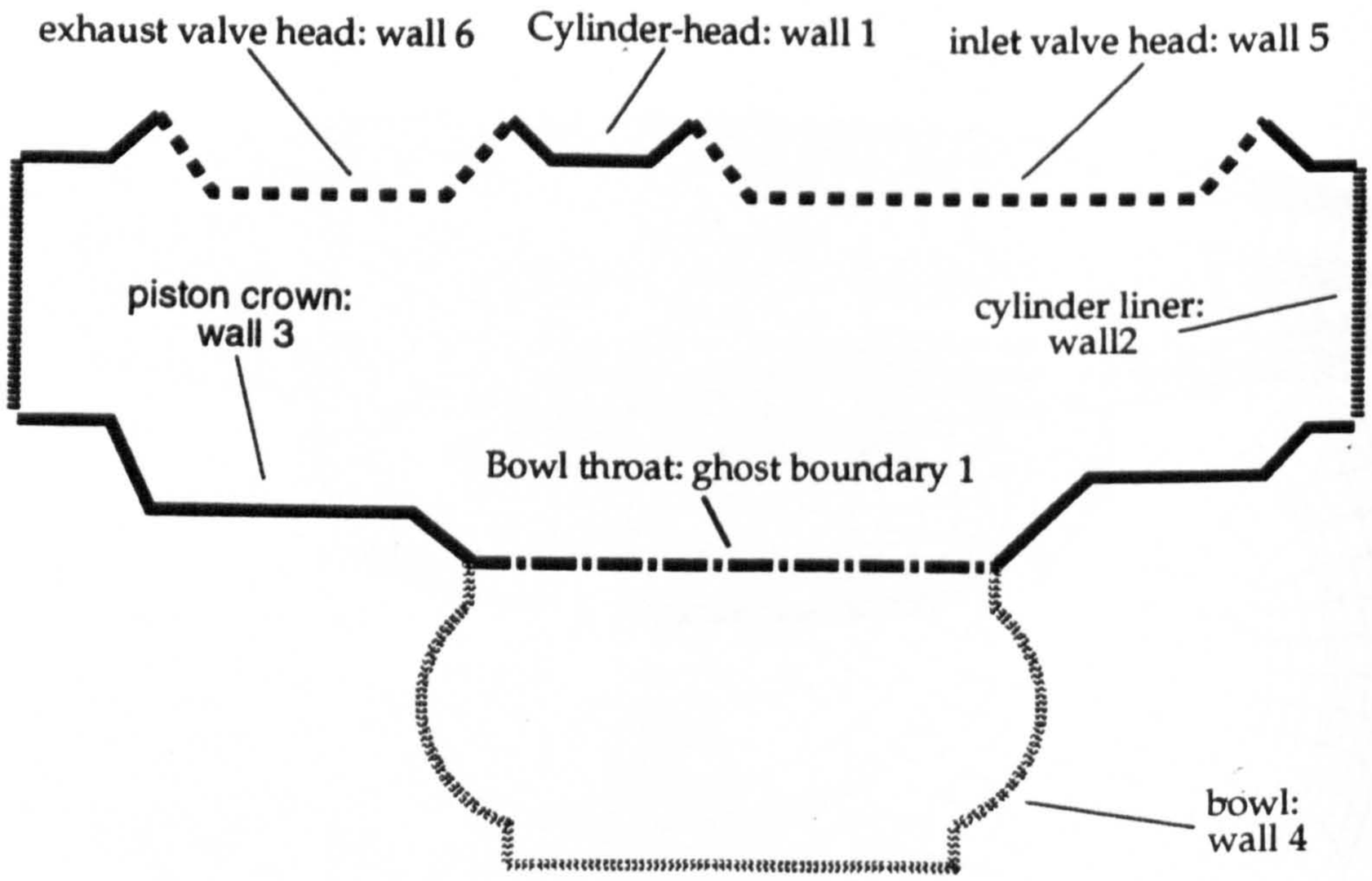


Figure 7.11 Schematic diagram of the combustion chamber, delineating the 6 wall boundaries and the ghost boundary at the entry of the bowl.

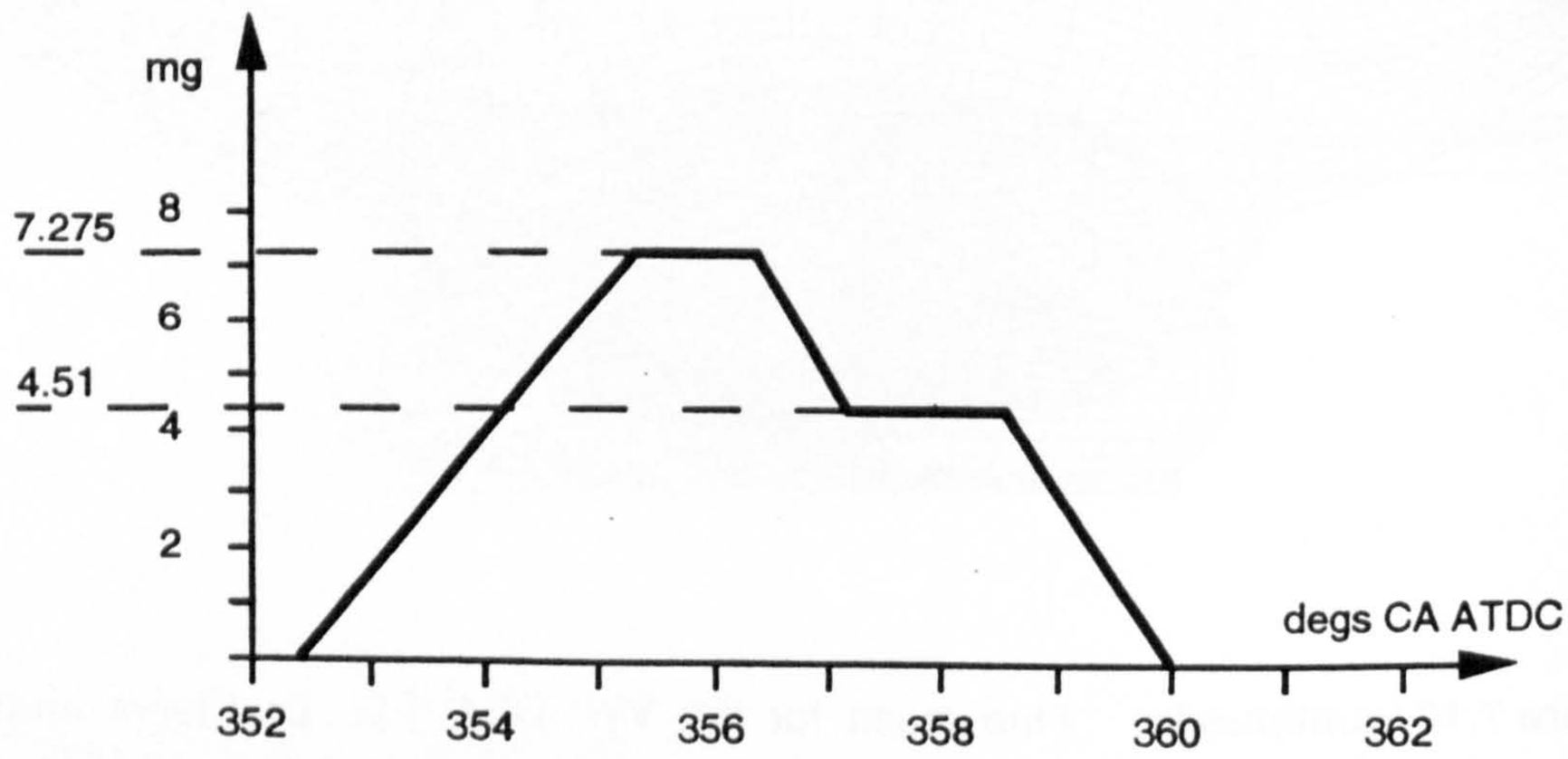


Figure 7.12 Flow rate for each injector hole as a function of crankshaft angle.

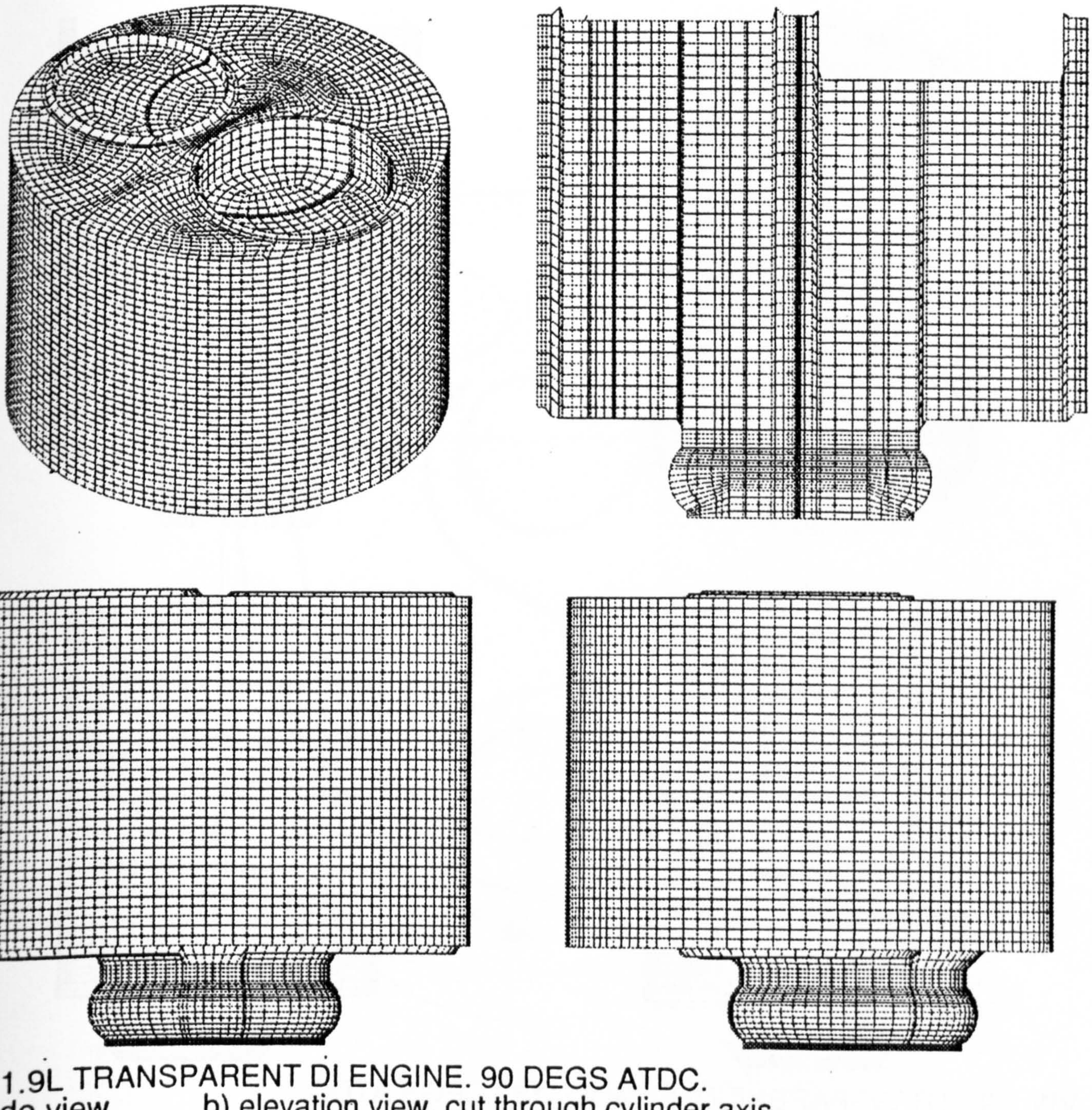
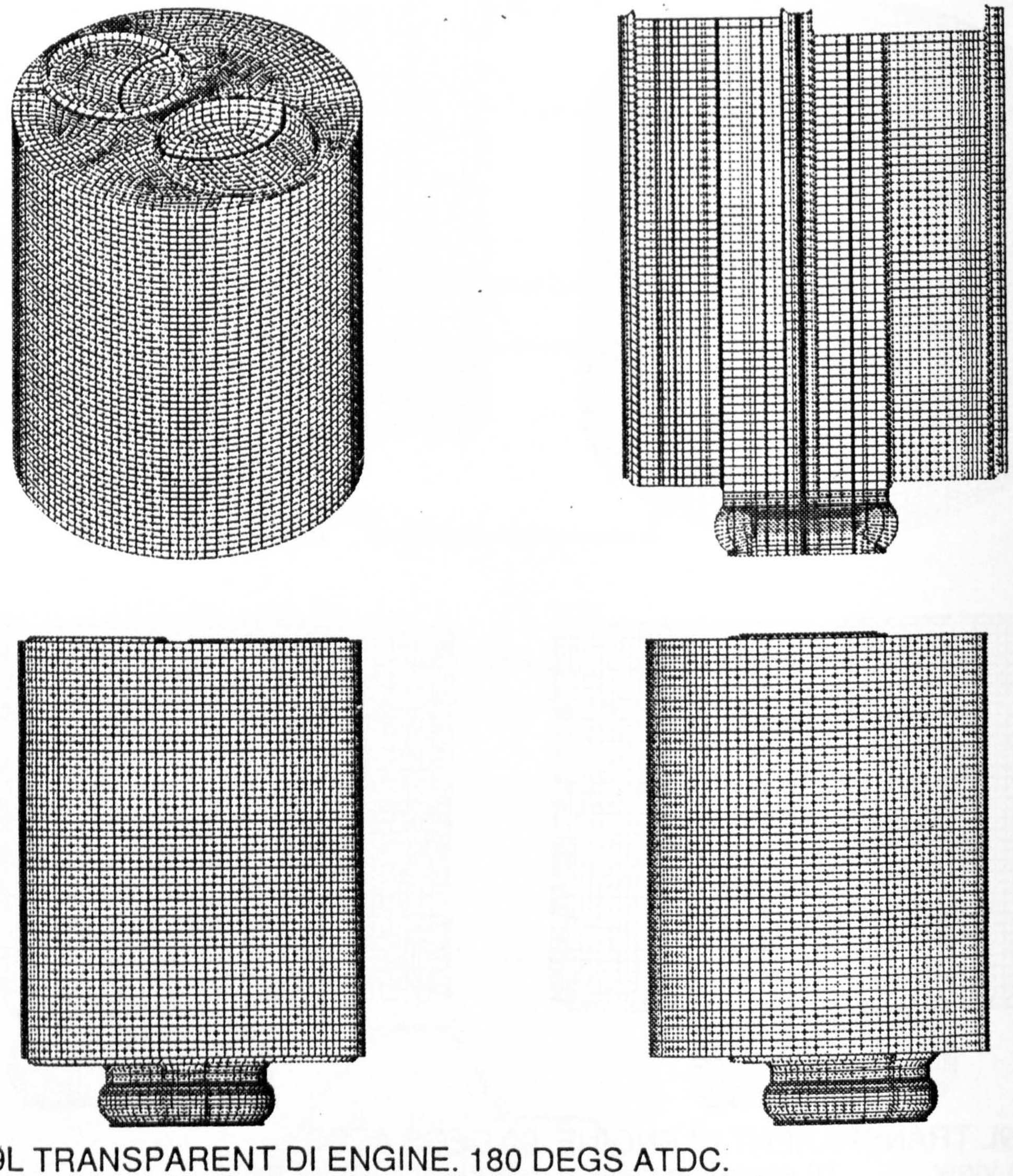


Figure 7.13 Computational mesh at midstroke of induction.



VW 1.9L TRANSPARENT DI ENGINE. 180 DEGS ATDC.
a) side view b) elevation view, cut through cylinder axis
c) elevation view d) elevation view

Figure 7.14 Computational mesh at BDC of induction.

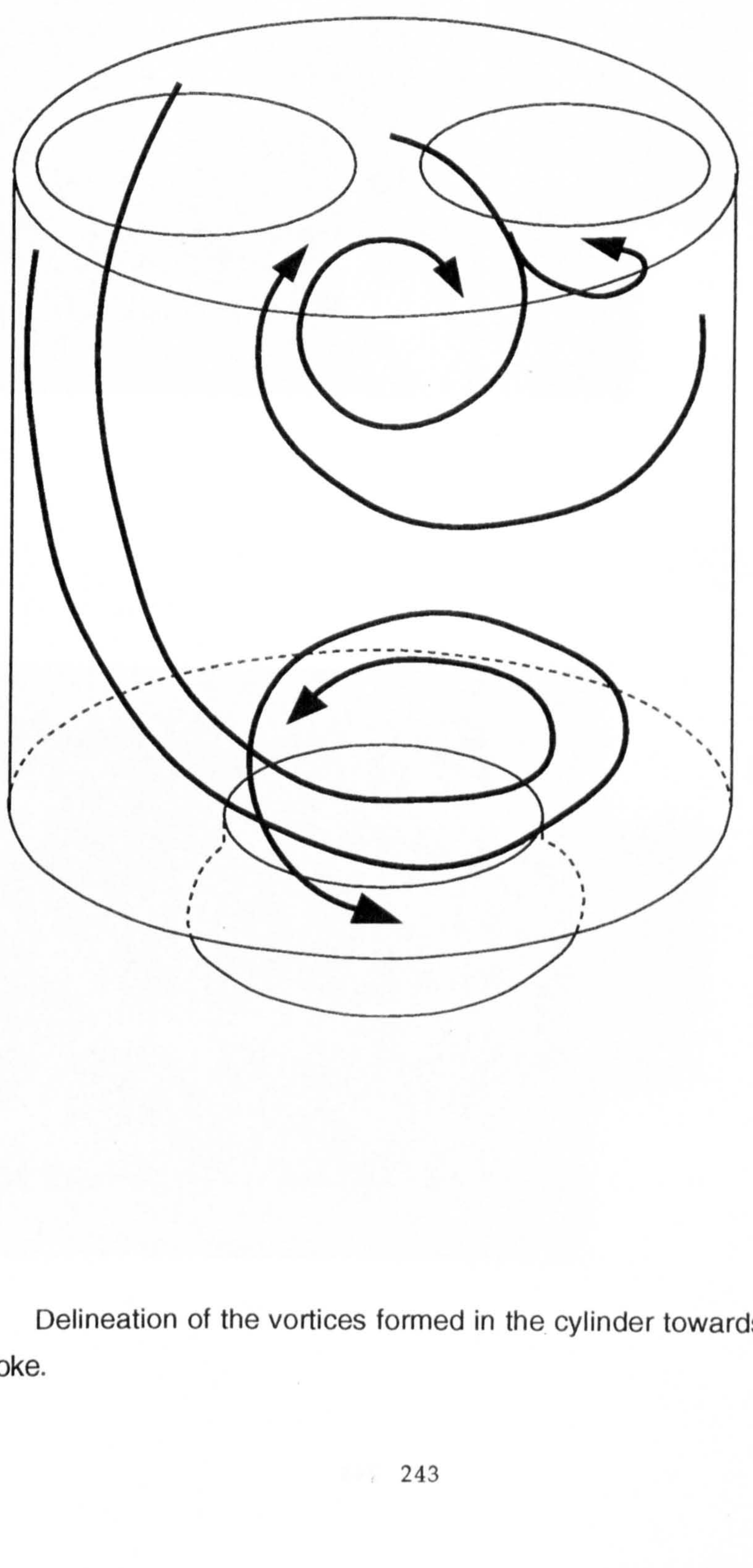


Figure 7.15 Delineation of the vortices formed in the cylinder towards the end of the induction stroke.

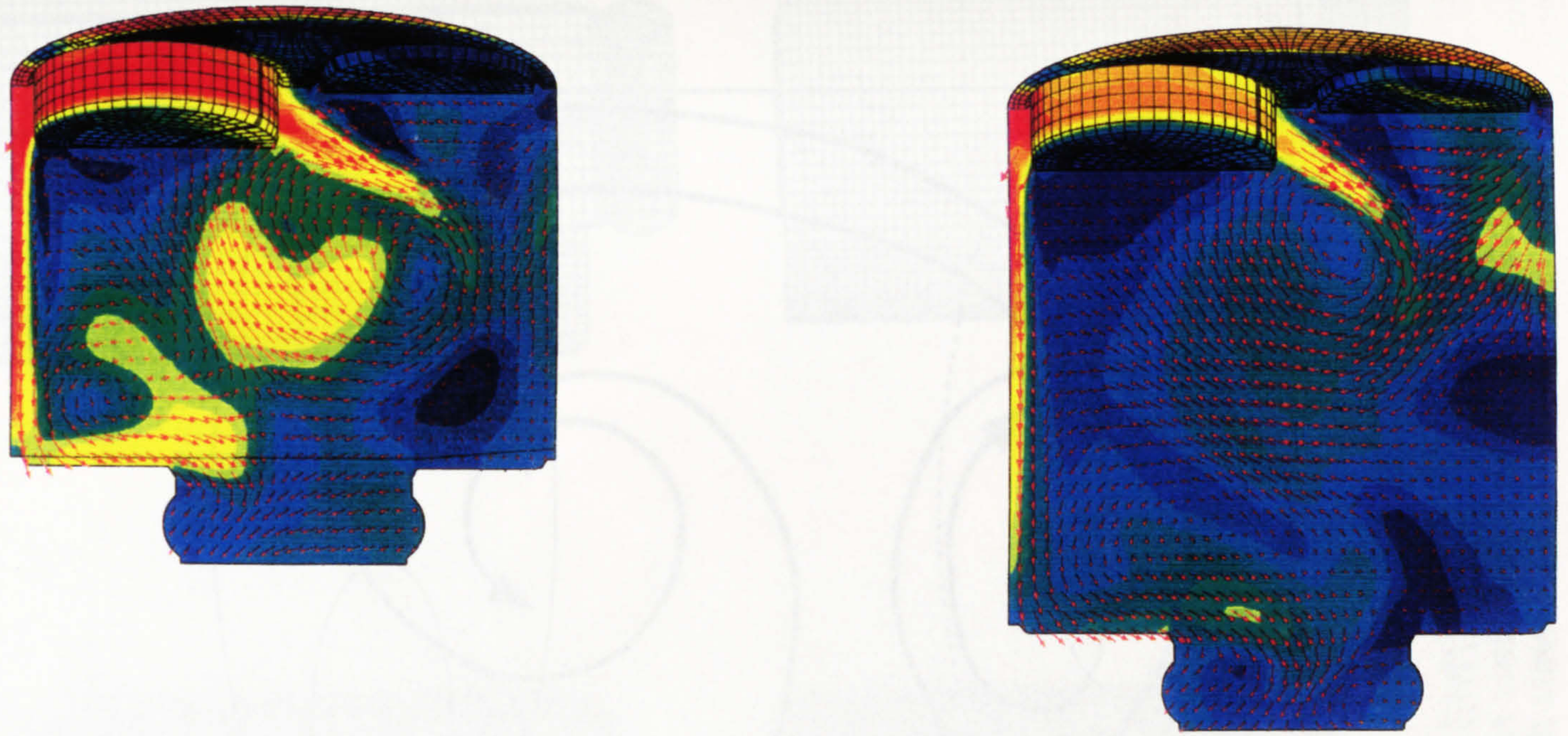


Figure 7.16 Gas flow velocity vectors on cutting planes along the axis of the bowl. The velocity magnitude is also visualised in the form of filled-area contours. Left: 90° CA ATDC. Right: 120° CA ATDC.

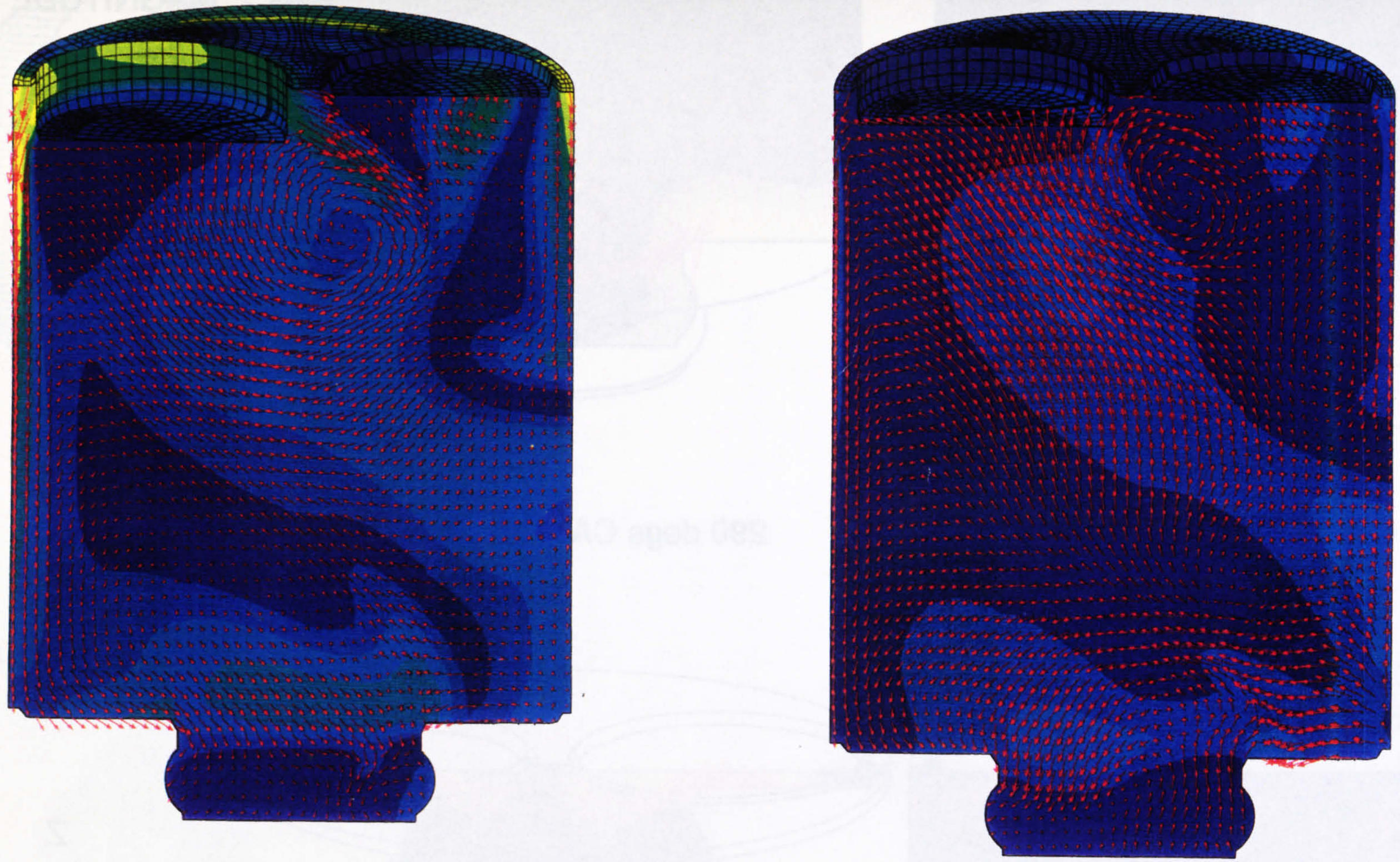


Figure 7.16 (continued) Gas flow velocity vectors on cutting planes along the axis of the bowl. The velocity magnitude is also visualised in the form of filled-area contours. Left: 150° CA ATDC. Right: 180° CA ATDC.

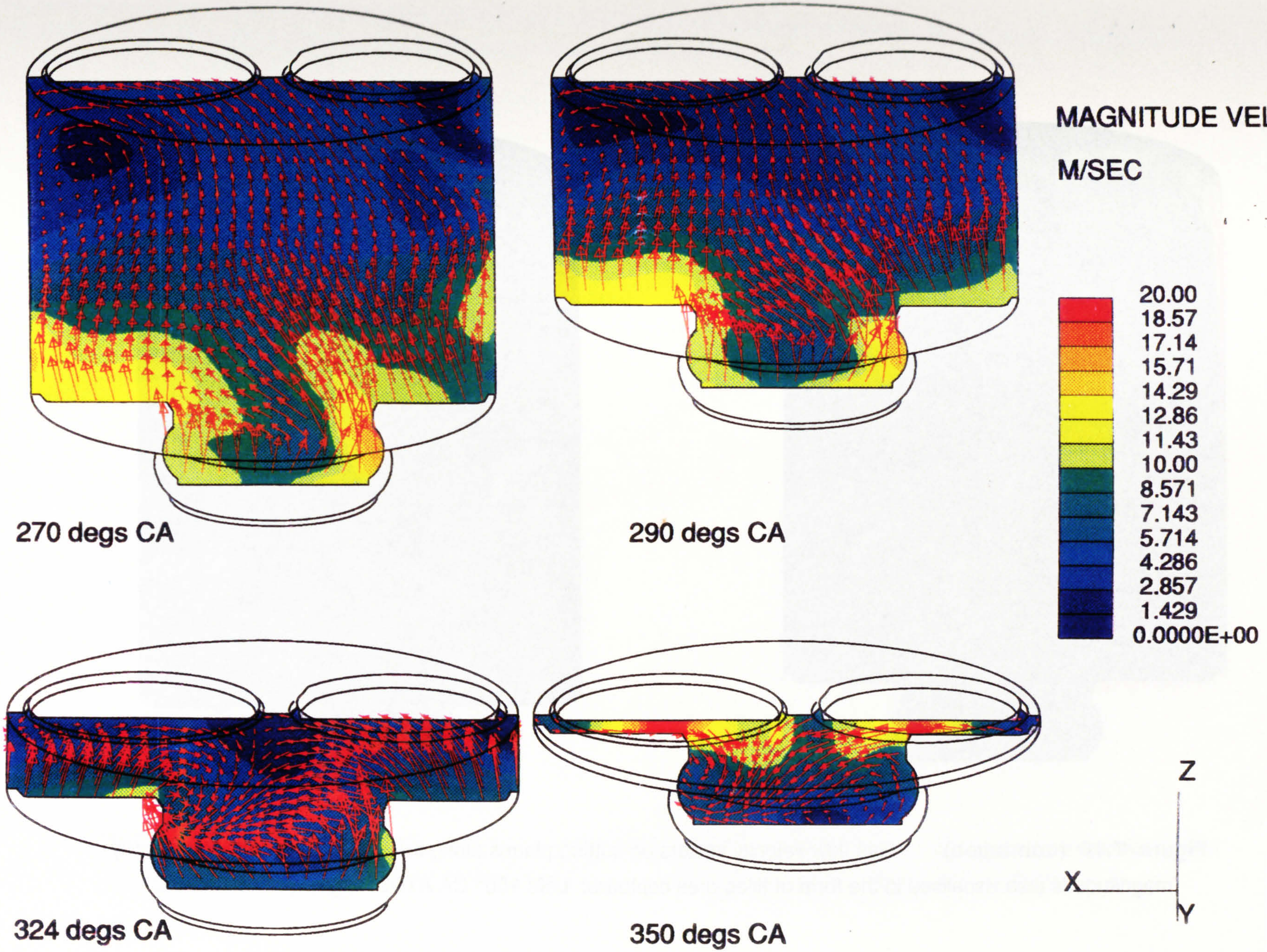


Figure 7.17 Gas flow velocity vectors (270°, 290°, 324°, 350° CA ATDC) on cutting planes along the axis of the bowl. The velocity magnitude is also visualised in the form of filled-area contours.

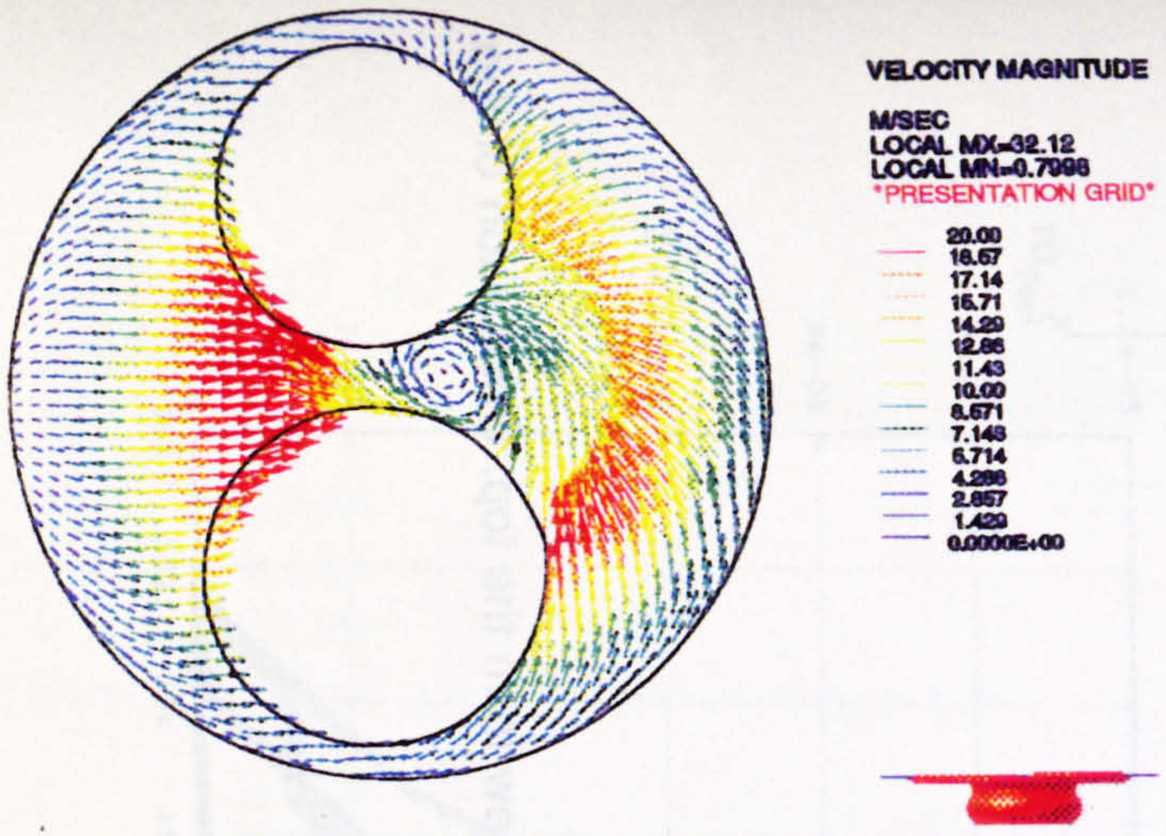


Figure 7.18 Gas flow velocity vectors at 352.0° CA ATDC on radial cutting planes in the squish and bowl regions.

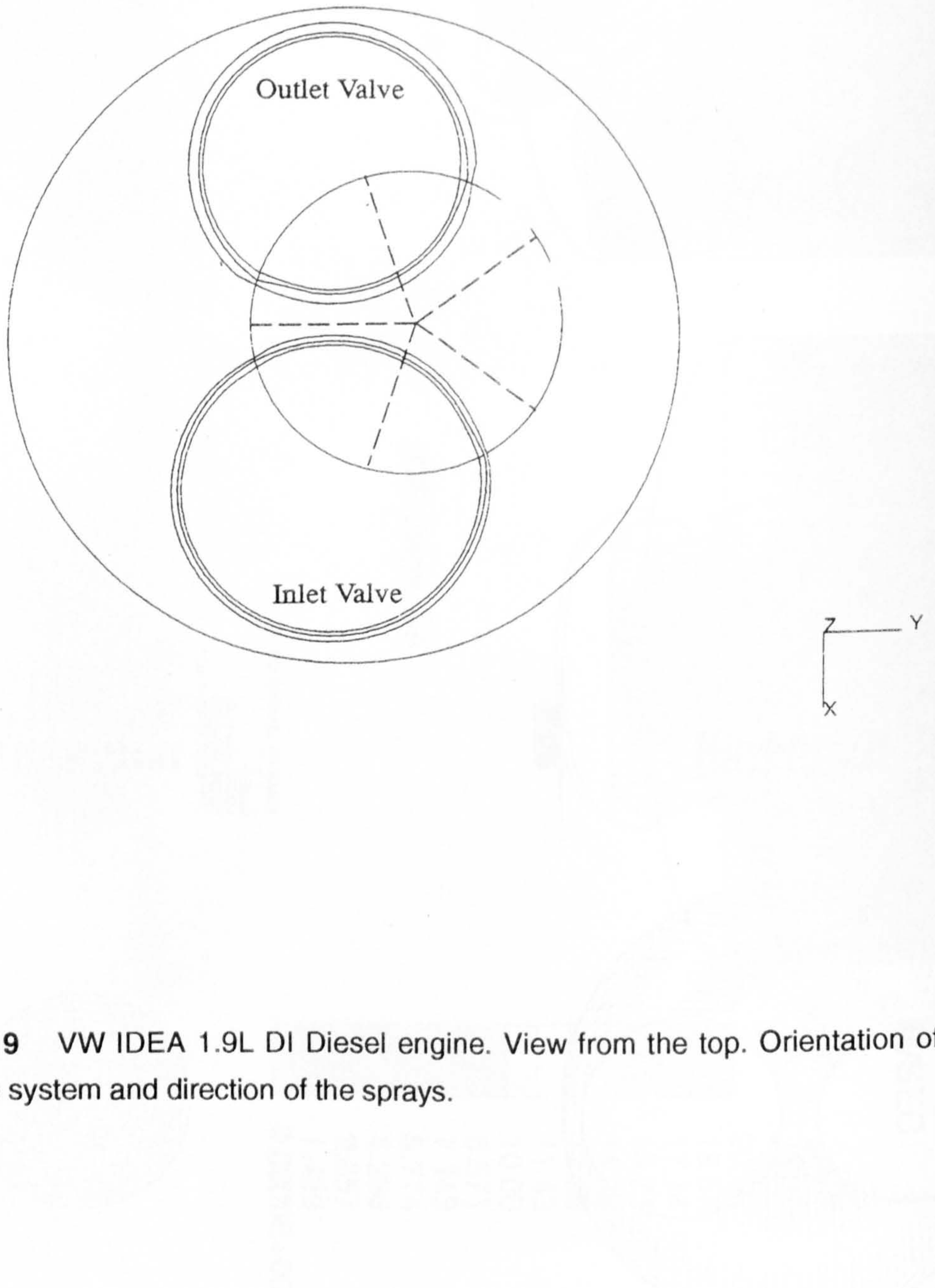


Figure 7.19 VW IDEA 1.9L DI Diesel engine. View from the top. Orientation of the coordinate system and direction of the sprays.

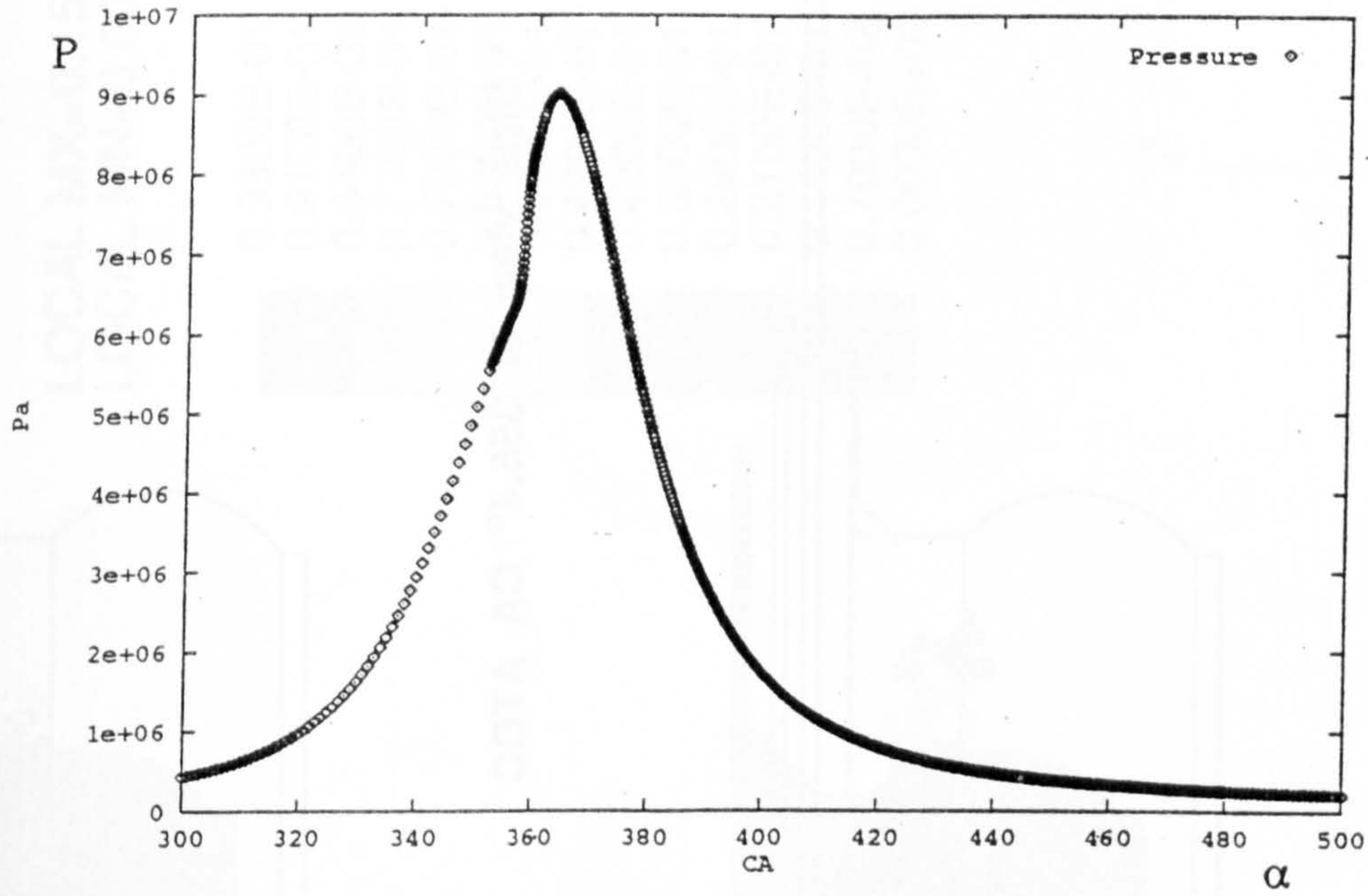


Figure 7.20 Calculated pressure trace.

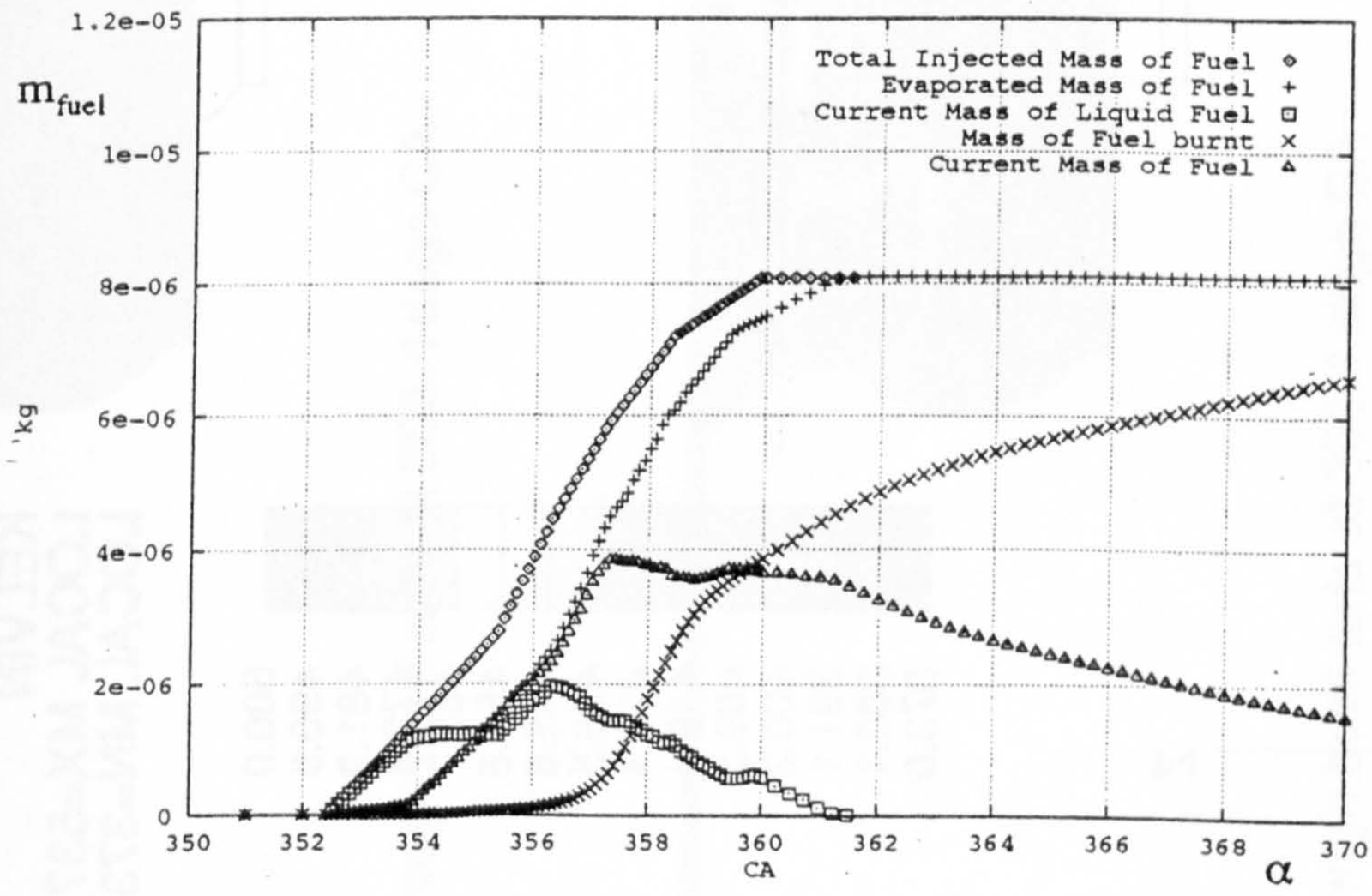


Figure 7.21 Monitoring of the injected, evaporated, and burnt mass of fuel.

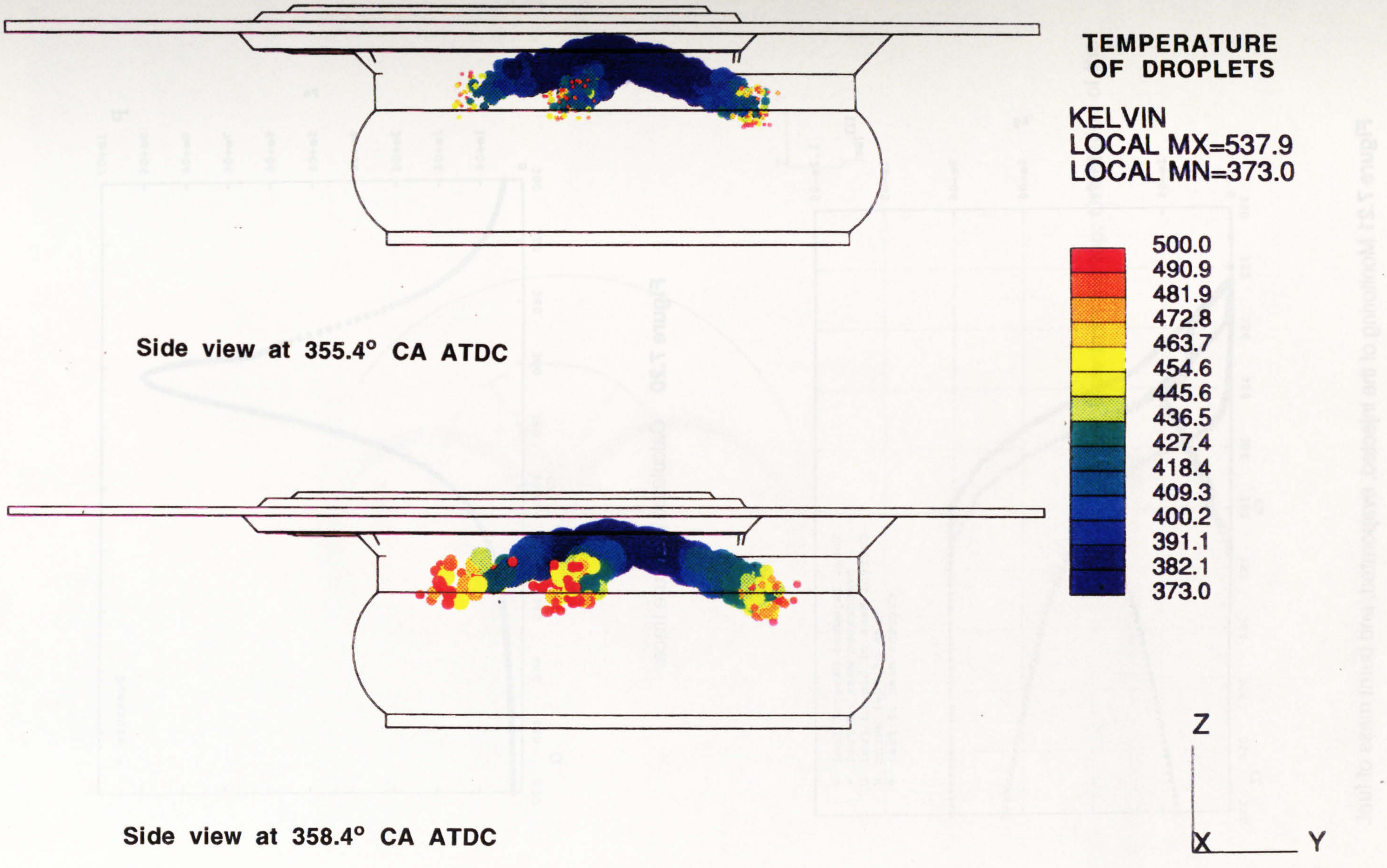


Figure 7.22 Side view of the fuel sprays at 355.4° and 358.4° CA ATDC.

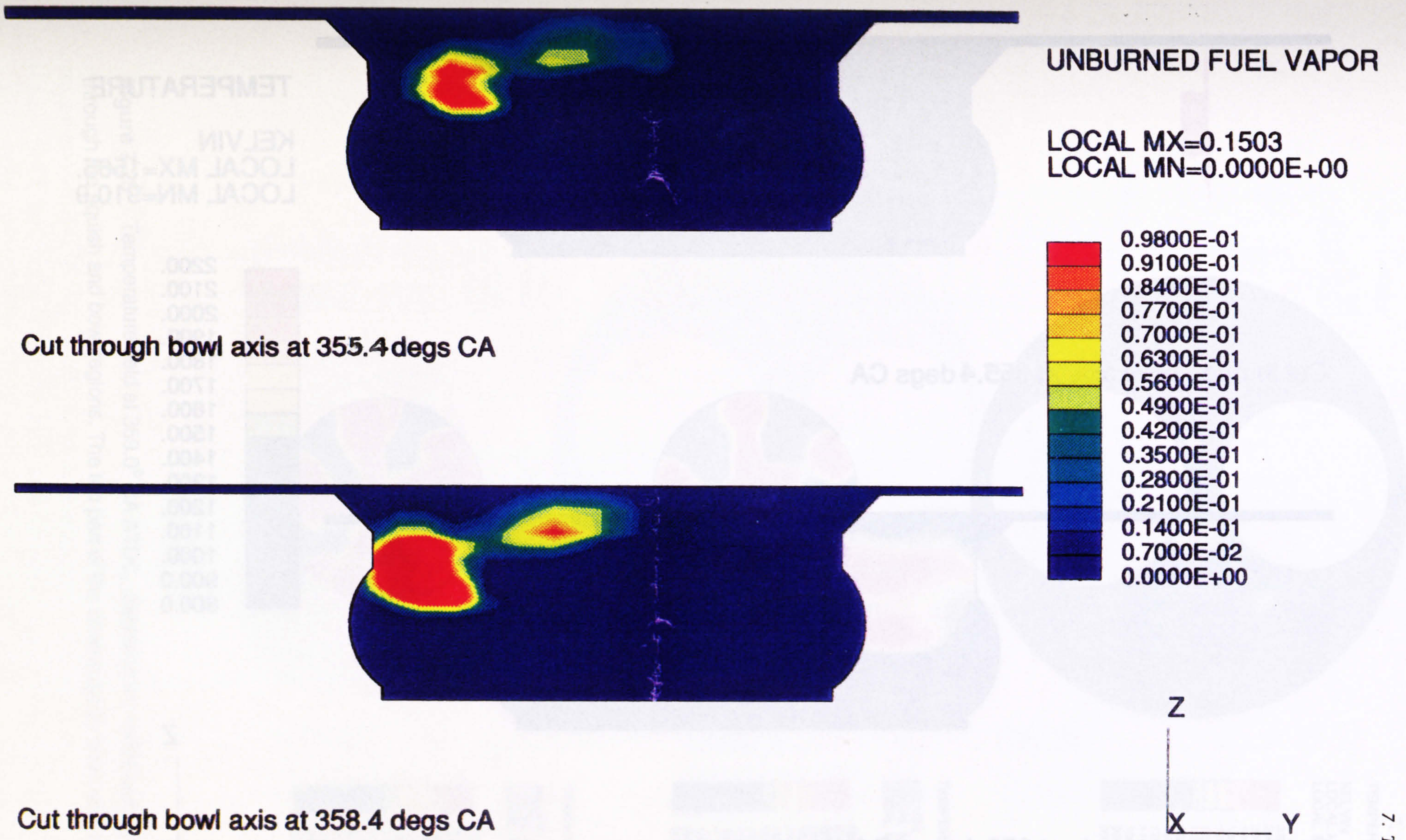


Figure 7.23 Unburnt fuel concentrations at 355.4° and 358.4° CA ATDC on axial cutting planes through the axis of the bowl.

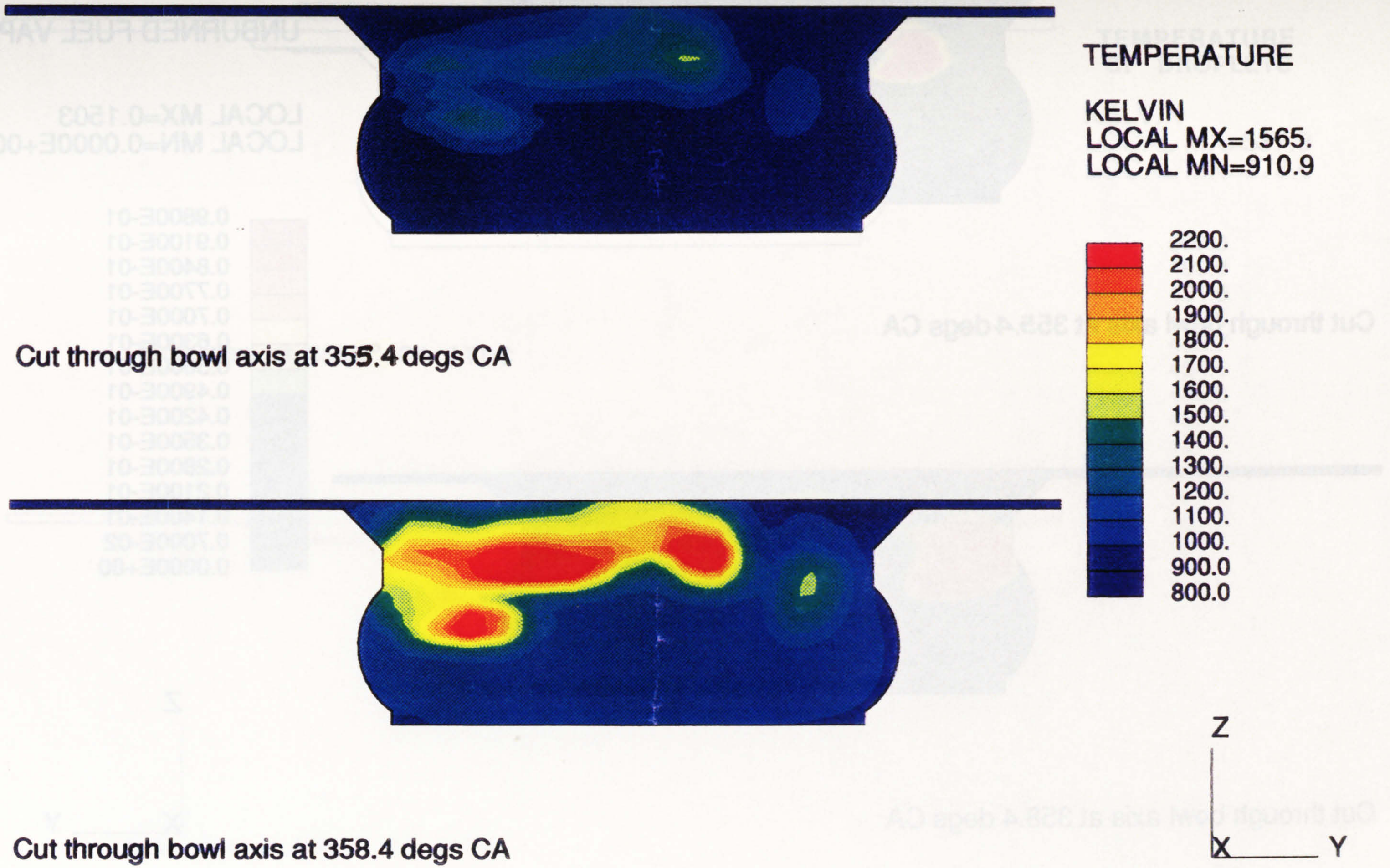


Figure 7.24 Temperature distribution at 355.4° and 358.4° CA ATDC on axial cutting planes through the axis of the bowl.

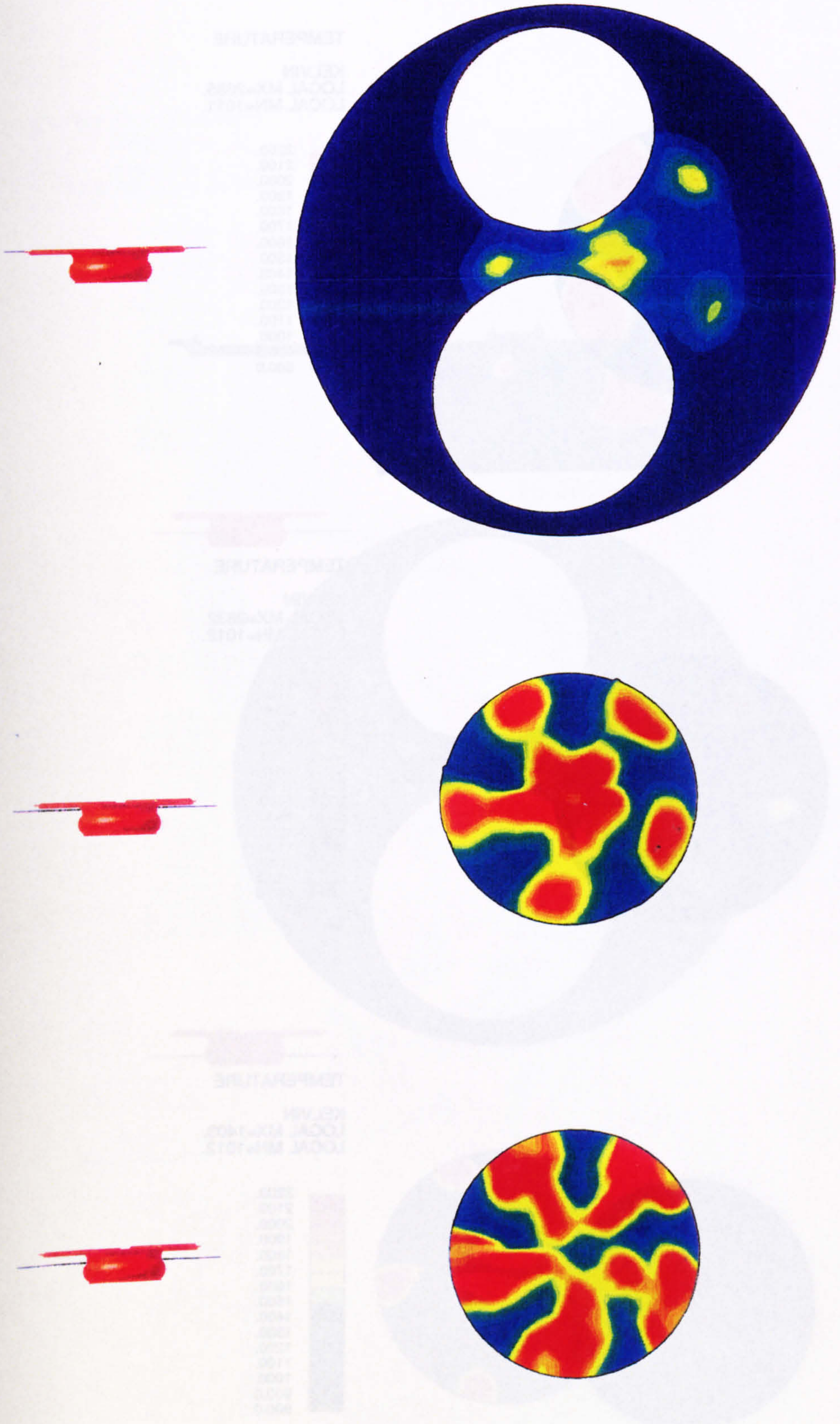


Figure 7.25 Temperature field at 359.0° CA ATDC, displayed on radial cutting planes through the squish and bowl regions. The top part of the bowl is partly filled with flame.

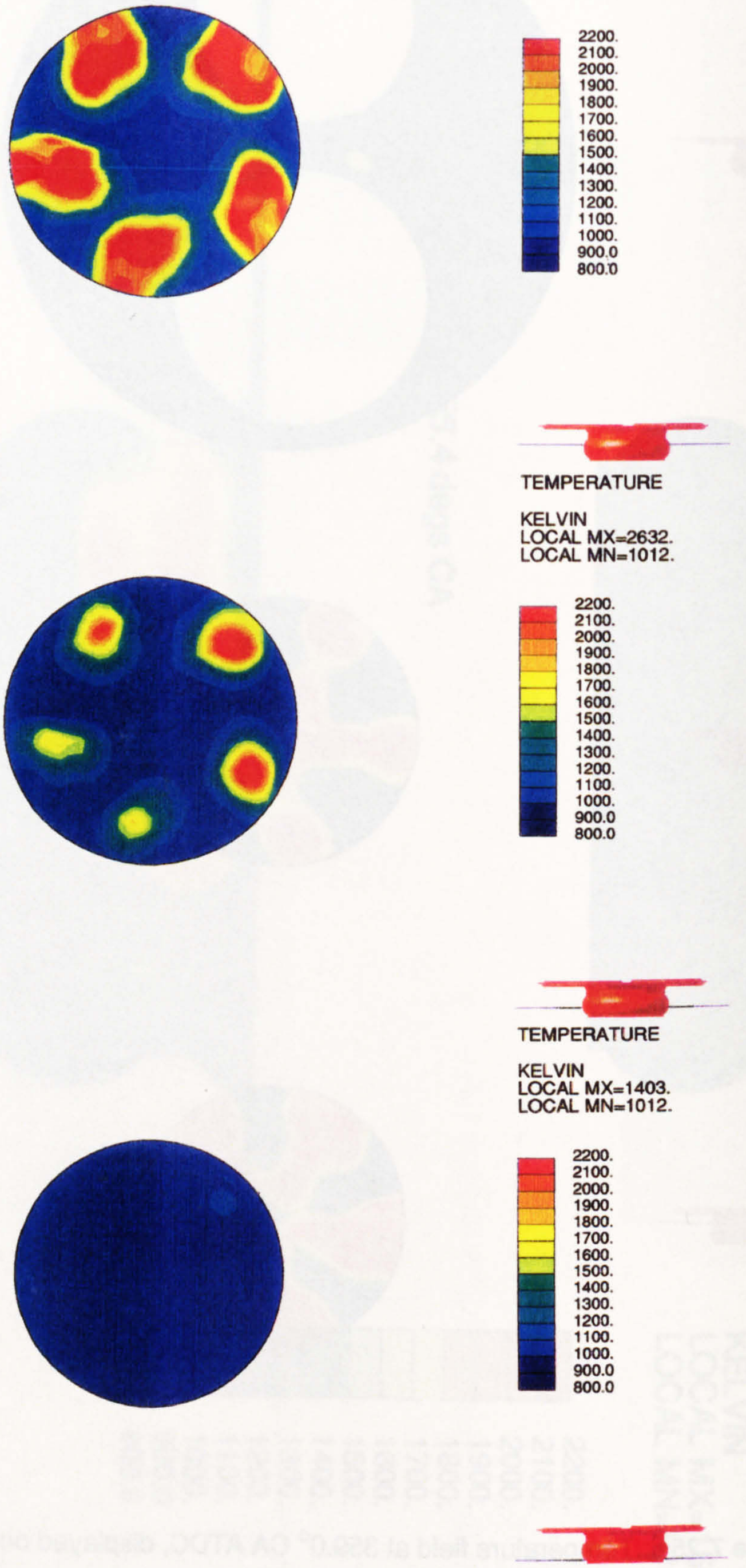


Figure 7.25 (continued) Temperature field at 359.0° CA ATDC, displayed on radial cutting planes through the squish and bowl regions. The top part of the bowl is filled with flame.

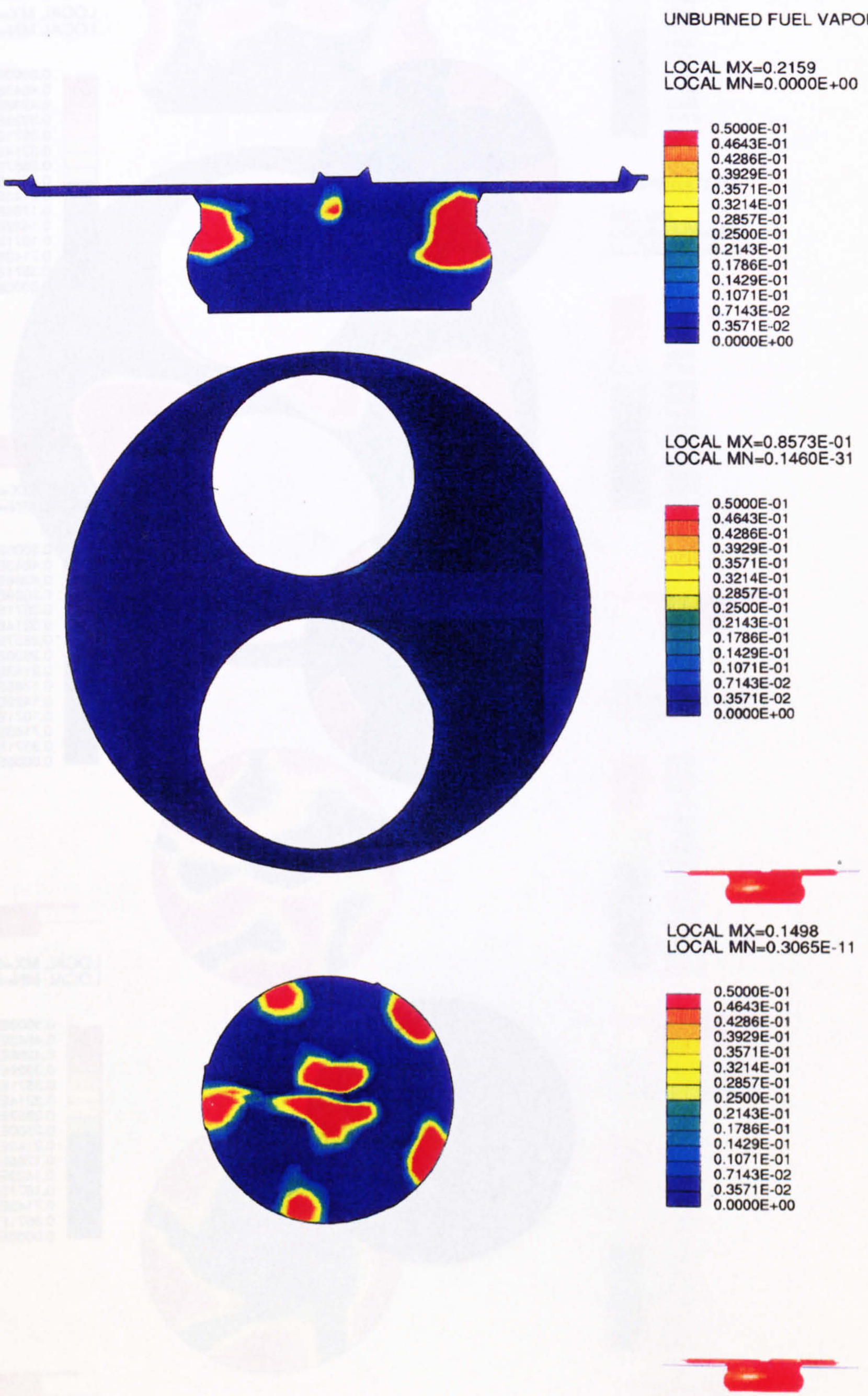


Figure 7.26 Unburnt fuel vapour concentration at 359.0° CA ATDC, displayed on radial cutting planes through the squish and bowl regions.

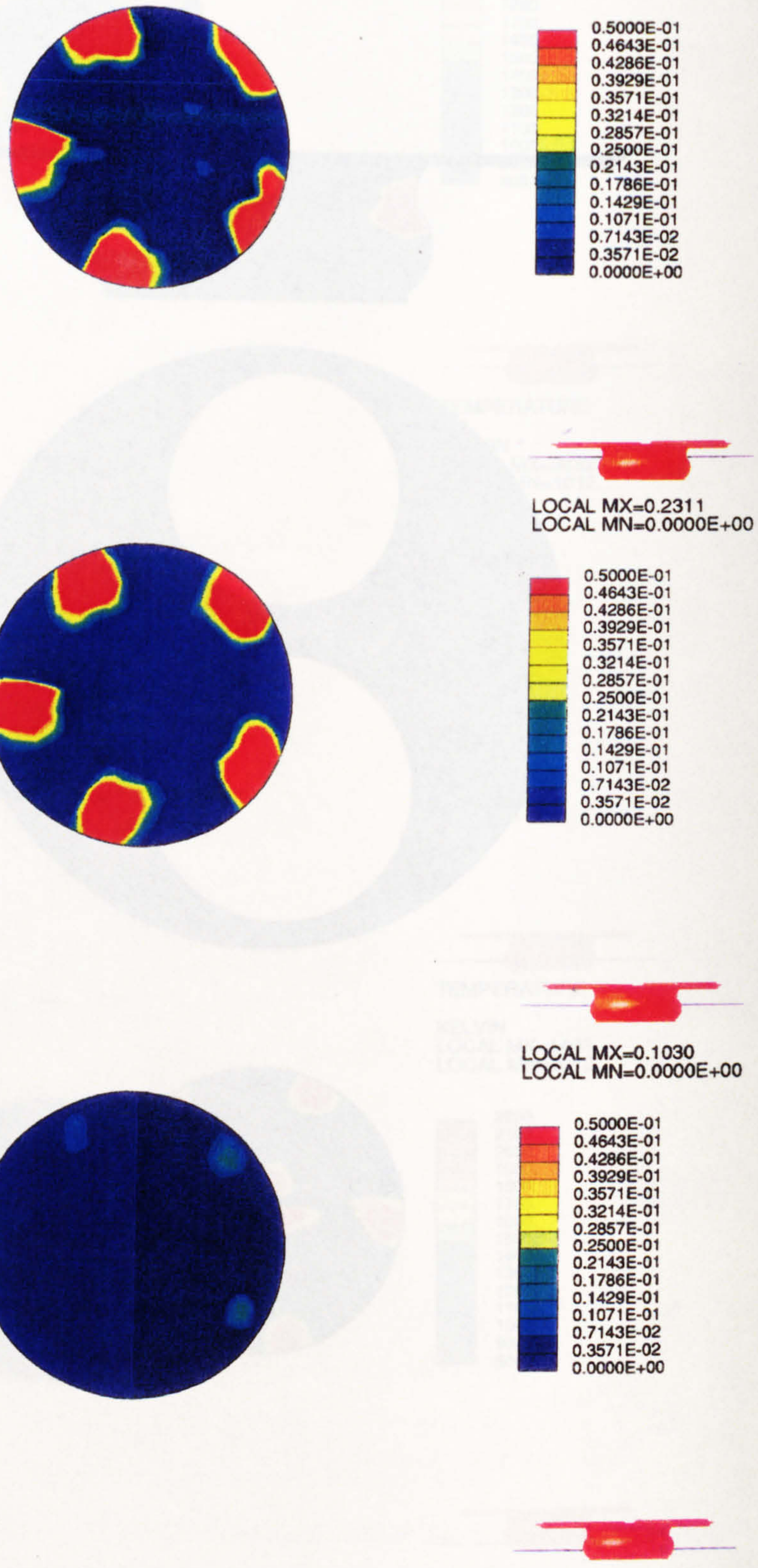


Figure 7.26 (continued) Unburnt fuel vapour concentration at 359.0° CA ATDC, displayed on radial cutting planes through the squish and bowl regions.

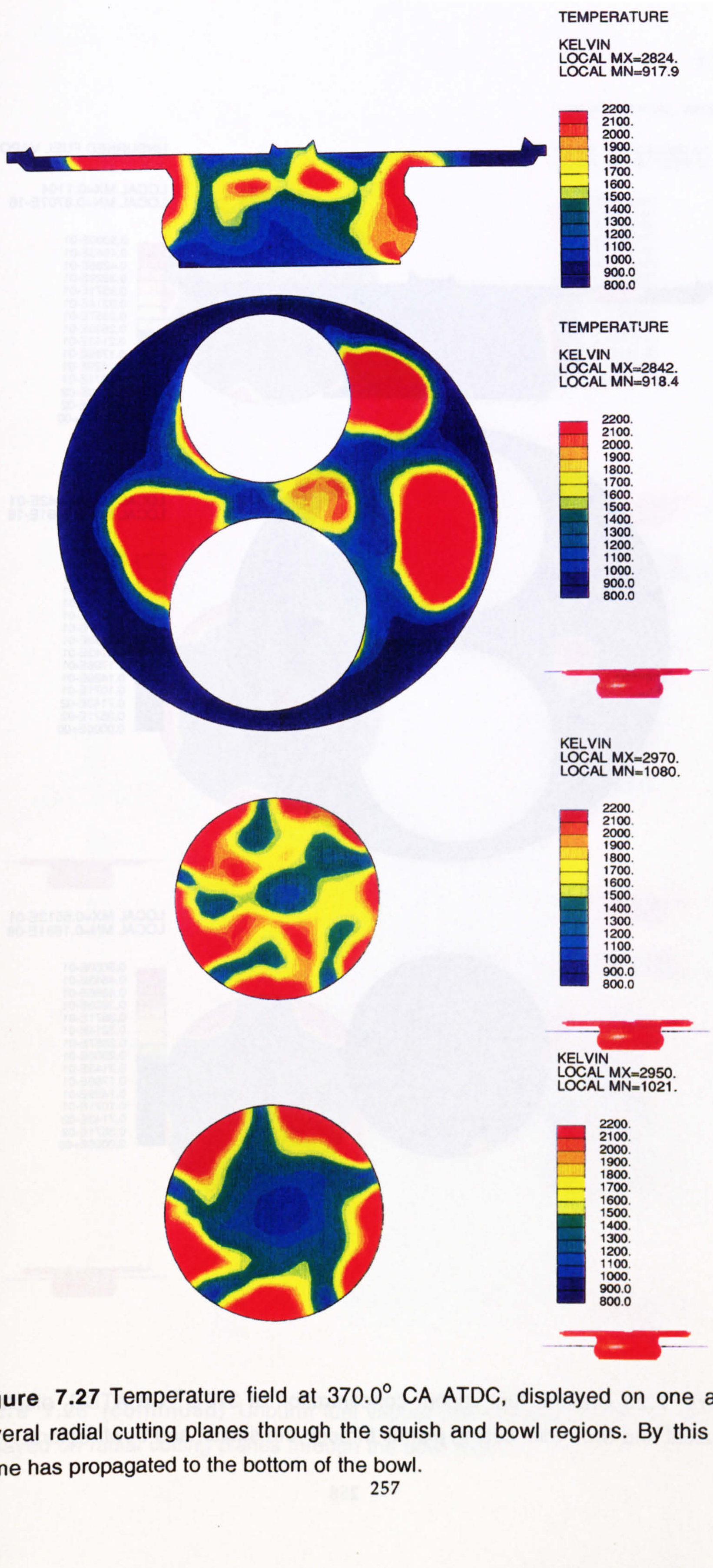


Figure 7.27 Temperature field at 370.0° CA ATDC, displayed on one axial and several radial cutting planes through the squish and bowl regions. By this time the flame has propagated to the bottom of the bowl.

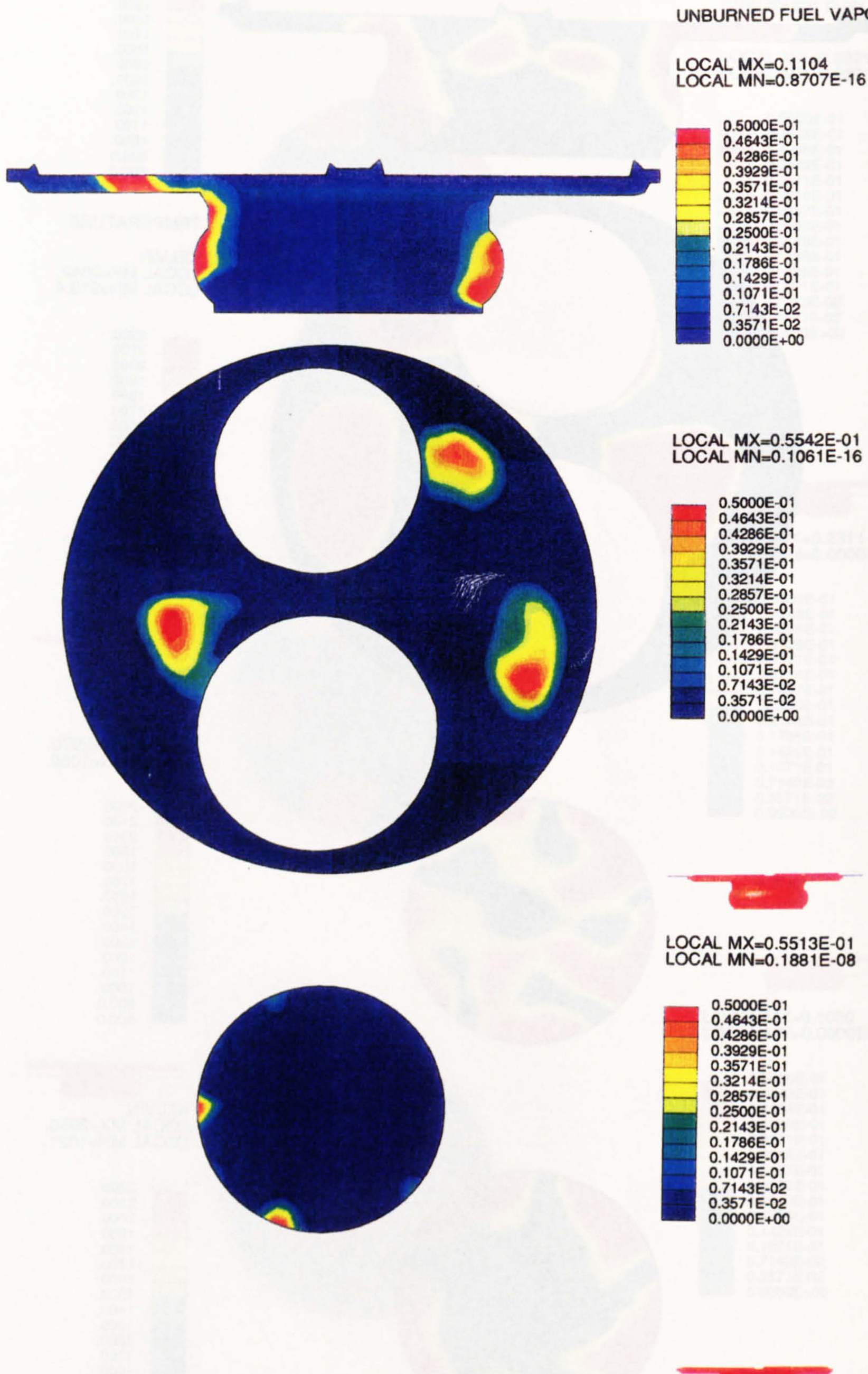


Figure 7.28 Unburnt fuel vapour concentration at 370.0° CA ATDC, displayed on one axial and two radial cutting planes through the squish and bowl regions.

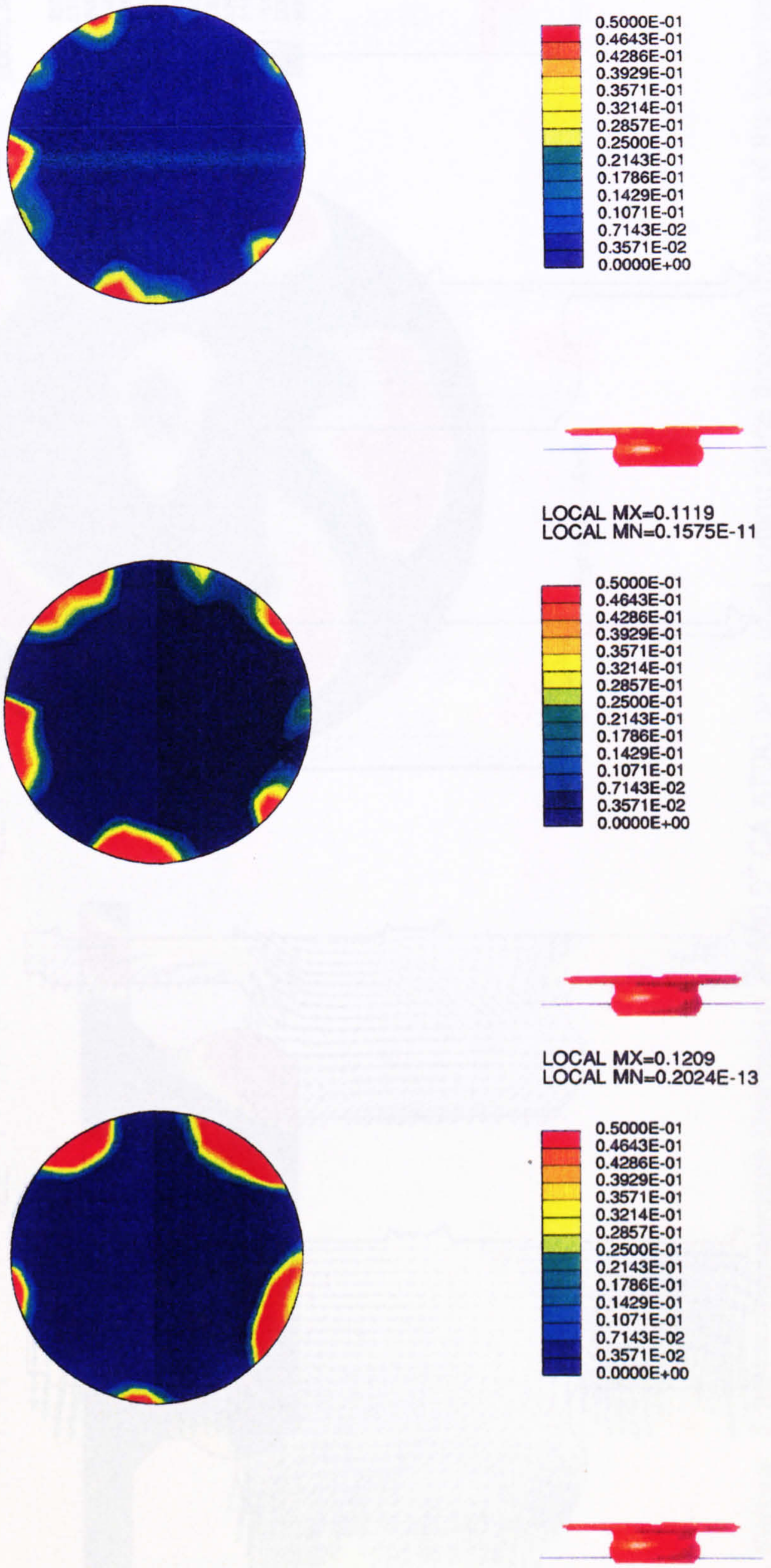


Figure 7.28 (continued) Unburnt fuel vapour concentration at 370.0° CA ATDC, displayed on radial cutting planes through the bowl region.

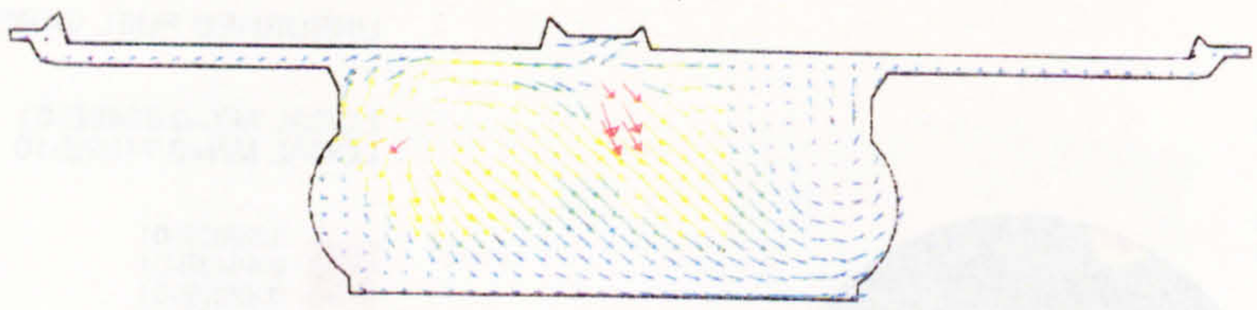


Figure 7.29 Gas flow velocity vectors at 357.2°, 358.4°, 361.2°, 380.0°, and 400.0° CA ATDC, displayed on axial cutting planes along the bowl axis.

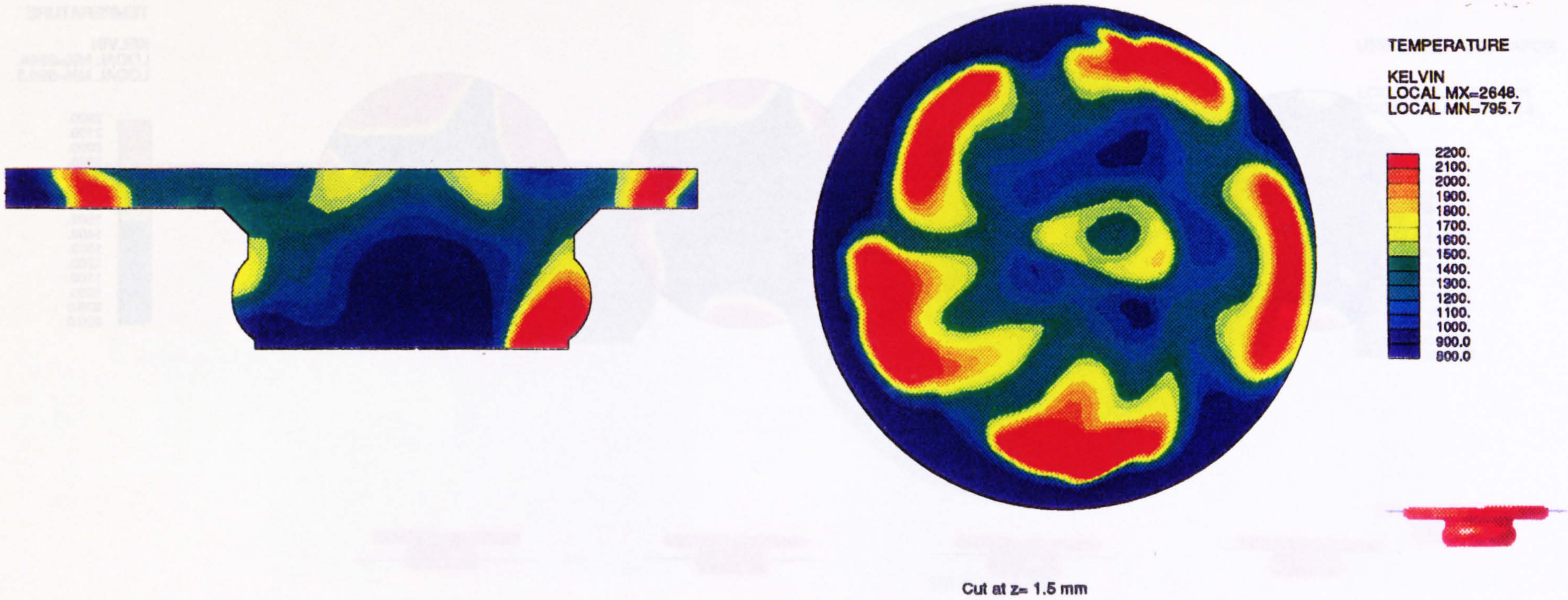


Figure 7.30 Temperature distribution at 380.0° CA ATDC on an axial cutting plane through the axis of the bowl (left) and on a radial cutting plane through the squish region (right).

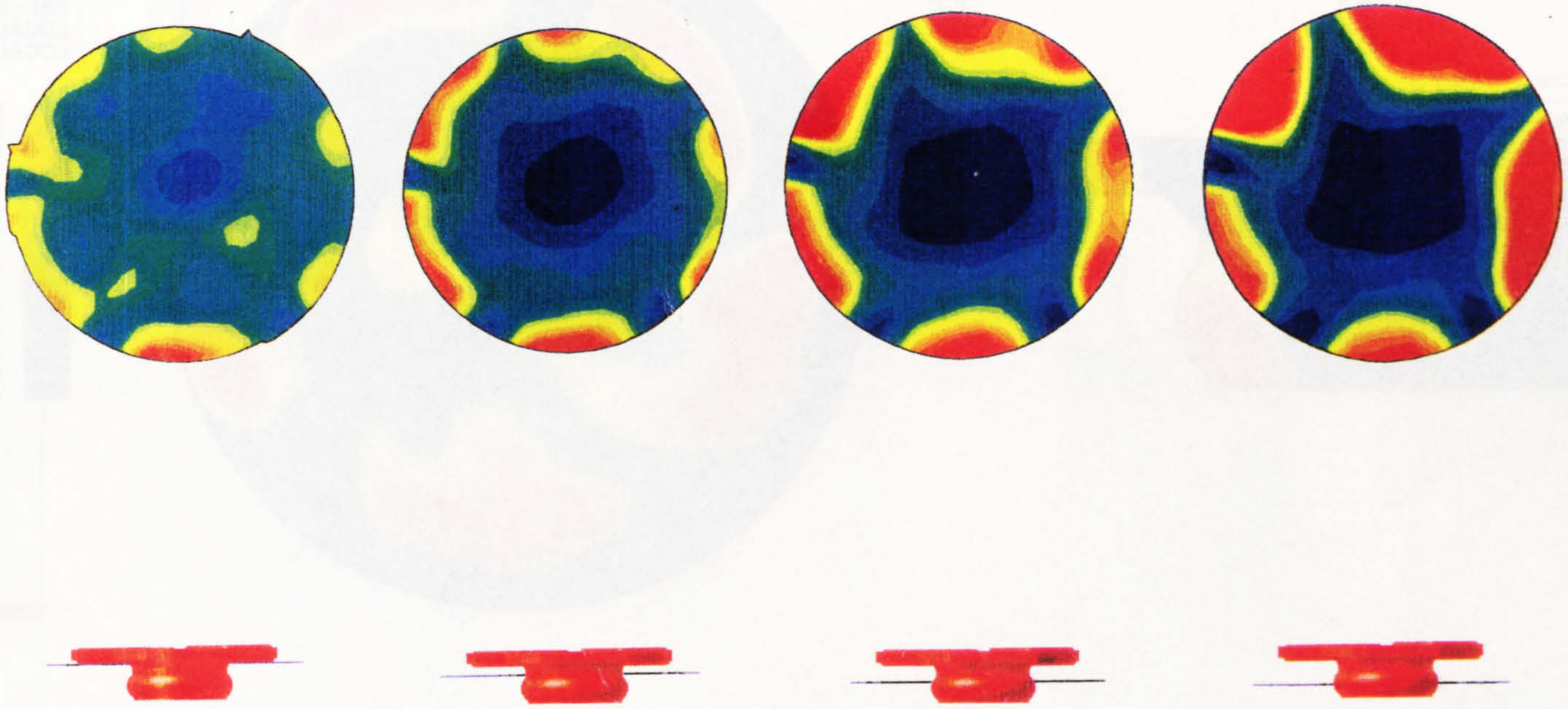


Figure 7.30 (continued) Temperature distribution at 380.0° CA ATDC on radial cutting plane through the bowl region.

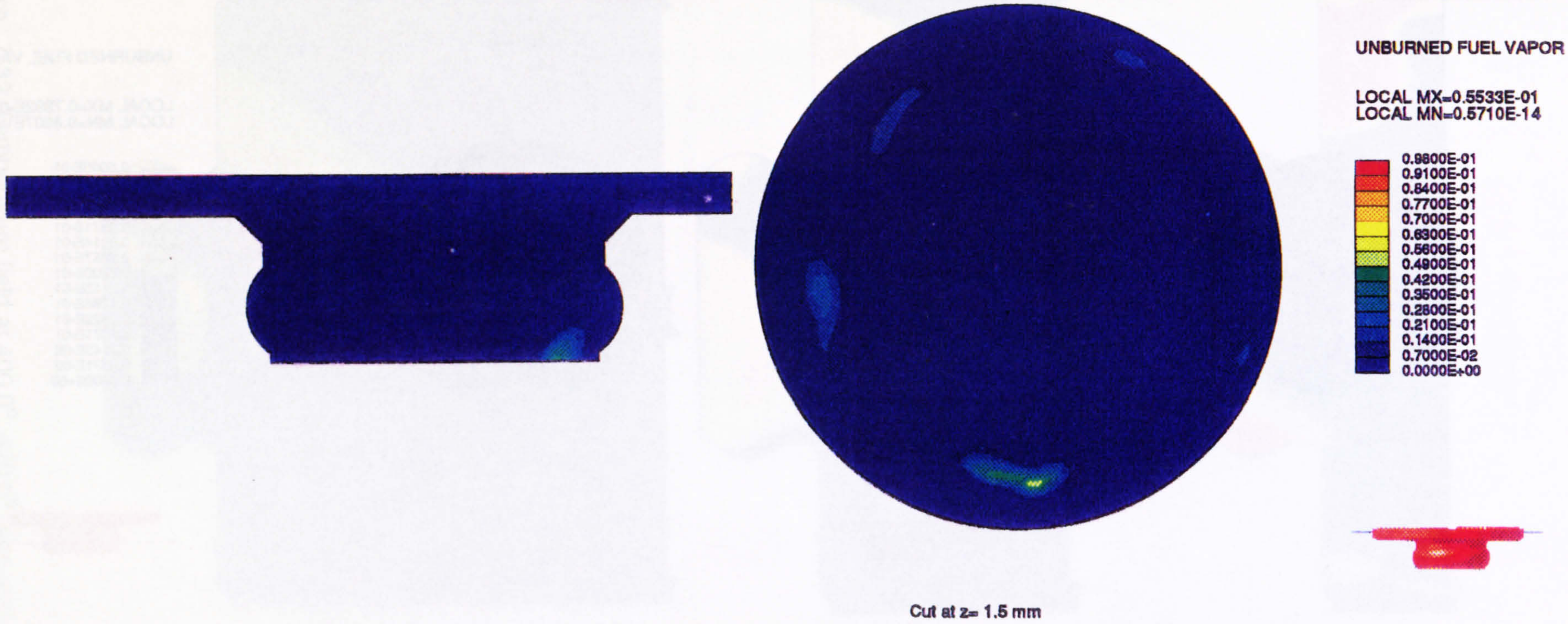
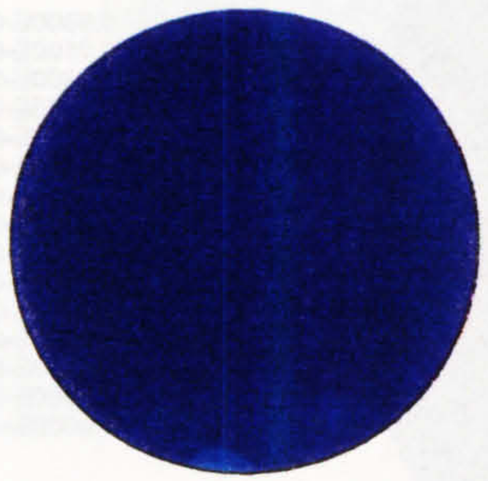
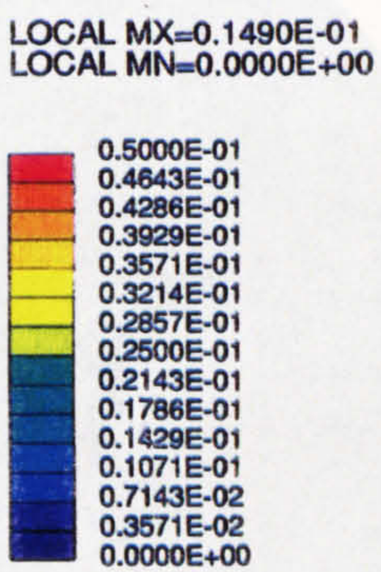
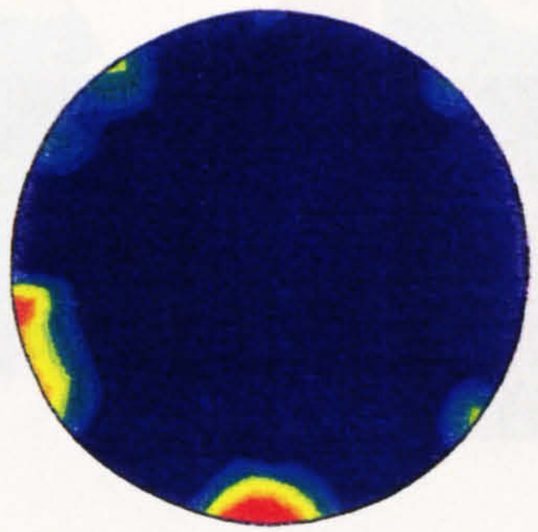


Figure 7.31 Unburnt fuel vapour concentrations at 380.0° CA ATDC on an axial cutting plane through the axis of the bowl (left) and on a radial cutting plane through the squish region (right).







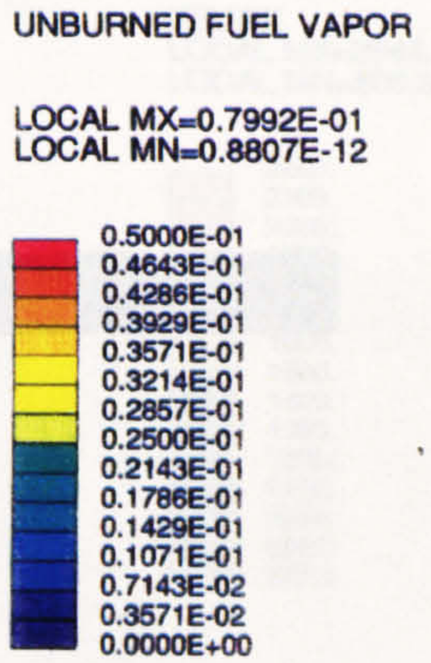




Figure 7.31 (continued) Unburnt fuel vapour concentrations at 380.0° CA ATDC on radial cutting plane through the bowl region.

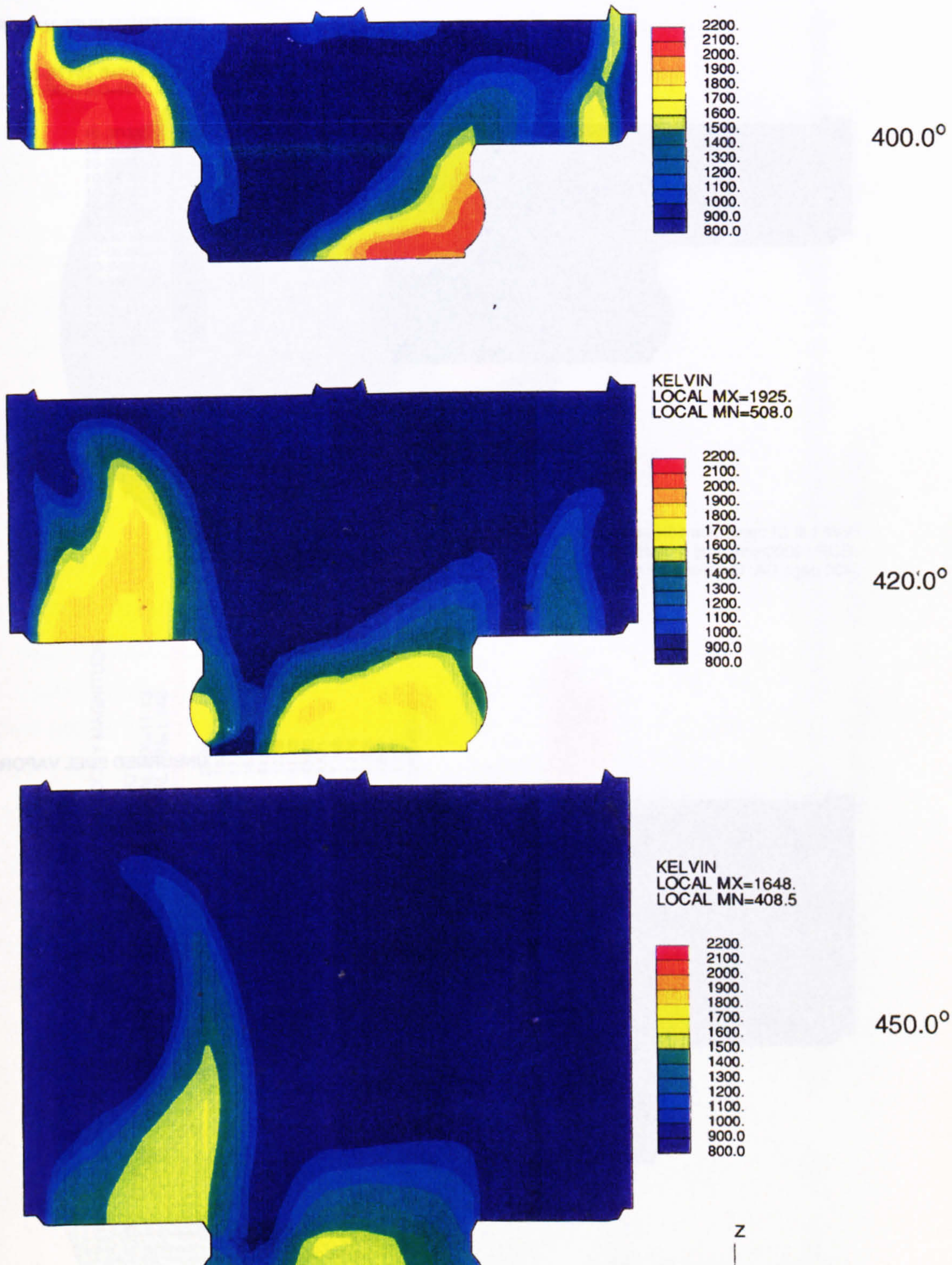


Figure 7.32 Temperature field at 400.0°, 420.0°, and 450.0° CA ATDC displayed on axial cutting planes along the bowl axis. There is a gradual diminution of the burning regions starting from the cylinder head.

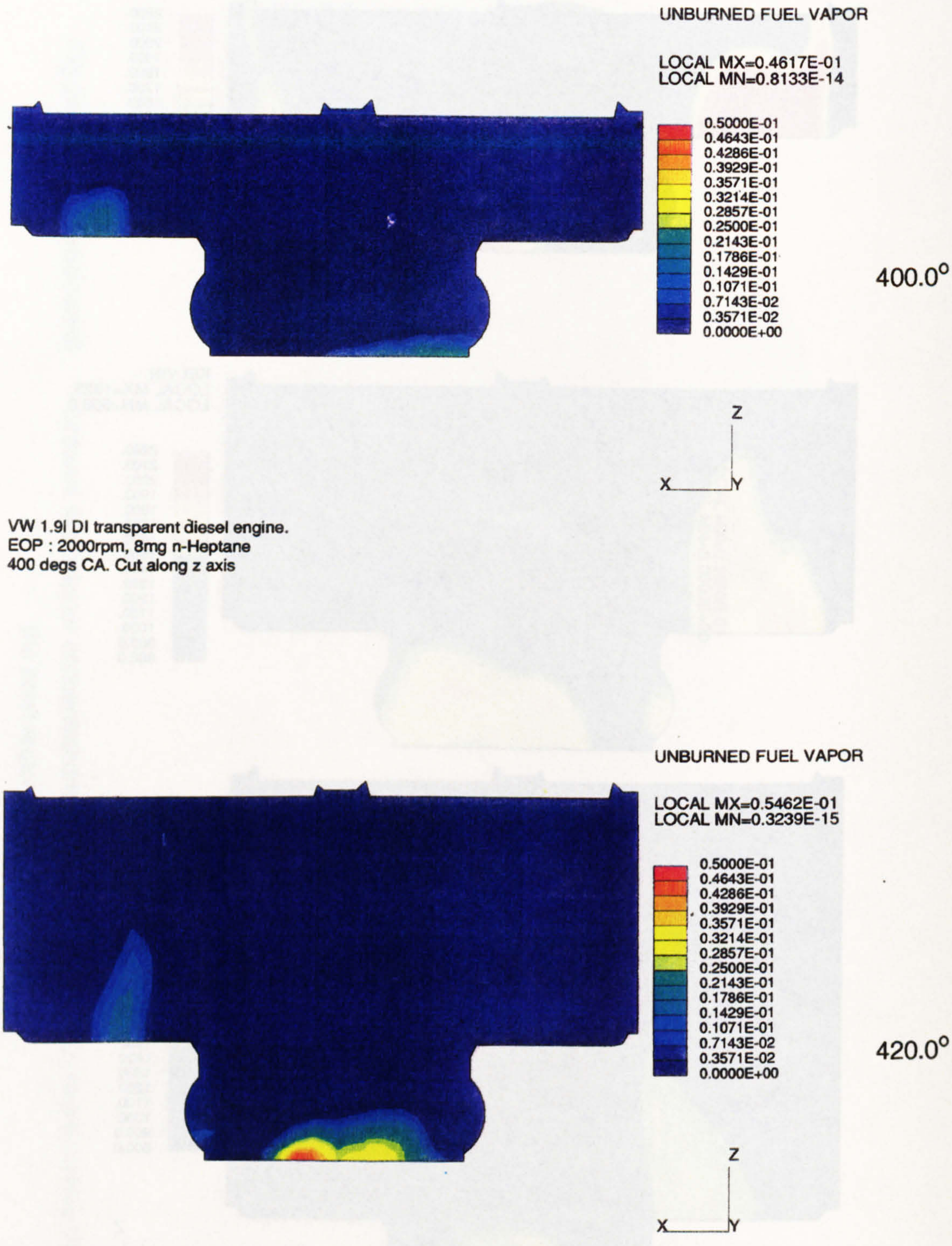
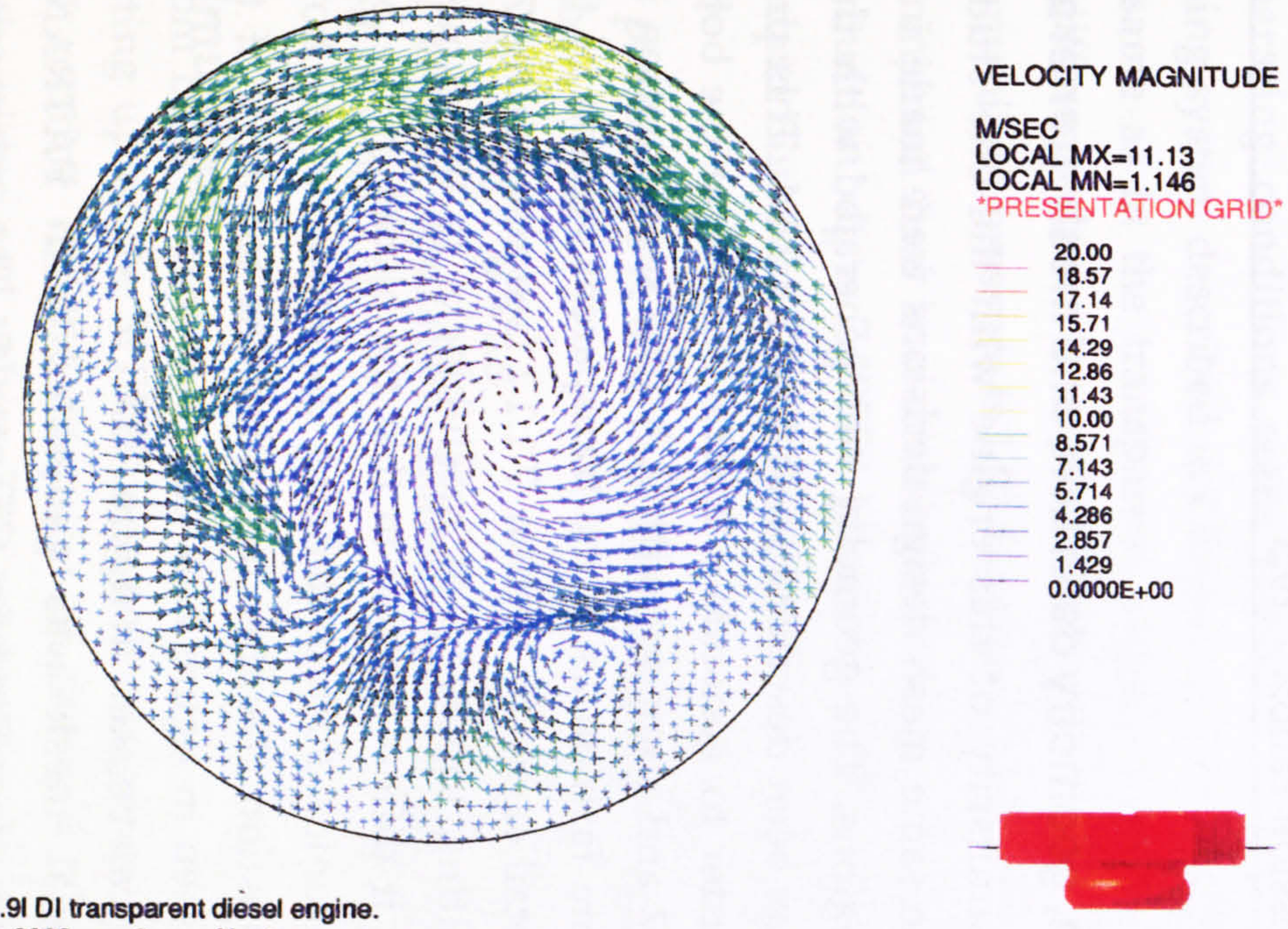
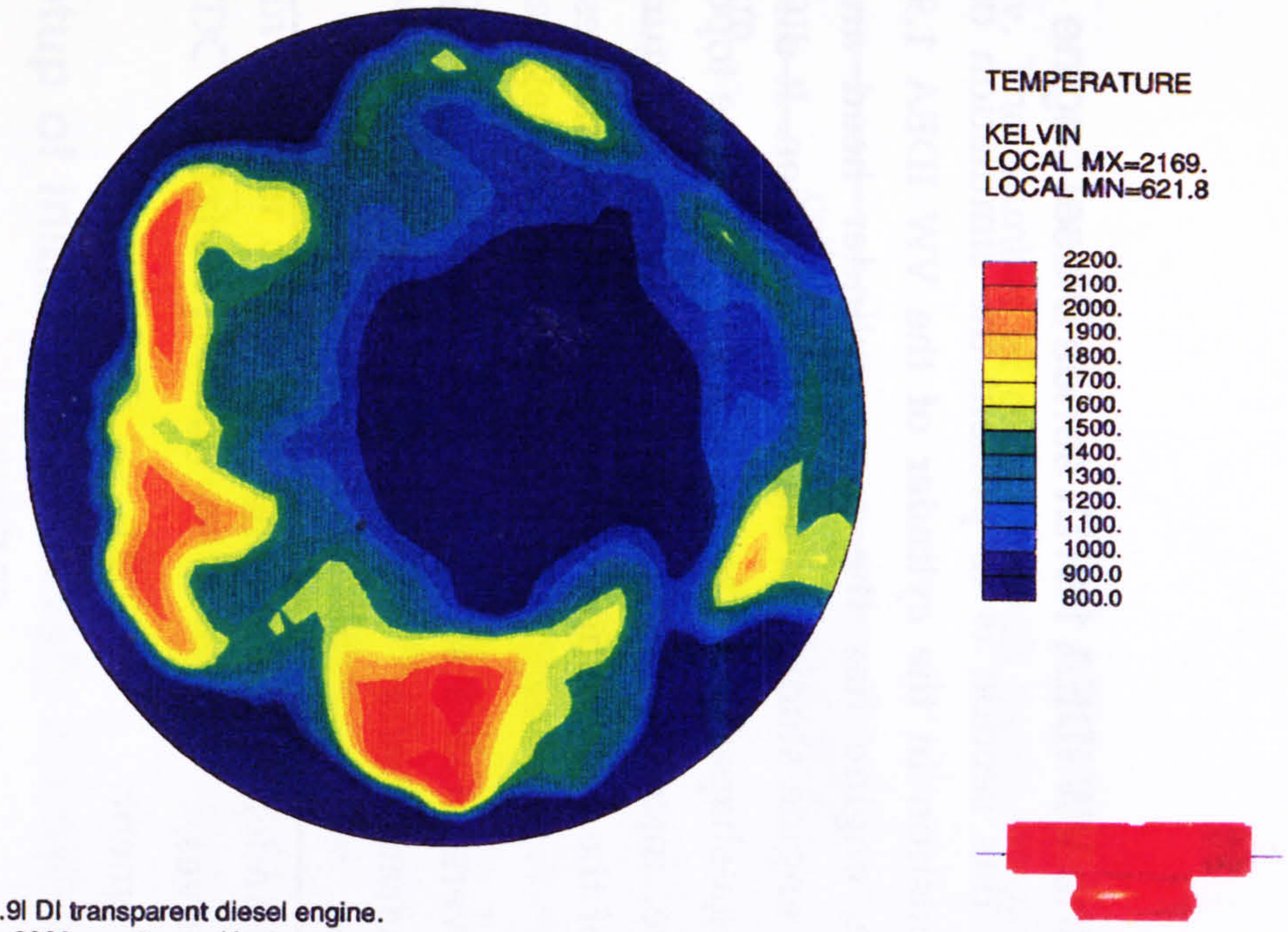


Figure 7.33 Unburnt fuel vapour concentration at 400.0° and 420.0° CA ATDC displayed on axial cutting planes along the bowl axis. The remaining fuel vapour is blown to the piston crown and bowl bottom regions.



VW 1.9l DI transparent diesel engine.
EOP : 2000rpm, 8mg n-Heptane
400 degs CA.



VW 1.9l DI transparent diesel engine.
EOP : 2000rpm, 8mg n-Heptane
400 degs CA. Top view.

Figure 7.34 Gas flow velocity vectors (left) and temperature distribution (right) on a radial cutting plane through the squish region at 400.0° CA ATDC.

7.2 Simulation of the VW IDEA 1.9l DI series Diesel engine

The objective in this section is to present the simulation of gas flow, injection, and combustion in the cylinder of the VW IDEA 1.9l DI series Diesel engine. This engine has the same cylinder head and piston configuration as the engine studied in the previous section. It differs from it in that it has an omega-shaped bowl recess in the piston. The topology of the cylinder head, valve, injector, and piston can be seen in figure 7.35.a. A schematic diagram of the combustion chamber at TDC of compression can be seen in figure 7.35.b.

This engine – a version of a series of Volkswagen production engines – has the following characteristics:

Four-cylinder in-line engine.	
Two valves.	
Displacement:	1.9 l.
Bore:	79.5 mm
Stroke:	95.5 mm
Compression ratio:	19.5

7.2.1 Mesh design, geometry definition, and mesh generation

Because of the similarity of this engine with the one studied in the previous chapter, the same mesh design decisions were made in the valve, squish, and bowl regions. The geometry was generated with the program 'GEOMETRY' that has been described in section 5.4.2.1.1. This program had to be modified in order to generate a conical surface at the bottom of the bowl (see figure 7.37 and compare with 7.7). The segmentation into blocks was done in the same fashion as in the transparent version of this engine, described in more detail earlier in section 7.1.1.1 (see also figures 7.4 and 7.5). Therefore, the resulting topology of mesh-blocks is the same as in the transparent version; it differs from it in that the blocks in the bowl region are bounded by the conical surface defining the bottom of the bowl. A solid model for the combustion chamber indicating the topology of the various mesh-blocks can be seen in figure 7.36. Gouraud-shaded solid models of the combustion chamber can be seen in figure 7.37.

The computational mesh was generated with PATRAN and the preprocessing system described in Chapter 5. The program 'MESH' – described in section 5.4.2.1.1 – was used to generate two meshes – a 'medium' and a 'fine' one – by setting the mesh-density factor to 2 and 3

respectively. The number of cells for each mesh are given in the following table:

Name	cells at TDC	cells at BDC	mesh density factor
Medium	4500	25400	2
Fine	21000	135000	3

The mesh generation process was completed by specification of wall and ghost boundaries in the fashion described in section 7.2.1.3. The mesh was checked for suitability for SPEED calculation and translated to the SPEED format by means of procedures provided in the preprocessing system described in Chapter 5. The 'medium' and 'fine' computational meshes for 120°CA ATDC crankshaft position can be seen in figures 7.38 and 7.39.

7.2.2 Setup of initial and boundary conditions

The initial and boundary conditions for the simulation as well as the engine operating conditions were specified in the 'problem file' of the preprocessing system described in Chapter 5. Most of the specified conditions were the same as in the transparent engine version, described in section 7.1.2. The differences from it are the following:

- The manifold pressure is 1.03 bar (instead of 1.0 bar)
- The injected fuel is n-dodecane (instead of n-heptane) with heat of combustion d_h = 7.395E9 J/kmol (or 43.5 MJ/kg) which is that of standard European Diesel fuel (density = 0.821 kg/l and cetane number = 55).
- The parameters A and B of the Magnussen combustion model were set to:

  A = 7.0 and B = 0.5

The main engine operation characteristics and setup conditions are summarised in table 7.3:

7.2.3 Simulation

The setting up of the SPEED code was completed with the creation of a FORTRAN routine for the adaption of the computational mesh to the motion of the piston and valves. This was accomplished in a fashion similar to the one described in section 7.1.3. A first analysis for motored conditions was made on the 'fine' mesh. Subsequently, combustion simulations were

performed on the 'medium' mesh. The full four-stroke cycle was calculated. The following models were used for simulation of the injection process (see section 3.2.2.2 and the user's manual of the SPEED code [173] for description and references thereof):

- Fuel injection and spray formation:
  - 'MPI' nozzle-flow cavitation model;
  - 'Gosman-Huh' atomisation model;
  - 'Reitz Diwakar' breakup model;
  - Droplet collision model;
  - 'MPI' droplet-wall interaction model;
- Modified 'Shell' ignition model;
- Modified 'Magnussen' combustion model.

7.2.4 Interpretation and visualisation of results

The flow patterns during induction and compression are very similar to the ones studied in the transparent engine. Towards the end of the compression stroke, swirl is substantially modified due to the squish-generated gas motion towards the bowl and the resulting vortex motion in the bowl following its omega shape (see the flow field on axial cutting planes at 330° and 350° CA ATDC in figure 7.40 and schematic diagram of figure 7.41).

Fuel is injected from 352.4° CA, directly into the bowl region in the form of five sprays, the directions of which can be seen in figure 7.19. The average pressure in the cylinder at this time is 53 bar (see figure 7.42). Due to the difference in compression ratio, this is remarkably lower than in the case of the transparent engine (63 bar, see figure 7.20). As can be seen in figure 7.43, the presence of the sprays cools down the gas in the region of the injector before the start of combustion.

Evaporation of fuel starts almost immediately after injection, albeit at a low rate. The evaporation rate increases noticeably at about 3.5° CA after the start of injection (see figure 7.44). As in the case of the transparent engine, two centres of evaporation are noticed for each spray: one near the injector and one near the tip of the spray (see figure 7.45).

As can be seen in figure 7.46, 6.0° CA after the start of injection the fuel sprays reach the wall of the bowl lip. This is in agreement with experimental experience for the same engine (Hentschel [242]), and different from the transparent engine case, where the n-heptane fuel does not reach the wall. The spray penetration at selected crank angle positions is given in the

following table. The peak droplet velocities are of the order of 136 m/s. The droplets at the spray tips have been slightly deflected by the swirl. Near the bowl walls the fuel vapour has been blown around by the swirl (see figures 34 and 35).

Table 7.2

Spray penetration at selected crank angle positions

Crank shaft position (° CA ATDC)	Penetration (mm)
353.5°	11.86
356.4°	18.12
385.5°	19.95
360.0°	21.25

Combustion starts 5° CA after the start of injection. This is deduced from the deviation of the combustion pressure trace from the one obtained under motored conditions shown in figure 7.44. As can be seen in the same figure, the calculated start of combustion is less than 0.5° CA after the measured one. A number of hot centres are present. At the first stages of combustion these are located near the injector and near the tips of the sprays (see figure 7.45), where the fuel-air mixture is stoichiometric (corresponding fuel vapour mass fraction = 0.063)

The fact that ignition delay is longer than that of n-heptane, and that n-dodecane droplets hit the wall whereas n-heptane ones do not can be explained by:

i. The slower evaporation of n-dodecane in comparison to n-heptane. This is so because the boiling point for n-dodecane (216.3° C) is more than twice that of n-heptane (98.4° C).

ii. The lower cylinder pressure and temperature (due to the lower compression ratio) during injection in the case of the series engine.

Burning spreads rapidly along the axis of the sprays and covers the top part of the bowl. As in the case of the transparent engine, after TDC the flame spreads towards the bottom of the bowl and out over the piston crown towards the cylinder wall due to combustion-produced gas expansion and the reverse-squish flow. The temperature distribution at 370° CA ATDC can be seen in figure 7.47. Now burning mainly takes place in the squish region, and near the bowl walls. As the piston speed increases the mixture is blown to the piston crown and the bottom of the bowl.

The peak combustion pressure is 74 bar. This is deduced from the calculated pressure trace from 270° to 450° CA ATDC given in figure 7.42.

The measured one for Diesel fuel (taken from Lepperhof [244]) is given in the same figure. As it can be seen in the figure, the calculated ignition delay and the rate of pressure rise during the early (premixed) phase of combustion is in good agreement with the measured one. The calculated peak pressure is 4.5 bar (6%) higher than the measured one, a result that is mainly attributed to the combustion model used.

The filtering procedures of the system described in Chapter 5 were used to extract additional information from the results of the simulation. These were the volumetric efficiency, the mean indicated pressure, and the specific fuel consumption. Their definitions are given in section 4.3.1.4. Their values for the above simulation are:

Volumetric efficiency (λ)	86.2%
Mean indicated pressure (p_i)	3.45 bar
Specific fuel consumption (b_i)	176.3 g/kWh

The above are in good agreement with the manufacturers specifications [187] for the specified operating conditions and Diesel fuel.

Additional filtering operations, described in section 4.3.1.4, were used for the calculation of the angular momentum and the swirl number. The angular momentum exhibits a rapid build-up to a peak due to the inflow of angular momentum through the inlet valve and then a gradual decay thereafter due to the viscous shear stresses acting at the boundaries (see figure 7.48). During early intake stages the inclination of the angular momentum (with respect to the axis of the cylinder) is from 140° to 35°, whereas after 50° CA ATDC it stabilises between values 35° and 5°, indicating the formation of a solid body rotating field (see figure 7.49).

The swirl ratio calculated in two different ways, one by assuming a solid body rotating disk and one with respect to the angular momentum axis (see section 4.3.1.4 for definitions), is given in plots 7.50 and 7.51 respectively). When calculated in the former way, the peak swirl ratio is 3. Also, during compression, when the mass of air in the cylinder remains constant, the behaviour of the swirl ratio follows that of the angular momentum.

The main differences between the results for the transparent and series versions of the VW 1.9L DI engine are summarised in the table 7.4. It should be noted that apart from the geometry of the bowl and the type of fuel, the two engines differ in the compression ratio, which in turn has an effect on the cylinder pressure.

Table 7.3

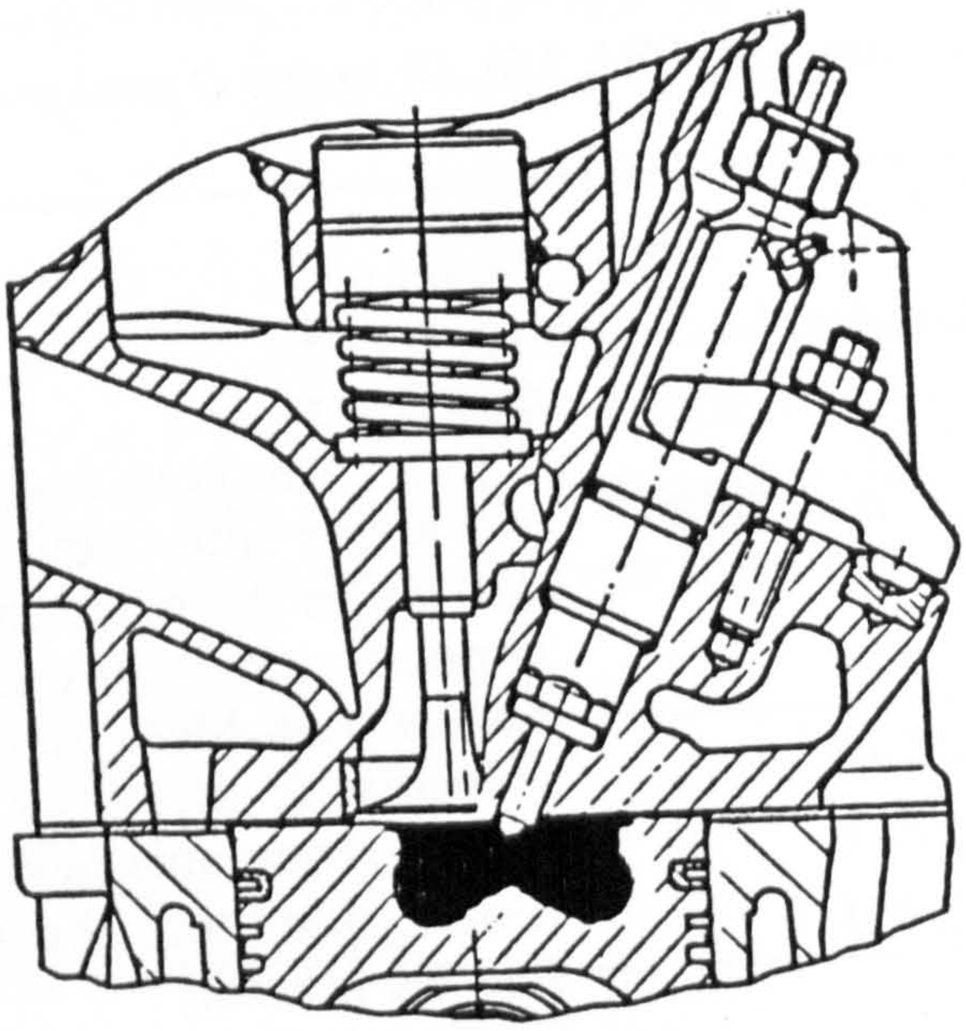
Main engine operation conditions for the simulation of gas flow and combustion in the VW IDEA 1.9L DI series Diesel engine

Compression ratio	19.5
Stroke	95.5 mm
Connecting rod length	144.0 mm
Engine speed	2000 rpm
Pressure in manifold	1.03 bar (for both manifolds)
Air temperature in manifold	300 K (for both manifolds)
Intake flow	Air enters the cylinder with initial angular momentum generated by a helical port. Measurements of gas velocities at the valve curtain were taken from Arcoumanis [243].
Fuel	n-dodecane (heat of combustion d_h = 7.395E9 J/kmol)
Injector	5 hole injector provided by Bosch
Nozzle length/diameter	5.38
Nozzle discharge coefficient	0.7
Temperature of injected fuel	373 K
Diameter of injector hole	0.21 mm
Injected parcels per second	964000 for each hole
Total injected quantity	8 mg
Start of injection	352.5° CA ATDC
End of injection	360.0° CA ATDC

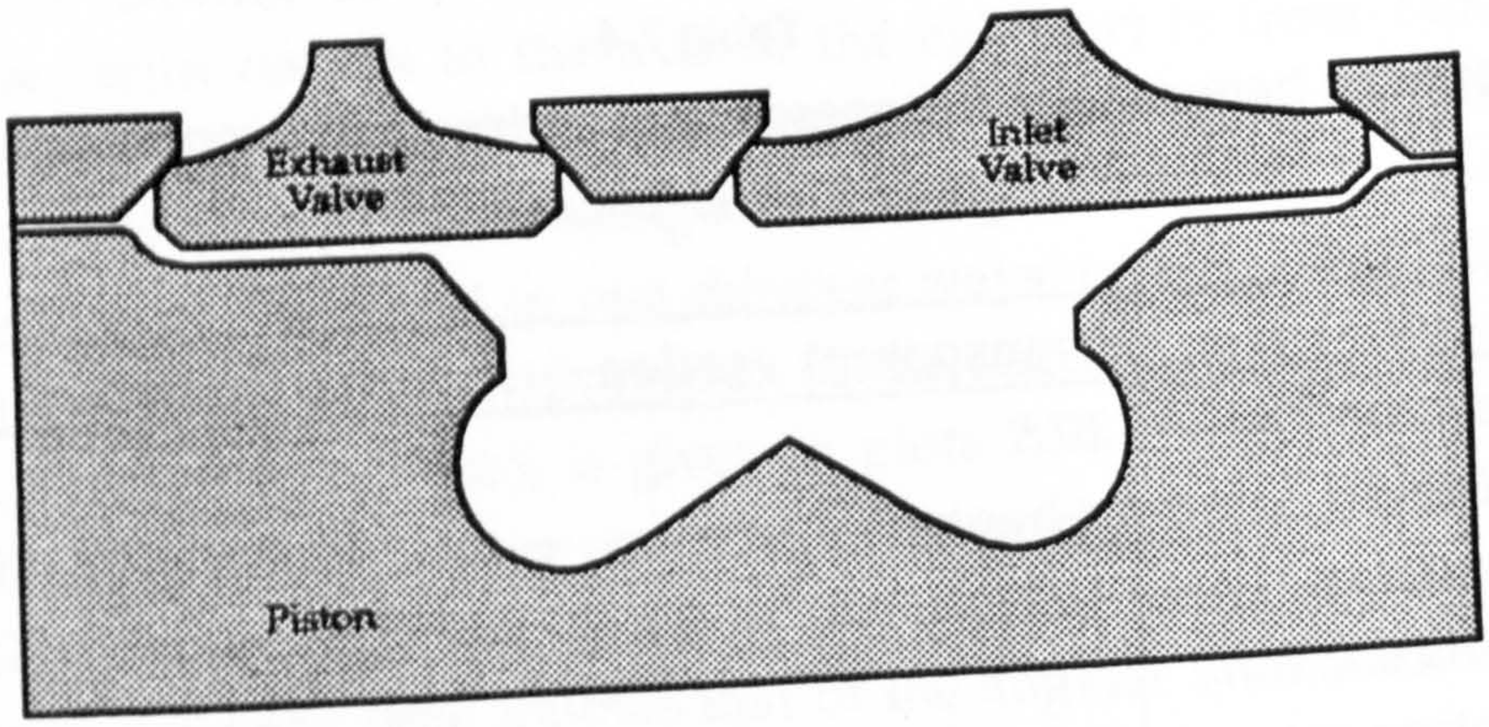
Table 7.4

Comparison between the transparent and series versions of the VW 1.9L DI Diesel engine simulations

	Transparent version	Series version
Compression ratio	20.5	19.5
Type of fuel	n-heptane	n-dodecane
Ignition delay	4.2° CA	5° CA
Spray hits the wall	No	Yes
Evaporation rate	Fast (all fuel evaporated 8.8° CA after injection start (see fig. 7.21)	Normal (all fuel evaporated 10.5° CA after injection start (see fig. 7.44)
Burning rate	Fast during early combustion (4 mg burnt by TDC see fig. 7.21)	Normal throughout combustion (1 mg burnt by TDC see figure 7.44)
Peak comb. pressure	74 bar	91 bar



(a)



(b)

Figure 7.35 (a) Elevation view of the VW IDEA 1.9L DI series Diesel engine with a cutting plane along the axis of the bowl. It features an omega shaped recess in the piston. (b) Schematic diagram of the combustion chamber.

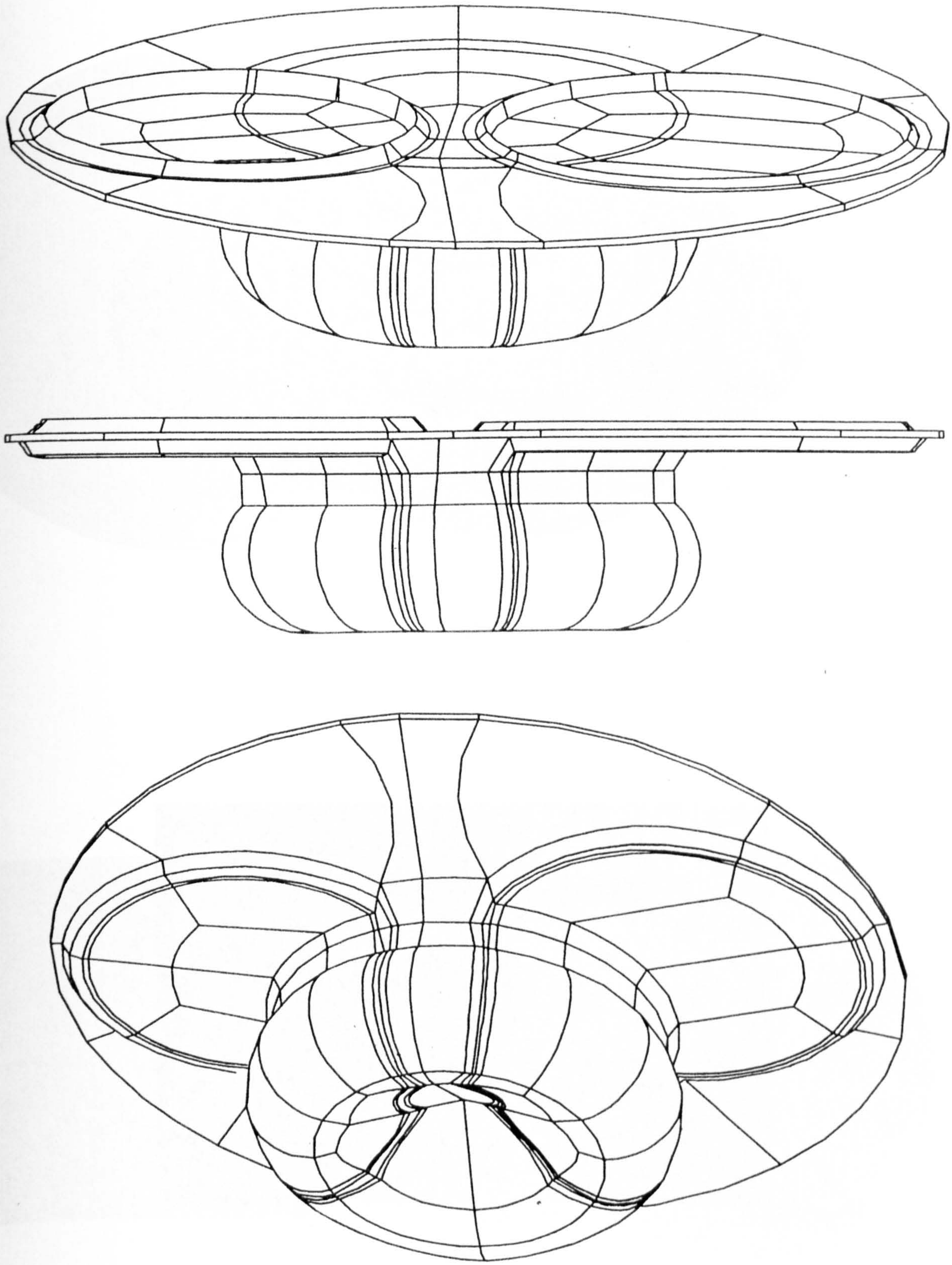
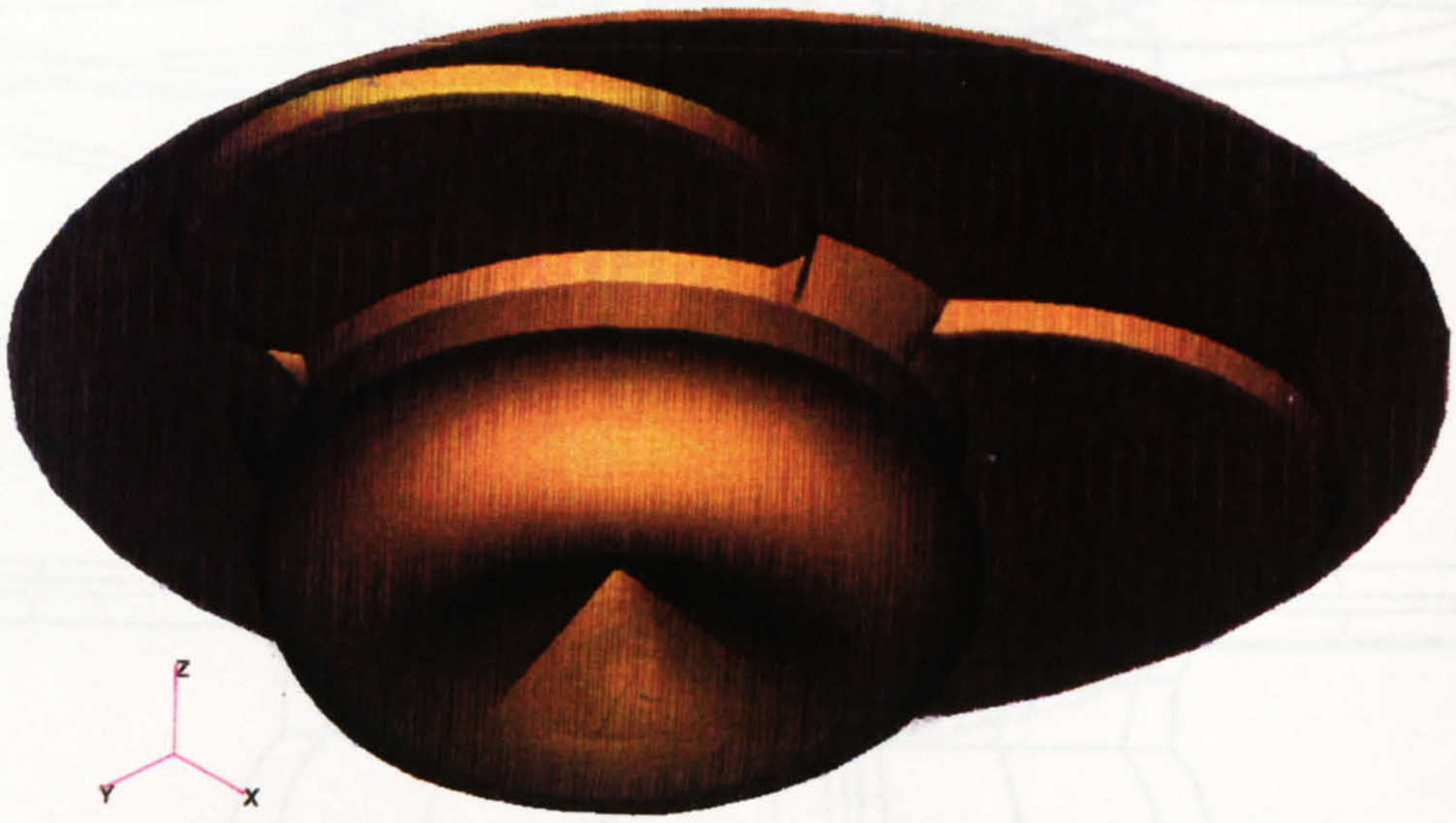
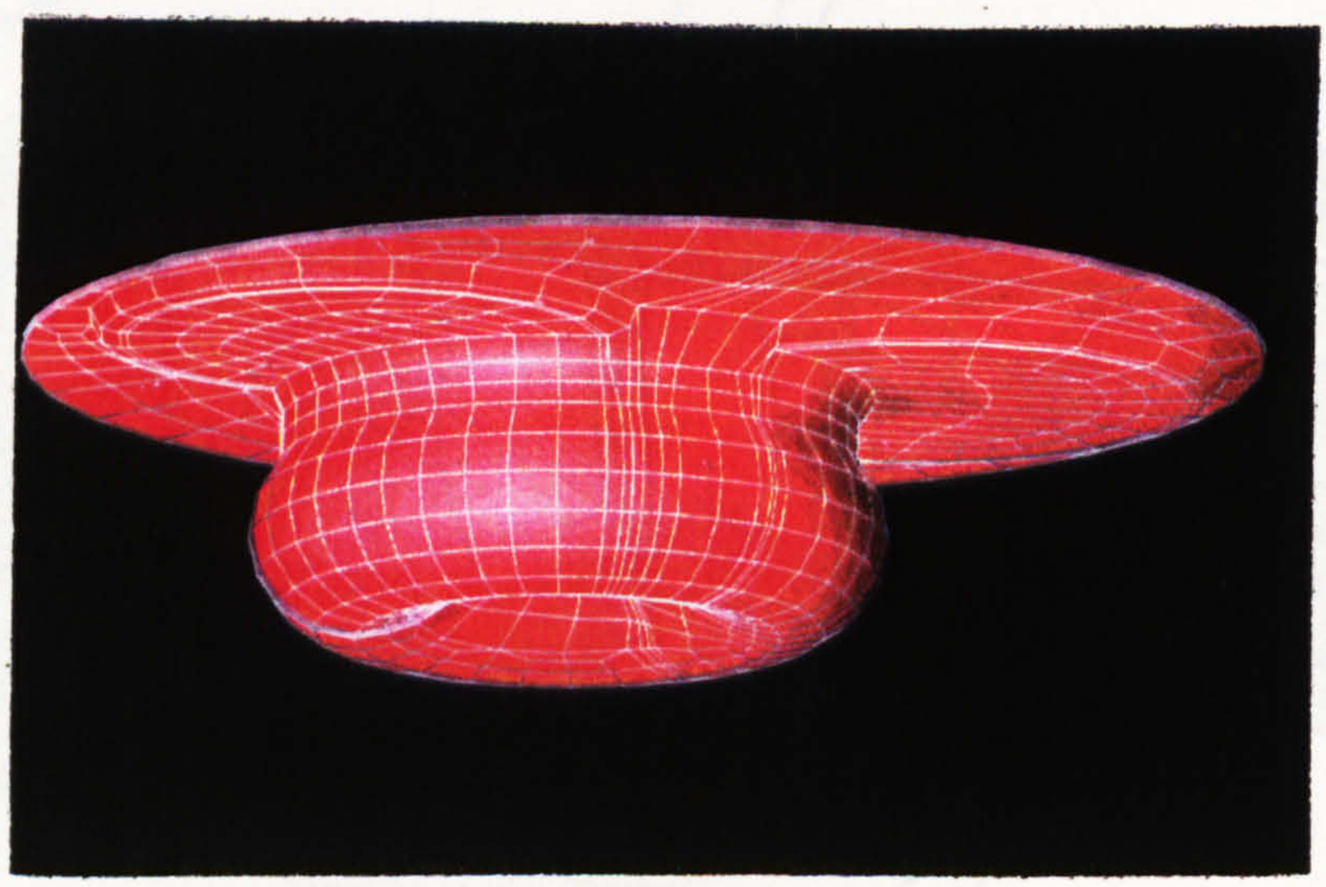


Figure 7.36 Solid model of the VW IDEA 1.9L DI series Diesel engine, indicating the topology of the mesh-blocks. Top: side view delineating the block topology at the cylinder head. Middle: elevation. Bottom: Side view showing the piston head region.



(a)



(b)

Figure 7.37 (a) Gouraud-shaded solid model of the combustion chamber of the VW IDEA 1.9L DI series engine. (b) Same as (a) but with superimposed mesh lines.



Figure 7.38 VW series engine. 'Medium' mesh at 120° CA ATDC. Bottom right: cutting section along the cylinder axis.



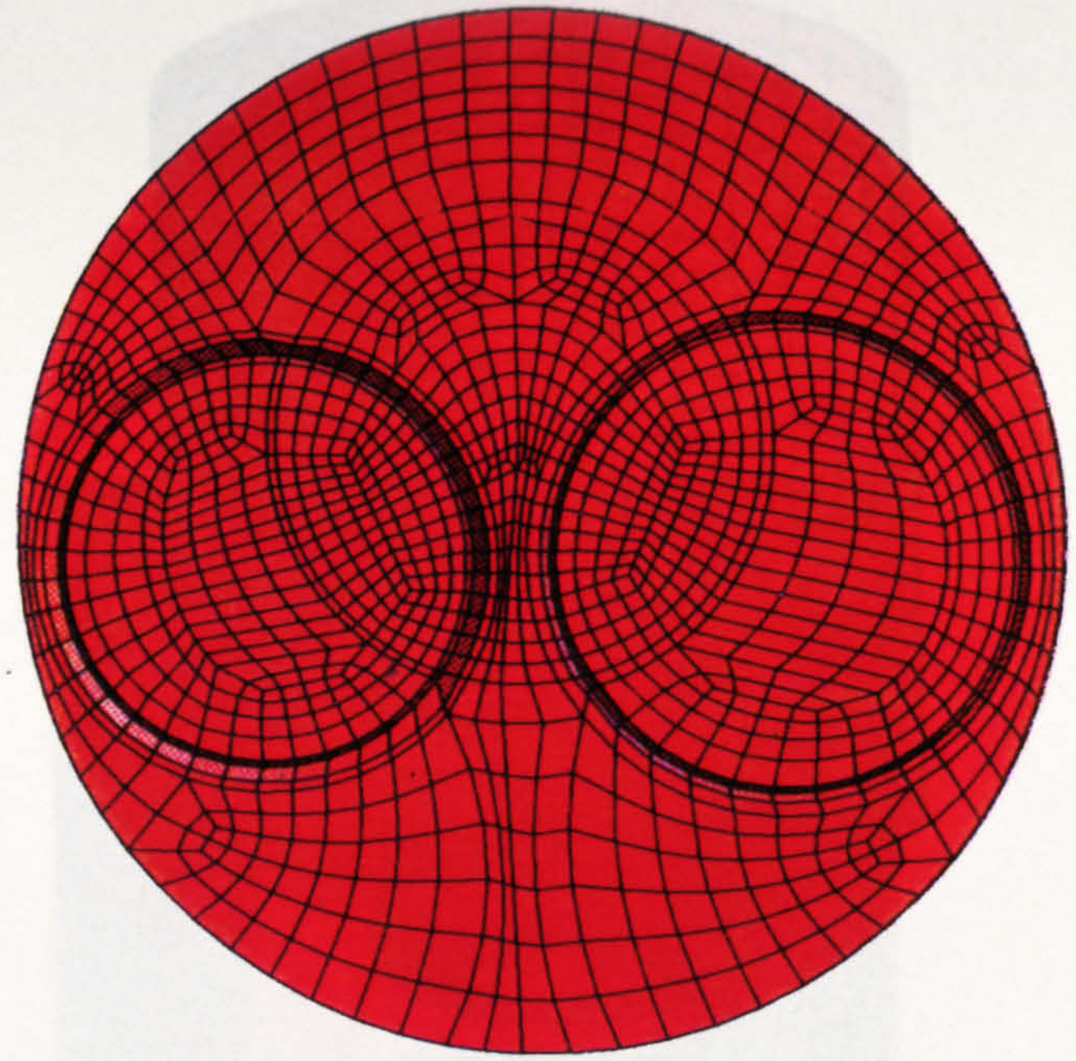
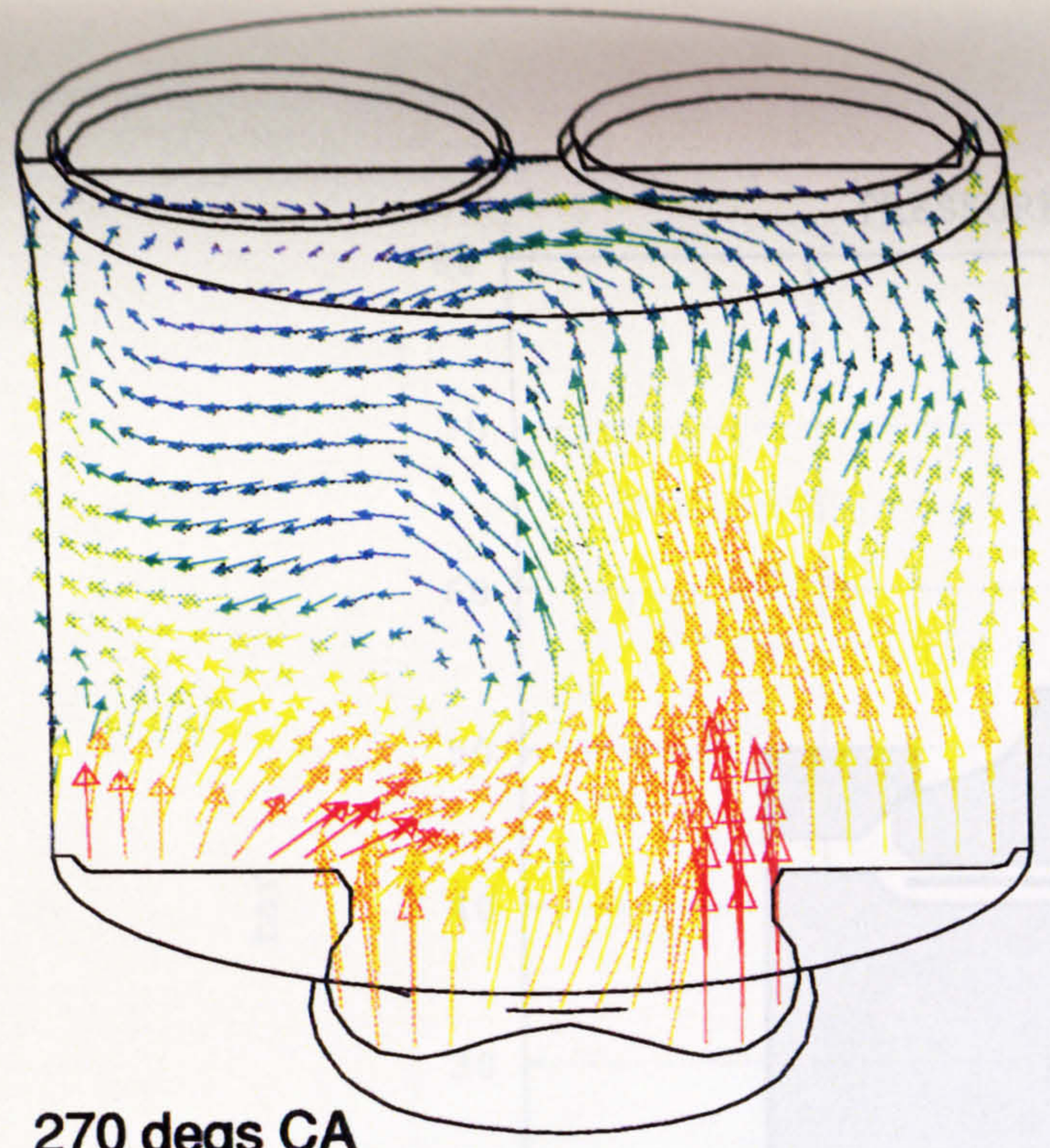


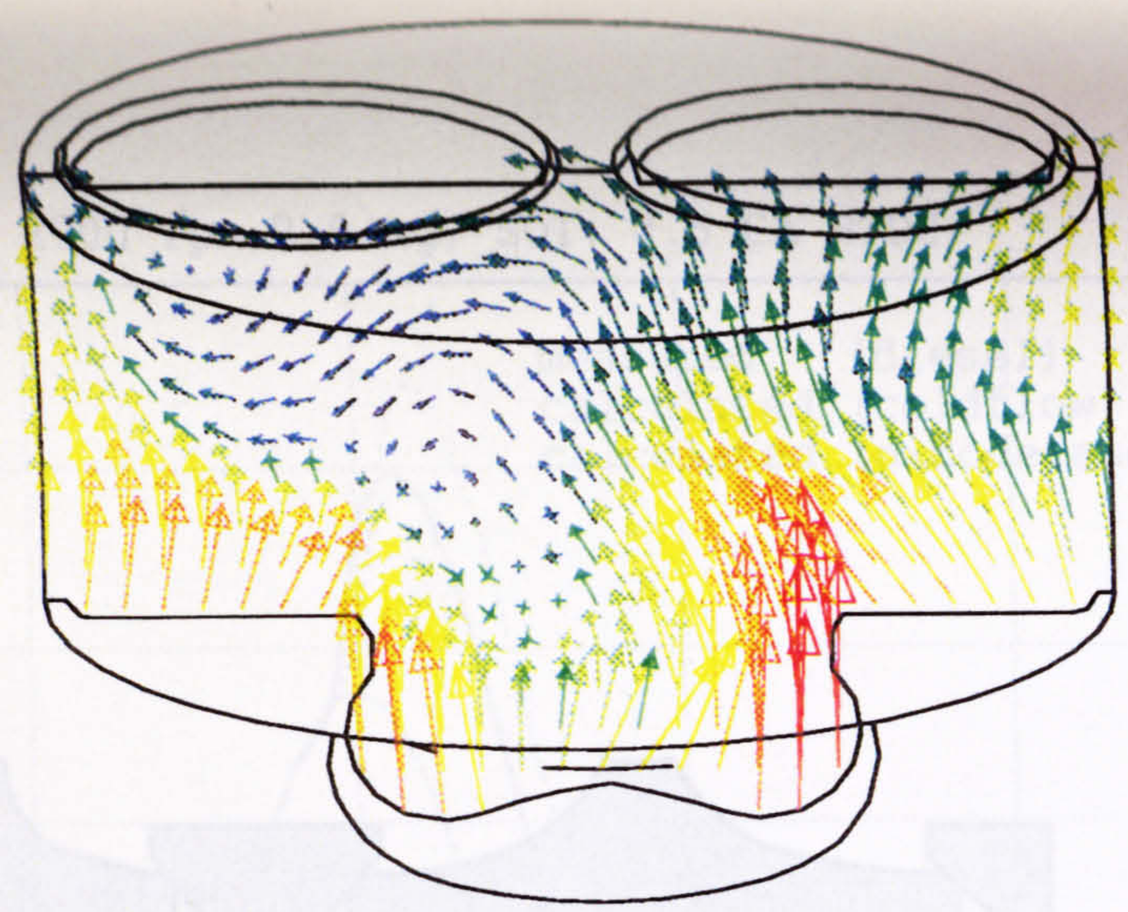




Figure 7.39 VW series engine. 'Fine' mesh at 120° CA ATDC. Bottom right: cutting section along the cylinder axis.



270 degs CA

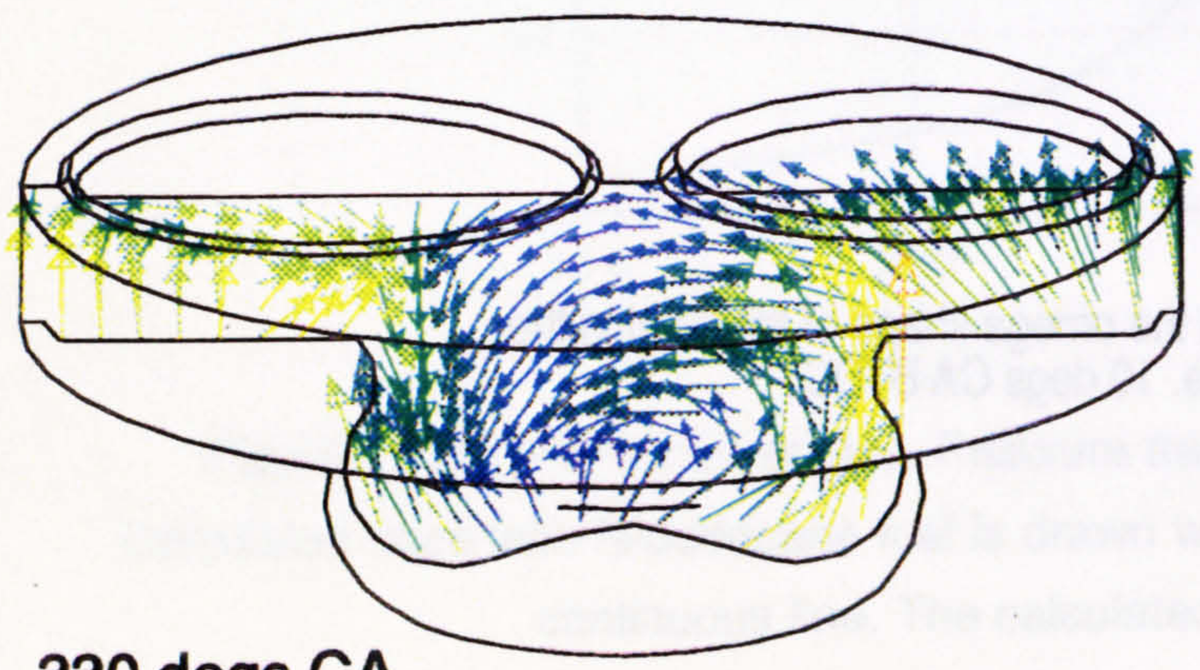


300 degs CA

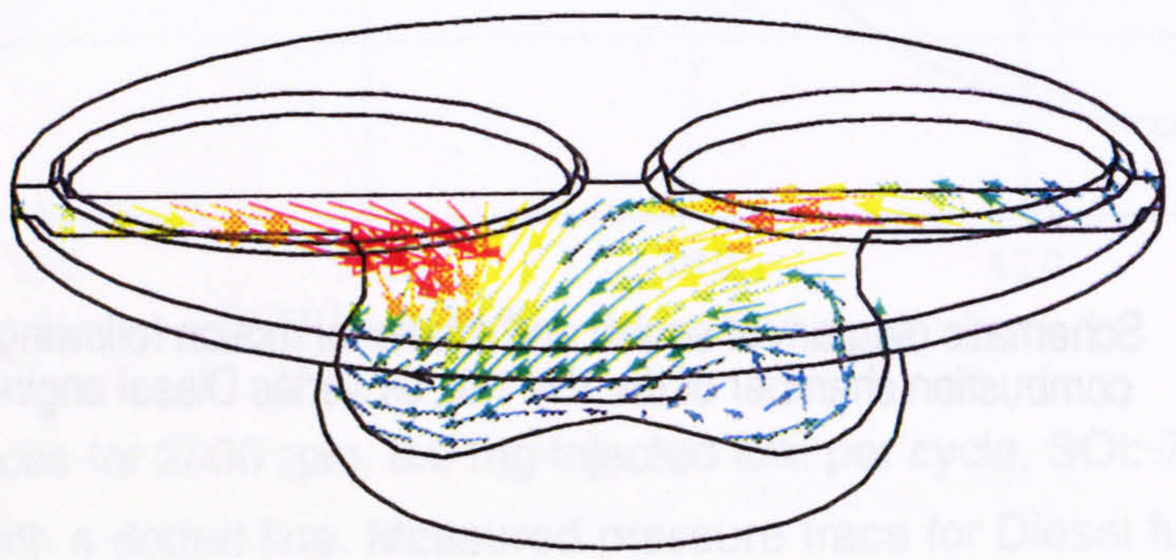
VELOCITY MAGNITUDE

M/SEC
LOCAL MX=16.00
LOCAL MN=0.1391
PRESENTATION GRID

14.00
13.00
12.00
11.00
10.00
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
0.0000E+00



330 degs CA



350 degs CA

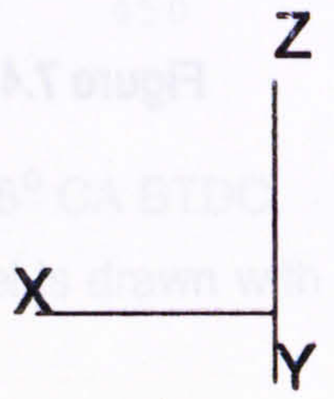


Figure 7.40 Gas flow velocity vectors at selected instances during compression at axial cutting planes along cylinder axis.

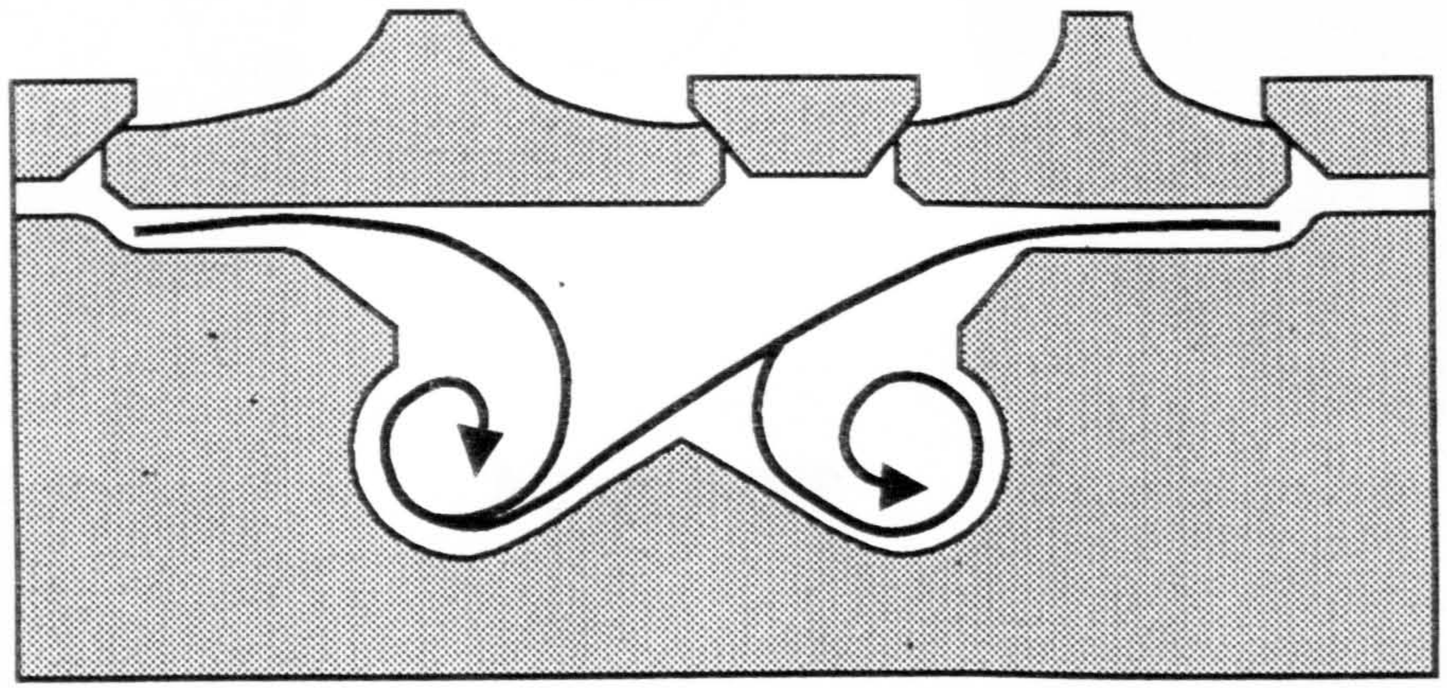


Figure 7.41 Schematic diagram of squish and rotational motion following the omega shape of the bowl in the combustion chamber of the VW 1.9L DI series Diesel engine. 10 degs CA BTDC.

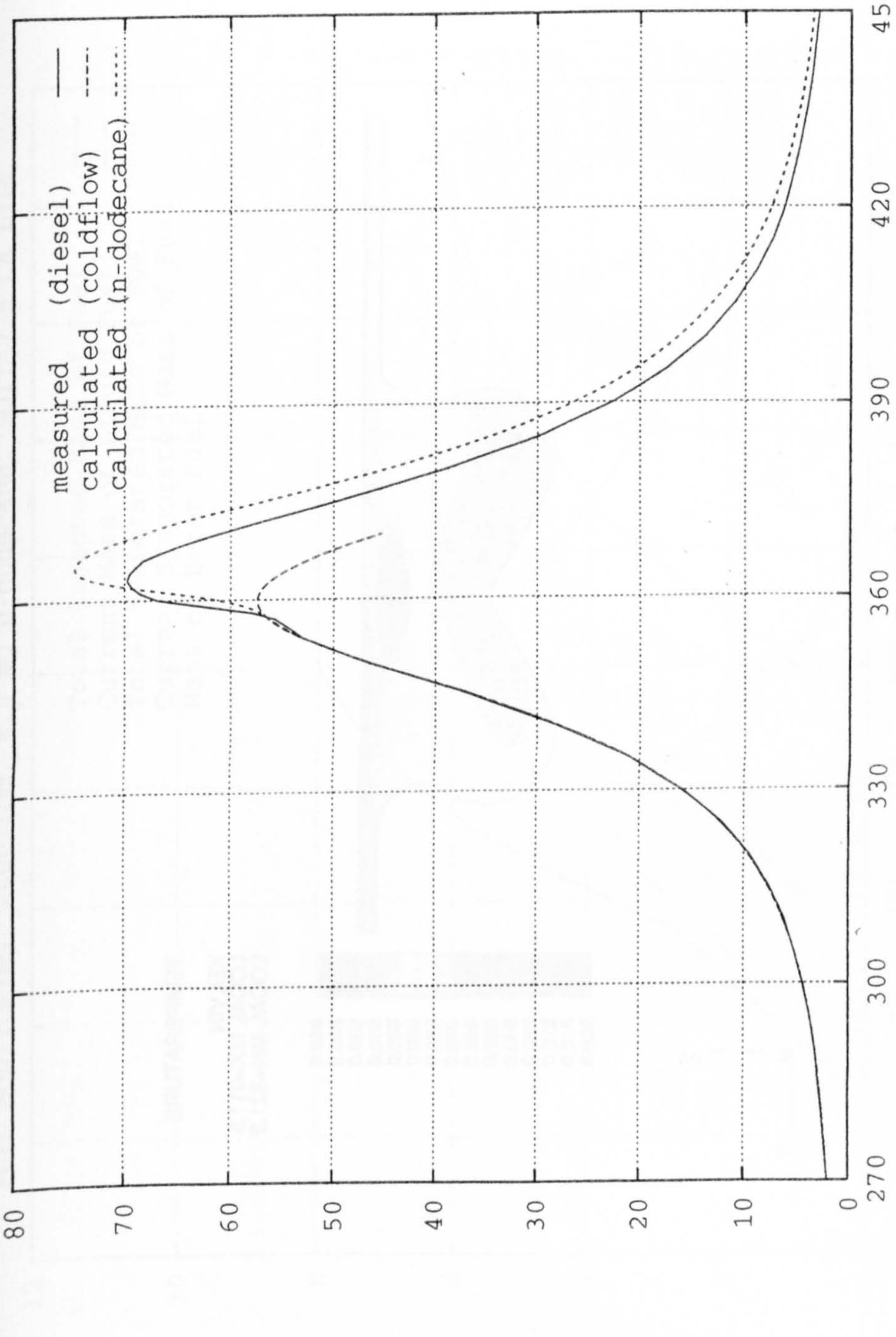


Figure 7.42 VW series engine. Pressure traces for 2000 rpm, 8.0 mg injected fuel per cycle, SOI: 7.5° CA BTDC. Calculated trace with N-dodecane fuel is drawn with a dotted line. Measured pressure trace for Diesel fuel is drawn with a continuous line. The calculated 'cold flow' pressure is also given till 370° CA ATDC.

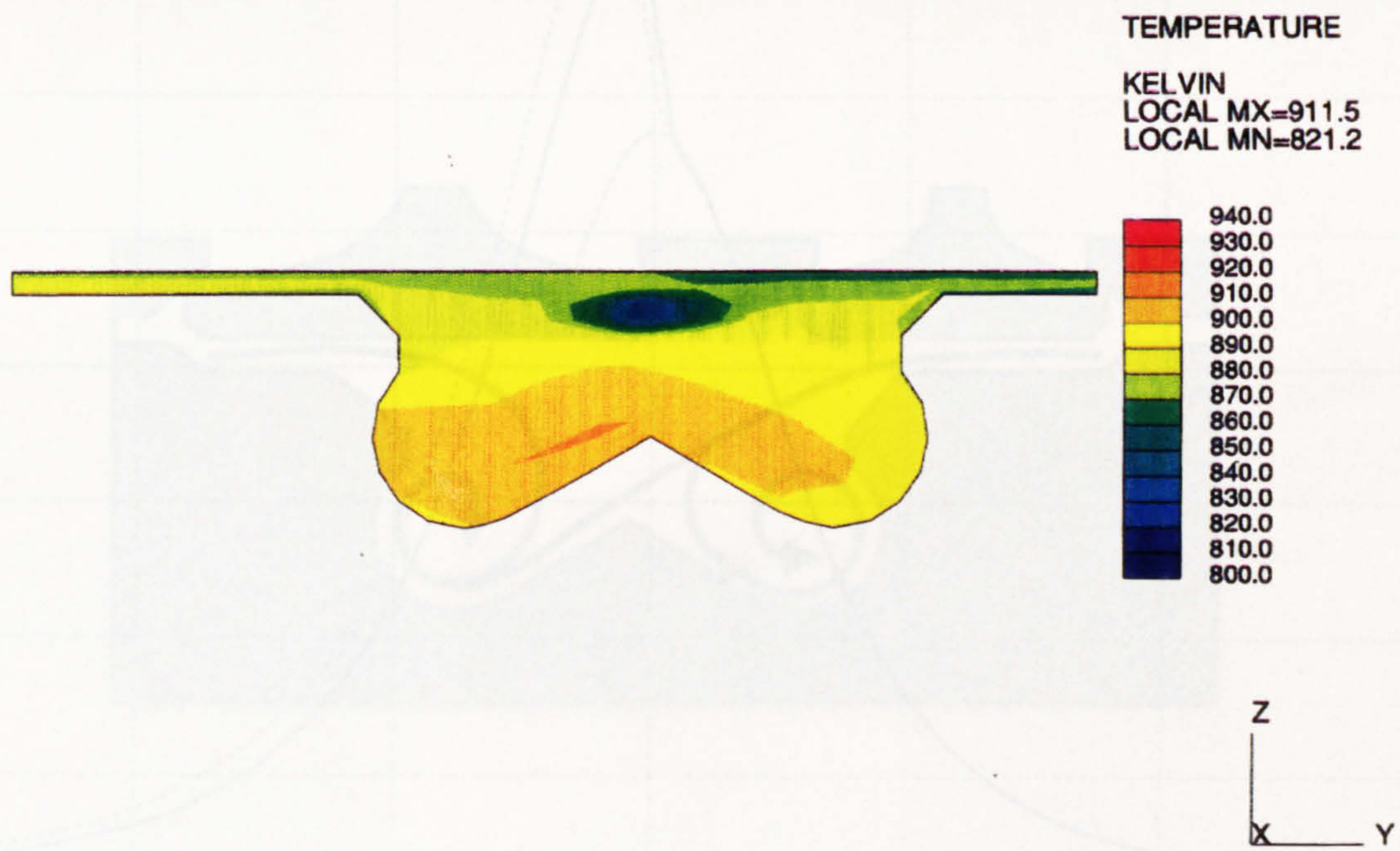


Figure 7.43 VW series engine. Gas temperature filled area contours 1° CA after the start of injection on an axial cutting plane through the bowl axis.

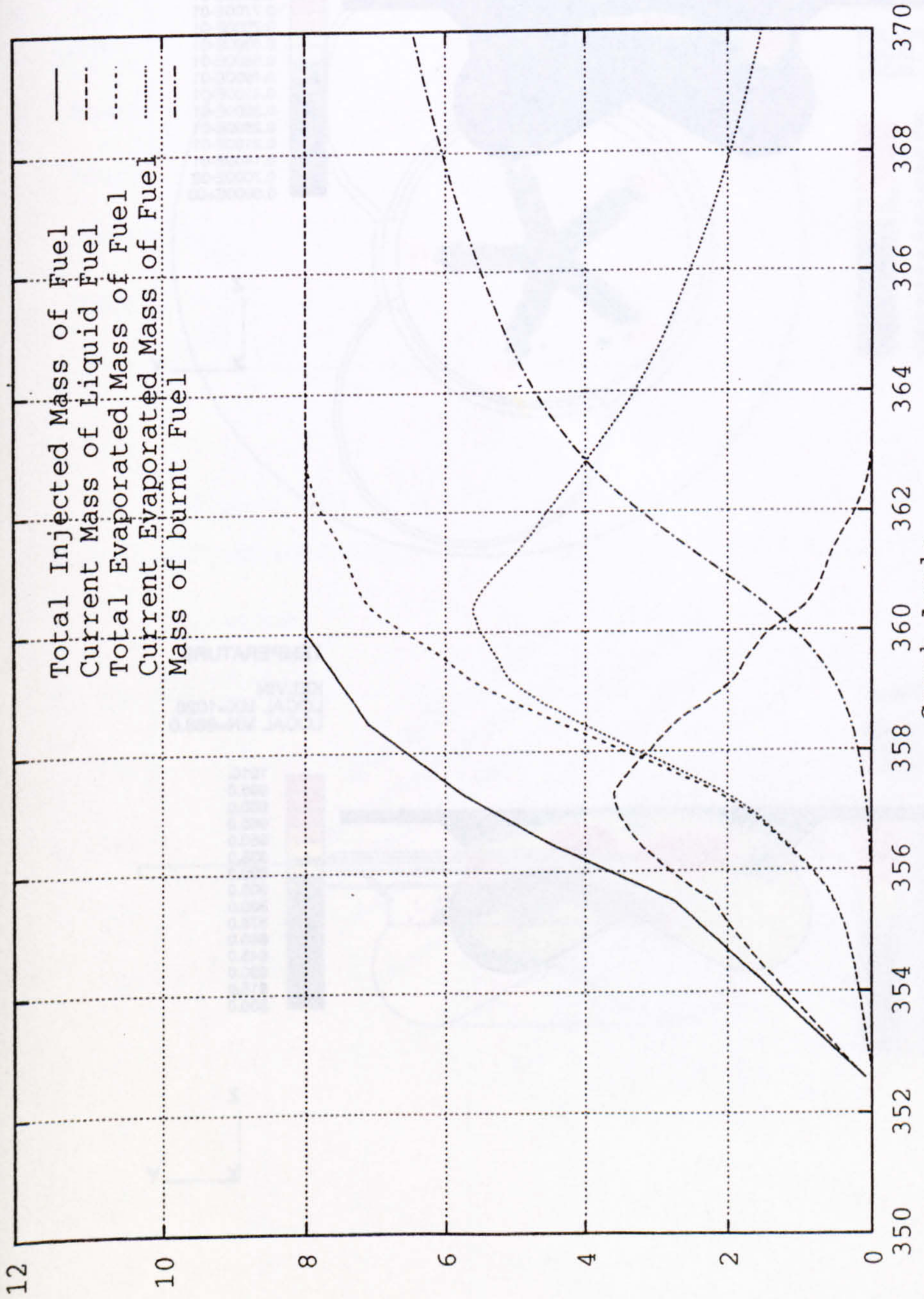


Figure 7.44 VW series engine. Fuel monitoring during the combustion simulation for 2000 rpm, 8.0 mg injected fuel per cycle, SOI: 7.5° CA BTDC.

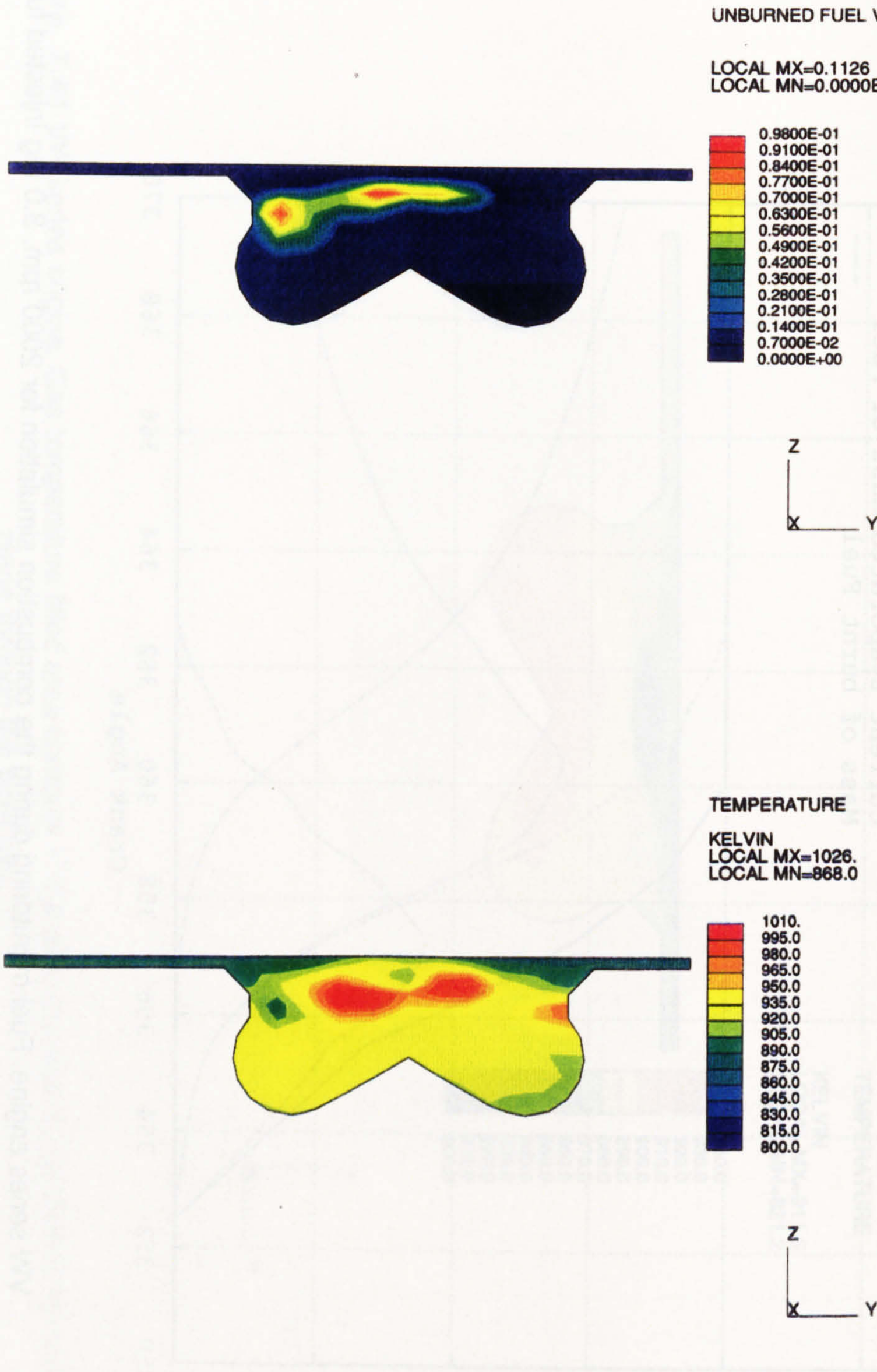


Figure 7.45 VW IDEA series engine. 358.5° CA ATDC. Top: filled-area contour plot of unburnt fuel concentration at an axial cutting plane along the bowl axis. Bottom: filled-area contour lot of gas temperature on the same cutting plane.

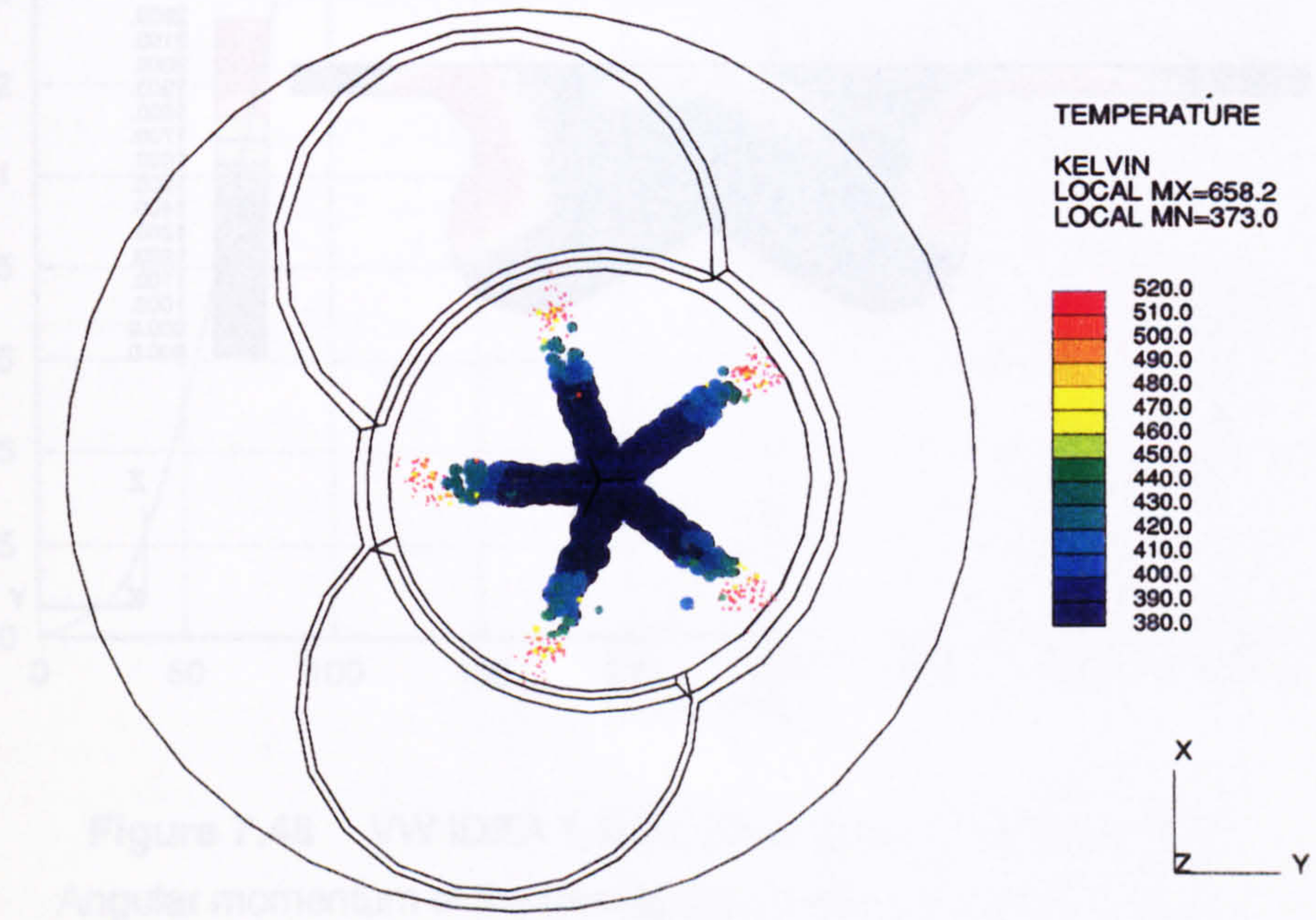


Figure 7.46 Fuel sprays at TDC of compression in the VW IDEA series engine. Each fuel droplet (parcel) is visualised as a circle, with radius that of the droplet and colour according to the droplet's temperature. Top: bottom view. Bottom: elevation.

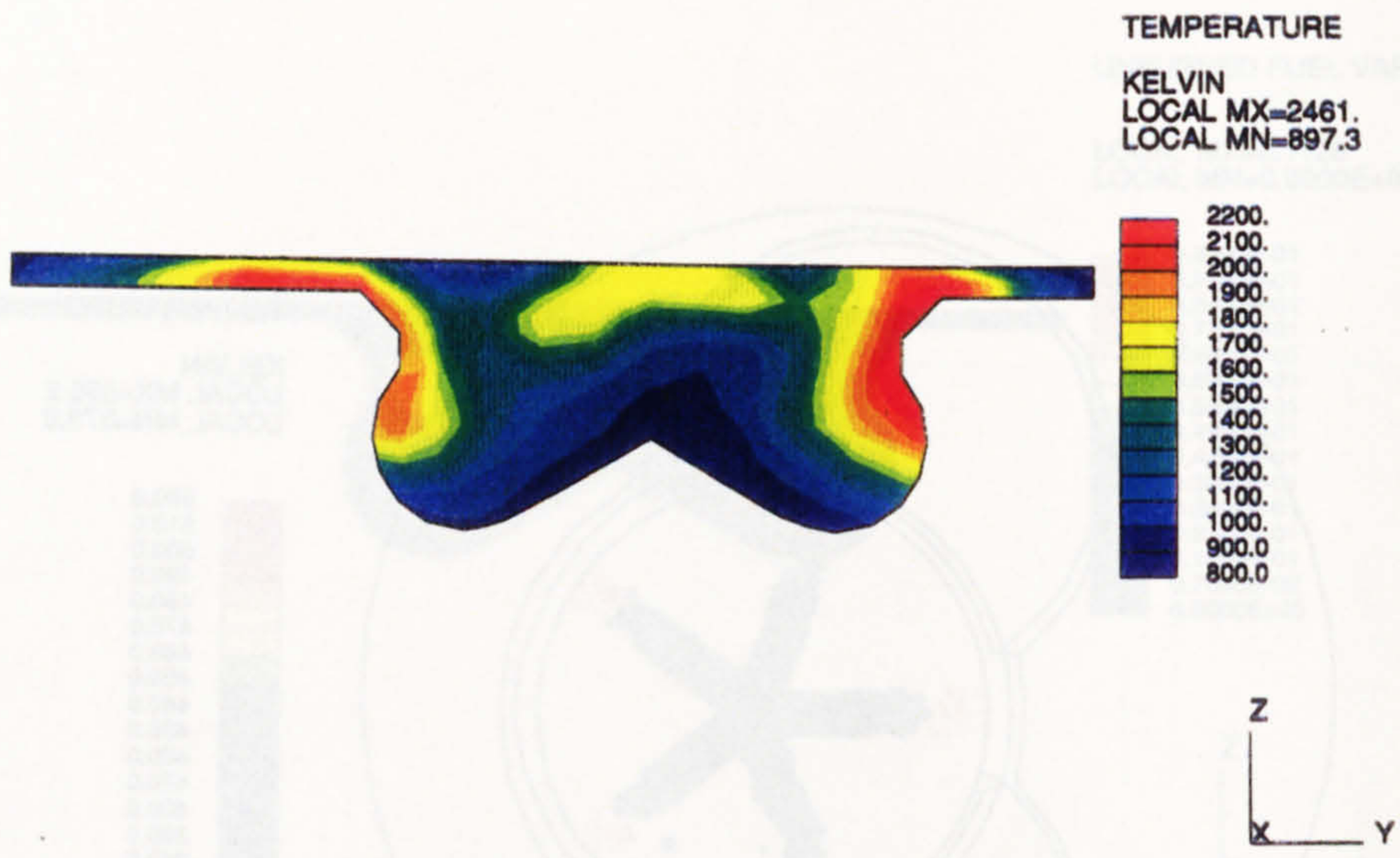


Figure 7.47 Gas temperature filled-area contours at 370.0° CA ATDC in the VW IDEA series engine. Top: elevation view of an axial cutting plane along the axis of the bowl. Bottom: multiple axial cutting planes parallel to the axis of the cylinder.

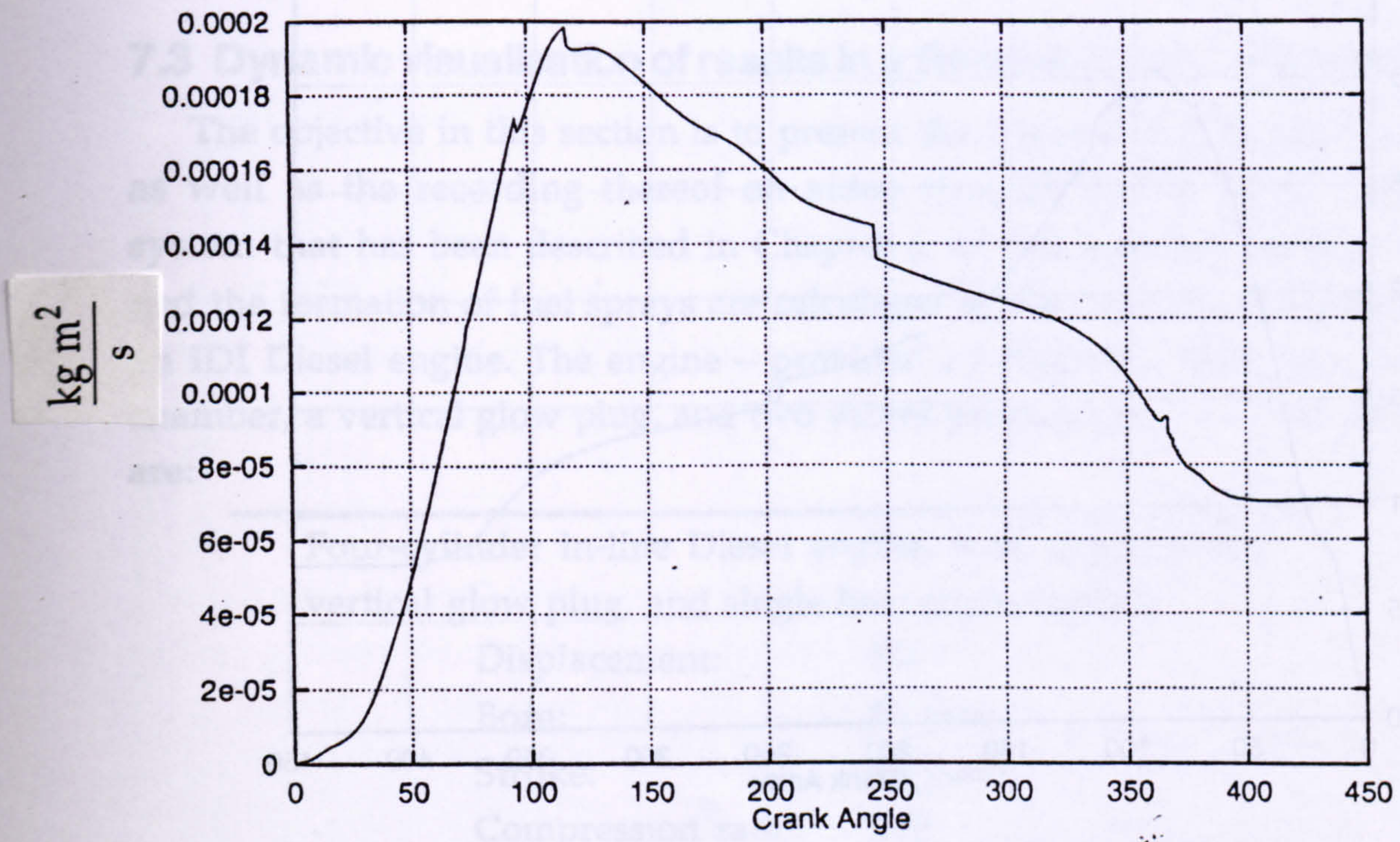
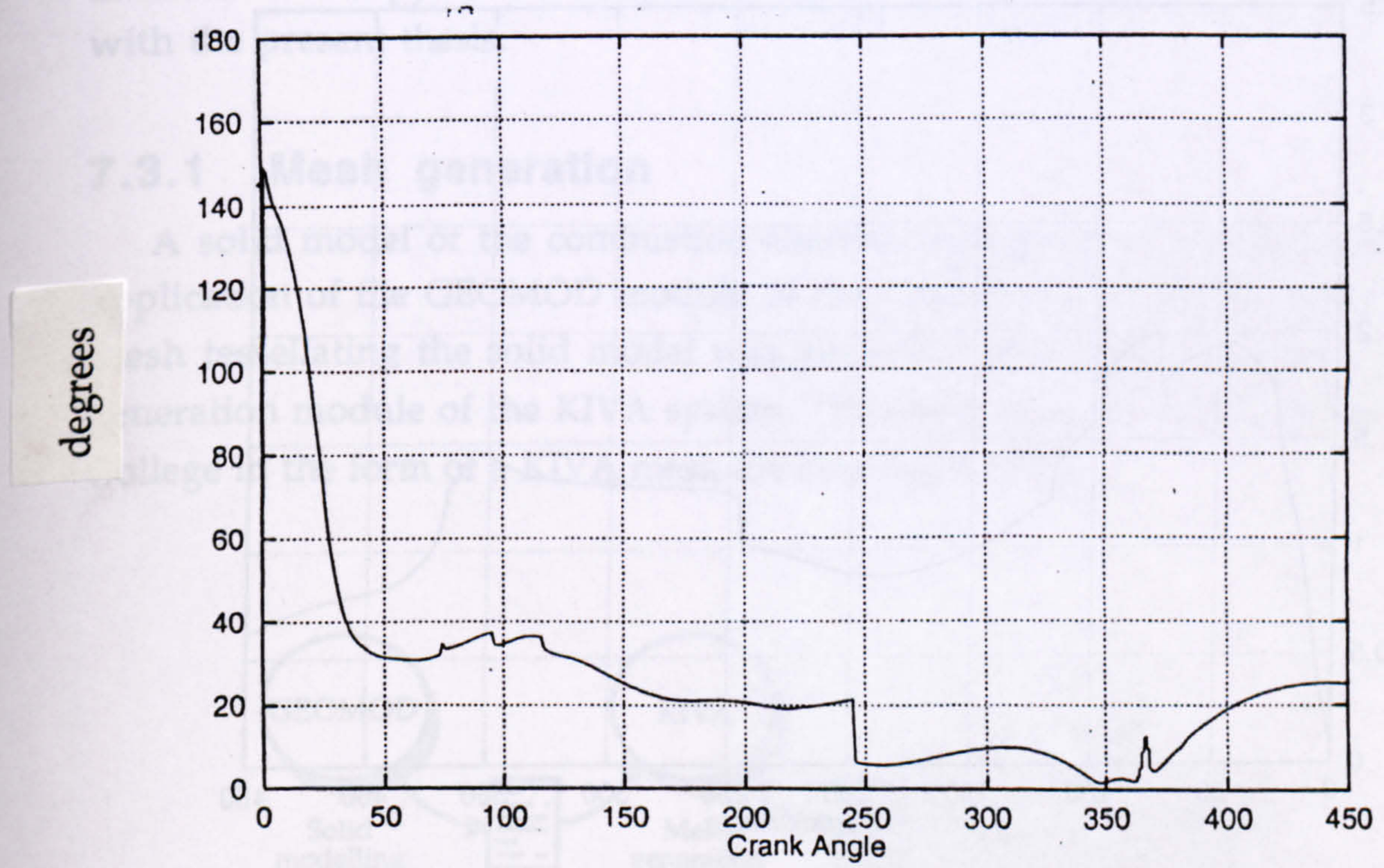


Figure 7.48 VW IDEA 1.9l DI Diesel engine, series version.
Angular momentum of the moving gas in the combustion chamber.



Inclination of the angular momentum vector, with respect to the axis of the cylinder.

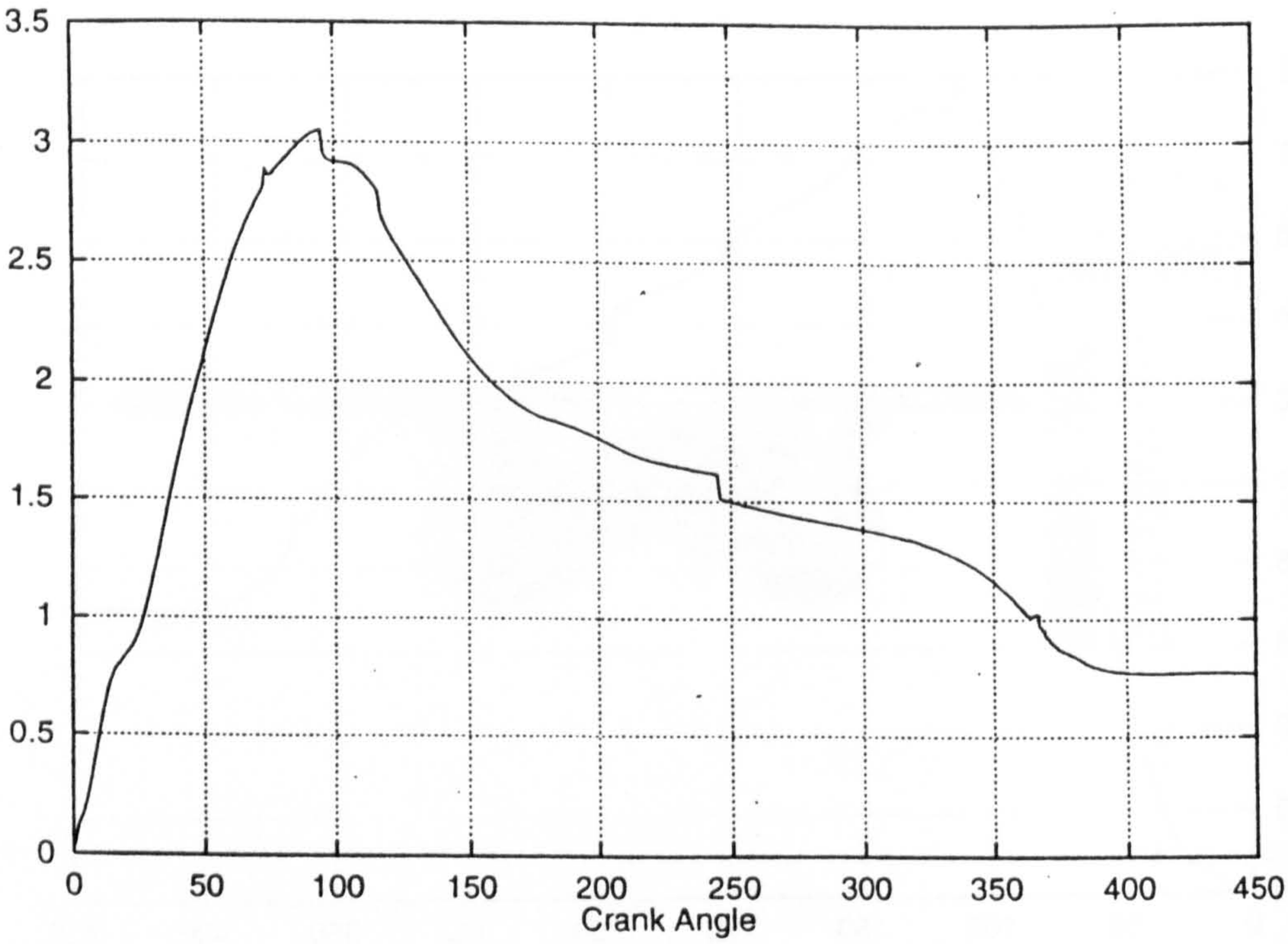


Figure 7.50 VW IDEA 1.9l DI Diesel engine, series version.
Swirl number calculated assuming solid body rotation .

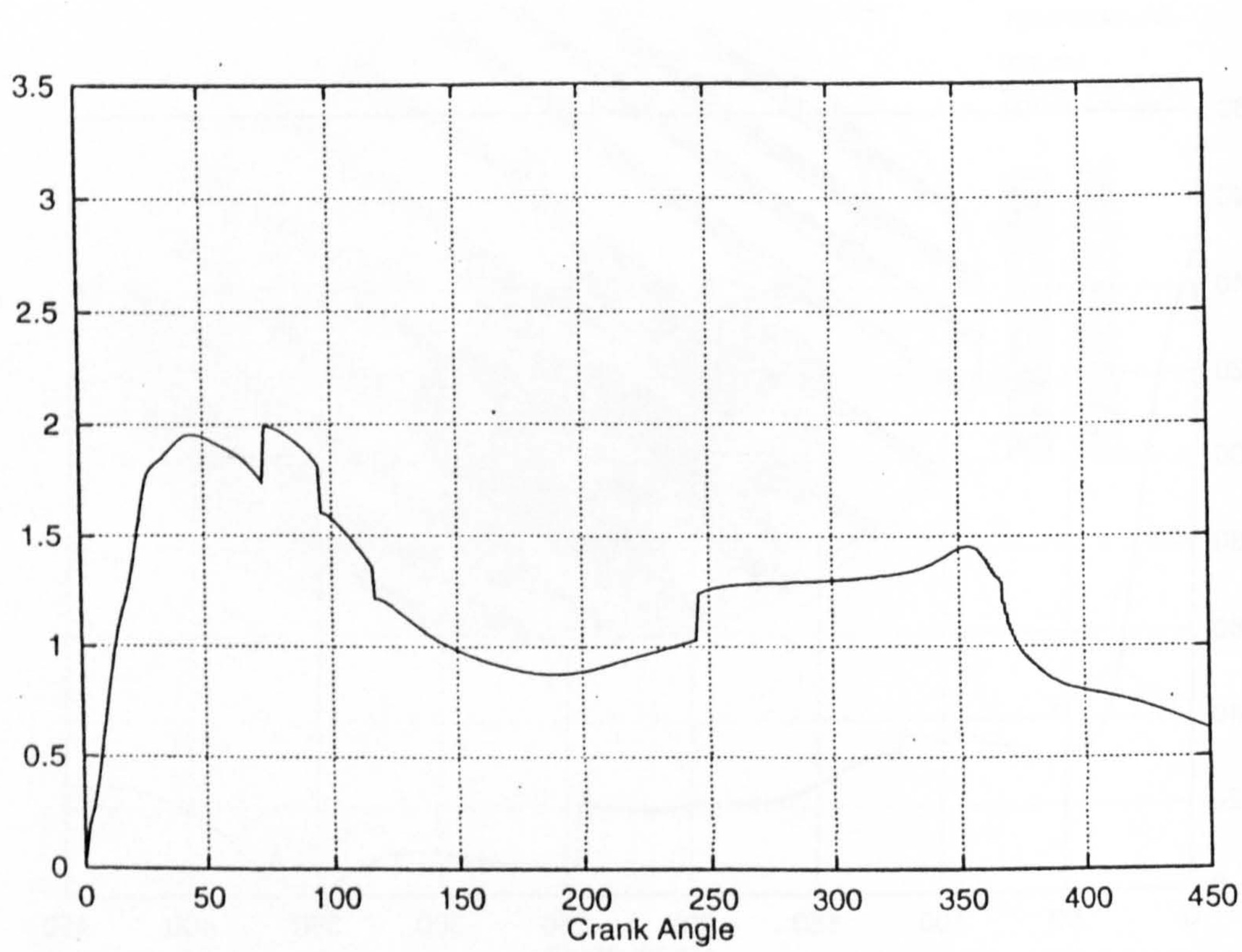


Figure 7.51 VW IDEA 1.9l DI Diesel engine, series version.
Swirl number with respect to the angular momentum axis.

7.3 Dynamic visualisation of results in a Renault IDI Diesel engine

The objective in this section is to present the animation of SPEED results, as well as the recording thereof on video tape, by means of the AERAS system that has been described in Chapter 6. In this exercise the flow field and the formation of fuel sprays are calculated in the combustion chamber of an IDI Diesel engine. The engine – provided by Renault – features a swirl-chamber, a vertical glow plug, and two valves per cylinder. Its characteristics are:

Four-cylinder in-line Diesel engine, with prechamber, vertical glow plug, and single-hole pintle injector.	
Displacement:	2.0 l
Bore:	86 mm
Stroke:	88 mm
Compression ratio:	22.5

The mesh generation and the setup of the initial and boundary conditions for the calculation are given in paragraphs 7.3.1 to 7.3.3. This is followed by a detailed presentation of the design and production of the animation. A copy of the animation video – in VHS format – is submitted with the present thesis.

7.3.1 Mesh generation

A solid model of the combustion chamber was generated at Renault by application of the GEOMOD module of the IDEAS/CAEDS CAD system. A mesh tessellating the solid model was generated at Renault with the mesh generation module of the KIVA system. The mesh was provided to Imperial College in the form of a KIVA mesh file (see figure 7.52).

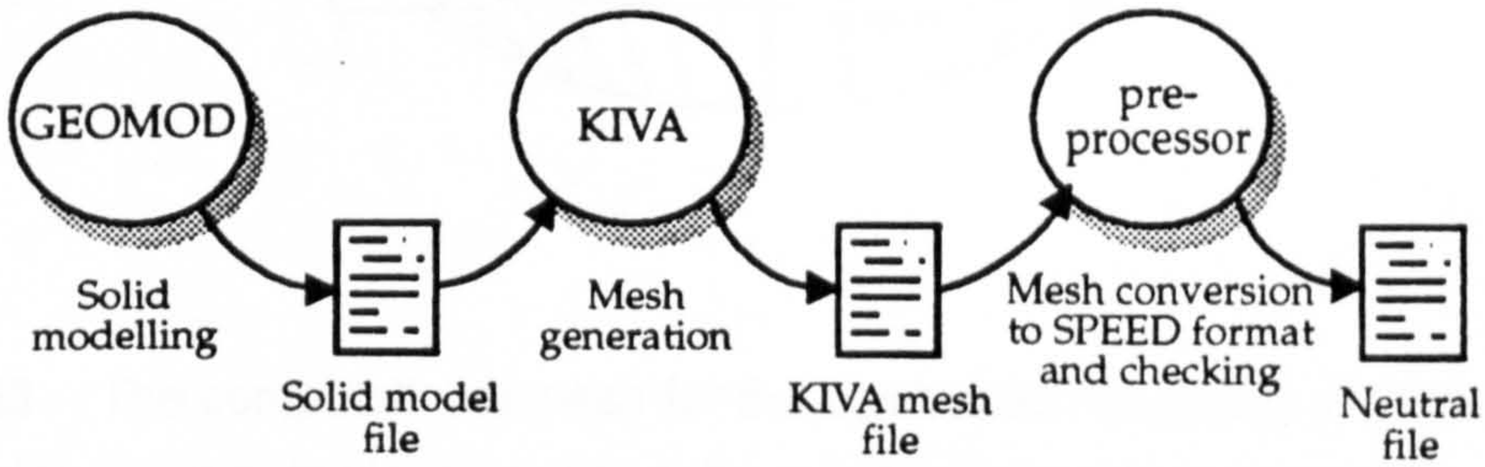


Figure 7.52. Data flow diagram of the mesh generation process.

The KIVA mesh was converted to a mesh that:

- complied with the notation, structure, and indexing nomenclature of the SPEED code;
- featured quadrilateral cells representing boundary faces.

For this, a KIVA-to-SPEED conversion program – called KIPAT – was developed. This program was integrated in the preprocessing system described in chapter 5, as a module thereof. The converted mesh can be seen in figure 7.53. It features:

- 4284 cells at TDC.
- A set of quadrilateral elements defined by those faces of internal cells that lie on boundaries, with indexing notation corresponding to four SPEED wall patches, located at the piston crown, cylinder liner, cylinder head and prechamber (see figure 5.16).
- A set of quadrilateral elements at the bowl and prechamber throats with notation corresponding to two ghost boundaries, required for the application of the mesh motion routines of SPEED (see figure 5.16).

The mesh, shown in figure 7.53, was stored in the form of a 'neutral file', for further processing with the preprocessing system presented in chapter 5.

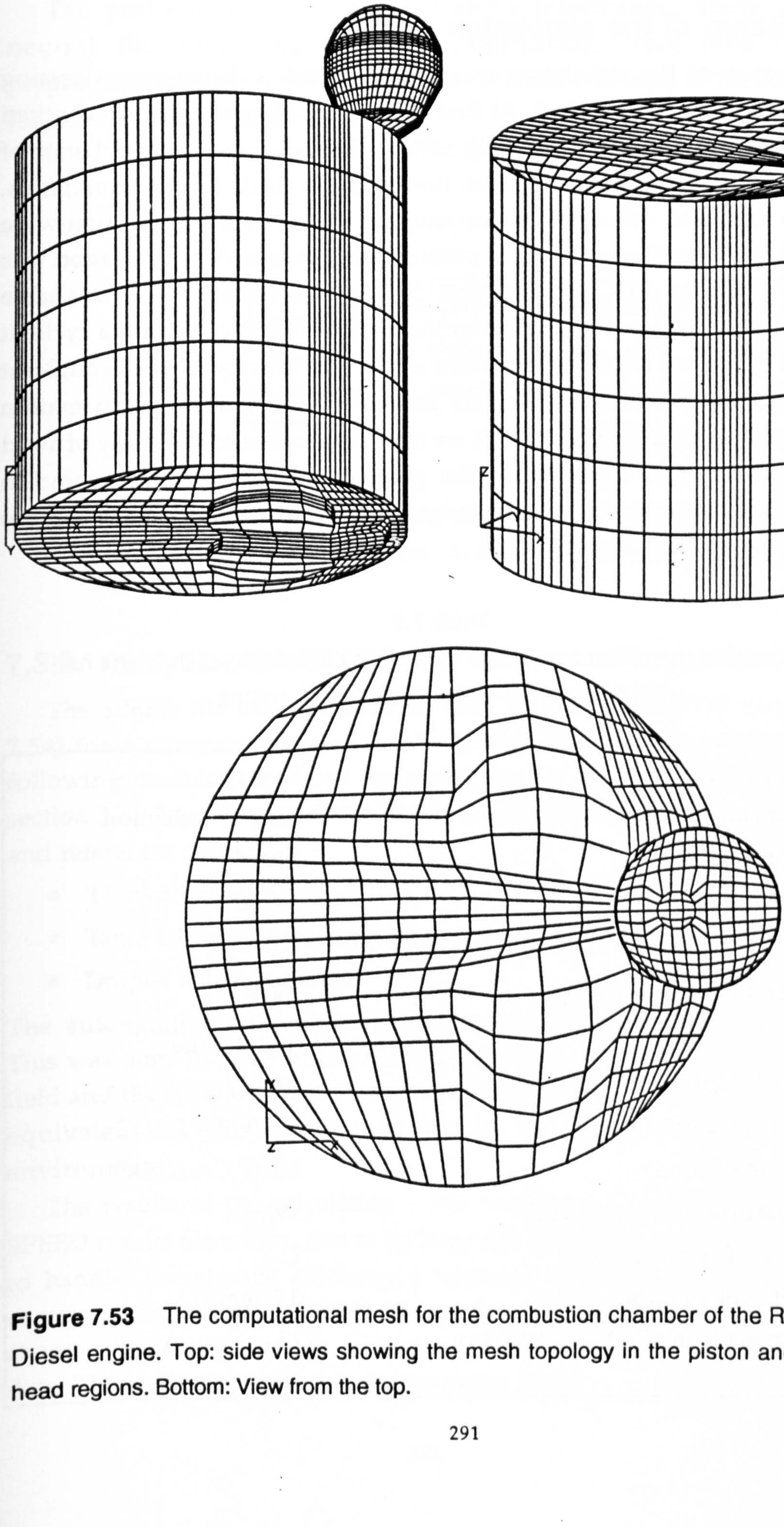


Figure 7.53 The computational mesh for the combustion chamber of the Renault IDI Diesel engine. Top: side views showing the mesh topology in the piston and cylinder head regions. Bottom: View from the top.

7.3.2 Setup of the simulation

The setup of the simulation was accomplished with the preprocessing system described in chapter 5. At first the computational mesh was checked for its suitability for calculation with SPEED. Subsequently, the initial state of the gas in the combustion chamber, the values of fields on the boundaries, injection data, and other data concerning the control of the calculation were specified in the problem file of the preprocessing system. The calculation was initiated from the BDC of induction. Since no information was available about the flow field in the cylinder at that (or any other) point in the cycle, it was decided to set all the components of the gas velocity to 0.0 throughout the calculation domain, and the bulk pressure to 1 bar. This approximation can be considered acceptable only if we restrict ourselves to the study of swirl formation and fuel injection in the prechamber. The engine operation conditions and the initial and boundary conditions for the calculation are summarised in the following table:

Table 7.5

Main engine operation conditions for the simulation of gas flow and fuel injection in the Renault IDI Diesel engine

Engine speed	2000 rpm
Start of calculation	BDC of induction
Bulk pressure	1 bar
Bulk temperature	300 K
Temperature on all walls (except glow plug)	420 K
Temperature of glow plug	750 K
Fuel	n-dodecane
Injected quantity per shot	20 mg/cycle
Start of injection	350.0° CA ATDC
End of injection	380.0° CA ATDC
Temperature of injected fuel	350.0 K
Number of injector holes:	1
Diameter of injector:	0.5 mm
No of parcels injected per timestep:	25

The problem file containing the above information, along with the neutral file containing geometry information, were read into the preprocessor for creation of a SPEED file. This file was used as the startup file for the calculation (figure 7.54).

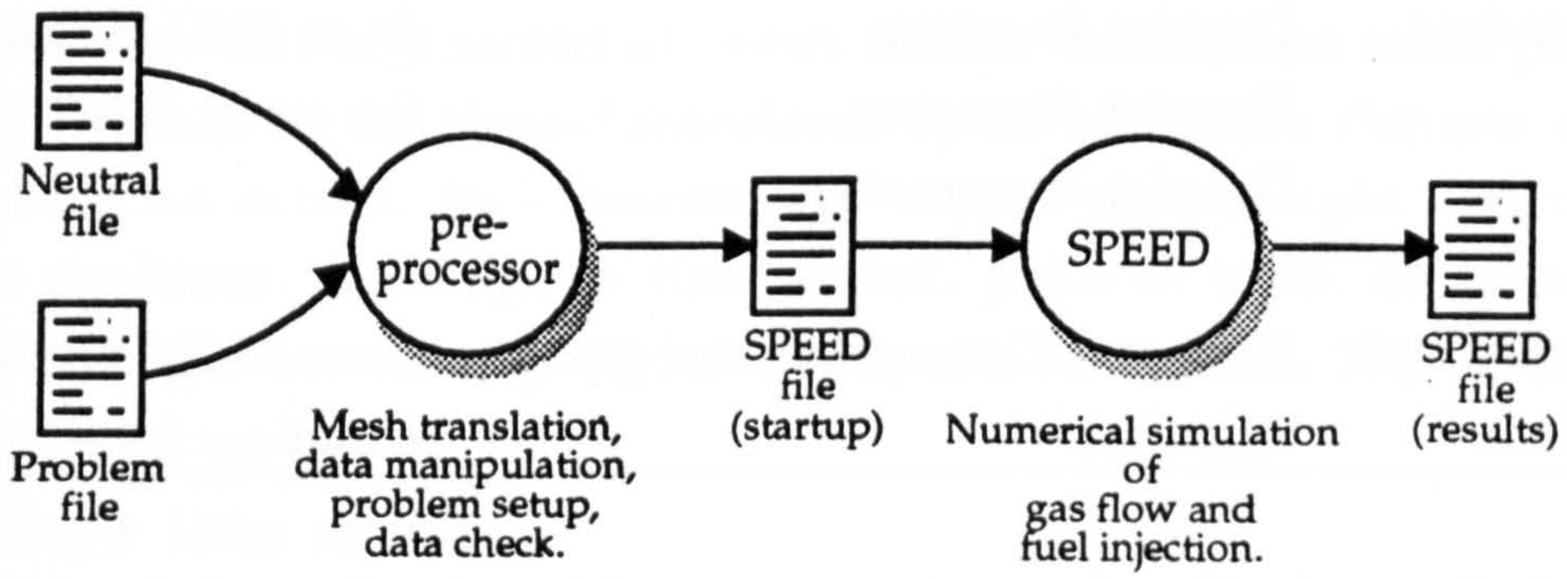


Figure 7.54 Data flow diagram of the preprocessing and analysis processes, with the preprocessor (presented in Chapter 5) and the analysis code (SPEED).

7.3.3 Analysis with SPEED

The SPEED file was subsequently read into the SPEED-DC code (figure 7.54) for a computation from 180.0° CA ATDC till 400.0° CA ATDC. The following models were used for simulation of the injection process (see section 3.2.2.2 and the user's manual of the SPEED code [173] for description and references thereof):

- 'Huh' atomisation model;
- 'Reitz Diwakar' breakup model;
- Droplet collision model.

The auto-ignition and combustion simulation models were switched off. This was done in order to be able to study the interaction between the flow field and the sprays, as these are modelled by the SPEED code. The physical equivalent of this simulation would be injection in a nitrogen environment.

The results of the calculation – 320 records – were stored in a single SPEED results file which, due to its large size (87 Mbytes), was very difficult to handle. For storage efficiency a trimmed version thereof was created containing only those data that were needed for the dynamic visualisation of chamber geometry, namely geometry, gas flow, and injected fuel droplet data. These are summarised in the following table:

	Data stored	times in degrees CA
Geometry data.	Vertex coordinates; Cell connectivity lists; Boundary connectivity lists; Boundary notation.	from 220° to 388° for every 2°
Spray data.	Droplet location; Droplet diameter; Droplet temperature.	from 350° to 388° for every 0.25°
Gas velocity data.	X, Y, and Z component of gas velocity.	from 220° to 388° for every 2°

By storing only the above data, up to 60% reduction of the size of the results file was achieved. The resulting file, occupying a total of 35 Mbytes on the data storage unit, was subsequently piped to the AERAS animation system for dynamic visualisation of gas flow and injected fuel droplets.

7.3.4 Dynamic visualisation

The dynamic visualisation of the combustion chamber geometry, the gas flow field, and the injected fuel spray were produced with the AERAS animation system, that has been described in Chapter 6 of the present work. The process of design and realisation of the animation is described in the remaining paragraphs.

7.3.4.1 Animation design

As a first step it was necessary to make decisions about what needed to be animated. This dictated what data sets had to be generated with SPEED and what models had to be built with AERAS. The decision was to:

- Animate the second half of the compression stroke, which is of interest to the development of swirl in the prechamber, and the first half of the power stroke showing the formation of the spray.
- To dynamically visualise the spray in the form of fuel droplets, from the commencement of injection till full evaporation.
- To dynamically visualise the flow field as a set of velocity vectors on a cutting plane coinciding with the symmetry plane of the combustion chamber.

The next stage was to draft a detailed storyboard of the animation. This was based on the visualisation requirements for gas flow and injection related phenomena that take place in the chamber.

7.3.4.2 Results interpretation and storyboard of the animation

The action was segmented in seven scenes. Each scene takes place at a specific location in the three-dimensional space and has a definite duration of continuous action. Its elements are objects-actors, light sources, and camera positions. A change in time period, point of view, or number and position of light sources signifies the start of a new scene. The seven scenes of the present animation were:

1. *Front titles sequence.*
2. *Engine cycling*. This scene is included in order to familiarise the user with the geometry features of the combustion chamber. It starts with a surface model of the liner, cylinder head, and swirl chamber. Then the front part of the surface model fades-out to reveal the piston and glow plug. Subsequently the camera zooms on the prechamber region, while the piston cycles at the same time. The duration of this scene is 37".
3. *Fuel injection.*

   This scene starts with the piston at 350° CA ATDC position. A title showing the time in crank angle values fades in. Fuel droplets are injected in the region of the prechamber. The tip of the spray is deflected towards the glow plug by the anti-clockwise swirl (see figure 7.56). The temperature of fuel droplets at the region of the glow plug increases to 690 K, due to their proximity to it (see figure 7.57). The spray impinges on the wall opposite from the injector at 370° CA ATDC. By 5° CA after the end of injection, fuel droplets are blown to the opposite corner of the swirl chamber – a phenomenon that would be unlikely in a firing engine – and into the throat (see figure 7.58). The droplets close to the top entry of the throat have diameters of the order of 3.0 microns.

   Fuel evaporation takes place throughout the prechamber. The main centre of evaporation is below the injector, along an axis joining the injector and the throat (see figure 7.59). 10° CA after the end of injection, most of the fuel has evaporated. As can be seen in figure 7.59, fuel vapour is blown to the wall of the prechamber below the injector and into the main chamber where it spreads throughout.

The duration of this scene is 30".

4. *Fuel injection.*

   This is a repetition of scene 3.

5. *Velocity vectors.*

   This scene starts with the piston at 220° CA ATDC position. A title showing the time in crank angle values fades in. This is followed by animation of gas velocity vectors, from 220° CA to 388° CA ATDC, on a cutting plane coinciding with the symmetry plane of the combustion chamber.

   As can be seen in figure 7.60 at 280° CA ATDC during compression a jet is formed in the throat and in the region of the prechamber between the throat and the injector. This jet increases in strength reaching velocities of the order of 200 m/s and causes swirl to develop in the prechamber as the piston approaches the TDC position. The swirl centre is in the region below the glow plug and the injector, very close to the axis of injection (see figure 7.61). At TDC, the swirl in the prechamber resembles solid-body rotation. The swirl centre is slightly displaced towards the entry of the throat, with maximum velocities on the aforementioned cutting plane of the order of 60 m/s. As the piston moves downwards, this rotational motion is modified due to gas expansion towards the main chamber. The swirl centre moves towards the entry of the throat. A jet – with velocities up to 230 m/s at 380° CA ATDC on the same cutting plane – is formed in the throat filling the main chamber with mixture (see figures 7.62 and 7.63)

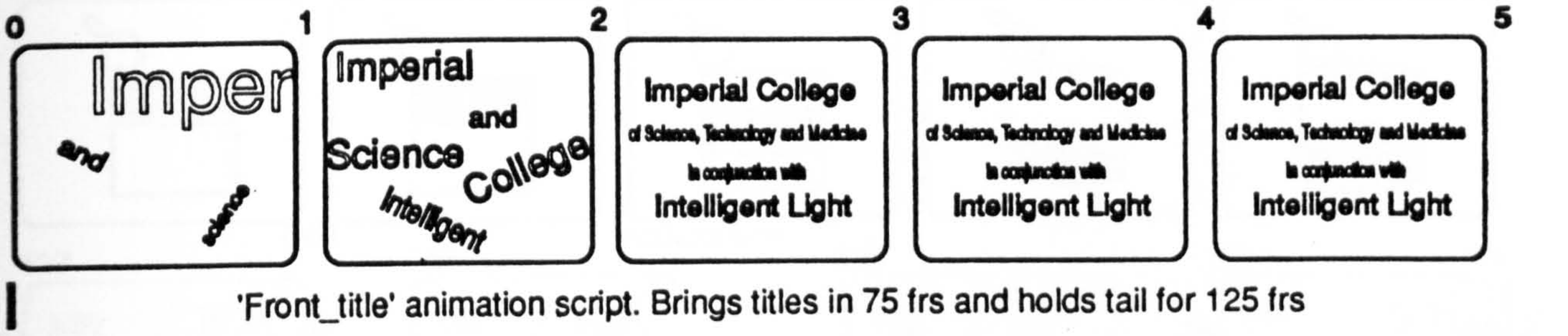
   The scene ends with the fade-out of the engine with vectors at 388° CA ATDC, after a total duration of 30".

6. *Velocity vectors.*

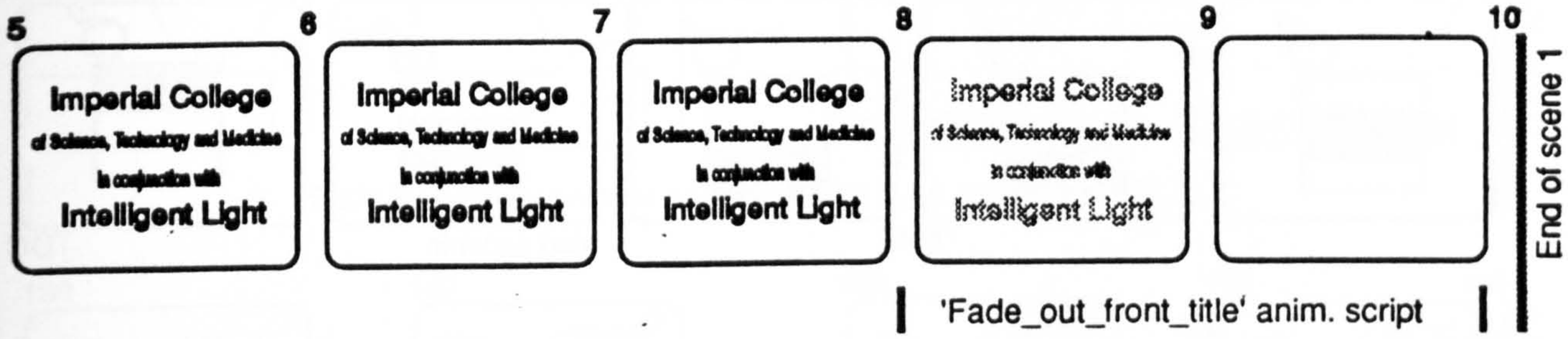
   This is a repetition of scene 5.

7. *End title sequence.*

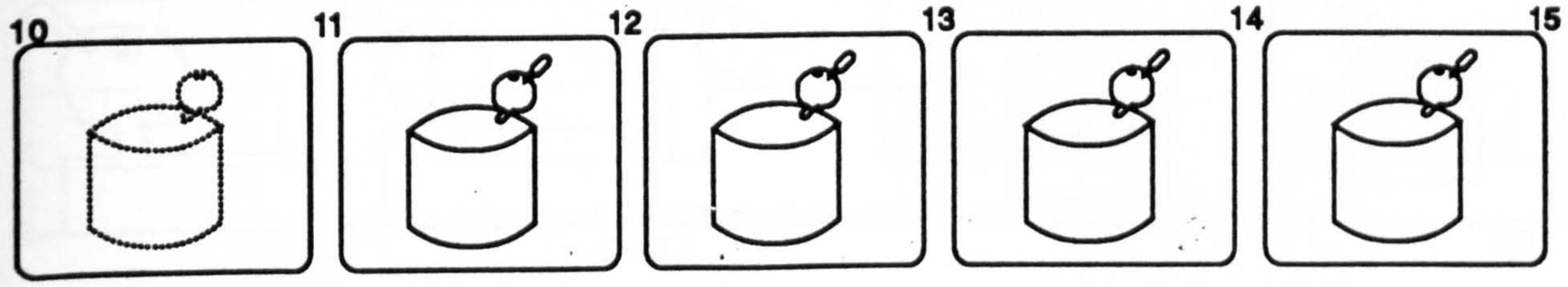
The story-board of the animation is shown in figure 7.55. Each frame in the story-board corresponds to the time (in seconds) shown at its top right side. Since 25 frames are needed for 1" of animation, there are 24 frames of animation missing between each pair of frames in the story-board. Comments on the action are written below the story-board frames. Long vertical lines notify the start and end of a scene. Short vertical lines notify the start and end of an AERAS animation script describing the action. A scene is often described by more that one animation scripts.



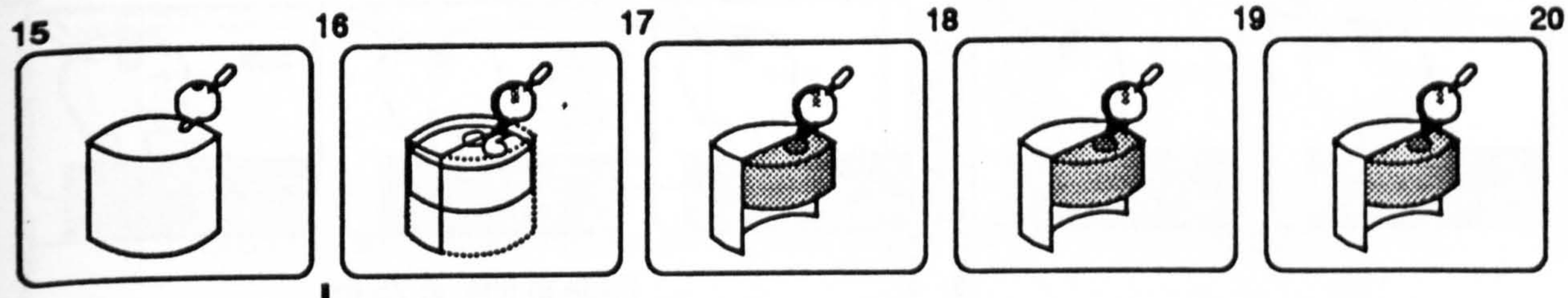
'Front_title' animation script. Brings titles in 75 frs and holds tail for 125 frs



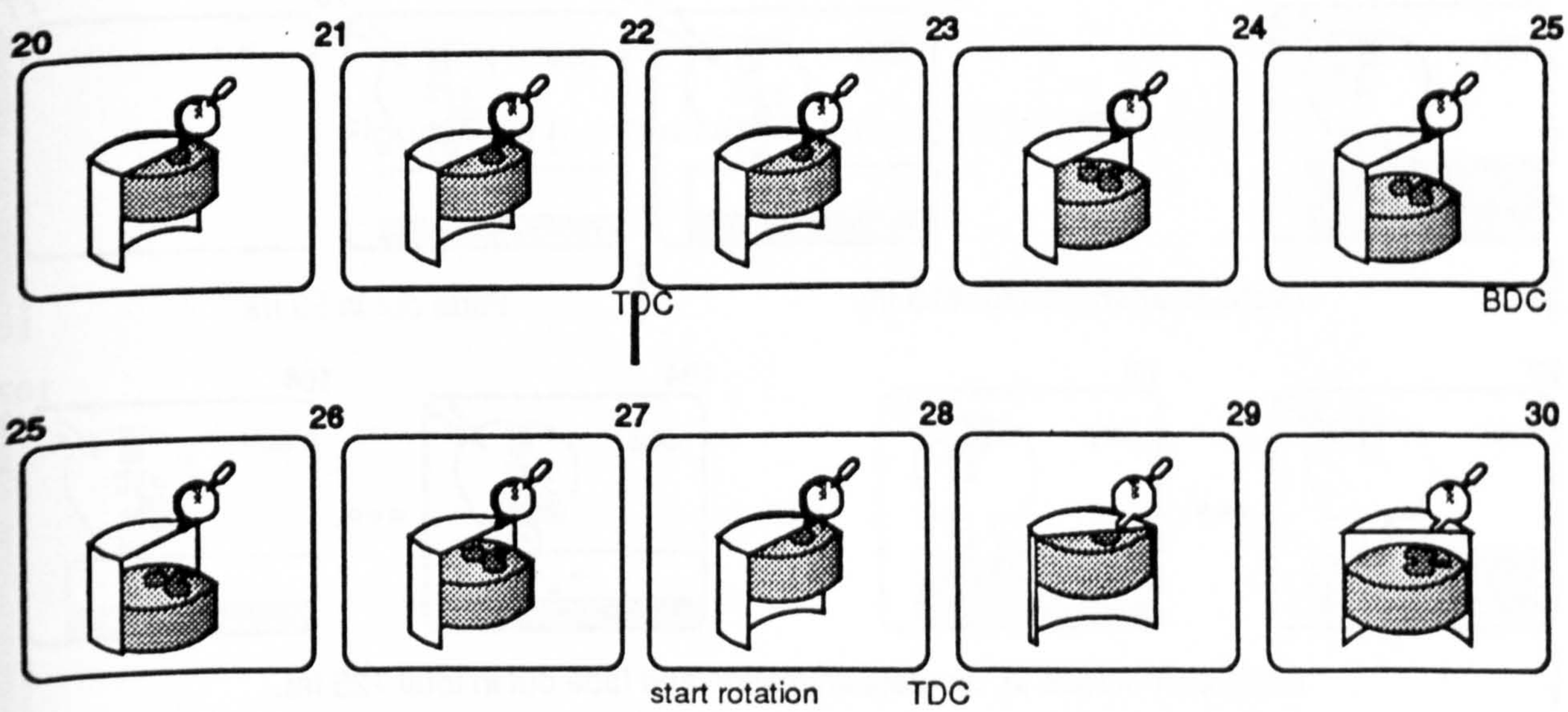
'Fade_out_front_title' anim. script



'Fade_in_cover' anim. script. Fades-in cover in 50 frs and holds tail for 100 frs.



'Fade_out_front' anim. script. Fades out front in 50 frs and holds tail for 100 frs.



'Engine_cycle' anim. script. Cycles engine and moves camera for 600 frs. and holds tail for 25 frs.

Figure 7.55 Storyboard of the animation.

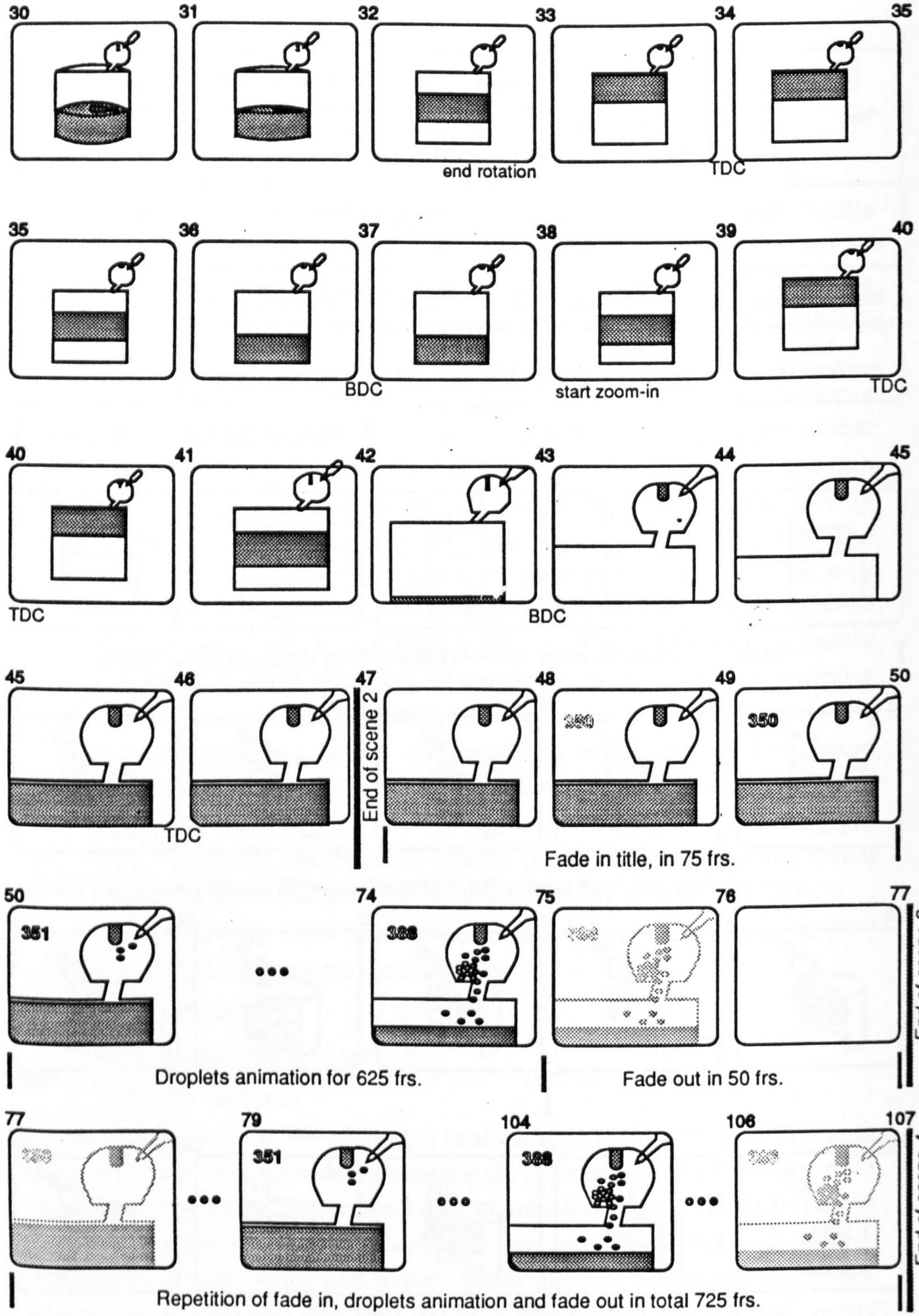


Figure 7.55 (continued) Storyboard of the animation.

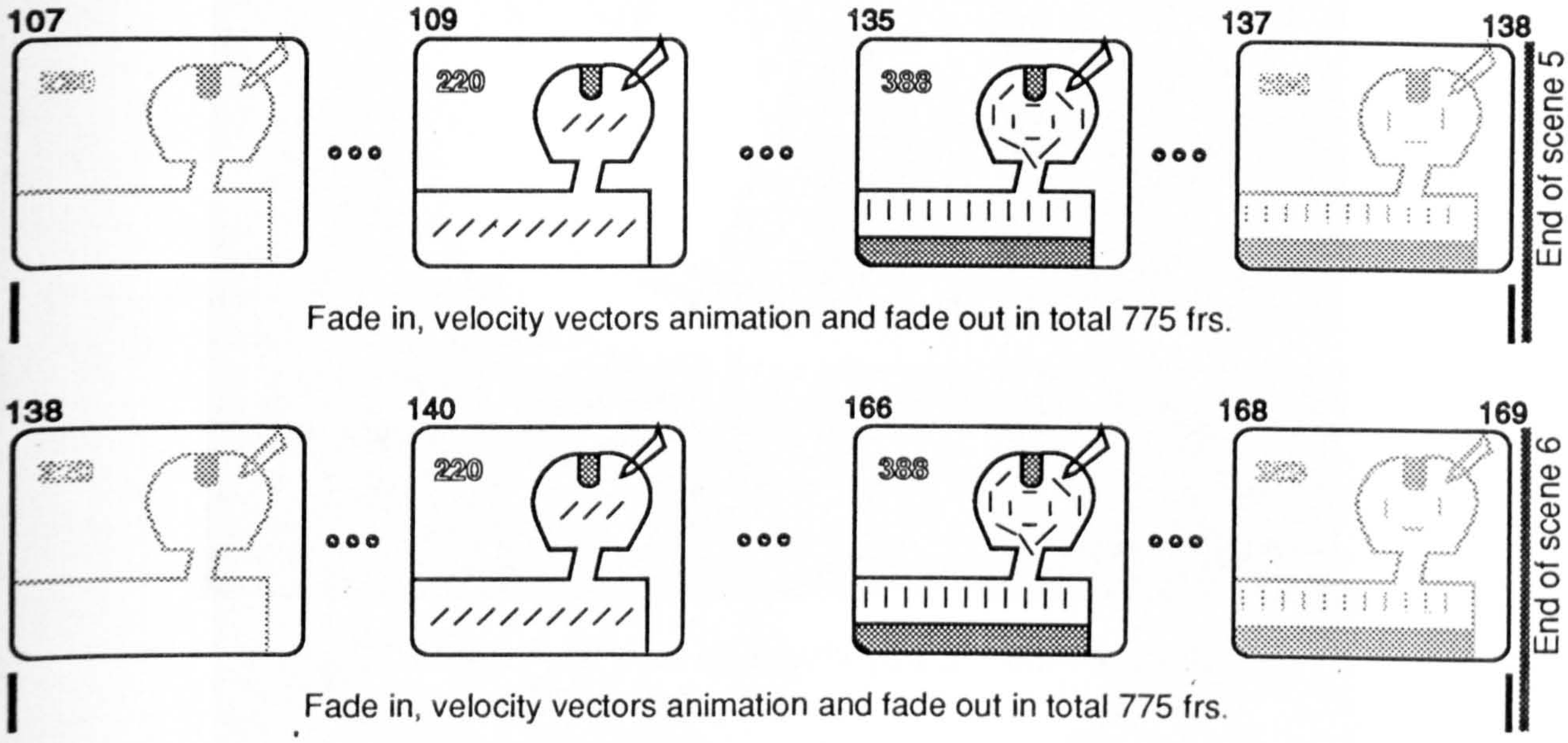


Figure 7.55 (continued) Storyboard of the animation.

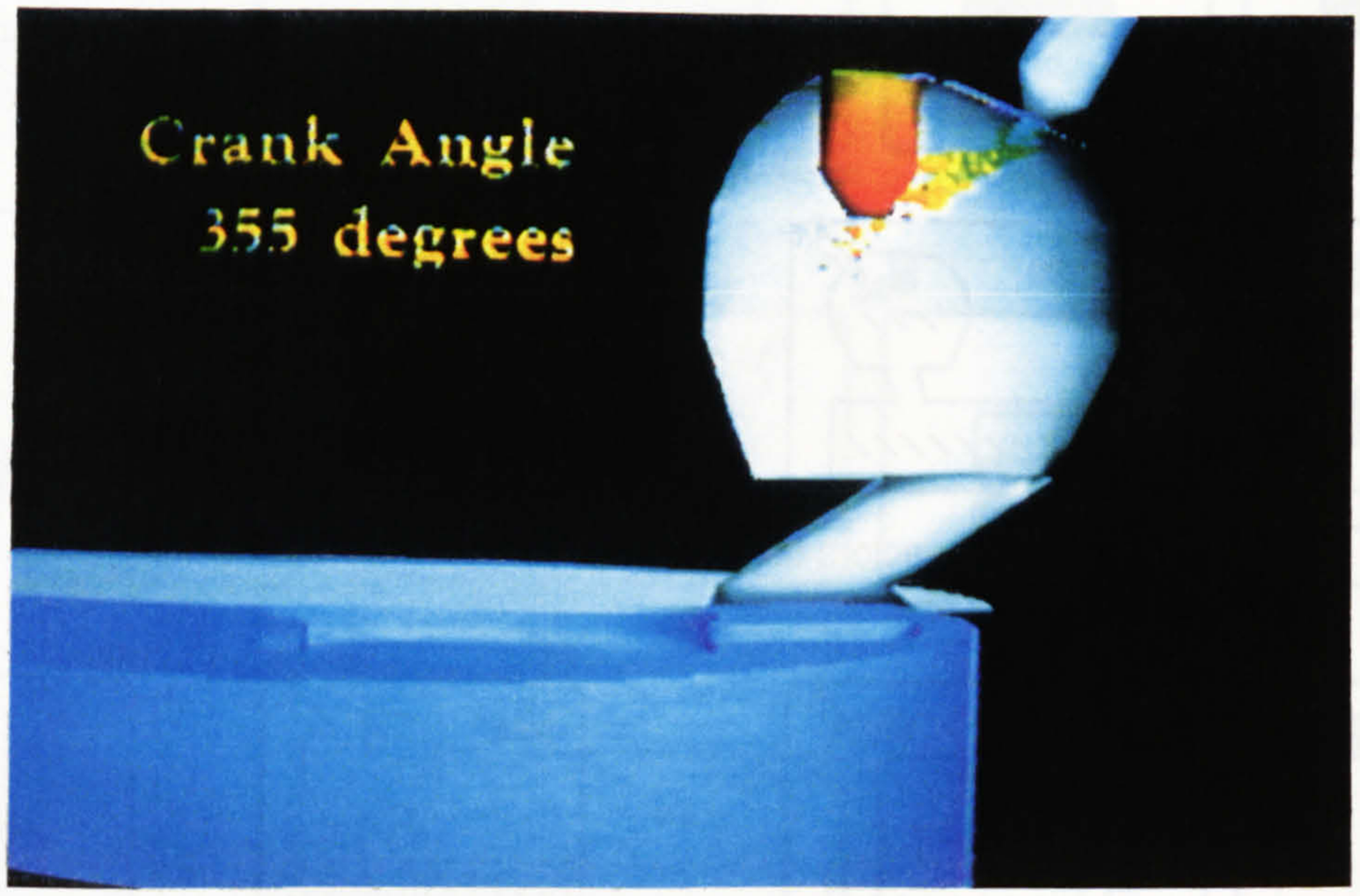


Figure 7.56 Spray visualisation at 355° CA ATDC

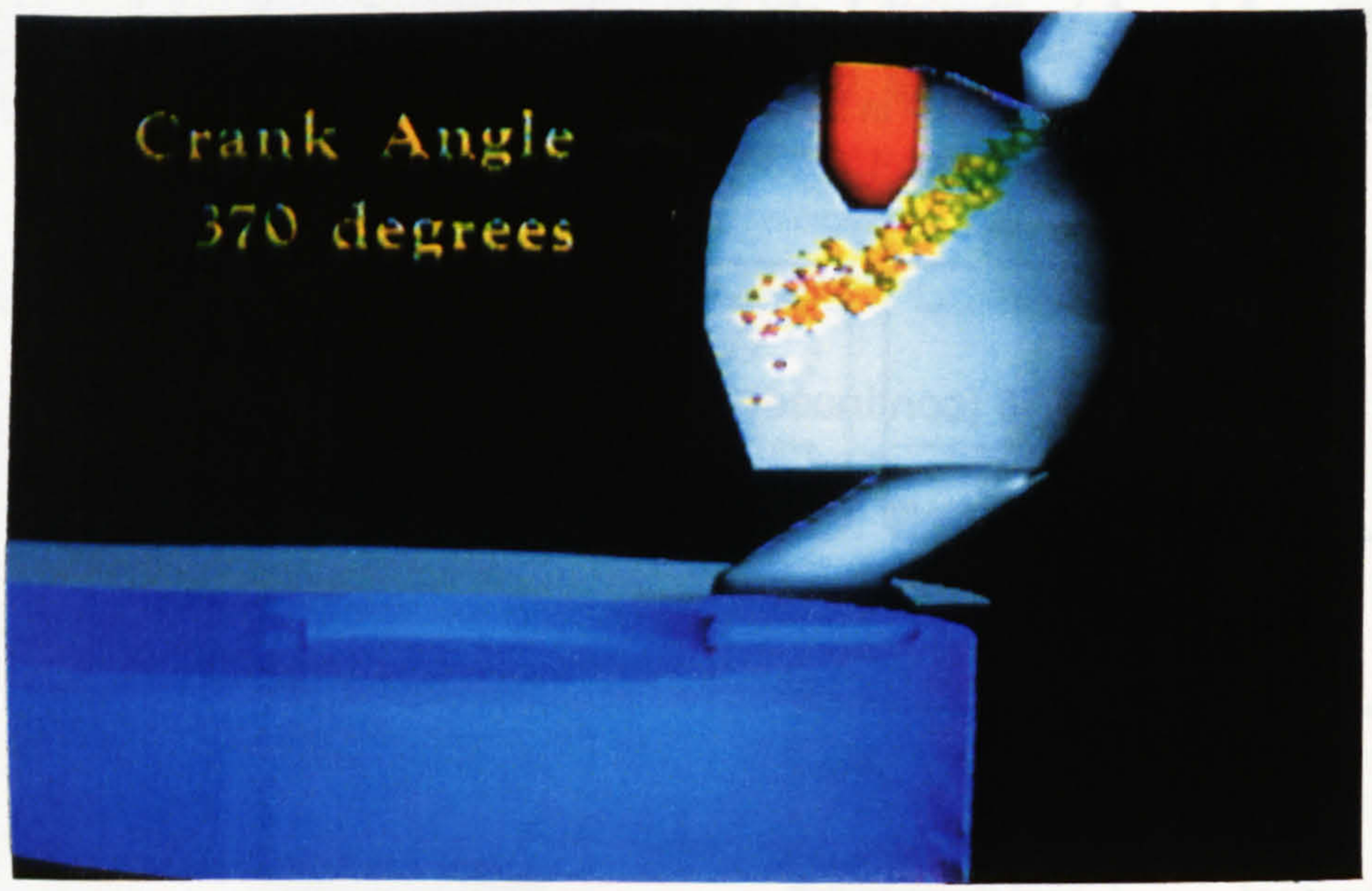


Figure 7.57 Spray visualisation at 370° CA ATDC

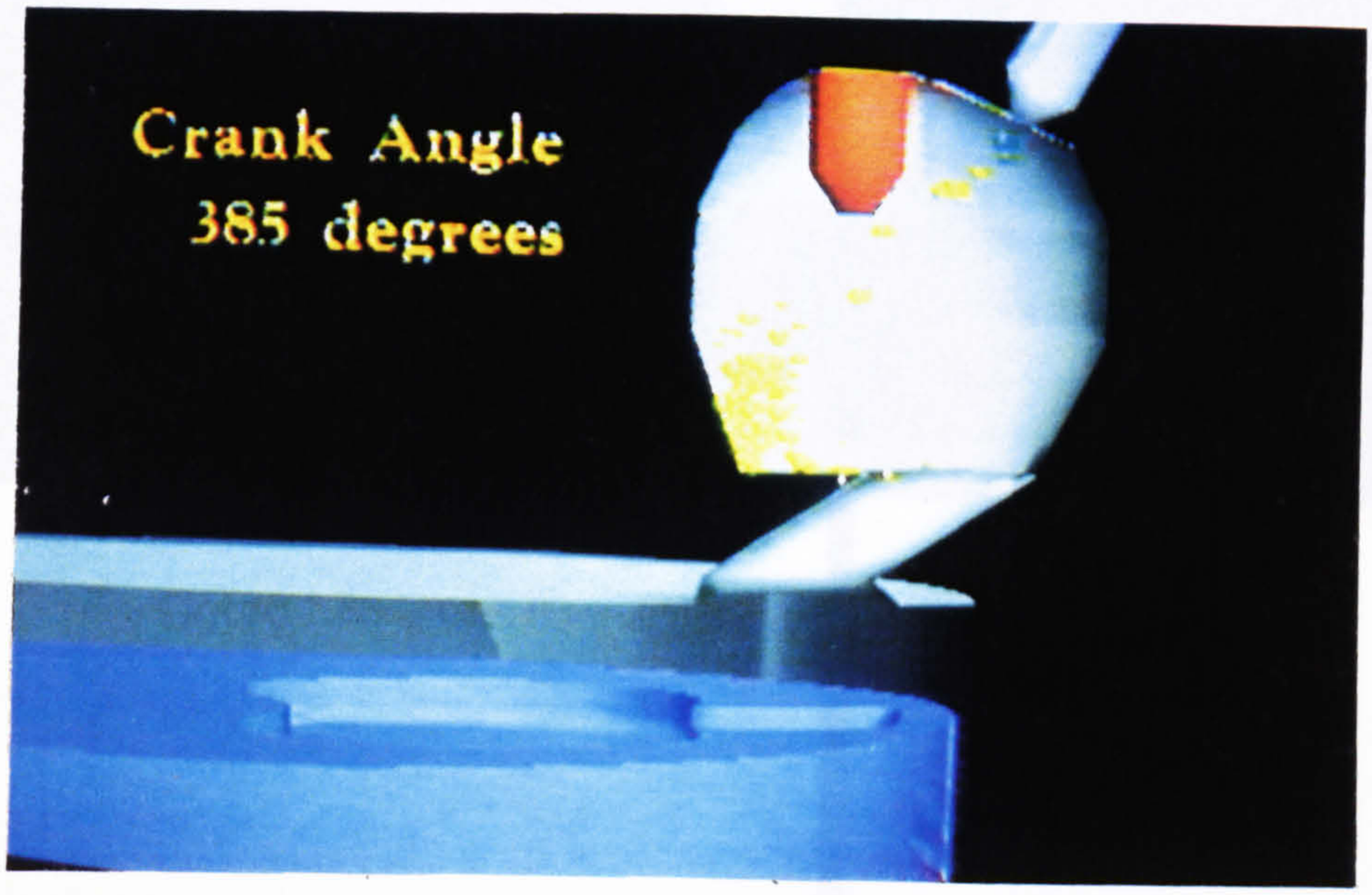


Figure 7.58 Spray visualisation at 385° CA ATDC

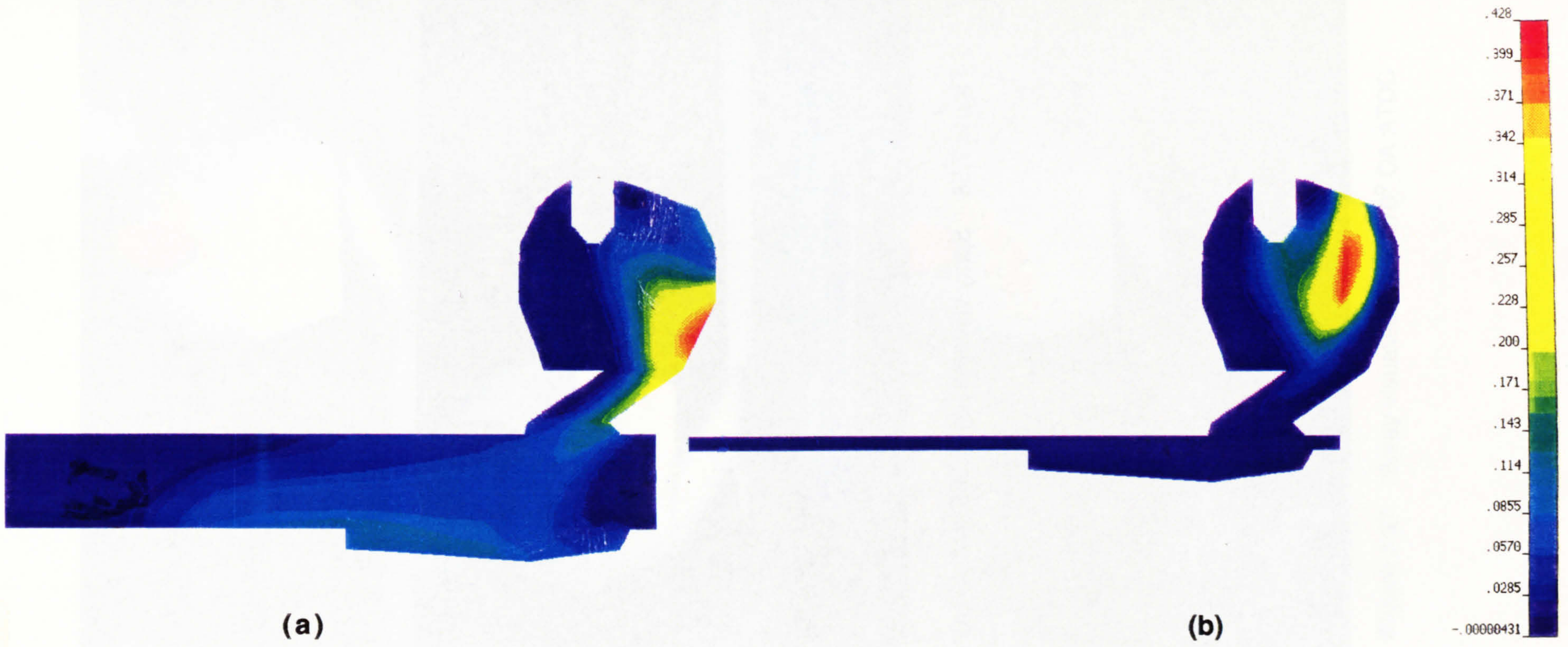


Figure 7.59 Renault IDI Diesel engine.
Fuel vapour concentration at (a) 360° CA ATDC and (b) 390° CA ATDC.

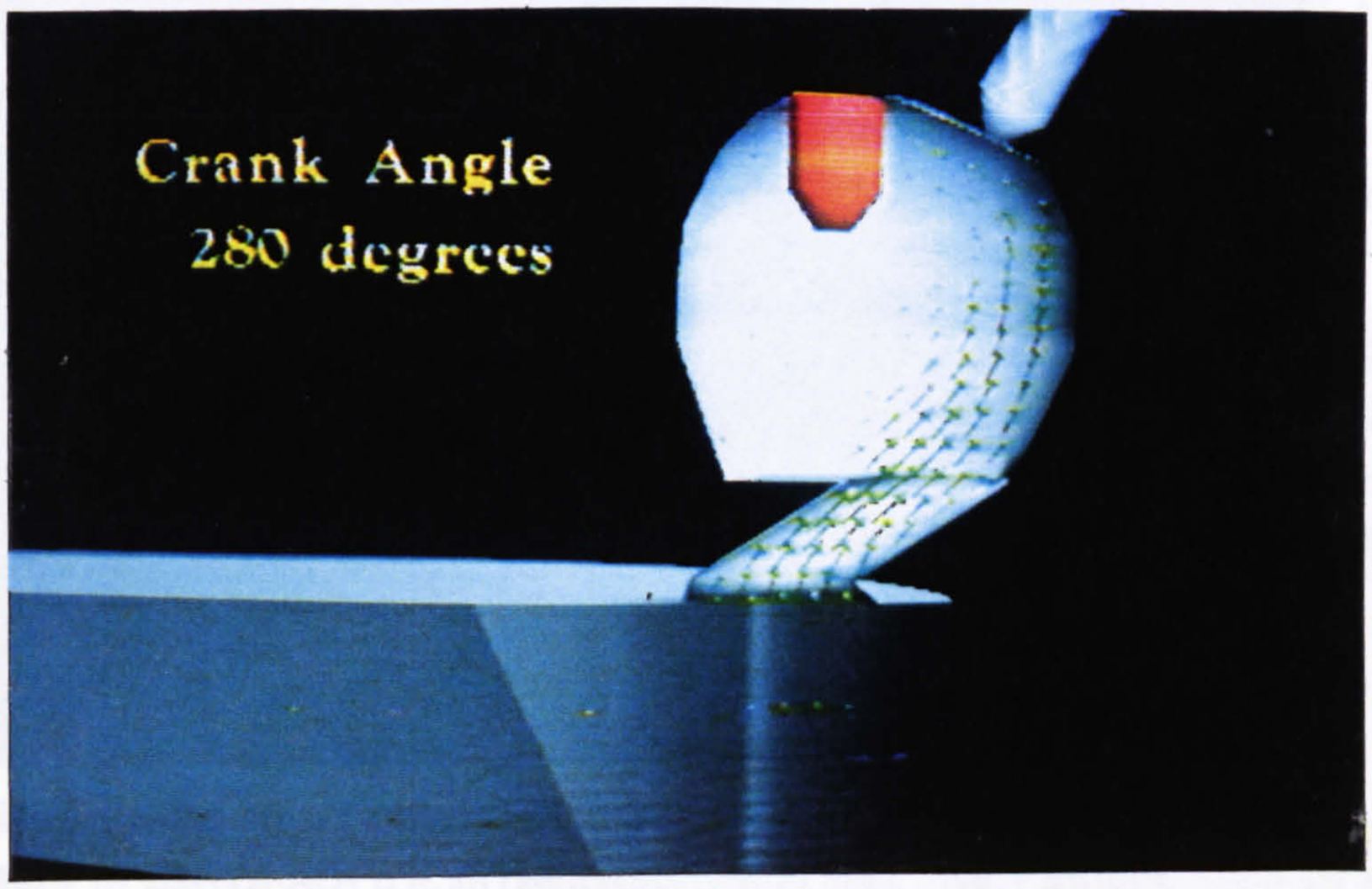


Figure 7.60 Gas flow visualisation at 280° CA ATDC

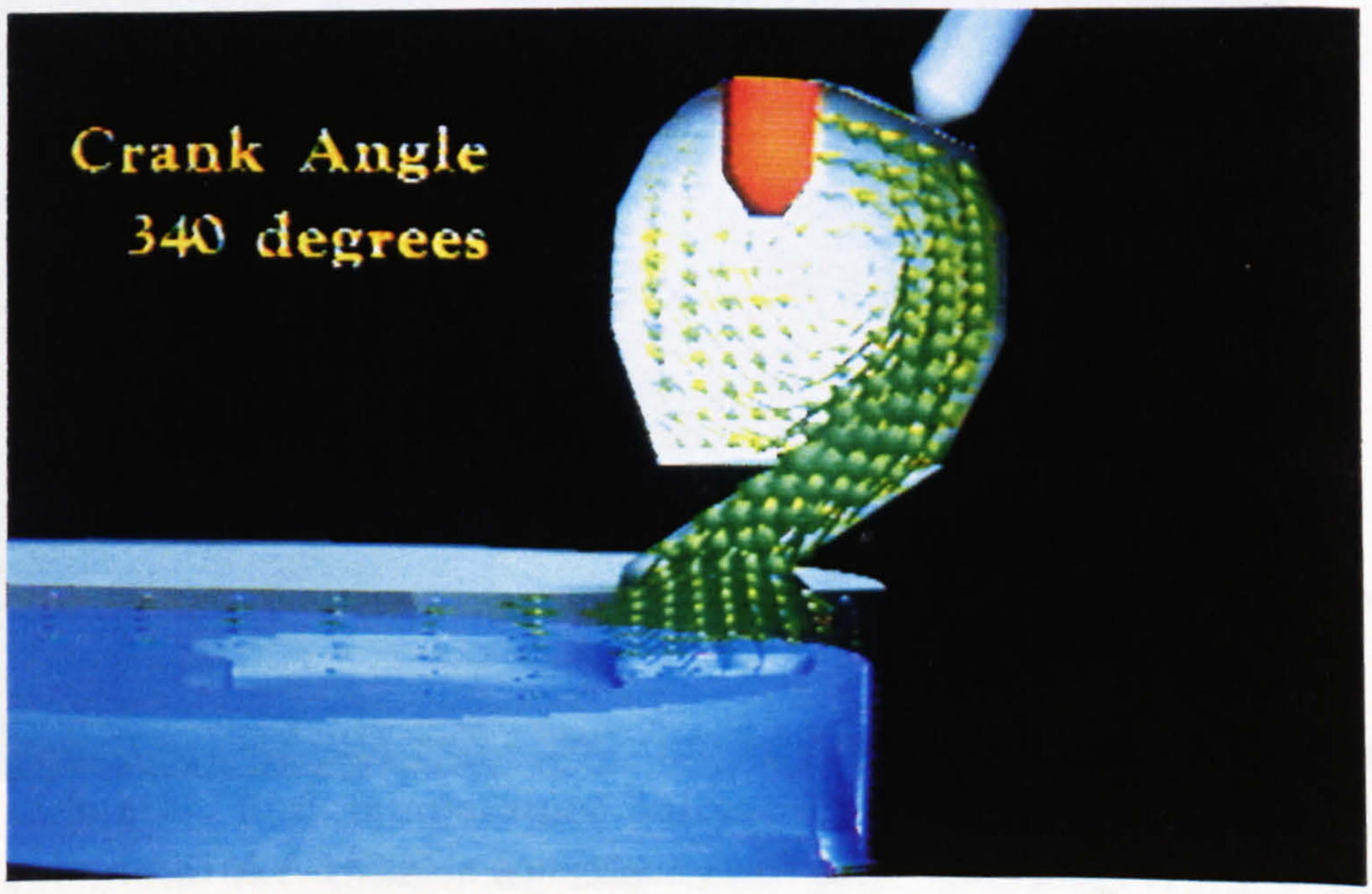


Figure 7.61 Gas flow visualisation at 340° CA ATDC

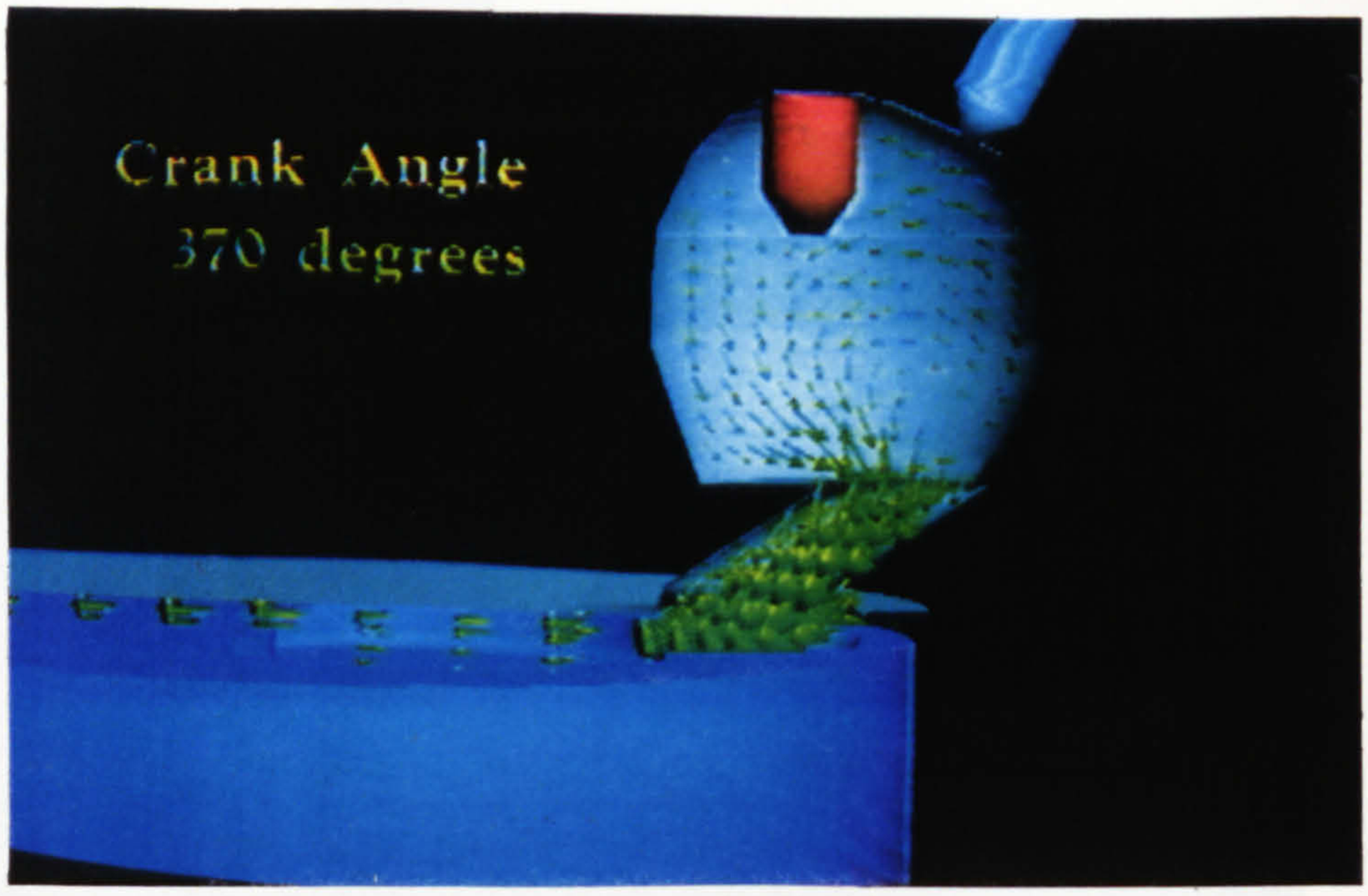


Figure 7.62 Gas flow visualisation at 370° CA ATDC

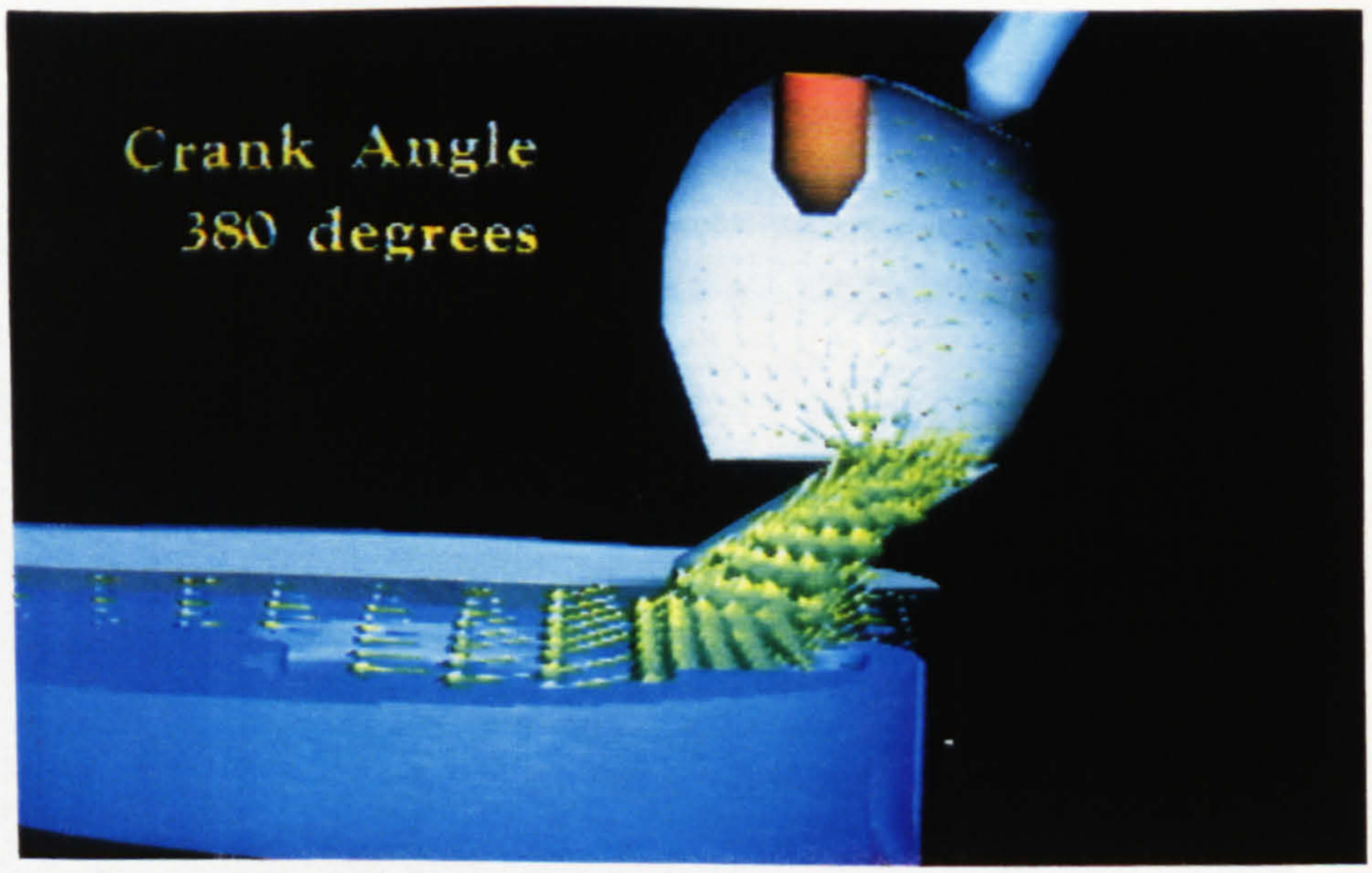


Figure 7.63 Gas flow visualisation at 380° CA ATDC

7.3.4.3 Production of the animation with the AERAS system

The AERAS animation system was used for the production of the animation and its layoff onto video tape. This was accomplished in the following stages.

7.3.4.3.1 Creation of geometry objects with the AERAS system

The SPIL module of AERAS was used to extract vertex and cell-connectivity information from the simulation data set and create surface models for the liner, cylinder head, and piston crown. Then the surface models of the liner and cylinder head were split in two sub-models for each model, representing the front and back parts of the initial models. These were stored in the form of rudimentary objects, or 'atoms'. These were combined with objects representing the injector and the glow plug, that were obtained from the AERAS library of primitives. The above objects were organised in the super-object 'chamber' of the AERAS system, that has been described in section 6.5.2.3.1. The structure of this object can be seen in figure 6.8. Its description can be found in appendix C1.

In order to make the animation more comprehensive, different colour and transparency values were assigned to each object as well as to the background. These values were (see also figure 6.9):

- Background: black, opaque;
- Piston: blue, opaque;
- Glow plug: red, opaque;
- Injector: white, opaque;
- Engine cover back: white opaque;
- Engine cover front: white, initially semitransparent, then fully transparent.

7.3.4.3.2 Creation of spray visualisations

Spray data for the animation were created with the SPIL module of the AERAS system. In more detail SPIL was used to:

- Scan the SPEED results file – for all timesteps – to extract location, diameter, and temperature information for each droplet and to calculate the minimum and maximum diameter values of the simulation droplets for each timestep.
- Calculate the minimum and maximum simulation temperature values over all the timesteps.

- Calculate droplet-diameter visualisation values that ensure efficient representation of sprays. This was accomplished by linearly fitting the SPEED calculated values to visualisation ones, ranging between minimum 0.3 mm and maximum 1.5 mm. This procedure has been described under the header 'data filtering' in section 6.5.2.3.2. This was necessary for two reasons:

i. The calculated from SPEED droplet size was too small (maximum diameter was 0.5 mm) to be visible without scaling.

ii. The smallest droplet was order of 10^3 times smaller than the biggest one. This means that without linear fitting of the visualisation diameter, for the smallest droplet to be visualised with the size of one pixel, the biggest droplet should cover the entire computer screen.

- Fit linearly droplet temperature on a colour scale from green to red.
- For each timestep create an object 'spray' defined from a set of visualisation droplets and store it in a file . Each droplet references a primitive sphere, that is taken from the AERAS primitives library and is scaled according to its diameter and coloured according to its temperature.

Spray visualisations can be seen in figures 7.56 to 7.58.

7.3.4.3.3 Creation of gas flow visualisations

In order to generate gas flow velocity vectors for the animation, the SPIL module of AERAS was used to:

- Scan the simulation data set to extract gas velocity data at the cell centroid of each cell and calculate minimum and maximum velocity magnitudes for each timestep.
- Define a cutting plane and store it in an object file.
- Create an atom vector – for each vertex in object 'cutting plane' – that is based on a primitive taken from the AERAS primitives library. Its length is the linear fit of simulation velocity values between minimum and maximum visualisation velocity values.

Gas flow visualisations can be seen in figures 7.60 to 7.63.

7.3.4.3.4 Creation of animation scripts

Once objects were created, they were organised in animation scenes. For this, lighting and motion attributes were assigned to each object. The

choreography of each scene was described in time and space – on a frame-by-frame basis – in animation scripts of the system. The animation script for scene two is given in appendix C2.

7.3.4.3.5 Lighting and raster tests

In order to properly light all the areas of action, four light sources were defined, named 'lite1', 'lite2', 'lite3', and 'lite4'. Their definition can be found in the 'chamber' object file in appendix C1. Their position and direction is:

lite1 located in the squish region and pointing towards the piston crown;

lite2 located in the squish region, illuminating the inner-top part of the cover;

lite3 located in front of the model, lighting the prechamber, the squish region, and the piston crown;

lite4 located at the top-left-rear side of the model, illuminating the top and side regions of the engine cover.

Once the position and attributes of light sources had been decided, storyboard reference or 'key frames' were rendered to check the position of the light sources and to form an idea of what the rendered images look like. The initial light source positions had to be adjusted, in order to illuminate regions that became shaded during the animation. The attributes of the light sources had to be adjusted to maintain the colour of objects, and – by addition of specular illumination – to reveal the metallic texture of surfaces. The illuminated model can be seen in figure 6.9.

7.3.4.3.6 Motion development, wireframe and raster flipbooks

At this stage it was necessary to preview the motion of the model and the camera. This was done by creating a wireframe flipbook animation of the action. This enabled refinement of the motion that produced a pleasant motion effect.

As a final examination of motion and illumination of objects, a Lores frame buffer animation[1] of each scene was produced. By doing so it was noticed that the computation of camera motion by linear interpolation[2] of

---

[1] A lores frame buffer animation is a flip-book type animation of low-resolution images that have been previously stored in the memory of the computer.

[2] Interpolation is used to calculate intermediate values between the starting and ending values of an action. The interpolation process is controlled by an *interpolation function* or *timing*

camera positions, according to which the camera position changed in equal steps in each frame, resulted to abrupt camera motion, mainly at the start and end *positions of the camera. To overcome this unpleasant effect an eased interpolation of camera motion was applied, so that the camera* appears to speed-up at the start and slow-down towards the end of motion.

7.3.4.3.7 Rendering of the animation

The objects and animation scripts were read into module RENDERER of AERAS (see figure 7.64) to:

- create intermediate values for the intervening frames between key frames;
- apply interpolation functions to control the interpolation process;
- render images with 24 bits of colour resolution per pixel to produce frames for the animation.

The rendering process for scenes two to six was time consuming, therefore it was computed in batch mode. Rendered images of key frames of the animation can be seen in figures 6.9, 7.56 to 7.58, and 7.60 to 7.63.

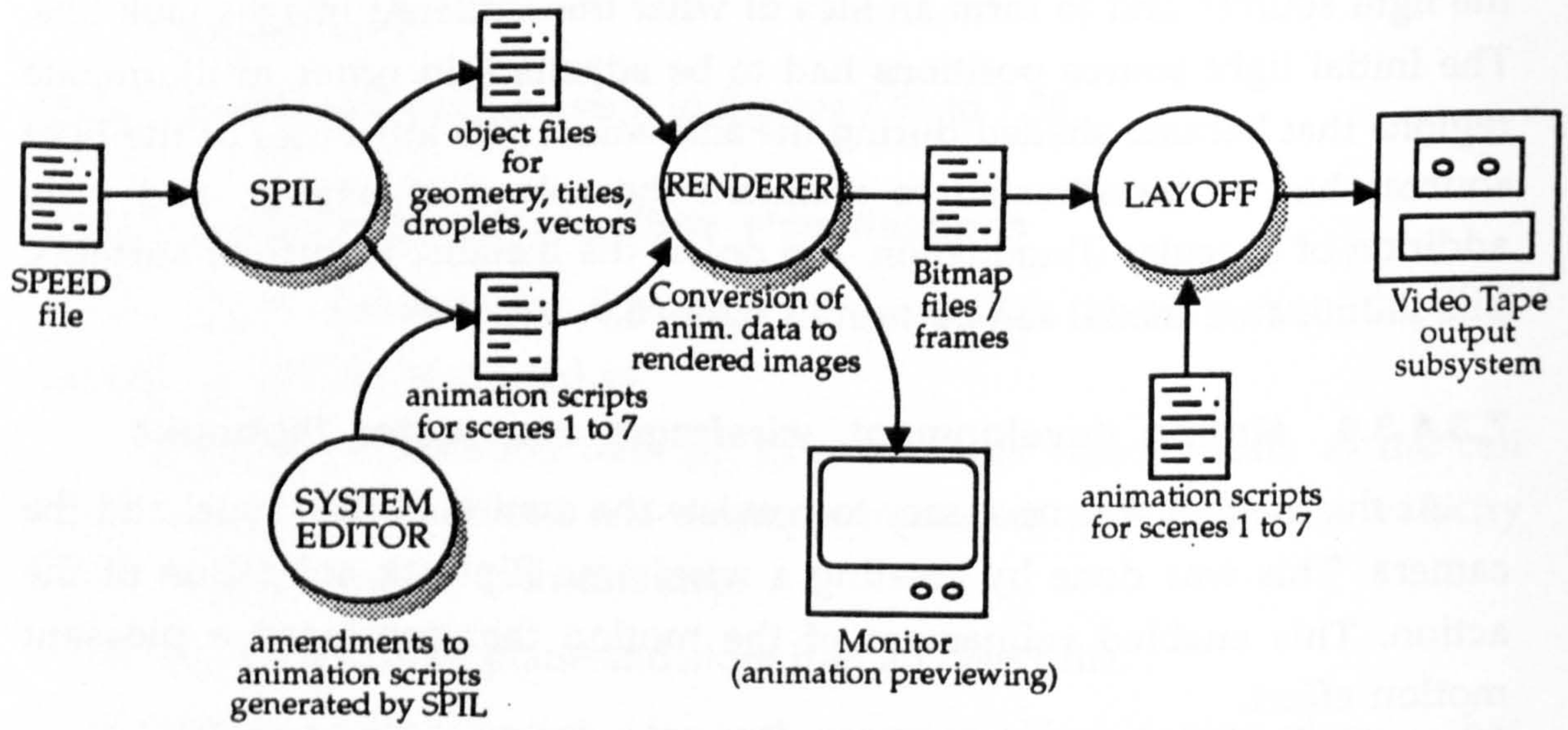


Figure 7.64 Data flow diagram of the animation process with the AERAS system.

---

curve, that determines the rate of transition through the path of values as determined by the set of start and end values and the interpolation method.

7.3.4.3.8 Recording of the animation and hardware implementation

The animation was rendered on a network of Apollo workstations, where frames were created simultaneously on several workstations using distributed parallel processing[1]. An Apollo DN10000 workstation was used as the main computing and storage component of the network. The display processor memory of the workstation was expanded by means of a frame buffer capture card. The animation was recorded frame-by-frame to videotape. Images – in the RGB signal format – output by the computer were encoded to the standard PAL composite video signal by means of a Lyon/Lamb encoding device. A Mini-Vas animation controller was used to control the rewind and pre-roll on the tape recorder required for placing each single frame on the tape. The controller itself was controlled from the workstation using an RS232 connection and the LAYOFF module of AERAS (see figure 7.64). The videotape recorder was a Sony 9850P with single-frame recording capability. The hardware used for signal conversion and control of the tape recording device has been described in detail in chapter 6. The interconnection diagram of the workstation and video output systems can be seen in figure 6.15.

7.3.4.3.9 Time and disk-space requirements

Depending on the complexity of the rendered image, it took about 30 CPU seconds to render a simple frame (titles) and up to about 17 CPU minutes to render the most complicated frame (that of the combustion chamber with hundreds of droplets in its domain). It should be noted that 25 frames are required for one second of animation and that it takes 20 seconds to record each frame to videotape. Therefore, a minute of animation required 8.3 hours to record to videotape.

The disk-space requirements for the above animation were very high, in comparison to other type of visualisation tasks. Each title image when stored required 0.2 MBytes, whereas each engine with droplets image required an average of 1.2 MBytes. This means that for the droplets animation sequence in scene three, and with picture repetition rate 8, 94 MBytes needed to be stored, whereas for the fade-in and out sequences, where the picture repetition rate was 1, an additional 65 MBytes needed to be stored. Therefore, the total storage required for scene three was 159 MBytes. The storage requirements for each scene are given in the following list:

---

1 Unused processing power on any computing units connected to the network was used by a facility called 'Apollo distributed renderer'.

Scenes 1 and 7.	20 MBytes
Scene 2.	151 MBytes
Scenes 3 and 4.	159 MBytes
Scenes 5 and 6.	188 MBytes

7.3.4.4 Problems encountered

Creating the animation was not a straight forward process. Animations often looked different than what was designed on paper and, in several cases, visualisation features had to be tried several times prior to recording. In this process the animation previewing facilities of the AERAS system proved very useful. Some of the more important problems that were encountered during the modelling, translation, and animation stages are presented here, on the grounds that they are indicative of the problems involved in flow visualisation with animation methods.

It was noticed that blue-coloured droplets were not clearly visible, because of the blue colour of the piston. The solution was to set the visualisation colour-spectrum from red to green – red for hot, green for cold droplets – excluding the values of blue from the spectrum.

When the first animation of the flow field was produced, the flow seemed to be changing direction, when it was actually only reducing in strength. The source of the problem was the fact that vectors were anchored at their base. Anchoring vectors at their centre solved the problem but not in a very satisfactory way. It became clear that the best way to dynamically visualise the flow field is not with velocity vectors but with 'zero mass' particles seeded into the flow, such that they would follow it.

7. Testcases

Chapter 8

Conclusions

8.1 Achievements and findings

A variety of tools, in the form of raw algorithms or procedures using existing CAD and animation systems have been assembled for mesh generation, simulation setup, and visualisation in conjunction with general-purpose computational fluid dynamics codes.

These tools have been demonstrated through application to the SPEED code, used for simulation of gas motion and combustion in internal combustion engines. The main features of SPEED have been summarised in Chapter 3. Its CFD methodology for calculation of gas flow, fuel injection, auto-ignition, and combustion on unstructured computational meshes with hexahedral cells have been shown to permit simulation of real engine geometries with meshes of controlled distribution.

The industrial use of the SPEED code had been – at the time of initiation of the present work – restricted by the existing pre- and post-processing systems, which:

- were confined to structured or block-structured meshes. These are not easy to generate and often not efficient for calculations in engine cylinders;
- were limited to machine dependent visualisation tools, mainly for 2D vector and line-contour plotting of fields that are defined over structured meshes;
- required programming of the pre- and post-processing tools needed for each application.

It has been shown that the above limitations can be overcome by a combination of solid-modelling, mesh-generation, and visualisation tools.

Solid modelling tools, mainly in the form of CAD systems, provide means of interactive definition of the surfaces of the combustion chamber enclosing the computational domain, in this way providing an input for subsequent mesh generation. Analytic solid modelling tools in particular result in a segmented-into-blocks domain, an advantage if block-meshing procedures are to follow.

Mesh Generation tools provide means of fitting a computational mesh in the chamber volume. Ideally, such tools should be applied directly in domains enclosed within the aforementioned surface models. Algebraic and

advancing-front techniques are computationally efficient and have been shown to be well suited for this task.

Visualisation tools provide means of conversion of raw simulation data to graphical images. Single or multiple cutting planes, isosurfaces, and particle tracking have been identified as the most appropriate tools for visualisation of 3D fields. A variety of tools for manipulation of calculation results prior to their visual presentation have been invented by the author. They allow efficient handling of voluminous results files, check of mesh quality, and calculation of integrated properties or additional fields from existing ones.

A consensus has been identified in the engine community that any further industrial application of CFD will require – CFD and physical modelling left aside – codes that are more easy to learn, use, and modify. A variety of techniques have been investigated in this direction. Graphical user interfaces, mainly of the desktop metaphor kind, have been found to be the most appropriate ones, because they comprise windows, pop-up menus, and mouse-controlled actions that users are familiar with. Shading, occlusion or partial transparency, perspective, and depth-cueing have been found to provide helpful perceptual cues for interpretation of 3D geometry and results visualisations. Finally, modular programming, object orientation, and database management have been found to enable the development of portable, flexible, and long-lasting codes.

A system has been developed by the author that combines most of the aforementioned tools in a pre- and post-processing environment for SPEED. This has been presented in chapter 5. In it:

- Graphical operations of CAD systems – PATRAN and IDEAS. are supported – have been employed for mesh generation in a graphical environment. These involve analytic solid modelling techniques for definition of the computational domain and decomposition thereof into an appropriate arrangement of large mesh-blocks. A mesh – unstructured in the general case – is subsequently generated in each block by means of mapping isoparametric techniques. Block-design and mesh-fitting strategies for the squish, bowl, and cylinder-head regions have been given. They produce meshes that are computationally efficient for a SPEED simulation and facilitate application of the piston and valve motion procedures thereof.
- Automatic mesh-generating procedures have been developed to complement the above meshing operations. They enable parametrical

definition of meshes, so that geometry features can be readily modified and fine meshes can be derived from coarse ones.

- A numerical pre- and post-processing code has been developed to handle numerical tasks related to:
  - Conversion of PATRAN or IDEAS meshes to the SPEED format. Additional translators have been developed for conversion of KIVA or PROSTAR-generated meshes.
  - Input of engine operation conditions (e.g. engine speed, valve lifts, type of fuel, injection conditions etc.), calculation control data (e.g. type of run), fuel injection parameters, and fuel and combustion model parameters.
  - Definition of initial conditions in the bulk and boundary conditions on inlet or outlet flow boundaries and walls.
  - Checking of the addressing, distortion, skewness, and aspect ratio of the mesh and the user-input simulation conditions for suitability for CFD engine calculations.
  - Calculation of engine performance related parameters (e.g. mean indicated pressure) and integral properties of the moving gas in the cylinder (e.g. centre of mass, kinetic energy, angular momentum, swirl ratio).
  - Data manipulation prior to visualisation (to select a 3D portion of the mesh, display data on user-selected surfaces, or to calculate fields from existing ones).
- Graphical tools of CAD systems have been employed for the visualisation of geometry and fields. They include shaded views of the geometry (surface or mesh) and portrayal of vector and filled-area or continuous-tone contour plots on user-specified cutting planes.
- Finally, a graphical user interface has been provided to the whole process.

The main advantages of this pre- and post-processing system over its preceeding ones have been identified as:

- It provides state-of-the-art tools for accurate definition of the surfaces bounding the combustion chambers of real engines (including valve chamfers, valve indentations on the piston crown, bowl lips etc.)
- It provides procedures that have been shown to greatly facilitate generation of unstructured, hexahedral-cell meshes of user-controlled distribution for optimum resolution. The effectiveness thereof has been shown in a variety of Diesel and petrol engine designs.

- In the case of DI Diesel engines, the above process is simplified by semi-automatic mesh-generation utilities.
- It is furnished with visualisation tools that have been shown to be suitable for interpretation of 3D fields.
- It provides utilities for definition of initial and boundary conditions, engine operation conditions, and data checking prior to calculation or visualisation.
- It provides utilities for calculation of engine performance related parameters and integral properties of the moving gas in the cylinder, that are commonly used in the automotive industry.

Finally, the following advantages of the system derive from the fact that it is based on CAD systems:

- The supported CAD systems are widely used in the automotive industry. This ensures full portability of the system throughout a wide range of hardware platforms and operating systems.
- The wide availability of CAD systems in design departments of automotive industries has resulted to high levels of familiarity with their user interface within the engineering community.
- CAD systems are widely used in other phases of the engine design process, such as stress analysis, heat transfer through engine parts, etc. Use of CAD systems in CFD engine simulation enhances its integration with other engine design processes.

Two major inadequacies, stemming from CAD system limitations, are the inability of the above system to efficiently visualise fuel sprays and to provide means of dynamic visualisation of time-dependent fields. The former has been overcome by customisation of the PROSTAR code for use with SPEED results. The latter has been overcome by development of the animation software system, named AERAS, that has been presented in Chapter 6. It provides a complete software and hardware environment for the dynamic visualisation of SPEED results. It combines a proprietary animation system, Intelligent Light (IL), and a numerical postprocessor, SPIL. The AERAS system has been shown to enhance the comprehension of SPEED results and enable the visualisation of 3D, time-dependent phenomena that could not be readily viewed with the existing system. One example is the dynamic portrayal of the development of fuel sprays.

The merits of AERAS that originate from the features of IL, as well as from the presented implementation and the SPIL code have been identified as:

- It provides a complete software and hardware environment for the dynamic visualisation of simulation results produced by the SPEED code.
- It is a powerful tool for high-quality rendering of large data sets.
- It produces animation scripts for the control of the animation.
- It provides facilities for previewing the animation prior to rendering and recording.
- It combines visualisation of fields with computer models of engine parts and titles. This renders AERAS a suitable system for presentations of engine simulation results.
- Finally, it makes use of existing hardware, in this way minimising the hardware expenses of the whole system.

The limitations of the presented system have been identified as:

- The elaborate rendering algorithms make rendering a time-consuming process, that is done in batch mode. In this way interactive live video is not possible and the usefulness of the presented system as a visualisation tool for everyday use is limited.
- The customisation of IL – a code fashioned for media applications – for use with SPEED has been achieved through the translator code SPIL. This has increased the complexity of AERAS (e.g. the type of the visualisation is selected in SPIL, not in IL) and imposed additional storage overheads.

The effectiveness of the above systems has been evaluated – in Chapter 7 – with reference to SPEED calculations of gas flow, fuel injection, auto-ignition, and combustion in three designs of Diesel engines: two of the direct-injection and one of the indirect-injection type. The direct-injection engines have the same cylinder-head and valve design. They differ in the piston configuration: a 'transparent' version has a piston bowl with a flat base, whereas the other, 'series', version has an 'omega' shaped bowl. Meshes of different density have been generated in all cases with the preprocessing system presented in chapter 5. In the case of the DI engines the automatic mesh-generation tools thereof have been applied. The chief figures of the resulting meshes have been identified as:

- The combustion chambers have been accurately modelled.
- The meshes have a layered topology, compatible with the routines of SPEED that adapt the mesh to the motion of the piston and valves.
- The meshes have been shown to satisfy the quality criteria for cell orthogonality, aspect ratio and warpage that are required by SPEED.

The pre- and post-processing system presented in Chapter 5 has been used for specification of engine operation conditions and initial and boundary conditions for the simulation. Calculations were performed with the SPEED code. In the case of the direct-injection engines, results have been visualised and analysed by means of the same pre- and post-processing system. The chief features have been identified as:

- Swirl is created by bringing the intake flow into the cylinder with an initial angular momentum. Intake-generated swirl persists through the compression and combustion processes resembling solid-body rotation within the bowl and around the bowl-axis towards the end of the combustion stroke. As the piston approaches the TDC position, this rotational motion is substantially modified due to squish-generated air motion into the bowl.
- In the transparent engine, injected n-heptane droplets approach the wall of the bowl without reaching it, whereas in the series version n-dodecane droplets hit the bowl wall at TDC of compression. The cause of this has been identified as the slower evaporation of n-dodecane (boiling point more than twice that of n-heptane) and the lower cylinder pressure and temperature (due to the lower compression ratio) during injection in the case of the series engine.
- The droplets at the tips of the sprays are slightly deflected by the swirl. Near the bowl walls the fuel vapour is blown around by the swirl.
- The ignition delay is shorter in the case of the transparent engine than in the series version. The reason has again been identified as the faster evaporation of fuel in the former case.
- At TDC of compression the top part of the bowl is almost filled with flame. The peak combustion pressure is 91 bar in the transparent engine and 74 bar in the series version. This follows the difference in the compression ratio (20.5 in the former and 19.5 in the latter).
- Shortly after TDC of compression the flame spreads out over the piston crown toward the cylinder wall due to combustion-produced gas expansion and the reverse squish flow. There is a gradual diminution of the burning regions starting from the cylinder head as the remaining fuel vapour mixes with air and burns.
- The calculated peak combustion pressure, spray penetration, ignition delay, mean indicated pressure, and specific fuel consumption in the series engine case are in good agreement with experimental measurements and the manufacturers specifications.

The SPEED calculated results in the case of air flow and non-combusting spray simulation in the combustion chamber of the indirect injection Diesel engine have been dynamically visualised by means of the AERAS animation system. The animation of gas velocity vectors and injected fuel droplets has indicated that:

- During compression a jet is formed in the throat and in the region of the prechamber below the injector. This jet increases in strength developing swirl in the prechamber as the piston approaches the TDC position. At TDC of compression the swirl in the prechamber resembles solid-body rotation. As the piston moves downwards this rotational motion is modified due to gas expansion towards the main chamber. The swirl centre moves towards the entry of the throat.
- The tip of the spray is deflected towards the glow plug by the swirl. Under motoring conditions, the spray impinges on the wall opposite of the injector and, subsequently, fuel droplets are blown into the throat.

8.2 Suggestions for further work

In the surface modelling field, CAD systems – widely used in the automotive industry – will remain the obvious option for engine-geometry definition for the foreseeable future. Alternative approaches, in the form of surface-modelling software libraries such as ACIS and SHAPES that can be integrated in existing pre-processing systems, might be expected to be of benefit, albeit at a very high licencing cost.

The weakness of the mesh-generation approach that has been suggested in chapter 5, is the requirement for an educated partitioning of the flow domain into mesh-blocks. The provided semi-automatic mesh-generation utilities are worthy of expansion to cater for a larger variety of engine designs. However, it seems clear that alternative mesh-generation procedures, capable of generating a mesh from the surface boundary definition with minimum user intervention, must be investigated. The advancing-front method would appear to be a potentially fruitful route of exploration.

Visualisation of the user-specified engine operation, initial, and boundary conditions is worthy of investigation. It can help in locating errors in the setup of the simulation, prior to calculation.

The data manipulation operations should be expanded to allow for colouring of the sprays and investigating the leading droplet of each spray or studying droplet size and temperature distributions on cutting planes through each spray. This may be a route towards understanding the

injection and spray formation processes and evaluation of the related models.

The most obvious aspect of the current visualisation methodology requiring improvement is the non-uniform user interface, split between the one of the CAD system or the Intelligent Light animation system and that of the numerical pre- and post-processors. A need to implement a single graphical user interface to the whole simulation process – geometry definition, mesh generation, simulation setup, analysis, and visualisation – is perceived. Such an approach is becoming increasingly possible because of the recent availability of new products with modular design and layered architecture that facilitate customisation for the particular needs of CFD engine simulations. Such products that ensure portability are:

- The Open-GL graphics library, which is becoming increasingly available in a variety of hardware platforms. It comprises high-quality graphics tools that can be readily exploited for the development of dedicated pre- and post-processing programs. On selected platforms it can be combined with the Explorer toolkit for easy access to tools well suited for visualisation tasks. It can be used with the Motif and X-windows user interface and windowing protocols to develop custom made user interface for CFD engine applications.
- Toolkit environments such as AVS, APE, FAST, and Data Visualiser. They have a modular design and a layered architecture that permits customisation for CFD engine analysis without the need for separate numerical codes.

The development of a pre- and post-processing system based on toolkit environments is restricted to visualisation operations, whereas a system based on the former approach can be extended to cover the mesh generation process. However, it cannot be emphasised too strongly, because of the high programming requirements it involves. Both approaches provide facilities for exploitation of computer networks via the X-windows and inter-process communications systems, and would allow steering of the distributed simulation and visualisation processes.

The aspect of the animation method for which investigation of alternatives is merited is the time needed to produce dynamic visualisations; the Intelligent Light approach employed here is useful for production of presentations but not for everyday use. The alternative options discussed earlier, combining real-time animation on the computer screen and recording in magnetic or digital media, would appear to be a fruitful alternative.

Appendix A

A.1 Algorithm for partitioning the domain in two regions, by means of a predefined cutting volume.

```
calculate face normal S_N for each surface of partitioning volume;
for each cell in the raw data set;
    for each face of the six faces of the partitioning volume;
        take face normal S_N;
        for each vertex associated with cell;
            define vector V from surface centre to vertex;
            if
                dot product V . S_N > 0 the vertex lies within region;
                continue calculation;
            else
                vertex is outside partitioning region;
                procede to next cell;
            endif;
        end;
since you have reached this point, cell lies within partitioning region;
place cell in new list of cells;
    end;
    end;
```

A.2 Algorithm for creation of a surface mesh on wall boundaries, based on the indirect addressing structure of the SPEED code:

```
Scan all boundary nodes that correspond to the wall patches of interest;
For each boundary node;
    determine the adjacent internal cell;
    create connectivity list for a quadrilateral element by examining the
    connectivity  list and boundary direction of the internal cell;
    for  each vertex on the connectivity list of quad element;
        interpolate value of scalar from the adjacent internal cell centroid;
    end;
end;
Save boundary quads data;
Save field data on boundary vertices.
```

A.3 A point-probing algorithm, determining the cell in which probe P lies, that is applicable on unstructured data with hexahedral cells. It is very fast and works in cases of highly distorted meshes, where the nearest vertex to point P may not belong to the cell point P lies within.

```
Find distances of all vertices from probing point P;

Sort all vertices according to their distance from point P, by using the 'bubble-
sort' algorithm and place them in a list vdist, the vertex with the smallest
distance from point P being at the top of the list;

for every vertex in vdist;
    set clist to be the list of cells connected to vertex;
    for every cell in clist;
        for every face in the cell;
            if P is not in the positive half-space of the face then;
                P is not in cell;
            endif;
        end;
    end;
end;
```

A.4 An algorithm for the creation of indirect-addressing information for each cell. It is applicable to any unstructured mesh with cells and faces of any shape, as long as there is only one neighbour to a cell at a face. Its implementation in the geometry model assembly module, so that it is applicable to hexahedral cells with quadrilateral faces, is done with the following settings:

```
number of vertices at cell   = 8
number of faces at cell      = 6
number of vertices at face   = 4
```

The algorithm is realised in two steps:

1. creation of a vertex-connectivity list (array NCV) for each vertex IV, storing all the cells that are adjacent to a vertex. This requires the cell-connectivity list (stored in array ICORN) for each cell IC.

```
do IC =1 to number of cells;
    do IV = 1 to number of vertices at cell;
        get vertex label from cell connectivity list (ICORN);
        store cell IC in list of cells adjacent to IV (NCV);
    enddo
enddo
```

2. Calculation, for each cell IC, of the neighbouring cell at each face, stored in array NEIGHB (see section 5.2.2 for definition) and the direction of the neighbouring cell, stored in array LDTHRF.

```
do IC = 1 to number of cells;
   do IF = 1 to number of faces at cell;
      do IV = 1 to number of vertices at face;
         get list of cells connected to vertex (NCV);
      enddo
      set the neighbour of cell IC at face IF to be the cell (unique) that
      appears on all lists of cells adjacent to the face; store in array
      NEIGHB;

      set the direction of neighour cell to be the direction as one looks
      from cell IC to neighbour; store in array LDTHRF;
   enddo;
enddo;
```

A.5 The outline of a program, called GEOMETRY, for definition of a solid model for the computational domain of a DI Diesel engine, and decomposition thereof into hyperpatches. In its present form it caters for two valves, valve chamfers and recesses on the piston, and bowl. The program is written in the PATRAN command language (PCL).

```
Program GEOMETRY;
   Read geometry dimension from input file;
   Define inlet-valve coordinate system;
   Define exhaust-valve coordinate system;
   Define bowl coordinate system;
   Define cylinder-liner coordinate system;
   Call function PISTON HEAD to define grids, lines, and patches on the piston
   head;
   Call function CYLINDER HEAD to define grids, lines, and patches on the
   cylinder head;
   Call function UPPER BOWL PLANE to define grids, lines, and patches on the
   upper bowl plane;
   Call function BOWL BULGE to define grids, lines, and patches on the bowl
   bulge;
   Call function LOWER BOWL SURFACE to define grids, lines, and patches on the
   lower bowl surface;
   Call function BOWL FLOOR to define grids, lines, and patches on the bowl
   floor;
   Call function MAKE LINERS to define grids, lines, and patches on the liners;
   Call function MAKE BLOCKS to create hyperpatches;
End of GEOMETRY.
```

Engine dimensions, as well as the z-coordinates of various planes are defined in a separate file. These are:

Geometry dimensions
cylinder bore
clearance
radius of bowl lip
radius of bowl bulge
radius of low bowl edge
radius of inlet valve recess on piston
radius of exhaust valve recess on piston
z-coord. of cylinder head plane
z-coord.of inlet valve head plane
z-coord. of exhaust valve head plane
z-coord. of inlet valve lip
z-coord. of exhaust valve lip
z-coord. of piston head plane
z-coord. of inlet valve recess plane
z-coord. of exhaust valve recess plane
z-coord. of upper bowl plane
z-coord. of bowl bulge
z-coord. of bowl floor plane

A.6 The outline of a program, called MESH for generation of a SPEED mesh in each hyperpatch that has been generated with program GEOMETRY. It is written in the Patran Command Language (PCL) and has the following structure:

```
Function MESH(enumber);

    Call CELLS(enumber) to generate a mesh of hexahedral cells in the
    computational domain;

    Call BOUNDARY_QUADS(enumber) to generate a mesh of quadrilateral   elements
    on the physical or reentrant boundaries;

    Call GHOST_QUADS(enumber) to generate a mesh of quadrilateral cells on
    the ghost boundary defined at the bowl throat;

End of MESH
```

'MESH' generates an isoparametric mesh in each hyperpatch of the solid model. To do so it calls command mesh of PATRAN, by specifying five agruments: The number of cells in each of the four edges of the base of the hyperpatch (four arguments), and the number of cell-layers along the height of the hyperpatch. These numbers are calculated from the input argument

enumber. The number of layers in each of the following four regions of the mesh, is:

Region	**number of layers**
Squish region at TDC	nearest integer of $(\frac{\texttt{enumber}}{6})$ + 1
Upper bowl region	enumber
Middle bowl region	3 * enumber
Lower bowl region is	nearest integer of $(\frac{\texttt{enumber}}{6})$ + 1

A.7 The algorithm implemented in SPIL for creation of object 'spray' is:

i. Scan the SPEED data set for all timesteps and calculate the maximum number of droplets per timestep *maxdrop*, and the minimum and maximum droplet temperatures and diameters *mint*, *maxt*, *mind* and *maxd* respectively .

ii. Create an object *'spray'* containing *maxdrop* droplet atoms with diameter= 0.0 m, and fully transparent.

iii. Setup the droplet visualisation colour scale from blue to red, corresponding to *mint* and *maxt* respectively.

iv. Return to beginning of SPEED data set.

```
for each timestep
begin
    read droplet data.
    for each droplet
    begin
    Create atom based on droplet primitive from primitives    library.
    Assign to atom colour value from the droplet   visualisation colour scale.
    Assign to atom visualisation diameter as a function of the droplet diameter.
    end
    Create object 'spray' referencing all droplet atoms
end
```

A.8 The algorithm implemented in SPIL for calculation of object 'flow' containing velocity vectors on a cutting plane through the data:

```
Scan the speed data set for all timesteps and calculate the minimum and maximum
velocity vector magnitudes minv and maxv respectively.
for each timestep
begin
    Read geometry and flow velocity data for each cell.
    Calculate object 'cutting plane' comprising a set of polygons defined as the
    intersection of the cutting plane with the mesh.
    Interpolate vector values from intersected cell vertices to vertices on the
    cutting plane.
    for each vertex in object 'cutting plane'.
    begin
    Create atom based on vector primitive from primitives     library.
    Assign to atom a visualisation length that is a function of the velocity
    vector magnitude.
    end
end
```

A.9 An algorithm for calculation of particle traces. It calculates the trajectory of a particle and stores it in array *trajectory* .

```
Set parameter nsubsub  for the calculation of sub-cycle timestep;
Get from user initial location of particle p ;
Find cell id ic  of cell in which the particle lies within;
Calculate the initial particle velocity u_p ;
for i =1 to number of timesteps do
begin
Get geometry data from the data base;
Get flow field data from the data base;
Calculate the time t, in crank angles, corresponding to timestep i ;
Calculate the time end_t, in cranke angle, corresponding to timestep i+1 ;
begin
j = 0
repeat
   j = j + 1 ;
   Calculate residence cell ic  ;
   Calculate residence length r_length  ;
   Calculate the volume cell_volume  of cell ic ;
   r_time  = cell_volume  ** (1/3) ;
   Δt = \F(r_time,nsubsub) ;
   particle_in_cell   = .true. ;
   k = 0 ;
   while particle_in_cell  do begin
      k  = k  + 1 ;
      t  = t   + Δt ;
      Interpolate the gas velocity u_p  at location p ,      in cell ic  ;
      Particle displacement   l  =  u_p · Δt ;
      New particle position   p  =  p  + l ;
      itrack  =  itrack + 1 ;
      trajectory  (itrack,  1 ) = p  ( 1 ) ;
      trajectory  (itrack,  2 ) = p  ( 2 ) ;
      trajectory  (itrack,  3 ) = p  ( 3 ) ;
      Check whether particle is in ic. If yes particle_in_cell  =..true.;
   end
   until ( t  < end_t  )
   end
end
```

Appendix B

Problem file for the conversion of a PATRAN mesh to a SPEED one, specification of engine operating conditions, definition of initial conditions and setting of boundary regions and the conditions on them. To keep this thesis in a sensible size, only selected valve lift, discharge coefficients, and valve profiles are included. Comment lines start with an exclamation mark.

```
! Read in geometry data

RUN,CONVERT,PATRAN


! Run type specification

RUNTYPE,NEW,DIESEL


! Starting time

STIME, 000.0


! Convergence requirements for coldflow

PISO, 4, 50, 1.0E-04
VPARAM, 100, 1.0E-5, 1.0E-5


! Injection and Fuel Control commands.
!-------------------------------------

! No of holes, no of parcels, start-end of injection, fuel temp

INJECTION, 5, 8, 352.4, 359.9, 373

! direction of injection and diameter of each hole

INJPARAM, 1, -6.379, -1.9,   -2.351, 0.21E-3, -14.01,  180
INJPARAM, 2, -7.396, -0.458, -2.543, 0.21E-3, -14.636, 111
INJPARAM, 3, -0.039, -0.901, -2.983, 0.21E-3, -15.78,  38
INJPARAM, 4, -9.039, -2.899, -2.983, 0.21E-3, -15.78,  322
INJPARAM, 5, -7.396, -3.342, -2.543, 0.21E-3, -14.636, 249

! fuel type
```

```
FUEL, N-HEPTANE

! Use Huh's atomisation model, with L/D=4.0, inj. disch. coeff.=0.7

ATOMISATION 4.0 0.7 /

! Stroke, connecting rod length and engine speed

ENGINE, 94.6E-3, 144.0E-3, 2000

! bulk initial conditions (gas velocity, pressure, temperature, k, e

BULK, 0.0, 0.0, 0.0, 1.0e5, 300.0, 3.696, 370.0

! definition of boundaries
!-------------------------

! cylinder head

REGION,11,WALL
1
1
350.0
1

! liner

REGION,12,WALL
1
1
360.0
3

! piston head

REGION,13,WALL
2
1
390.0
2

! piston recess

REGION,14,WALL
2
1
400.0
5

! inlet valve head
```

```
REGION,15,WALL
3
1
340.0
4
21

! exhaust valve head

REGION,16,WALL
4
1
390.0
4
22

! ghost boundary

REGION,61,GHOST
1


! specification of valve lifts
! ----------------------------

! Valve lift table for intake valve

vlift, 21,  0.0,0.00000
vlift, 21, 10.0,0.00109
vlift, 21, 20.0,0.00234
vlift, 21, 30.0,0.00367
vlift, 21, 40.0,0.00489
vlift, 21, 50.0,0.00596
vlift, 21, 60.0,0.00685
vlift, 21, 70.0,0.00757
vlift, 21, 80.0,0.00811
vlift, 21, 90.0,0.00848
vlift, 21,100.0,0.00867
vlift, 21,110.0,0.00869
vlift, 21,120.0,0.00853
vlift, 21,130.0,0.00820
vlift, 21,140.0,0.00769
vlift, 21,150.0,0.00701
vlift, 21,160.0,0.00615
vlift, 21,170.0,0.00512
vlift, 21,180.0,0.00392
vlift, 21,190.0,0.00261
vlift, 21,200.0,0.00133
vlift, 21,210.0,0.00042
vlift, 21,220.0,0.00013
vlift, 21,230.0,0.00007
vlift, 21,240.0,0.00001
vlift, 21,250.0,0.00000
```

```
vlift, 21,700.0,0.00001
vlift, 21,710.0,0.00007
vlift, 21,720.0,0.00028


! Discharge coefficient for intake valve

vdcoeff, 21, 0.000, 0.592
vdcoeff, 21, 0.002, 0.592
vdcoeff, 21, 0.004, 0.682
vdcoeff, 21, 0.006, 0.609
vdcoeff, 21, 0.008, 0.510
vdcoeff, 21, 0.0087,0.455


! Valve lift table for exhaust valve

vlift, 22,   0.0,0.00000
vlift, 22, 469.0,0.00001
vlift, 22, 479.0,0.00006
vlift, 22, 489.0,0.00023
vlift, 22, 499.0,0.00090
vlift, 22, 509.0,0.00203
vlift, 22, 519.0,0.00331
vlift, 22, 529.0,0.00453
vlift, 22, 539.0,0.00563
vlift, 22, 549.0,0.00657
vlift, 22, 559.0,0.00736
vlift, 22, 569.0,0.00800
vlift, 22, 579.0,0.00848
vlift, 22, 589.0,0.00880
vlift, 22, 599.0,0.00897
vlift, 22, 609.0,0.00899
vlift, 22, 619.0,0.00885
vlift, 22, 629.0,0.00855
vlift, 22, 639.0,0.00810
vlift, 22, 649.0,0.00750
vlift, 22, 659.0,0.00674
vlift, 22, 669.0,0.00583
vlift, 22, 679.0,0.00476
vlift, 22, 689.0,0.00356
vlift, 22, 699.0,0.00229
vlift, 22, 709.0,0.00111
vlift, 22, 719.0,0.00035
vlift, 22, 729.0,0.00012
vlift, 22, 739.0,0.00006
vlift, 22, 749.0,0.00001
vlift, 22, 759.0,0.00000


! Discharge coefficient for exhaust valve

vdcoeff, 22, 0.000, 0.6
vdcoeff, 22, 0.009, 0.6
```

```
! valve boundary conditions
! ----------------------

! inlet manifold conditions

MANIFOLD, 21, 000.0,  1.0E5, 300.0, 1.2, 164, 0.0
MANIFOLD, 21, 050.0,  1.0E5, 300.0, 1.2, 164, 0.0
MANIFOLD, 21, 162.0,  1.0E5, 300.0, 1.2, 164, 0.0
MANIFOLD, 21, 244.0,  1.0E5, 300.0, 1.2, 164, 0.0

! exhaust manifold conditions

MANIFOLD, 22, 900, DELETE
MANIFOLD, 22, 000.0,  1.0E5, 300.0, 1.2, 164, 0.0
MANIFOLD, 22, 1080.0, 1.0E5, 300.0, 1.2, 164, 0.0

! inlet valve profiles
! definition of 10 points axial direction and six points in the
! radial direction for valve curtain 1 (region 21)

POINTS, 21, 10, 6, 0, 0.0, 45.0, 90.0, 180.0, 270.0, 315.0

! Definition of alpha and beta angles and radial velocity profiles
! at 0, 50, 162, and 244 degs CA ATDC.

! 0 degs CA ATDC

ALPHA, 21, 000.0, 1, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 2, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 3, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 4, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 5, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 6, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 7, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 8, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 9, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 000.0, 10,0, 45.0, ,45.0, 45.0, 45.0, 45.0, 45.0
BETA, 21, 000.0, 1, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 2, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 3, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 4, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 5, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 6, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 7, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 8, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 9, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 000.0, 10,0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
RADVEL, 21, 000.0, 1, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
RADVEL, 21, 000.0, 2, 0, 00.1, 00.1, 00.1, 00.1, 00.1, 00.1
RADVEL, 21, 000.0, 3, 0, 00.1, 00.1, 00.2, 00.2, 00.2, 00.1
RADVEL, 21, 000.0, 4, 0, 00.1, 00.1, 00.3, 00.3, 00.3, 00.1
RADVEL, 21, 000.0, 5, 0, 00.1, 00.1, 00.5, 00.6, 00.5, 00.3
```

```
RADVEL, 21, 000.0, 6, 0, 00.3, 00.5, 00.9, 01.0, 00.9, 00.5
RADVEL, 21, 000.0, 7, 0, 00.2, 00.3, 00.5, 00.6, 00.5, 00.3
RADVEL, 21, 000.0, 8, 0, 00.2, 00.2, 00.3, 00.3, 00.3, 00.2
RADVEL, 21, 000.0, 9, 0, 00.1, 00.1, 00.2, 00.2, 00.2, 00.1
RADVEL, 21, 000.0, 10,0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0

! 50 degs CA ATDC

ALPHA, 21,  50.00,  1, 0,   45.0,   45.0,   45.0,   45.0,   45.0,   45.0
ALPHA, 21,  50.00,  2, 0,   40.25,  52.37,  53.92,  70.11,  43.87,  60.24
ALPHA, 21,  50.00,  3, 0,   40.36,  52.39,  53.88,  65.76,  44.08,  59.47
ALPHA, 21,  50.00,  4, 0,   36.04,  54.99,  51.01,  53.50,  42.07,  57.40
ALPHA, 21,  50.00,  5, 0,   36.04,  54.99,  51.01,  53.50,  42.07,  57.40
ALPHA, 21,  50.00,  6, 0,   32.76,  58.61,  57.67,  48.06,  41.76,  52.42
ALPHA, 21,  50.00,  7, 0,   32.76,  58.61,  57.67,  48.06,  41.76,  52.42
ALPHA, 21,  50.00,  8, 0,   27.81,  56.51,  59.74,  44.78,  41.50,  43.27
ALPHA, 21,  50.00,  9, 0,   27.81,  56.51,  59.74,  44.78,  41.50,  43.27
ALPHA, 21,  50.00,  10,0,  103.11,  24.03,  84.17,  31.70,  45.76,  32.28
BETA, 21,  50.00,  1, 0,    0.00,   0.00,   0.00,   0.00,   0.00,   0.00
BETA, 21,  50.00,  2, 0,  -30.86, -19.95, -16.09, -13.60, -27.97, -11.78
BETA, 21,  50.00,  3, 0,  -29.97, -18.84, -16.62, -17.76, -29.08, -10.15
BETA, 21,  50.00,  4, 0,  -32.66, -18.76, -18.38, -25.27, -29.15, -11.90
BETA, 21,  50.00,  5, 0,  -32.66, -18.76, -18.38, -25.27, -29.15, -11.90
BETA, 21,  50.00,  6, 0,  -32.54, -18.32, -20.64, -21.13, -29.03, -15.70
BETA, 21,  50.00,  7, 0,  -32.54, -18.32, -20.64, -21.13, -29.03, -15.70
BETA, 21,  50.00,  8, 0,  -32.27, -17.78, -25.94, -19.25, -25.79, -18.20
BETA, 21,  50.00,  9, 0,  -32.27, -17.78, -25.94, -19.25, -25.79, -18.20
BETA, 21,  50.00,  10,0,  -25.07, -41.09, -60.04, -27.58, -18.70, -21.57
RADVEL, 21,  50.00,  1, 0,    0.00,   0.00,   0.00,   0.00,   0.00,   0.00
RADVEL, 21,  50.00,  2, 0,    0.40,   0.71,   0.78,   0.69,   0.56,   0.32
RADVEL, 21,  50.00,  3, 0,    0.52,   0.90,   1.00,   0.87,   0.71,   0.47
RADVEL, 21,  50.00,  4, 0,    0.49,   0.93,   0.95,   0.84,   0.73,   0.59
RADVEL, 21,  50.00,  5, 0,    0.49,   0.93,   0.95,   0.84,   0.73,   0.59
RADVEL, 21,  50.00,  6, 0,    0.48,   0.94,   0.89,   0.79,   0.72,   0.65
RADVEL, 21,  50.00,  7, 0,    0.48,   0.94,   0.89,   0.79,   0.72,   0.65
RADVEL, 21,  50.00,  8, 0,    0.45,   0.81,   0.50,   0.73,   0.71,   0.67
RADVEL, 21,  50.00,  9, 0,    0.45,   0.81,   0.50,   0.73,   0.71,   0.67
RADVEL, 21,  50.00,  10,0,    0.37,   0.13,   0.07,   0.41,   0.70,   0.51

! 162 degs CA ATDC

ALPHA, 21, 162.00,  1, 0,   45.0,   45.0,   45.0,   45.0,   45.0,   45.0
ALPHA, 21, 162.00,  2, 0,   40.25,  52.37,  53.92,  70.11,  43.87,  60.24
ALPHA, 21, 162.00,  3, 0,   40.36,  52.39,  53.88,  65.76,  44.08,  59.47
ALPHA, 21, 162.00,  4, 0,   36.04,  54.99,  51.01,  53.50,  42.07,  57.40
ALPHA, 21, 162.00,  5, 0,   36.04,  54.99,  51.01,  53.50,  42.07,  57.40
ALPHA, 21, 162.00,  6, 0,   32.76,  58.61,  57.67,  48.06,  41.76,  52.42
ALPHA, 21, 162.00,  7, 0,   32.76,  58.61,  57.67,  48.06,  41.76,  52.42
ALPHA, 21, 162.00,  8, 0,   27.81,  56.51,  59.74,  44.78,  41.50,  43.27
ALPHA, 21, 162.00,  9, 0,   27.81,  56.51,  59.74,  44.78,  41.50,  43.27
ALPHA, 21, 162.00,  10,0,  103.11,  24:03,  84.17,  31.70,  45.76,  32.28
BETA, 21, 162.00,  1, 0,    0.00,   0.00,   0.00,   0.00,   0.00,   0.00
BETA, 21, 162.00,  2, 0,  -30.86, -19.95, -16.09, -13.60, -27.97, -11.78
BETA, 21, 162.00,  3, 0,  -29.97, -18.84, -16.62, -17.76, -29.08, -10.15
```

```
BETA, 21, 162.00,  4, 0,  -32.66, -18.76, -18.38, -25.27, -29.15, -11.90
BETA, 21, 162.00,  5, 0,  -32.66, -18.76, -18.38, -25.27, -29.15, -11.90
BETA, 21, 162.00,  6, 0,  -32.54, -18.32, -20.64, -21.13, -29.03, -15.70
BETA, 21, 162.00,  7, 0,  -32.54, -18.32, -20.64, -21.13, -29.03, -15.70
BETA, 21, 162.00,  8, 0,  -32.27, -17.78, -25.94, -19.25, -25.79, -18.20
BETA, 21, 162.00,  9, 0,  -32.27, -17.78, -25.94, -19.25, -25.79, -18.20
BETA, 21, 162.00,  10,0,  -25.07, -41.09, -60.04, -27.58, -18.70, -21.57
RADVEL, 21, 162.00,  1, 0,    0.00,   0.00,   0.00,   0.00,   0.00,   0.00
RADVEL, 21, 162.00,  2, 0,    0.40,   0.71,   0.78,   0.69,   0.56,   0.32
RADVEL, 21, 162.00,  3, 0,    0.52,   0.90,   1.00,   0.87,   0.71,   0.47
RADVEL, 21, 162.00,  4, 0,    0.49,   0.93,   0.95,   0.84,   0.73,   0.59
RADVEL, 21, 162.00,  5, 0,    0.49,   0.93,   0.95,   0.84,   0.73,   0.59
RADVEL, 21, 162.00,  6, 0,    0.48,   0.94,   0.89,   0.79,   0.72,   0.65
RADVEL, 21, 162.00,  7, 0,    0.48,   0.94,   0.89,   0.79,   0.72,   0.65
RADVEL, 21, 162.00,  8, 0,    0.45,   0.81,   0.50,   0.73,   0.71,   0.67
RADVEL, 21, 162.00,  9, 0,    0.45,   0.81,   0.50,   0.73,   0.71,   0.67
RADVEL, 21, 162.00,  10,0,    0.37,   0.13,   0.07,   0.41,   0.70,   0.51

! 244 degs CA ATDC

ALPHA, 21, 244.0, 1, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 2, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 3, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 4, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 5, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 6, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 7, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 8, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 9, 0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
ALPHA, 21, 244.0, 10,0, 45.0, 45.0, 45.0, 45.0, 45.0, 45.0
BETA, 21, 244.0, 1, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 2, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 3, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 4, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 5, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 6, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 7, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 8, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 9, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
BETA, 21, 244.0, 10,0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
RADVEL, 21, 244.0, 1, 0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
RADVEL, 21, 244.0, 2, 0, 00.1, 00.1, 00.1, 00.1, 00.1, 00.1
RADVEL, 21, 244.0, 3, 0, 00.1, 00.1, 00.2, 00.2, 00.2, 00.1
RADVEL, 21, 244.0, 4, 0, 00.1, 00.1, 00.3, 00.3, 00.3, 00.1
RADVEL, 21, 244.0, 5, 0, 00.1, 00.1, 00.5, 00.6, 00.5, 00.3
RADVEL, 21, 244.0, 6, 0, 00.3, 00.5, 00.9, 01.0, 00.9, 00.5
RADVEL, 21, 244.0, 7, 0, 00.2, 00.3, 00.5, 00.6, 00.5, 00.3
RADVEL, 21, 244.0, 8, 0, 00.2, 00.2, 00.3, 00.3, 00.3, 00.2
RADVEL, 21, 244.0, 9, 0, 00.1, 00.1, 00.2, 00.2, 00.2, 00.1
RADVEL, 21, 244.0, 10,0, 00.0, 00.0, 00.0, 00.0, 00.0, 00.0
```

Appendix C

C.1 Object 'chamber' for the animation of the Renault IDI Diesel engine. It icludes definition of the piston, the cover, the injector, the glow plug, and four light sources. Comments are enclosed in curly brackets. Atoms taken from the AERAS library of primitives start with the symbol #.

```
object  chamber
    piston x -90
    cover_front    x -90
    cover_back     x -90
    injector
    glow_plug
    lite1
    lite2
    lite3
    lite4
end

object  injector
    #inj_tip   y 90
    #inj_base
end

object  glow_plug
    #glow_tip  y 90
    #glow_base
end

{Define light source in the squish region, pointing to the axis. Parallel.Uniform
profile.}

light  lite1
    m(0, 0, 20)
    aim (0, 0, 20, 0, 0, 0)
    c(0.66, 0.9, 0.2)
    spread (0, 1, 1, 0)
end

{ Define a light source at squish region, illuminating the inner-top part of the
cover }
light  lite2
    m(0, 2, 0)
```

```
    aim (0, 2, 0, 0, 5, 0)
    c(0.66, 0.2, 0.2)
    spread (180, 0.1, 1, 1)
    range (0, 0, 15)
end

{ Define light source at top-right-front of model, lighting the prechamber and
partly the squish region and piston crown. }

light  lite3
    m(50, 120, 100)
    aim (50, 120, 100, 20, 120, 0)
    c(0.66, 1, 0)
    spread (180, 0.3, 1, 0.2)
    range (0, 0,150)
end

{ Define light source at top-left-rear of model, illuminating the top and side
regions of the cover.

light  lite4
    m(-100, 100, -20)
    aim (-100, 100, -20, 50, 150, 0)
    c(0.66, 1, 0)
    spread (180, 0, 1, 0.1)
    range (0.5, 30,100)
end
```

C.2 Animation script for scene 2, as described in section 7.2. It defines the motion of the camera and the elements of object 'chamber' in 700 animation frames. Comments are enclosed in curly brackets.

```
{ANIMATION SCRIPT FOR SCENE 2}

{Should be used with object cover}

{ camera cranes-down and tracks right from 226 to 375.
Camera zooms-in from 500 to 700 }

CAMERA 1      225    m (-500, 500, 500) (-500, 500, 500)
CAMERA 226    375    m (-500, 500, 500) (0, 100, 800)
CAMERA 376    500    m (0, 100, 800) (0, 100, 800)
CAMERA 501    700    m (0, 100, 800) (25, 200, 1400) * inout (0.2, 0.8)
```

```
{Centre camera move from 500 to 700}

CAMERA 1        500         center (10, 50, 0) (10, 50, 0)
CAMERA 501      700         center (10, 50, 0) (25, 100, 0) * inout (0, 1)

{Zoom-in from 501 to 700}

CAMERA 1        500         mag 2.5
CAMERA 501      700         mag 2.5  15 * inout (1, 0)


{fade in engine cover}

cover_front     1   50      t (1, 1, 0) (0, 0, 1)
cover_back      1   50      t (1, 1, 0) (0, 0, 1)

{fade-out front part of engine cover. Keep it transparent till end}

cover_front     51      100         t (0, 0, 1) (1, 1, 0)
cover_front     101     700         t (1, 1, 0)


{Cycle piston 4 times, stop at 350 degrees C.A.}

piston 101      175         m (0, 0, 0) (0, 0, -88) * cosine (0, 0)
piston 176      250         m (0, 0, -88) (0, 0, 0) * cosine (0, 0)

piston 251      325         m (0, 0, 0) (0, 0, -88) * cosine (0, 0)
piston 326      400         m (0, 0, -88) (0, 0, 0) * cosine (0, 0)

piston 401      475         m (0, 0, 0) (0, 0, -88) * cosine (0, 0)
piston 476      550         m (0, 0, -88) (0, 0, 0) * cosine (0, 0)

piston 551      625         m (0, 0, 0) (0, 0, -88) * cosine (0, 0)
piston 626      700         m (0, 0, -88) (0, 0, -1.034158E+00) * cosine (0, 0)
```

Appendix D

Session file of translation and checking of the VW 1.9l DI transparent Diesel engine mesh. User commands are typed after the prompt (*>).

```
*>run,con,pat
**********************************************************************
              PATSPEED version 2.3.5,     ??/??/1990  release

              Fluids Section,  Dept. of Mechanical Engineering
            IMPERIAL COLLEGE OF SCIENCE,TECHNOLOGY AND MEDICINE
**********************************************************************

 Please enter the name of the "PATRAN NEUTRAL FILE" to be read:
*>VW_coarse_mesh.ntr

 Did you use WEDGE elements around the poles?
 1:YES,      2:NO,      3:THERE ARE NO POLES
*>3

 Please type in the units of the current neutral file
 e.g. MEtres, CEntimetres,CM,, MMetres
 The default is milimetres.

  Neutral file in millimetres
  Error opening file VW_coarse_mesh.ntr
  Try again...

 Please enter the name of the "PATRAN NEUTRAL FILE" to be read:
*>VW_coarse.ntr

 Reading the neutral file titled: VW_coarse.ntr

 Reading data for    1349 elements
                      963 nodes
 Reading      963 nodes...
  Reading node:
                     500
 Reading     1349 elements...
  Reading element:
                     500
                     1000
  Neutral file in millimetres
  Converting neutral file in SI units...

 Creating mesh addressing data...
    Total number of cells is :                609
    Total number of wall patches is:      4
    Total number of valve boundaries is: 0
    Total number of flow sources is:      0
    Total number of symmetry planes:      0
    Total number of poles:                0

*>pri,size
 -------DESCR.--------------------SIZE-----LIMIT----
 No. of cells          (NC)      =    609     18000
```

```
 No. of vertices          (NVRTEX) =     963     20000
 No. of bound.nodes       (NB)     =     666      5000
 No. of wall patches      (NW)     =       4         9
        No. of nodes on wall  1=       239      2100
        No. of nodes on wall  2=        28      2100
        No. of nodes on wall  3=       165      2100
        No. of nodes on wall  4=       234      2100
 No. of valves            (NVALVE) =       0         1
 No. of poles             (NAXIS)  =       0         1
 No. of I/O sources       (NVELS)  =       0         1
 No. of bowlthr. nodes(NCSQBO) =       9      2100

*>check,addressing

*****The code is currently checking the values of addressing arrays*****

Checking the internal cells related addressing arrays...

Checking the values of wall related addressing arrays...
     Checking wall patch :  1
     Checking wall patch :  2
     Checking wall patch :  3
     Checking wall patch :  4

Checking the values of valve related addressing arrays...

Checking for duplicate vertices...

Yes!!!  Addressing in the interior is O.K.
**************************************************************

*>check,orthog,short

 TOLERANCE ANGLE IS : 45.00
 Do you wish to overwrite ? (y/[n])
*>y

 Please type in the new tolerance angle (in degreess)
*>30

 Checking for non-orthogonal cells...
 Cells with faces forming angles less than  30.00
                          or greater than 150.00
 are considered to be distorted cells.


 Checking    609 cells...
 Remaining cells :
                   109

  16 distorted cells found

 These cells are:
        27        48        89        90        93        94        95        96
        98        99       100       117       122       123       128       195
*>check,orthog,long

 TOLERANCE ANGLE IS : 45.00
 Do you wish to overwrite ? (y/[n])
*>y

 Please type in the new tolerance angle (in degreess)
*>20.

 Checking for non-orthogonal cells...
 Cells with faces forming angles less than  20.00
                                or greater than 160.00
 are considered to be distorted cells.


 Checking    609 cells...
 Remaining cells :
                   109

   2 distorted cells found
```

```
 These cells are:
   Cell     Angle1 Angle2 Angle3 Angle4 Angle5 Angle6 Angle7 Angle8
     98      63.84 107.92 164.78  23.46  90.00  86.24  99.56  90.00
    122      78.74 104.46  89.94  86.86 160.76  90.00  90.00  19.24

*>check,aspect,long

 TOLERANCE ASPECT RATIO IS : 20.00
 Do you wish to overwrite ? (y/[n])
*>n

    8 cells are found with an aspect ratio greater than: 20.00

 These cells (in order of descending aspect ratio) are:
 Aspect ratio                    Cell label
        24.11                             9
        22.00                            16
        21.55                            10
        20.57                             3
        20.55                             1
        20.43                             8
        20.23                             2
        20.02                            15
 Max. aspect ratio= 24.11   at cell:     9
 Min. aspect ratio=  1.04   at cell:   138

*>check,warp,long

 TOLERANCE FOR WARPAGE IS :5.0000
 Do you wish to overwrite ? (y/[n])
*>y

 Please type in the new tolerance for warpage (0. to 90.)
*>20.

 Checking for warped cells...
 Cells with faces having warpage greater than 0.0603
 are considered to be distorted cells with warped faces.

 Checking    609 cells...
 Remaining cells :
                   109

    6 distorted cells with warped faces found

 These cells are:
   Cell      Face-3   Face-2   Face-1    Face1   Face2   Face3
     90      0.0027   0.0000   0.0000   0.0000   0.0000  0.4142
     92      0.0000   0.0000   0.0000   0.0000   0.0000   0.3245
     93      0.0061   0.0000   0.0000   0.0000   0.0000   0.7107
     96      0.0413   0.0000   0.0000   0.0000   0.0000   0.6421
     97      0.0000   0.0000   0.0000   0.0000   0.0000   0.3165
     99      0.0015   0.0000   0.0000   0.0000   0.0000   0.4218

*>check,warp,long

 TOLERANCE FOR WARPAGE IS :5.0000
 Do you wish to overwrite ? (y/[n])
*>y

 Please type in the new tolerance for warpage (0. to 90.)
*>45.

 Checking for warped cells...
 Cells with faces having warpage greater than 0.2929
 are considered to be distorted cells with warped faces.


 Checking    609 cells...
 Remaining cells :
                   109

    6 distorted cells with warped faces found
                                    339
 These cells are:
```

Cell	Face-3	Face-2	Face-1	Face1	Face2	Face3
90	0.0027	0.0000	0.0000	0.0000	0.0000	0.4142
92	0.0000	0.0000	0.0000	0.0000	0.0000	0.3245
93	0.0061	0.0000	0.0000	0.0000	0.0000	0.7107
96	0.0413	0.0000	0.0000	0.0000	0.0000	0.6421
97	0.0000	0.0000	0.0000	0.0000	0.0000	0.3165
99	0.0015	0.0000	0.0000	0.0000	0.0000	0.4218

```
*>quit
```

Blank

In

Original

Bibliography

[1] H.S. Anderson, J.A. Berton, J.R.Peter, G. Carswell, D.S.Dyer, J.T. Faust, J.L. Kempf, R.E. Marshal,
"The animation production environment: A basis for visualization and animation of scientific data", Ohio Supercomputer Graphics Project Technical Report, March 1989.

[2] R.B. Haber,
"Visualization techniques for Engineering Mechanics", *Computing Systems in Engineering* Volume 1. No. 1. pp. 37-50, 1990.

[3] R.Marshall, J.Kempf, S.Dyer, C.Yen,
"Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie", *Computer. Graphics*, Volume 24, No..2, March 1990, pp. 89-97.

[4] RB Haber, DA McNabb,
"Visualization Idioms: a Conceptual Model for Scientific Visualization Systems", *Visualisation in Scientific Computing*, G.M.Nielson and B.Shriver, and L.J.Rosenblum, Eds, IEEE Computer Society Press, 1990, pp. 174-189.

[5] G.V. Bancroft, F.J. Merritt, T.C. Plessel, P.G. Kelaita, R.K. McCabe, and A. Globus "FAST: A Multi-Precossed Environment for Visualization", *Proceedings of the first IEEE Conference on Visualization*, San Francisco, California, October 23-26,1990.

[6] "Patran User's Guide", PDA engineering, 1987.

[7] "SUPERTAB User's Guide", Structural Dynamics Research Corp., 1986.

[8] M.L.C. Sluiter, D.L. Hansen, Structural Dynamics Research Corporation,
"A general purpose automatic mesh generator for shell and solid finite elements", *Computers in Engineering*, Volume 3, 1982.

[9] A.D. Gosman,
"Multidimensional modelling of cold flows and turbulence in reciprocating engines", SAE Paper 850344, 1985.

[10] A.D. Gosman,
"Computer modelling of flow and head transfer in engines -Progress and prospects", *Proceedings of International Conference on Diagnostics and Modelling of Combustion in Reciprocating Engines*, Tokyo 1985.

[11] R.Marshall, J.Kempf, S.Dyer, C.Yen,
"Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie", *Computer Graphics*, Volume 24, No. 2, March 1990, pp. 89-97.

[12] McCormick, B.H., DeFanti, T.A., and Brown, M.D. (editors),
"Visualisation in Scientific Computing", *Computer Graphics*, Volume 21, No. 6, 1987.

[13] C. Marooney, Private communication, 1988.

[14] A. Kaufman, editor,
"Volume Visualization", *IEEE Computer Society Press Tutorial*, 1991.

[15] D. Fitzgeorge and J.L. Allison,
"Air swirl in a road vehicle Diesel engine", *Proceedings I. Mech. E.*, No. 4, pp. 151-177 (1962-63).

[16] R.B. Bird, W.E. Steward, and W.E. Lightfoot,
"Transport Phenomena", *Wiley & Sons*, New York, 1960.

[17] S. V. Patankar
"Numerical heat transfer and fluid flow", *University Press*, NY, 1979.

[18] P. Bradshaw (ed.),
"Turbulence", *Topics in applied physics*, Volume 12, Springer, Berlin, 1978.

[19] D.R. Lancaster,
"Effects of engine variables on turbulence in a spark-ignition engine", S.A.E. Paper No. 760159, 1976.

[20] A. Favre,
"Statistical equations of turbulent gases", *Problems of hydrodynamics and continum mechanics*, Society of Industrial and Applied Mathematics, 1969.

[21] A.D. Gosman,
"Flow processes in cylinders", in *The Thermodynamics and Gas Dynamics of Internal Combustion Engines, Vol. II* (Eds. J.H. Horlock and D.E. Winterbone), Clarendon Press, Oxford (1986).

[22] W.C. Reynolds,
"Modelling of fluid motions in engines - an introductory overview", in *Combustion modelling in reciprocating engines* (eds. J. Mattavi & C. Amann), Plenum Press, New York, 1980.

[23] B. Ahmadi-Befrui,
"Analysis of flow evolution in the cylinders of motored reciprocating engines", Thesis submitted for the degree of Doctor of Philosophy of the Univ. of London, Imperial College, May 1985.

[24] T. Theodoropoulos,
"Prediction of Three-Dimensional engine flow on Unstructured Meshes", Thesis submitted for the degree of Doctor of Philosophy of the Univ. of London, Imperial College, June 1990.

[25] A.D. Gosman,
"Aspects of the simulation of combustion in reciprocating engines", from *Proceedings of Symposium on Numerical simulation of combustion phenomena*, France, 1985, Lecture Notes in Physics, Volume 241 (Springer-Verlag, Berlin).

[26] H.G. Weller,
"The development of a new flame area combustion model using conditional averaging", Technical report, Imperial College, mechanical engineering depatrment., thermofluids section, TF9307, 1993.

[27] R. Borghi,
"Turbulent combustion modelling", *Prog. Energy Combustion Science*, Volume 14, pp. 245-292, 1988.

[28] D.B. Spalding,
"Mixing and Chemical Reaction in Steady Confined Turbulent Flames", in the *Thirteenth Symposium (International) on Combustion*, pp. 649-657, The Combustion Institute, 1970.

[29] B. F. Magnussen, and B.H. Hjertager,
"On Mathematical Modelling of Turbulent Combustion with Special Emphasis on Soot Formation and Combustion", in *Sixteenth Symposium (International) on Combustion*, pp. 719-729. The Combustion Institute.

[30] L. Delamare, R. Borghi, and M. Gonzalez,
"The modelling and calculation of a turbulent premixed flame propagation in a closed vessel: comparison of three models with experiments", Technical Report 910265, SAE, 1991.

[31] R. Borghi, B. Argueyrolles, S. Gauffie, and P. Souhaite,
"Application of a Presumed p.d.f. Model of Turbulent Combustion to Reciprocating Engines", in *Twenty-first Symposium (International) on Combustion*, pp. 1591-1599. The Combustion Institute.

[32] N. Peters,
"Laminar Flamelet Concepts in Turbulent Combustion", in *Twenty-first Symposium (International) on Combustion*, pp. 1231-1250. The Combustion Institute.

[33] K.N.C. Bray,
"Methods of Including Realistic Chemical Reaction Mechanisms in Turbulent Combustion Models", in Warnatz J., editor, *Complex Chemical Reaction Systems*. Springer-Verlag, 1987.

[34] R. Borghi and D. Dutoya,
"On the Scales of the Fluctuations in Turbulent Combustion", in *Seventeenth Symposium (International) on Combustion*, pp. 235-244, The Combustion Institute.

[35] S.B. Pope,
"Computations of Turbulent Combustion: Progress and Challenges", in *23rd Symposium (International) on Combustion*, pp. 591-612. The Combustion Institute, 1990.

[36] S.B. Pope,
"The Evolution of Surfaces in Turbulence", in International Journal of Engineering and Science, Vol. 26, 1988, pp. 445-469.

[37] S.M. Candel and T.J. Poinsot,
"Flame Stretch and the Balance Equation for Flame Area", *Combustion Science and Technology*, 70:1-15, 1990.

[38] H.G. Weller, C.J. Marooney, and A.D. Gosman,
"A new spectral method for calculation of the time-varying area of a laminar flame in homogeneous turbulence", in *Twenty-third Symposium (International) on Combustion*, pp. 629-636, The Combustion Institute, 1990.

[39] A.D. Gosman and A.P. Watkins,
"A computer prediction method for turbulent flow and heat transfer in piston/cylinder assemblies", *Proceedings of Symposium on Turbulent Shear Flows*, Pennsylvania State University, April 1977. Also Imperial College, Dept. of Mechanical Engineering Report No. FS/77/10 (1977).

[40] A.D. Gosman and R.J.R. Johns,
"Computer analysis of fuel-air mixing in direct injection engines", S.A.E. Paper No 800091, 1980.

[41] T.D. Butler, A.A. Amsden, P.J. O'Rourke, J.D. Ramshaw,
"KIVA: A comprehensive Model for 2D and 3D Engine Simulations", SAE 850554, 1985.

[42] A.D.Gosman, Tsui Y-Y, Watkins AP
" Calculation of Three-Dimensional Air Motion in Model Engines", SAE 840229, 1984.

[43] P. Begleris,
"Three-Dimensional flow predictions in motored diesel engines", Thesis submitted for the degree of Doctor of Philosophy, University of London, Imperial College, December 1986.

[44] B. S. Tabrizi,
"Three-dimensional Diesel spray predictions in combustion chamber flows", Thesis submitted for the degree of Doctor of Philosophy, University of London, Imperial College, June 1991.

[45] "The FIRE (Flow In Reciprocating Engines) code",
AVL, Graz, Austria, 1993.

[46] A.D. Gosman, C.J. Marooney,
"Development and Validation of Computer Models of In-cylinder Flow and Combustion in Diesel and Spark Ignition Engines", *Proc. Autotech 89* (IMechE C399/169), Birmingham, England, 1989.

[47] B.N. Adamson, A.D. Gosman, C.J. Marooney, B. Nasseri, T. Theodoropoulos,
"A new Unstructured Mesh Method for Flow Prediction in Internal Combustion Engines", *Proc. Comodia 1990*, Kyoto, Japan, 1990.

[48] BN Adamson, AD Gosman, C Hill, CJ Marooney, B Nasseri, S Sarantinos, T Theodoropoulos, HG Weller,
"Simulation of In-Cylinder Flow and Combustion in Reciprocating Engines", *Science and Engineering on Supercomputers*, EJ Pitcher, ed., *Computational Mechanics Publications & Springer Verlag*, October 1990.

[49] A.D. Gosman, C.Hill, C.J. Marooney, M. Sarantinos,
"Interactive computational fluid dynamics tools for engine design", *IMechE Proc. Int. Conf. on Computers in Engine Tech*, Cambridge 1991.

[50] K.L. Smit and T.Y. Su,
"A Geometry-Integrated Approach to Multiblock Grid Generation", *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, A.S. Arcilla, J. Hauser, P.R. Eiseman, J.F. Thompson (eds.), Elsevier Science Publishers B.V. (North-Holland), 1991.

[51] Mark Zellat,
Direction des Etudes Moteur Modelisation, Renault Automobiles, Private communication.

[52] F.D. Lange,
"Berechnung des Strömungsfeldes und der Wärmeübertragung im Zylinder eines geschleppten Motors", Rheinish-Westfälische technische Hochschule Aachen, Diplomarbeit, Mai 1993.

[53] R. Williams,
"Adaptive Parallel Meshes with Complex Geometry", Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, A.S. Arcilla, J. Hauser, P.R. Eiseman, J.F. Thompson (eds.), *Elsevier Science Publishers B.V. (North-Holland)*, 1991.

[54] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes,
"Computer Graphics: Principles and practice", *AddisonWesley*, 2nd edition, 1990.

[55] I.D.Faux, M.J.Pratt,
"Computational Geometry for Design and Manufacture", *Ellis Horwood Publishing Company* , 1987 reprint.

[56] K. Weiler,
"Edge-based data Structures for Solid Modelling in Curved-Surface Modelling environments" *IEEE Computer Graphics and App.*, Vol. 5, No.1, Jan. 1985, pp 21-40.

[57] A. Requicha, H. Voelker,
"Constructive Solid Geometry", Technical Memo 25, Production Automation Project, Univ. of Rochester, Rochester N.Y. 1977.

[58] M.S. Casale, E.L.Stanton,
"An overview of Analytic Solid Modelling". *IEEE Computer Graphics and Applications*, Feb. 1985.

[59] M.S.Casale,
"Free-Form Solid Modelling with Trimmed Surface Patches", *IEEE Computer Graphics and Applications*, Jan 1987.

[60] W. Tiller,
"Rational B-splines for Curve and Surface Reepresentation", *IEEE Computer Graphics and Applications*, 3(6), September 1983, pp. 61-66.

[61] E. Cohen,
"Some mathematical tools for a modeller's workbench", *IEEE Computer Graphics and Applications*, 3(7), October 1983, pp. 63-66.

[62] "ACIS Geometric Modeller: Technical Overview", Spatial Technology Inc., Boulder Colorado, 1991.

[63] P. Sinha,
"Geometric Computing with SHAPES", XOX Corp., Minneapolis, 1989.

[64] W.A. Cook and W.R. Oakes,
"A mapping method for generating three-dimensional meshes: past and present", *Computers in Engineering*, Vol. 3, 1982 pp. 9-16.

[65] W.A. Cook,
"Ingen: A general Purpose Mesh Generator for Finite Element Codes", Los Alamos Scientific Laboratory Report, LA 7135 MS, March 1978.

[66] A.D. Gosman, R.J.R. Johns, and A.P.Watkins,
"Development of prediction methods for in-cylinder processes in reciprocating engines", in *Combustion modelling in reciprocating engines.* (ed. J.N. Mattavi and C.A. Amann). *Plenum Press*, New York, 1980.

[67] J.F. Thompson, Z.U.A Warsi, C.W. Mastin,
"Numerical Grid Generation, Foundations and Applications", *North-Holland*, 1985.

[68] R.J.R.Johns,
"A unified method for calculating engine flows", *ASME* , 84-DGP-18, 1984.

[69] W.J. Gordon and L.C. Thiel,
"Transfinite mappings and their application to Grid Generation", in *Numerical Grid Generation*, Elsevier Science Publishing Company, pp. 171-192, 1982.

[70] R. Haber and J.J. Abel,
"Discrete Transfinite Mappings for the Description and meshing of Three-Dimensional Surfaces using Interactive Computer Graphics", *International Journal for Numerical Methods in Engineering*, 17 (7), 1015-1044, 1982.

[71] P.R. Eiseman,
"A Multi-Surface method of coordinate Generation", *Journal of Computational Physics*, Volume 33, No.1, October 1979.

[72] R.E. Smith and B.L. Weigel,
"Analytic and Approximate Boundary-Fitted Coordinate Systems for Fluid Flow Simulation", AIAA paper 80-0192, *AIAA 18th Aerospace Sciences Meeting*, Pasadena California, January 1980.

[73] C Dener, C Hirsh,
"IGG – an Interactive 3D Surface Modelling and Grid Generation System", Paper presented at the *Third International Conference on Numerical Grid Generation*, Barcelona, Spain, 3-7 June 1991.

[74] A.D. Gosman and R.J.R. Johns,
"A simple method for generating curvilinear-orthogonal grids for numerical fluid mechanics calculations", Imperial College, Technical Report FS/79/23, Dept. of Mechanical Engineering.

[75] J.F. Thompson, F.C. Thames, and C.W. Mastin,
"TOMCAT- A code for Numerical Generation of Boundary-Fitted Curvilinear Coordinate Systems on Fields Containing any Number of Arbitrary Two-Dimensional Bodies", *Journal of Computational Physics*, 24, 274, 1977.

[76] R.L. Sorenson,
"A Computer Program to Generate Two-Dimensional Grids About Airfoils and other shapes by the use of Poisson's Equation", NASA, TM-81198, 1980.

[77] Z.U.A. Warsi,
"Basic differential models for coordinate generation", in *Numerical grid generation*, North-Holland, Elsevier Science Publishing Co, 1982.

[78] P.R. Eiseman,
"Orthogonal grid generation", in *Numerical grid generation*, North-Holland Publishing Company, 1982.

[79] R.J.R. Johns,
"Prediction of flow in Diesel engine cylinders", Thesis submitted for the degree of Doctor of Philosophy, University of London, Imperial College, April 1980.

[80] A.D. Gosman and R.J.R. Johns,
"Development of a Predictive Tool for In-Cylinder Gas Motion in Engines", SAE Technical Paper Series, 780315, 1978.

[81] G. Marshall, P. Eiseman, and J.T. Kuo,
"A General Collapsing Technique for Three-Dimensional Algebraic Grid Generation", *Journal of Computational Physics*, 62, pp. 180-198, 1986.

[82] J.M. de la Viuda, J. Diet, and G. Ranoux,
"Patch-Independent Structured Multiblock Grids for CFD Computations", in *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Arcilla, Hauser, Eiseman and Thompson eds., Elsevier Science Publishers B.V. (North-Holland), 1991.

[83] K.L. Smit and T.Y. Su,
"A Geometry-Integrated Approach to Multiblock Grid Generation", in *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Arcilla, Hauser, Eiseman and Thompson eds., Elsevier Science Publishers B.V. (North-Holland), 1991.

[84] K. Becker,
"Interactive Algebraic Mesh Generation for Twin Jet Transport Aircraft",
in *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Arcilla, Hauser, Eiseman and Thompson eds., Elsevier Science Publishers B.V. 1991.

[85] C Dener, C Hirsh,
"IGG – an Interactive 3D Surface Modelling and Grid Generation System",
Paper presented at the Third International Conference on Numerical Grid Generation, Barcelona, Spain, 3-7 June 1991.

[86] Hirsch C, Lacor C, Dener C, VucinicD,
"An Integrated CFD System for 3D Turbomachinery Applications", *77th Symp. on CFD Techniques for Propulsion Applications*, San Antonio, Texas, May 1991.

[87] J.Q. Cordova,
"VisualGrid: a Software Package for Interactive Grid Generation", *AIAA 21st Fluid Dynamics, Plasma Dynamics and Lasers Conf.*, Seattle, 1990.

[88] K. Tsubui, K. Miyakishi, and K. Kuwahara,
"Incompressible flow simulations with solid models in rectangular grid systems", in *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Arcilla, Hauser, Eiseman and Thompson eds., Elsevier Science Publishers B.V. , 1991.

[89] E.K. Buratinski,
"A Fully Automatic Three-Dimensional Mesh Generator for Compex Geometries", in *International journal for numerical methods in engineering*, Volume 30, 1990.

[90] N.P. Weatherill,
"A method for generating irregular computational grids in multiply connected planar domains", in *International journal for numerical methods in fluids*, Vol. 8, pp. 181-197, 1988.

[91] R. Lohner and P. Parikh,
"Generation of Three Dimensional Meshes by the Advancing Front Method", in *International. journal for numerical methods in fluids*, Vol. 8, 1988.

[92] J.F. Greenleaf, T.S. Tu, and E.H. Wood,
"Computer generated 3D oscilloscopic images and associated techniques for display and study of the spatial distribution of pulmonary blood flow",
IEEE Transactions on nuclear sciences, NS-17, 1970, pp. 353-359.

[93] M.W.Vannier, J.L.Marsh, J.O.Warren,
"Three dimensional computer graphics for craniofacial surgical planning and evaluation", *Computer Graphics (IEEE)*, Vol. 17, No. 3, pp. 263-273, July 1983.

[94] G. T. Herman and J. K. Udupa,
"Display of 3D Digital Images: Computational Foundations and Medical Applications", *IEEE Computer Graphics and Applications*, Volume 3, No 5, pp. 39-46, August 1983.

[95] R. Yagel and A. Kaufman,
"Annotated Bibliography", Volume Visualisation, A.Kaufman ed., IEEE Computer Society Press Tutorial, pp 439-465, 1991.

[96] L. Gelberg, D. Kamins, and J. Vroom,
"VEX: A Volume Exploratorium. An Integrated Toolkit for Interactive Volume Visualisation", *Proceedings of the Chapel Hill Workshop on Volume Visualisation*, pp. 21-26, University of North Carolina Press, Chapel Hill, NC,. May 1989.

[97] "Scientific Visualisation: Bringing data into focus",
Computer Magazine, IEEE Computer Society, August 1989.

[98] R. Bakalash and A. Kaufman,
"MediCube: A 3D medical imaging architecture", *Computers and Graphics*, 13, 2, pp. 151-157, 1989.

[99] D. Cohen, A. Kaufman, R. Bakalash, and S. Bergman,
"Real time discrete shading", *The Visual Computer*, Feb. 1990, pp 16-27.

[100] K. Hoehne, M. Bomans, A. Pommert, M. Riemer, C.Schiers, U. Tiede, and G. Wiebecke,
"3D Visualisation of Tomographic Volume Data Using the Generalized Voxel Model", in *The Visual Computer*, pp 28-36, February 1990.

[101] A. Kaufman,
"Introduction to Volume Visualisation", *State of the Art in Data Vizualisation*, Course Notes 27, SIGGRAPH'90, August 1990.

[102] M. Levoy,
"A hybrid ray tracer for rendering polygon and volume data", *IEEE Computer Graphics and Aplications*, 10, 3, March 1990, pp33-40.

[103] P. Shirley and H. Neeman,
"Volume Visualisation at the Center for Supercomputing Research and Development", *Proceedings of the Chapel Hill Workshop on Volume Visualisation*, University of North Carolina Press, Chapel Hill, NC. May 1989, pp. 17-20.

[104] L. Brewster, S. Trivedi, H. Tuy, J. Udupa,
"Interactive Surgical Planning", *IEEE Computer Graphics and Applications*, Volume 4, Number 3, March 1984, pp. 31-40.

[105] A. Farrell, W.Yang, R.Zappulla,
"Animated 3D CT Imaging", *IEEE Computer Graphics and Applications*, Volume 5, Number 12, December 1985, pp. 26-32.

[106] G.T.Herman and H.K. Liu,
"Three-Dimensional Display of Human Organs from Computed Tomograms", *Computer Graphics and Image Processing*, Jan. 1979, pp1-12.

[107] LJ Rosenblum,
"Scientific Visualisation at Research laboratories", *Computer (IEEE)* , Volume 22, Number 8, August 1989, pp. 68-70.

[108] D.S.Goodsell, I.S.Mian and A.J.Olson,
"Rendering Volumetric data in molecular systems", *Journal of Molecular Graphics*, March 1989, pp. 35-36.

[109] V Pereyra, JA Rial,
"Visualizing Wave Phenomena with Seismic Rays", *Visualisation in Scientific Computing*, G.M.Nielson, B.Shriver, and L.J.Rosenblum, Eds, IEEE Computer Society Press, 1990, pp. 174-189.

[110] R.H.Wolfe and C.N.Liu,
"Interactive Visualisation of 3D Seismic Data: A Volumetric Method", *IEEE Comp. Graphics and Applications*, Vol. 8, No 7, July 1988, pp.24-30.

[111] T.V.Papathomas, J.A. Schiavone, and B.Julesz,
"Stereo Animation for Very Large Data Bases: Case Study-Meteorology", *IEEE Computer Graphics and Applications*, Vol. 7, No. 9, September 1987.

[112] W Hibbard, D Santek,
"Visualizing Large Meteorological Data Sets", *Computer (IEEE)*, Volume 22, Number 8, August 1989, pp. 53-59.

[113] M.K.Seager, N.E.Werner, M.E.Zosel, and R.E.Strout,
"Graphical Analysis of Multi/Micro-Tasking on Cray Multiprocessor", *Visualisation in Scientific Computing*, G.M.Nielson, B.Shriver, and L.J.Rosenblum, Eds, *IEEE Computer Society Press*, 1990, pp. 160-174.

[114] N.Hirsh and B.L.Brown,
"A Holistic Method for Visualizing Computer Performance Measurements", *Visualisation in Scientific Computing*, Nielson, Shriver, and Rosenblum, Eds, *IEEE Computer Society Press*, 1990, pp. 174-189.

[115] G.M. Nielson, B. Shriver,and Rosenblum eds.,
"Visualization in Scientific Computing", *IEEE Computer Society Press*, 1990.

[116] H.Chernoff,
"The Use of Faces to Represent Points in a K-Dimensional Space Graphically", *Journal of the American Statistical Association*, Volume 68, Number 342, 1973, pp. 361-368.

[117] S.L.Crawford and T.C.Fall,
"Projection Pursuit Techniques for the Visualisation of High Dimensional Datasets", *Visualisation in Scientific Computing*, G.M.Nielson, B.Shriver, and L.J.Rosenblum, Eds, *IEEE Computer Society Press*, 1990, pp. 94-108.

[118] S.Cunningham, J.R.Brown, and M.McGrath,
"Visualisation in Science and Engineering Education", *Visualisation in Scientific Computing*, G.M.Nielson, B.Shriver, and L.J.Rosenblum, Eds, *IEEE Computer Society Press*, 1990, pp. 48-57.

[119] P. Bunning and J. L. Steger,
"Graphics and Flow Visualization in Computation Fluid Dynamics", *AIAA* , 85-1507, 1985.

[120] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurtwitz, A. Van Dam STELLAR COMPUTER,
"SCIENTIFIC VISUALIZATION: The Application Visualization System: A Computational Environment for Scientific Visualisation", *IEEE Computer Graphics and Applications, July 1989.*

[121] DS Dyer,
"A Dataflow Toolkit for Visualisation", *IEEE Computer Graphics and Applications,* July 1990.

[122] T. Plessel,
"GAS User's Manual", NASA Ames Research Center, Code RFW, 1988.

[123] SE Rogers, PG Buning, FJ Merritt,
"Distributed Interactive Graphics Applications in Computational Fluid Dynamics", *International Journal of Supercomputing*, 1, 4, Winter 1987.

[124] T. Plessel,
"SURF Users Manual", NASA Ames Research Center, Code RFW, 1988.

[125] "IRIS Explorer™ User's Guide",
Silicon Graphics Computers Systems, Doc. No. 007-1371-010, June 1992.

[126] DL Brittain, J Aller, M Wilson, S-LC Wang,
"Design of an End-User Data Visualization System", Wavefront Technologies Inc., Santa Barbara, 1990.

[127] M. Giles M. & R. Haimes,
"Advanced Interactive Visualisation for CFD", *Computing Systems in Engineering* Vol. 1. No. 1. pp. 51-62, 1990.

[128] F. Lange, Private communication.

[129] American Cinematographer, 1990 Special Effects special, December 1990.

[130] P Burger, D Gillies,
"Interactive Computer Graphics", Addison Wesley 1989.

[131] Evans & Sutherland Computer Corporation,
"The Breadth of Visual Simulation Technology", Evans & Sutherland Computer Corporation, Salt Lake City, UT, 1989.

[132] H Fuchs, M Levoy, SM Pizer,
"Interactive Visualization of 3D Medical Data", *Computer (IEEE)* Volume 22, Number 8, August 1989, pp. 46-52.

[133] NC Kluksdahl, AM Kriman, DK Ferry,
"The Role of Visualization in the Simulation of Quantum Electronic Transport in Semiconductors", *Computer (IEEE),* Volume 22, Number 8, August 1989, pp. 60-67.

[134] D.E. Edwards,
"Three-Dimensional Visualisation of Fluid Dynamic Problems", AIAA-89-0136.

[135] E. Catmull,
"A System for Computer Generated Movies", in *Proc. ACM Annual Conference*, ACM, New York, NY, August 1972, pp. 422-431.

[136] C.W. Reynolds,
"Computer Animation with Scripts and Actors", SIGGRAPH 1982, pp. 289-296.

[137] R.M. Baecker,
"Picture Driven Animation", SJCC, AFIPS Press, Montvale, JN, 1969, pp. 273-288.

[138] S. Feiner, D. Salesin, and T. Banchoff,
"DIAL: A diagrammatic Animation Language", *CG & A*, 2(7), September 1982, pp. 43-54.

[139] Symbolics, Inc.
"S-Dynamics", Symbolics, Inc. Cambridge MA, 1985.

[140] P. Strauss,
"BAGS: The Brown Animation Generation System", Ph. D. Thesis, Techincal Report CS-88-22, Computer Science Departement, Brown University, Providence, RI, May 1988.

[141] G. Stern,
"Bbop - A System for 3D Keyframe Figure Animation", in *Introduction to Computer Animation*, Course Notes 7 for SIGGRAPH 83, New York, July 1983, pp. 240-243.

[142] W.T. Reeves and R. Blau,
"Approximate and Probabilistic Algorithms for Shading and Rendering Particle Systems", SIGGRAPH 85, pp. 313-322.

[143] A. Borning,
"Defining Constraints Graphically", in *SIGGHI'86 Conference Proceedings, ACM*, New York, 1986, pp. 137-143.

[144] Pixar Corporation,
"Luxo, Jr.", film, Pixar Corporation, San Rafael, CA, 1986.

[145] C.M. Ginsberg and D. Maxwell,
"Graphical Marionette", in *Proceedings of the SIGGRAPH/SIGART Interdisciplinary Workshop on Motion: Representation and Perception*, Toronto, April 4-6, 1983, pp. 172-179.

[146] R.L.Phillips,
"Distributed Visualization at Los Alamos National Laboratory", *Computer (IEEE)*, Volume 22, Number 8, August 89, pp. 70-77.

[147] K. Akeley,
"The Silicon Graphics 4D/240GTX Superworkstation", *CG & A*, 9(4), July 1989, pp. 71-83.

[148] P. Buning,
"Flow Visualisation of CFD Using Graphics Workstations", *AIAA* , 87-1180, *Proceedings of 8th Computational Fluid Dynamics Conference*, June 9-11, 1987.

[149] Intelligent Light Inc.,
"Fieldview fact sheet", Intelligent Light Inc., Fair Lawn, NJ, January 1991.

[150] PV-Wave Advantage user's manual, Visual Numerics inc., Houston, USA, 1992.

[151] L. Carrabine,
"Visualize Space Station vibrations", *CAE in Action*, Penton Publishing Inc. Cleveland Ohio, 1990.

[152] "Intelligent Light, reference manual for software release 2.2", Intelligent Light Inc., POB65, Fair Lawn, New Jersey 1990.

[153] GV Bancroft, T Plessel, F Merritt, PP Walatka, V Watson,
"Scientific Visualization in Computational Aerodynamics at NASA Ames Research Center", *Computer (IEEE)* , 22, 8, August 1989.

[154] R.B.Haber,
"Scientific Visualisation and the Rivers Project at the National Center for Supercomputing Applications", *Computer (IEEE)*, Volume 22, Number 8, August 89, pp. 84-89.

[155] P.O.Witze,
"Influence of air motion variation on the performance of a direct injection stratified Charge engine", *I. Mech. E.* C394/80, 1980.

[156] H.W.Barnes, C. Moss,
"A designer's viewpoint", In *Passenger Car Engines*, Institution of Mechanical Engineers, London, 1975.

[157] J.B. Heywood,
"Internal Combustion Engine Fundamentals", McGraw-Hill, 1988.

[158] M.Namazian and J.B.Heywood,
"Flow in the piston-cylinder-ring crevices of a spark-ignition engine: effect on hydrocarbon emissions, efficency and power", *SAE Technical Paper Series*, 820088, International Congress and Exposition, Detroit, Michigan, February 22-26, 1982.

[159] M. Namazian, S. Hansen, E. Lyford-Pike, J. Sanchez-Barsee, J. Heywood, and J. Rife,
"Schlieren Visualization of the Flow and Density Fields in the Cylinder of a Spark-Ignition Engine", SAE paper 800044, *SAE Trans.*, volume 89, 1980.

[160] A.F. Bicen, C. Vafidis, and J. H. Whitelaw,
"Steady and Unsteady Airflow through the Intake Valve of a Reciprocating Engine", *ASME Trans., J. Fluids Engng*, vol 107, pp. 413-420, 1985.

[161] T. Uzkan, C. Borgnakke, and T. Morel,
"Characterization of Flow Produced by a High-Swirl Inlet Port", SAE paper 830266, 1983.

[162] P. O. Witze,
"The effect of Spark Location on Combustion in a Variable Swirl Engine", SAE paper, 820044, *SAE Trans.*, vol 91, 1982.

[163] M. W. Scott,
"Understanding Diesel Combustion through the use of High Speed Moving Pictures in Color", SAE paper 690002, *SAE Trans.*, vol 78, 1969.

[164] T. Kamimoto, H. Kobayashi, and S. Matsuoka,
"A Big Size Rapid Compression Machine for Fundamental Studies of Diesel Combustion", SAE paper 811004, *SAE Trans.*, vol. 90, 1981.

[165] C. Arcoumanis and J.H. Whitelaw,
"Fluid mechanics of internal combustion engines - a review", Proc. I. Mech. E, Vol. 201, No C1, 1987.

[136] C.W. Reynolds,
"Computer Animation with Scripts and Actors", SIGGRAPH 1982, pp. 289-296.

[137] R.M. Baecker,
"Picture Driven Animation", SJCC, AFIPS Press, Montvale, JN, 1969, pp. 273-288.

[138] S. Feiner, D. Salesin, and T. Banchoff,
"DIAL: A diagrammatic Animation Language", *CG & A*, 2(7), September 1982, pp. 43-54.

[139] Symbolics, Inc.
"S-Dynamics", Symbolics, Inc. Cambridge MA, 1985.

[140] P. Strauss,
"BAGS: The Brown Animation Generation System", Ph. D. Thesis, Techincal Report CS-88-22, Computer Science Departement, Brown University, Providence, RI, May 1988.

[141] G. Stern,
"Bbop - A System for 3D Keyframe Figure Animation", in *Introduction to Computer Animation*, Course Notes 7 for SIGGRAPH 83, New York, July 1983, pp. 240-243.

[142] W.T. Reeves and R. Blau,
"Approximate and Probabilistic Algorithms for Shading and Rendering Particle Systems", SIGGRAPH 85, pp. 313-322.

[143] A. Borning,
"Defining Constraints Graphically", in *SIGGHI'86 Conference Proceedings, ACM*, New York, 1986, pp. 137-143.

[144] Pixar Corporation,
"Luxo, Jr.", film, Pixar Corporation, San Rafael, CA, 1986.

[145] C.M. Ginsberg and D. Maxwell,
"Graphical Marionette", in *Proceedings of the SIGGRAPH/SIGART Interdisciplinary Workshop on Motion: Representation and Perception*, Toronto, April 4-6, 1983, pp. 172-179.

[146] R.L.Phillips,
"Distributed Visualization at Los Alamos National Laboratory", *Computer (IEEE)*, Volume 22, Number 8, August 89, pp. 70-77.

[147] K. Akeley,
"The Silicon Graphics 4D/240GTX Superworkstation", *CG & A*, 9(4), July 1989, pp. 71-83.

[148] P. Buning,
"Flow Visualisation of CFD Using Graphics Workstations", *AIAA* , 87-1180, *Proceedings of 8th Computational Fluid Dynamics Conference*, June 9-11, 1987.

[149] Intelligent Light Inc.,
"Fieldview fact sheet", Intelligent Light Inc., Fair Lawn, NJ, January 1991.

[150] PV-Wave Advantage user's manual, Visual Numerics inc., Houston, USA, 1992.

[151] L. Carrabine,
"Visualize Space Station vibrations", *CAE in Action*, Penton Publishing Inc. Cleveland Ohio, 1990.

[152] "Intelligent Light, reference manual for software release 2.2", Intelligent Light Inc., POB65, Fair Lawn, New Jersey 1990.

[153] GV Bancroft, T Plessel, F Merritt, PP Walatka, V Watson, "Scientific Visualization in Computational Aerodynamics at NASA Ames Research Center", *Computer (IEEE)* , 22, 8, August 1989.

[154] R.B.Haber, "Scientific Visualisation and the Rivers Project at the National Center for Supercomputing Applications", *Computer (IEEE)*, Volume 22, Number 8, August 89, pp. 84-89.

[155] P.O.Witze, "Influence of air motion variation on the performance of a direct injection stratified Charge engine", *I. Mech. E.* C394/80, 1980.

[156] H.W.Barnes, C. Moss, "A designer's viewpoint", In *Passenger Car Engines*, Institution of Mechanical Engineers, London, 1975.

[157] J.B. Heywood, "Internal Combustion Engine Fundamentals", McGraw-Hill, 1988.

[158] M.Namazian and J.B.Heywood, "Flow in the piston-cylinder-ring crevices of a spark-ignition engine: effect on hydrocarbon emissions, efficency and power", *SAE Technical Paper Series*, 820088, International Congress and Exposition, Detroit, Michigan, February 22-26, 1982.

[159] M. Namazian, S. Hansen, E. Lyford-Pike, J. Sanchez-Barsee, J. Heywood, and J. Rife, "Schlieren Visualization of the Flow and Density Fields in the Cylinder of a Spark-Ignition Engine", SAE paper 800044, *SAE Trans.*, volume 89, 1980.

[160] A.F. Bicen, C. Vafidis, and J. H. Whitelaw, "Steady and Unsteady Airflow through the Intake Valve of a Reciprocating Engine", *ASME Trans., J. Fluids Engng*, vol 107, pp. 413-420, 1985.

[161] T. Uzkan, C. Borgnakke, and T. Morel, "Characterization of Flow Produced by a High-Swirl Inlet Port", SAE paper 830266, 1983.

[162] P. O. Witze, "The effect of Spark Location on Combustion in a Variable Swirl Engine", SAE paper, 820044, *SAE Trans.*, vol 91, 1982.

[163] M. W. Scott, "Understanding Diesel Combustion through the use of High Speed Moving Pictures in Color", SAE paper 690002, *SAE Trans.*, vol 78, 1969.

[164] T. Kamimoto, H. Kobayashi, and S. Matsuoka, "A Big Size Rapid Compression Machine for Fundamental Studies of Diesel Combustion", SAE paper 811004, *SAE Trans.*, vol. 90, 1981.

[165] C. Arcoumanis and J.H. Whitelaw, "Fluid mechanics of internal combustion engines - a review", Proc. I. Mech. E, Vol. 201, No C1, 1987.

[166] C. Vafidis, G. Vorropoulos, and J. H. Whitelaw,
"Valve and in-cylinder flow in a motored engine with a directed port and cylindrical piston bowl", Imperial College, Mech. Eng. Dept., Fluids Section Report, FS/87/17.

[167] M. Sarantinos and T. Strauss,
"The use of the SPEED-CODE in the period from May 1992 to April 1993",
Status report, Volkswagen AG, Forshung und Entwicklung, Motorische Arbeitsverfahren, May 1993.

[168] M.J. Tindal, T.J. Williams, and M. Aldoory,
"The Effect of Inlet Port Design on Cylinder Gas Motion in Direct Injection Diesel Engines", in *Flows in Internal Combustion Engines*, pp. 101-111, ASME, New York, 1982.

[169] A. Ekchian and D.P. Hoult,
"Flow Visualization Study of the Intake Process of an Internal Combustion Engine",
SAE paper 790095, *SAE Trans.*, vol 88, 1979.

[170] AD Gosman, C. Kralj, CJ Marooney, P. Theodossopoulos,
"The development of the SPEED code for Diesel combustion simulation",
I. Mech. E., C448/035, 1992.

[171] B. E. Launder and D. B. Spalding,
"The numerical computation of turbulent flows",
Computer Methods in Applied Mechanics and Engineering, vol. 3, pp. 269, 1974.

[172] S. H. El Tahry,
"k-ε equations for compressible reciprocating engine flows", *AIAA Journal of Energy*, 7, No. 4, pp 345, 1983.

[173] "SPEED-DC user's manual", Computational Dynamics Ltd, Olympic House, 317 Latimer Road, London W10, 6RA.

[174] F. Obermeier,
"Modelling of nozzle-flow", IDEA subprogramme A1, March 1991.

[175] K.Y. Huh & A.D. Gosman,
"A phenomenological model of Diesel spray atomisation", International Conference on multiphase flows, Tsukuba, September 1991.

[176] R.D. Reitz & R. Diwakar,
"Effects of drop breakup on fuel sprays", SAE 860469, 1986.

[177] G.L. Borman and J.H. Johnson,
"Unsteady vaporization histories and trajectories of fuel drops injected into swirling air", SAE (598C), 1962.

[178] M.P. Halstead, L.J. Kirsch, A. Prothero, and C.P. Quinn,
"A mathematical model for hydrocarbon autoignition at high pressures"

[179] M.A. Theobald, W.K. Cheng,
" A numerical study of Diesel Ignition", ASME report No. 87-FE-2.

[180] N. Zarzalis,
"Berechnung des instationären Strömungsfeldes und der Flammen-ausbreitung im Zylinder eines Otto-Motors", Ph.D. thesis, T.H. Karlsruhe, Germany, 1986.

[181] N. Fateh, Private communication, 1986.

[182] R.I. Issa,
"Solution of the implicitly discretised fluid flow equation by operator splitting", *Journal of Computational Physics*, volume 62, pp. 40-62, 1986.

[183] A.R. Forrest,
"The Twisted Cubic Curve: A Computer-Aided Geometric Design Approach", *Computer Aided Design*, 12(4), pp. 165-172, July 1980.

[184] "STAR user's manual", Computational Dynamics Ltd.

[185] S.A. Coons,
"Surfaces for Computer Aided Design of Space Forms", Report No. MAC-TR-41, Massachusetts Institute of Technology, Cambridge, Mass., 1967.

[186] L.Gelberg, D. Kamins, D. Parker, J Sacks,
"Visualisation techniques for structured and unstructured scientific data", *SIGGRAPH 1990, 17th International Conference On Computer Graphics and Interactive Techniques*, Course Notes, Dallas August 1990.

[187] G.W. Schweimer, Volkswagen AG, F-MA, combustion research, private communication.

[188] F. Pischinger,
"Verbrennungsmotoren", Vorlesungsumdruck, Rheinisch -Westfälische Technische Hocchschule, Aachen, 1987.

[189] A.R. Smith,
"Color Gamut Transform Pairs", SIGGRAPH 78, pp. 12-19, 1978.

[190] R. Lohner,
"Fast Graphics Algorithms Using Unstructured Grids",
SIGGRAPH 1990, 17th International Conference On Computer Graphics and Interactive Techniques, Course Notes, Dallas August 1990.

[191] W.E.Lorenson and H.E.Cline,
"Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Proceedings of *SIGGRAPH '87*, in *Computer Graphics*, Volume 21, No. 4, July 1987.

[192] C. Hill,
"Software design for interactive engineering simulation", Thesis submitted for the degree of Doctor of Philosophy, University of London, Imperial College, 1993.

[193] H. Gouraud,
"Continuous shading of curved surfaces", *IEEE Trans on Computers*, C-20(6), June 1971.

[194] B.T. Phong,
"Illumination for computer generated pictures", CACM, 18(6), June 1975, pp.311-317.

[195] Apple Computer Inc.
"Human Interface Guidelines: The Apple Desktop Interface", Addison-Wesley, 1987.

[196] B. Shneiderman
"Direct Manipulation: A Step Beyond Programming Languages", *Computer (IEEE)* 16, 8, 1983.

[197] J. Johnson, T.L. Roberts, W. Verplank, D.C. Smith, C. Irby, M. Beard, K. Mackey
"The Xerox Star: a Retrospective", *IEEE Computer*, 22, 9, September 1989.

[198] "O.S.F. Motif: Programmer's Guide, revision 1.1",
Prentice Hall, New Jersey, 1991.

[199] BF Webster,
"The NeXt Book", Addison Wesley, 1982.

[200] Sun Microsystems Inc. and AT&T,
"Open Look Graphical User Interface Functional Specification", Addison-Wesley 1989.

[201] MA Linton, JM Vlissides, PR Calder,
"Composing User Interfaces with InterViews", *Computer (IEEE)* 22, 2, 1989.

[202] Telesoft-HIS,
"TeleUse: the OSF/Motif Graphical User Interface Builder", Publicity material, Telesoft, San Diego CA, 1991.

[203] Apollo Computer Inc.,
"Creating User Interfaces with Open Dialogue", Apollo Computer Inc., Chelmsford MA

[204] H. Hill, Private communication.

[205] E.P. Glinert,
"Visual Programming Environments: Applications and Issues" IEEE Computer Society Press, 1990.

[206] P.E .Haeberli,
"ConMan: a Visual Programming Language for Interactive Graphics", *Computer Graphics (IEEE)*, 22, 4, August 1988.

[207] N Hunt,
"IDF: a Graphical Data Flow Programming Language for Image Processing and Computer Vision", Technical report TR-90-05, Teleos Research, Palo Alto, CA, 1990.

[208] Iconicon Inc.,
"Product Description: Iconicode Image Processing Development Tools", Iconicon, 1991.

[209] Symbolics Inc.
"Handbooks of Symbolics Programming Environment, 7a, Programming the User Interface – Concepts", Symbolics Inc., 1988.

[210] G.W. Cherry,
"Software Construction by Object-Oriented Pictures: Stimulus-Response Machines", ACM SIGSOFT 16, 2, April 1991.

[211] B.A. Myers,
"Taxonomies of Visual Programming and Program Visualization", in *Journal on Visual Languages and Computing*, Volume 1, March 1990.

[212] P.H. Ang, P.A. Ruetz, D. Auld,
"Video Compression Makes Big Gains", IEEE Spectrum, October 1991.

[213] Philips International Inc.,
"Compact Disk Interactive", McGraw-Hill, 1989.

[214] S. Rockman, C. Cain, N. Walker,
"Commodore CDTV", *Personal Computer world* 13, 8, August 1990.

[215] Sony Microsystems,
"The Multimedia Connection", Sony Literature, 1990.

[216] R.L. Phillips,
"MediaView: a General Multimedia Digital Publication System", *Communications ACM*, 34, 7, July 1991.

[217] Negreponti,
"Beyond the desktop metaphor", Lecture at Imperial Coll., London, 1990.

[218] S.R. Levine, S.F. Ehrlich,
"The Freestyle System: a Design Perspective", *Human-Machine Interactive Systems*, ed. A Klinger, Plenum Press NY, 1991.

[219] C.R. Strathmeyer,
"Voice in Computing: an Overview of Available Technologies" *Computer (IEEE)*, 23, 8, August 1991.

[220] R.J .Abbott,
"A Fully Integrated Natural Language Knowledge Base Management System", *Human-Machine Interactive Systems*, Plenum Press, NY, 1991.

[221] D.L. Parnas,
"On the Criteria to be Used in Decomposing Systems into Modules", *Comm. ACM*, 15, 1972, reprinted in Tutorial on Software Design Techniques, P Freeman, and AI Wasserman, 4th ed., IEEE, 1983.

[222] A. Goldberg,
"Smalltalk-80: The interactive programming environment", Addison-Wesley, 1984.

[223] B. Stroustrup,
"The C++ Programming Language", Addison-Wesley 1987.

[224] M.T. Ozsu, P. Valdinez,
"Distributed Database Systems: where are we now", *Computer (IEEE)*, 24, 8, Aug. 1991.

[225] R.S.Gallagher and J.C.Nagtegaal,
"An Efficient 3D Visualisation Technique for Finite Element Models and Other Coarse Volumes", *Computer Graphics (IEEE)*, Volume 23, No. 3, July 1989.

[226] A.D. Gosman, C.J. Marooney, and M. Sarantinos,
"A user's guide for the pre-processing system of the SPEED code", Imperial College, Dept. of Mech. Eng., June 1991.

[227] A.D. Gosman, C.J. Marooney, and M. Sarantinos,
"A user's guide for the GRAFIS code", Imperial College, Dept. of Mech. Eng., June 1991.

[228] P. Buning and J. L. Steger,
"Graphics and Flow Visualization in CFD", *AIAA* , 85-1507, 1985.

[229] M. McKinley,
"Computer Graphics: On the fast track", *World Breadcast News*, Oct. 1990, pp. 40-44.

[230] S.Lewis,
"Animated Research, Use of visualization technology for interpretation of complex multi-dimensional data", *Computer Graphics World*, Pennwell 1988.

[231] Intelligent Light,
"Reference manual for software release 2.2", Intelligent Light, POB65, Fair Lawn, New Jersey 1990.

[232] L. Carrabine,
"Visualize Space Station vibrations" *CAE in Action*, Penton Publishing Inc. 1990.

[233] D.E. Edwards,
"Three-Dimensional Visualisation of Fluid Dynamic Problems" *AIAA-89-0136.*

[234] "Video Production Techniques. A handbook for television, film and audio-visual program makers", Kluwer Publishing Limited, in association with ITV (UK), 1990.

[235] G. White,
"Video Techniques", *Newnes Technical Books*, 1982.

[236] P. Freeman, A.I. Wasserman,
" Software Design Techniques", Fourth edition, IEEE Computer Society Press, 1983.

[237] D. Voorhies,
"RISC Rendering: Reduced-Complexity Graphics", *IEEE Computer Graphics and Applications*, Volume 9, No. 4, July 1989.

[238] PAL Genlock Encoder 606P MKII, operations manual, Acron Ltd, 1989.

[239] "LYON-LAMB PAL MiniVas Animation Controller", operations manual, LYON-LAMB Video Animation Systems, 1989.

[240] LYON-LAMB ENC-7, Operations Manual, LYON-LAMB Video Animation Systems, 1990.

[241] SONY VO-9850P U-matic Editing Video Tape Recorder (SP), Specifications Manual, Sony Corporation.

[242] W. Hentschel, A. Hesse, and K.P. Schindler,
"Experimental investigation of spray formation and combustion in an actual Diesel engine", Volkswagen AG, Research, periodic report for the Diesel combustion research progremme IDEA, subprogramme C4, for the period 1.1.1992 – 30.6.1992.

[243] C. Arcoumanis and K.S. Jeong,
"Steady flow characterisation of the Volkswagen 1.9l Diesel cylinder head",
Technical report submitted to Volkswagen AG, Imperial College, Dept. of Mech. Eng. London 1991.

[244] G. Lepperhof and M. Houben,
"Formation and oxidation of soot and unburnt hydrocarbons inside a Diesel combustion chamber", Lehrstuhl für angewandte Thermodynamik, IDEA subprogramme D.8, periodic report, 1989.

[245] M. Sarantinos and C. Kralj,
"Study of the formation of fuel sprays and combustion in an axisymmetric DI Diesel engine", Techinical report no. 102001 prepared for Volkswagen AG, by Icon Computer Graphics Ltd, December 1993.

[246] "Data Visualiser User's Guide",
Fourth edition, Wavefront Technologies Inc., Santa Barbara CA, May 1993.

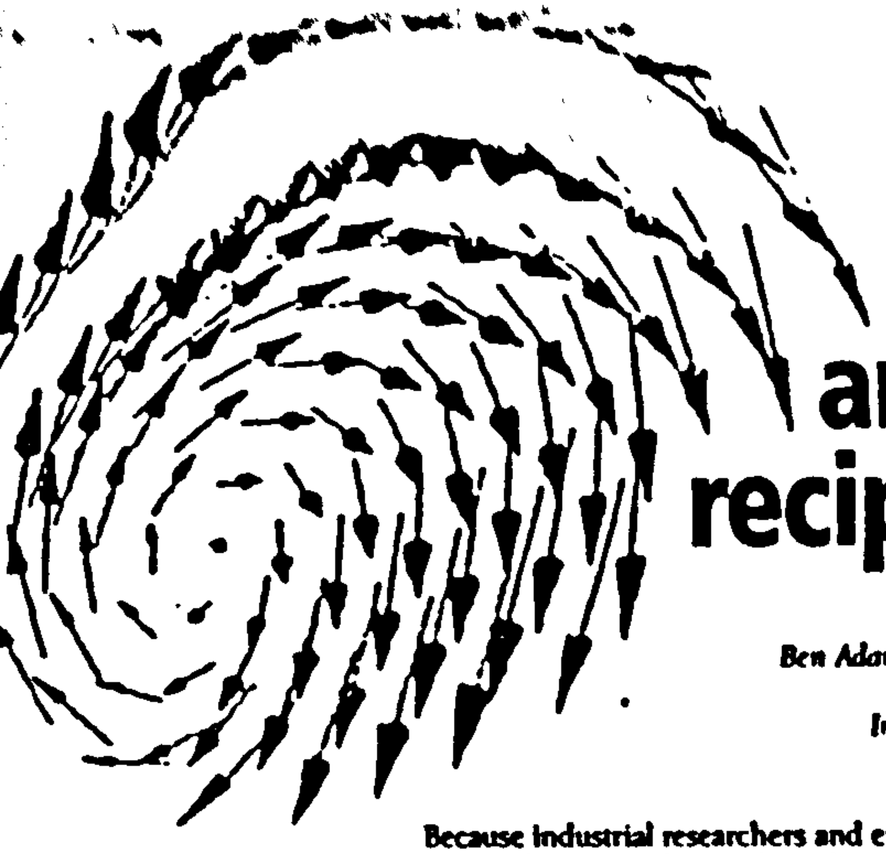
[247] M. Sarantinos and F.Lange,
"Flow, injection, and combustion simulation for the 1.9L DI Diesel engine, series version", Techinical report no. 101001 prepared for Volkswagen AG, by Icon Computer Graphics Ltd, December 1993.

[248] M. Sarantinos,
"Gas flow and combustion study in a variable compression ratio petrol engine", Techinical report no. 103001 prepared for Volkswagen AG, by Icon Computer Graphics Ltd, February 1994.

Blank

In

Original



Simulation of in-cylinder flow and combustion in reciprocating engines

Ben Adamson, David Gosman, Charles Hill, Chris Marooney, Behrooz Nasseri, Michael Sarantinos, Theodor Theodoropoulos, and Henry Weller
Imperial College of Science, Technology and Medicine, London, England

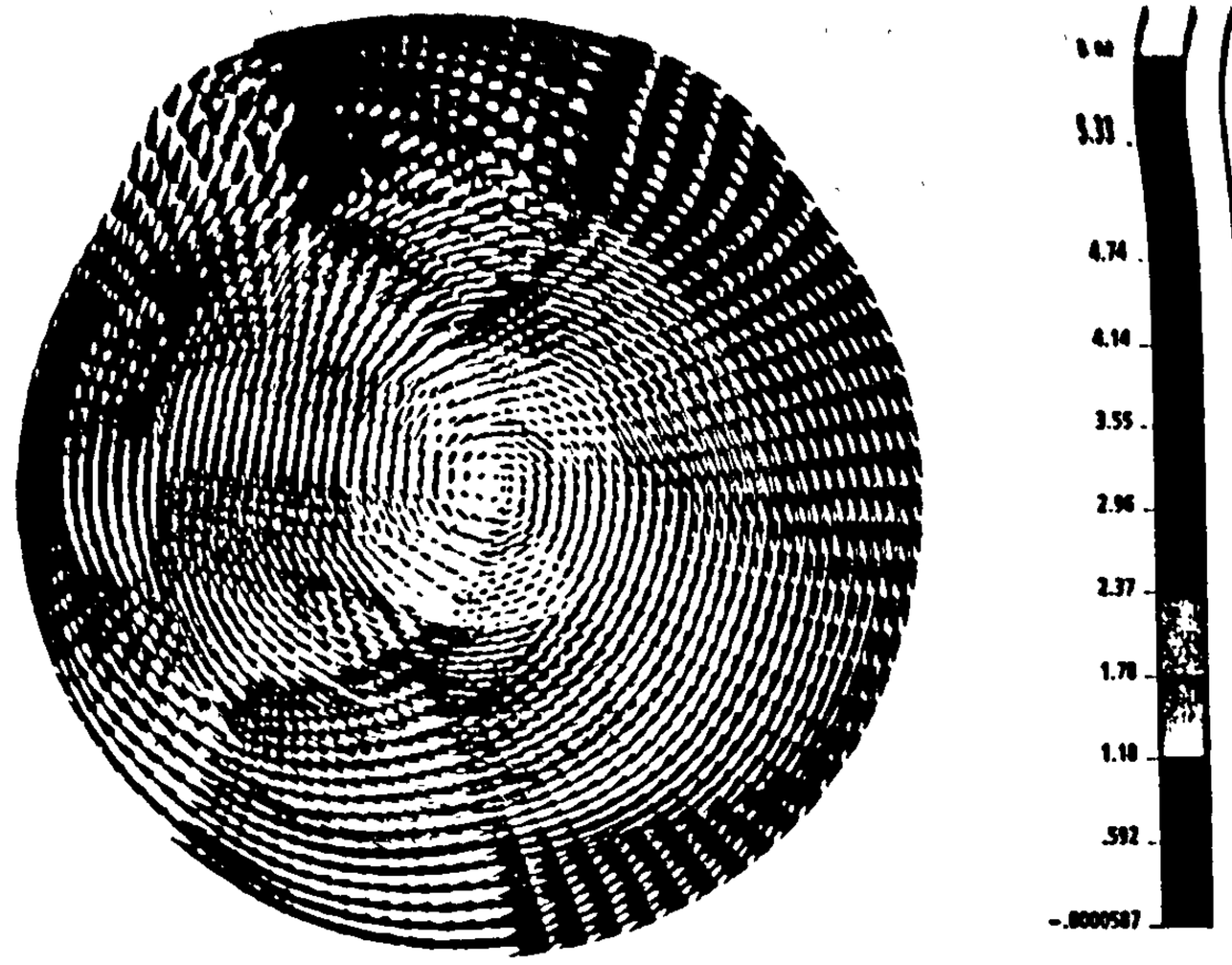
Because industrial researchers and engineers require precise solutions to their technical problems, computational fluid dynamics (CFD) methodologies must accommodate real flow geometries. In many cases, including that of the reciprocating internal combustion engine, these geometries are difficult to mesh with conventional regular grids, which are susceptible to distortion. Unstructured body-fitted meshes best address such cases because they provide the flexibility to fit boundary and flow features without introducing large amounts of non-orthogonality into the mesh. This meshing strategy is used in the SPEED (Simulation Programme for Engine Evaluation and Design) code developed at Imperial College.[1] The development of the SPEED code has involved not only the adaptation for use on unstructured meshes of the finite-volume discretization method (FVM) and the conjugate gradient (CG) solution method for matrix equations, but also techniques for creating the mesh, describing its continuity, moving and restructuring it in response to boundary and flow changes, and visualizing the four-dimensional property fields generated by the code. The applications of the code described here were run on the CRAY X-MP/28 computer system at the University of London Computer Centre.

Discretization, numerics, and modeling

The SPEED code uses the finite-volume method to discretize the CFD equations for mass, momentum, internal energy, turbulence, and the chemical species for ignition and combustion. The equations are solved in a fully implicit manner so that large time steps can be used. This implicitness is necessary to accommodate the large computational demands made by engine simulation codes. These demands exist due to two primary factors: (1) the variable sets for the models of turbulence, fuel-spray, droplet motion, and combustion chemistry, as well as the variable set for the basic aerodynamic flow, are large and will increase as more detailed models come into use, and (2) the unstructured mesh requires a database of addresses that describe its connectivity. This requirement, plus the larger supporting storage needed for efficient computation with unstructured meshes, also yields a large variable set. In combination, these factors create a substantial memory requirement, and as a result CPU time must be minimized to achieve an acceptable turnaround time.

The interactions among the various models are represented in a predictor-corrector manner after the PISO algorithm of Issa.[2] Corrector steps are solved neighbor-explicitly; that is, in the correction of a given node value, its neighbor's values are set to the latest values already available. For neighbor-explicit calculations, the algorithm is highly vectorizable because in an unstructured mesh the vector length equals the mesh size; the standard version of the code already is conditioned suitably for vector machines in this way. For parallel processing by subdomain decomposition, this part of the code also is well-conditioned for Cray Research hardware architectures, provided the neighbors on the subdomain boundary are duplicated in the data. The major architecture-conditioning challenge comes from the fully implicit predictor stages. The solution of the unstructured sparse matrix equations generated for these messages is best accomplished by a CG solver, which in principle is structurally neither vectorizable nor parallelizable. However, we have found that if the mesh node numbering is preconditioned such that the difference between neighbor numbers exceeds the vector length of the machine, then the CG solution algorithm can be substructured to yield a vectorizable inner loop, though the increased bandwidth increases the iterations to solution by about 20 percent. The technique is not available in parallel architectures, but research continues in this area.

For dispersed-phase calculations, Lagrangian equations of motion for a representative sample of droplets are used to model atomization, evaporation, breakup, and coalescence. Most of the Lagrangian code is independent of the mesh structure, and the extension to unstructured meshes involves defining a method of particle tracking. In the SPEED code, a local coordinate system is defined for each cell, on which a conventional search procedure can be used.



Sample engine results

The engine calculations described here illustrate the range of phenomena that can be calculated with the SPEED code and the visualization options available. Figure 1 shows the top-dead-center swirl flow in a disc chamber geometry. The vector plot display of velocity demonstrates the value of color in decoding the basic velocity magnitude information conveyed by arrow length. Comparisons of calculations with experimental results are shown in Figure 2. The significant improvement seen in the swirl velocity comparison on moving from a coarse mesh of 8000 cells to a fine mesh of 30,000 cells is expected from the properties of the upwind differencing technique used to model convection. A similar improvement in the turbulence intensity (not shown) is due to improved resolution of the generating flow gradients.

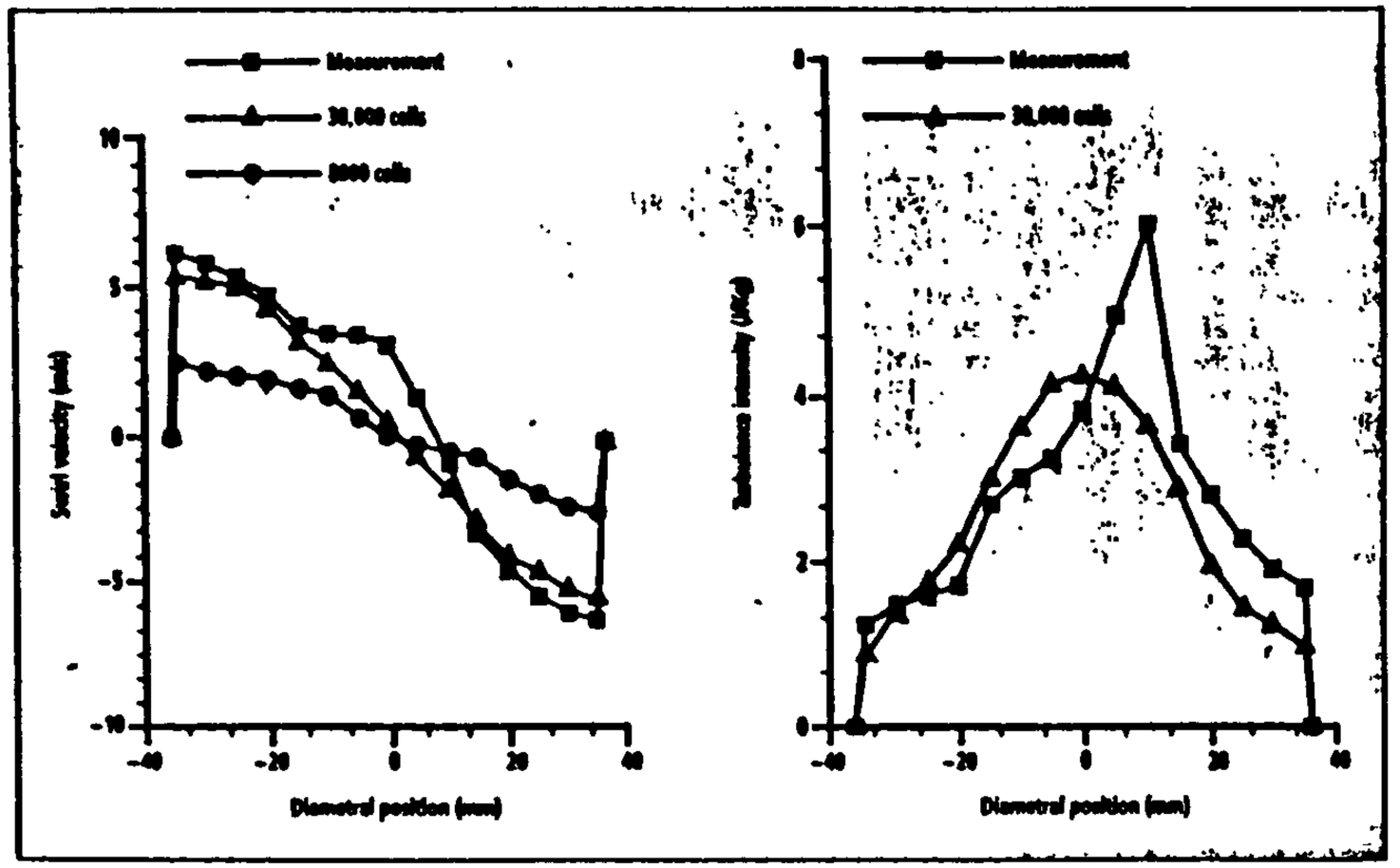
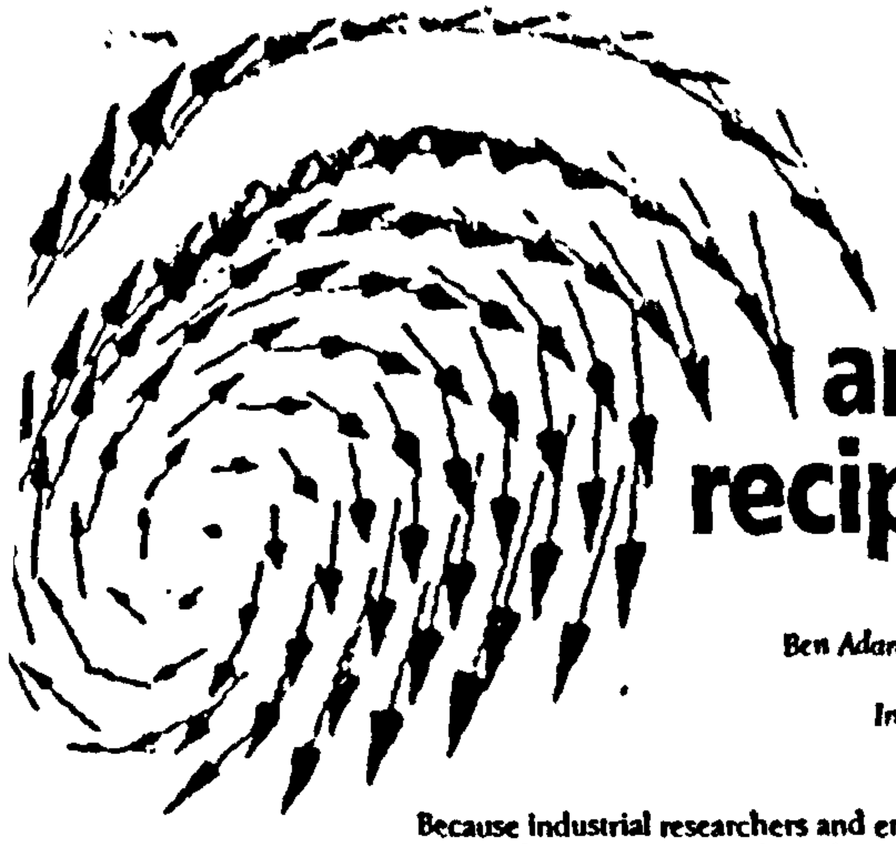


Figure 2. Comparison of measurements and computational results for top-dead-center compression in a disc chamber engine.

Best Copy
Available

Print bound close to the spine



Simulation of in-cylinder flow and combustion in reciprocating engines

Ben Adamson, David Gosman, Charles Hill, Chris Marooney, Behrooz Nasseri, Michael Sarantinos, Theodor Theodoropoulos, and Henry Weller
Imperial College of Science, Technology and Medicine, London, England

Because industrial researchers and engineers require precise solutions to their technical problems, computational fluid dynamics (CFD) methodologies must accommodate real flow geometries. In many cases, including that of the reciprocating internal combustion engine, these geometries are difficult to mesh with conventional regular grids, which are susceptible to distortion. Unstructured body-fitted meshes best address such cases because they provide the flexibility to fit boundary and flow features without introducing large amounts of non-orthogonality into the mesh. This meshing strategy is used in the SPEED (Simulation Programme for Engine Evaluation and Design) code developed at Imperial College.[1] The development of the SPEED code has involved not only the adaptation for use on unstructured meshes of the finite-volume discretization method (FVM) and the conjugate gradient (CG) solution method for matrix equations, but also techniques for creating the mesh, describing its continuity, moving and restructuring it in response to boundary and flow changes, and visualizing the four-dimensional property fields generated by the code. The applications of the code described here were run on the CRAY X-MP/28 computer system at the University of London Computer Centre.

Discretization, numerics, and modeling

The SPEED code uses the finite-volume method to discretize the CFD equations for mass, momentum, internal energy, turbulence, and the chemical species for ignition and combustion. The equations are solved in a fully implicit manner so that large time steps can be used. This implicitness is necessary to accommodate the large computational demands made by engine simulation codes. These demands exist due to two primary factors: (1) the variable sets for the models of turbulence, fuel-spray, droplet motion, and combustion chemistry, as well as the variable set for the basic aerodynamic flow, are large and will increase as more detailed models come into use, and (2) the unstructured mesh requires a database of addresses that describe its connectivity. This requirement, plus the larger supporting storage needed for efficient computation with unstructured meshes, also yields a large variable set. In combination, these factors create a substantial memory requirement, and as a result CPU time must be minimized to achieve an acceptable turnaround time.

The interactions among the various models are represented in a predictor-corrector manner after the PISO algorithm of Issa.[2] Corrector steps are solved neighbor-explicitly; that is, in the correction of a given node value, its neighbor's values are set to the latest values already available. For neighbor-explicit calculations, the algorithm is highly vectorizable because in an unstructured mesh the vector length equals the mesh size; the standard version of the code already is conditioned suitably for vector machines in this way. For parallel processing by subdomain decomposition, this part of the code also is well-conditioned for Cray Research hardware architectures, provided the neighbors on the subdomain boundary are duplicated in the data. The major architecture-conditioning challenge comes from the fully implicit predictor stages. The solution of the unstructured sparse matrix equations generated for these messages is best accomplished by a CG solver, which in principle is structurally neither vectorizable nor parallelizable. However, we have found that if the mesh node numbering is preconditioned such that the difference between neighbor numbers exceeds the vector length of the machine, then the CG solution algorithm can be substructured to yield a vectorizable inner loop, though the increased bandwidth increases the iterations to solution by about 20 percent. The technique is not available in parallel architectures, but research continues in this area.

For dispersed-phase calculations, Lagrangian equations of motion for a representative sample of droplets are used to model atomization, evaporation, breakup, and coalescence. Most of the Lagrangian code is independent of the mesh structure, and the extension to unstructured meshes involves defining a method of particle tracking. In the SPEED code, a local coordinate system is defined for each cell, on which a conventional search procedure can be used.

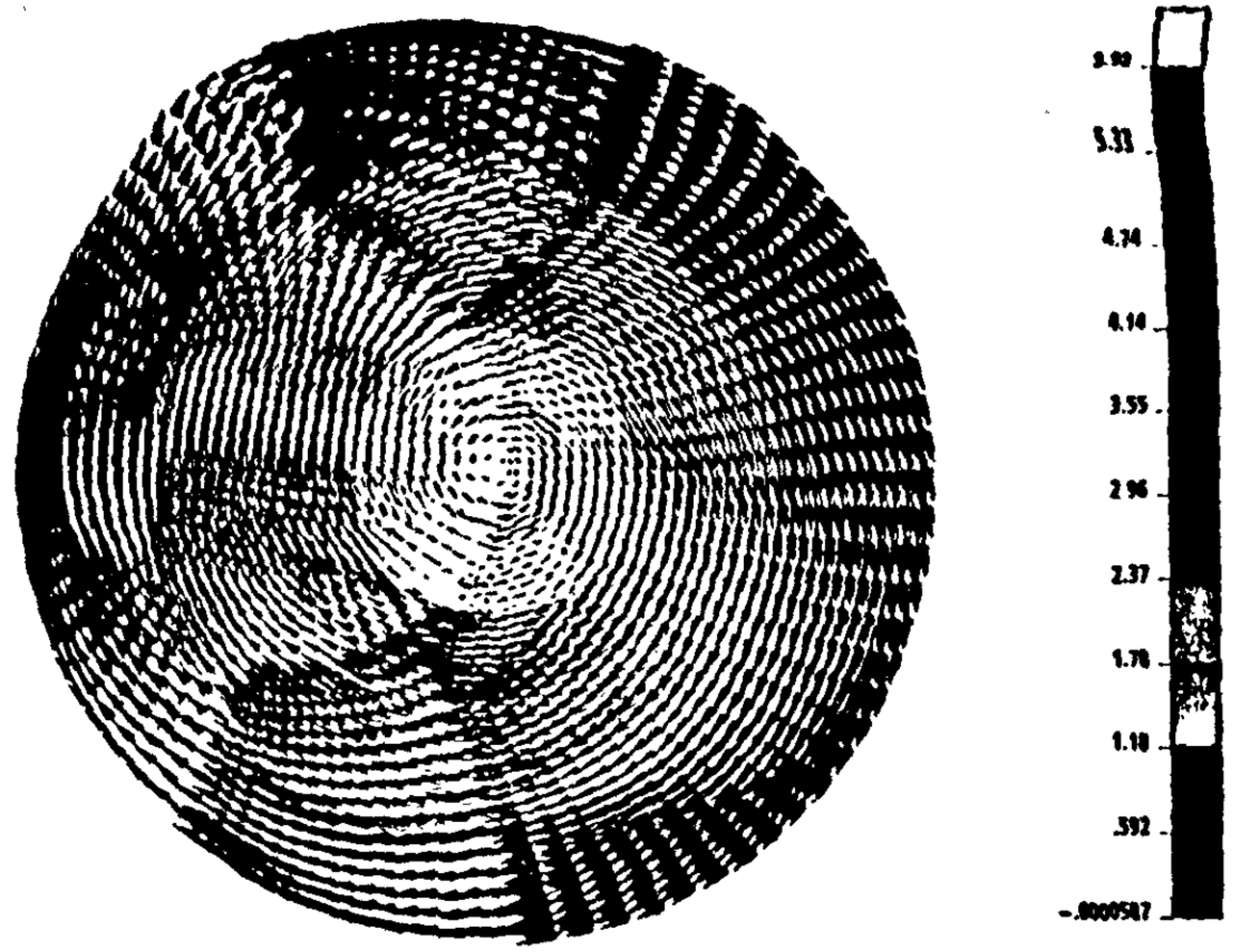


Figure 1. Bird chamber engine calculated swirl velocity field at top-dead-center.

Sample engine results

The engine calculations described here illustrate the range of phenomena that can be calculated with the SPEED code and the visualization options available. Figure 1 shows the top-dead-center swirl flow in a disc chamber geometry. The vector plot display of velocity demonstrates the value of color in decoding the basic velocity magnitude information conveyed by arrow length. Comparisons of calculations with experimental results are shown in Figure 2. The significant improvement seen in the swirl velocity comparison on moving from a coarse mesh of 8000 cells to a fine mesh of 30,000 cells is expected from the properties of the upwind differencing technique used to model convection. A similar improvement in the turbulence intensity (not shown) is due to improved resolution of the generating flow gradients.

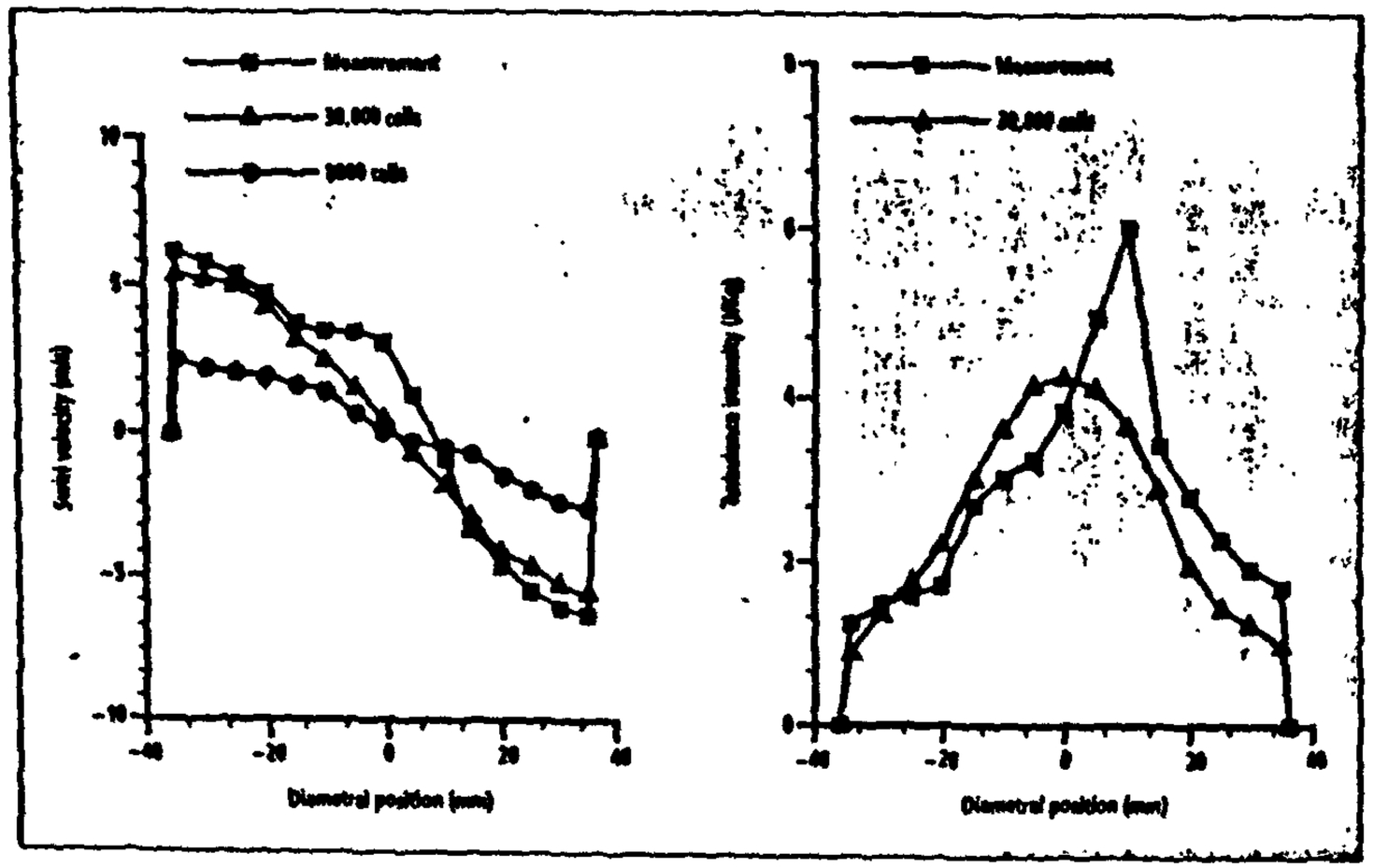
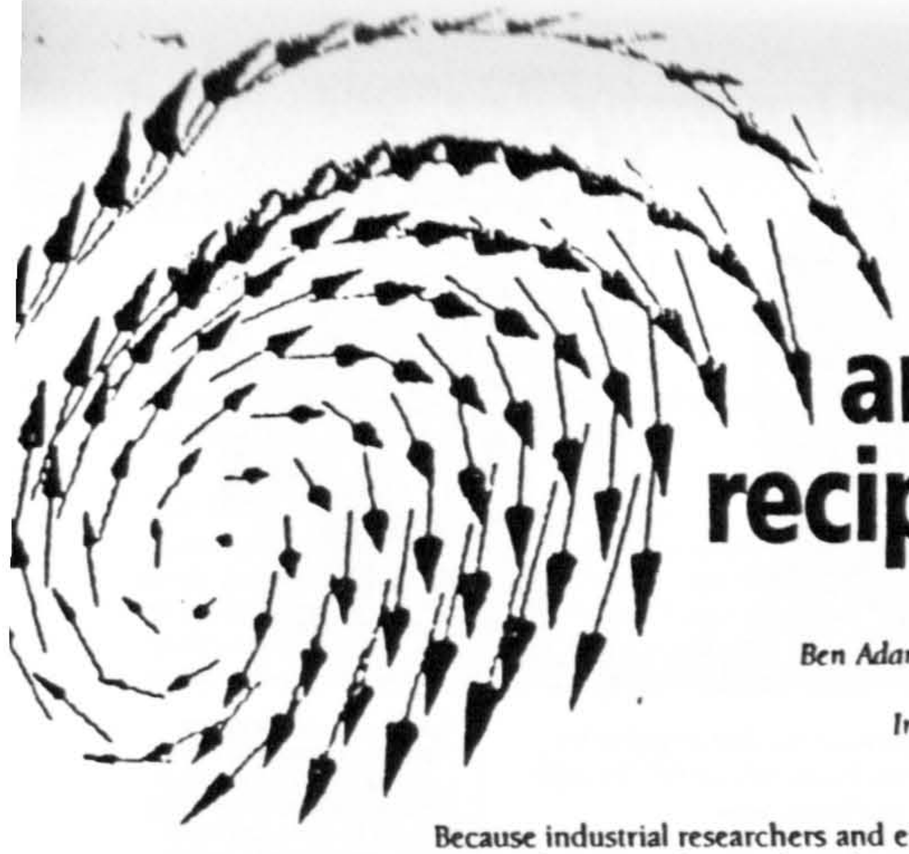


Figure 2. Comparison of measurements and computational results for top-dead-center compression in a disc chamber engine.

Best Copy
Available

Print bound close to the spine



Simulation of in-cylinder flow and combustion in reciprocating engines

Ben Adamson, David Gosman, Charles Hill, Chris Marooney, Behrooz Nasseri, Michael Sarantinos, Theodor Theodoropoulos, and Henry Weller
Imperial College of Science, Technology and Medicine, London, England

Because industrial researchers and engineers require precise solutions to their technical problems, computational fluid dynamics (CFD) methodologies must accommodate real flow geometries. In many cases, including that of the reciprocating internal combustion engine, these geometries are difficult to mesh with conventional regular grids, which are susceptible to distortion. Unstructured body-fitted meshes best address such cases because they provide the flexibility to fit boundary and flow features without introducing large amounts of non-orthogonality into the mesh. This meshing strategy is used in the SPEED (Simulation Programme for Engine Evaluation and Design) code developed at Imperial College.[1] The development of the SPEED code has involved not only the adaptation for use on unstructured meshes of the finite-volume discretization method (FVM) and the conjugate gradient (CG) solution method for matrix equations, but also techniques for creating the mesh, describing its continuity, moving and restructuring it in response to boundary and flow changes, and visualizing the four-dimensional property fields generated by the code. The applications of the code described here were run on the CRAY X-MP/28 computer system at the University of London Computer Centre.

Discretization, numerics, and modeling

The SPEED code uses the finite-volume method to discretize the CFD equations for mass, momentum, internal energy, turbulence, and the chemical species for ignition and combustion. The equations are solved in a fully implicit manner so that large time steps can be used. This implicitness is necessary to accommodate the large computational demands made by engine simulation codes. These demands exist due to two primary factors: (1) the variable sets for the models of turbulence, fuel-spray, droplet motion, and combustion chemistry, as well as the variable set for the basic aerodynamic flow, are large and will increase as more detailed models come into use, and (2) the unstructured mesh requires a database of addresses that describe its connectivity. This requirement, plus the larger supporting storage needed for efficient computation with unstructured meshes, also yields a large variable set. In combination, these factors create a substantial memory requirement, and as a result CPU time must be minimized to achieve an acceptable turnaround time.

The interactions among the various models are represented in a predictor-corrector manner after the PISO algorithm of Issa.[2] Corrector steps are solved neighbor-explicitly; that is, in the correction of a given node value, its neighbor's values are set to the latest values already available. For neighbor-explicit calculations, the algorithm is highly vectorizable because in an unstructured mesh the vector length equals the mesh size; the standard version of the code already is conditioned suitably for vector machines in this way. For parallel processing by subdomain decomposition, this part of the code also is well-conditioned for Cray Research hardware architectures, provided the neighbors on the subdomain boundary are duplicated in the data. The major architecture-conditioning challenge comes from the fully implicit predictor stages. The solution of the unstructured sparse matrix equations generated for these messages is best accomplished by a CG solver, which in principle is structurally neither vectorizable nor parallelizable. However, we have found that if the mesh node numbering is preconditioned such that the difference between neighbor numbers exceeds the vector length of the machine, then the CG solution algorithm can be substructured to yield a vectorizable inner loop, though the increased bandwidth increases the iterations to solution by about 20 percent. The technique is not available in parallel architectures, but research continues in this area.

For dispersed-phase calculations, Lagrangian equations of motion for a representative sample of droplets are used to model atomization, evaporation, breakup, and coalescence. Most of the Lagrangian code is independent of the mesh structure, and the extension to unstructured meshes involves defining a method of particle tracking. In the SPEED code, a local coordinate system is defined for each cell, on which a conventional search procedure can be used.

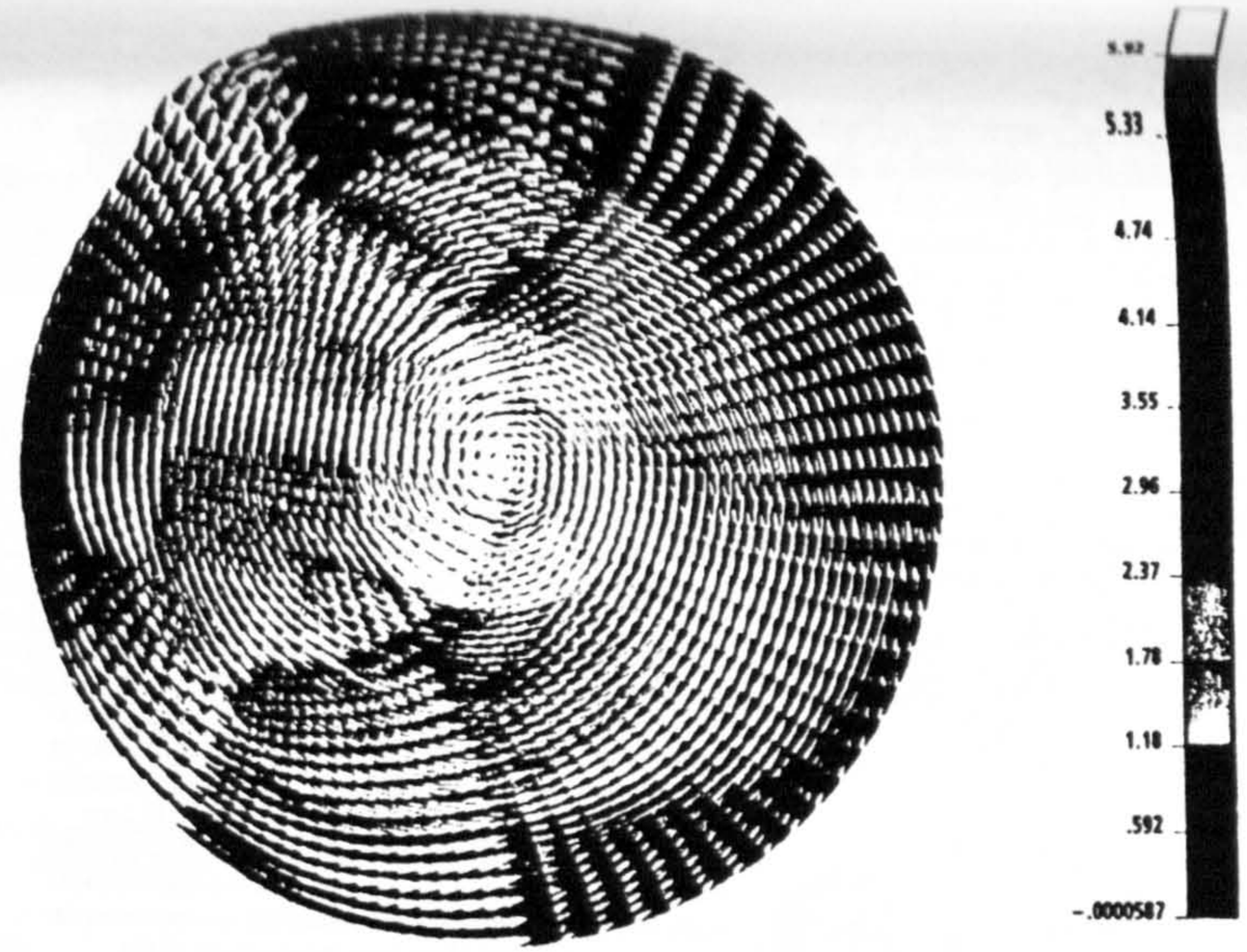


Figure 1. Velocity vectors showing calculated swirl velocity field at top-dead-center.

Sample engine results

The engine calculations described here illustrate the range of phenomena that can be calculated with the SPEED code and the visualization options available. Figure 1 shows the top-dead-center swirl flow in a disc chamber geometry. The vector plot display of velocity demonstrates the value of color in decoding the basic velocity magnitude information conveyed by arrow length. Comparisons of calculations with experimental results are shown in Figure 2. The significant improvement seen in the swirl velocity comparison on moving from a coarse mesh of 8000 cells to a fine mesh of 30,000 cells is expected from the properties of the upwind differencing technique used to model convection. A similar improvement in the turbulence intensity (not shown) is due to improved resolution of the generating flow gradients.

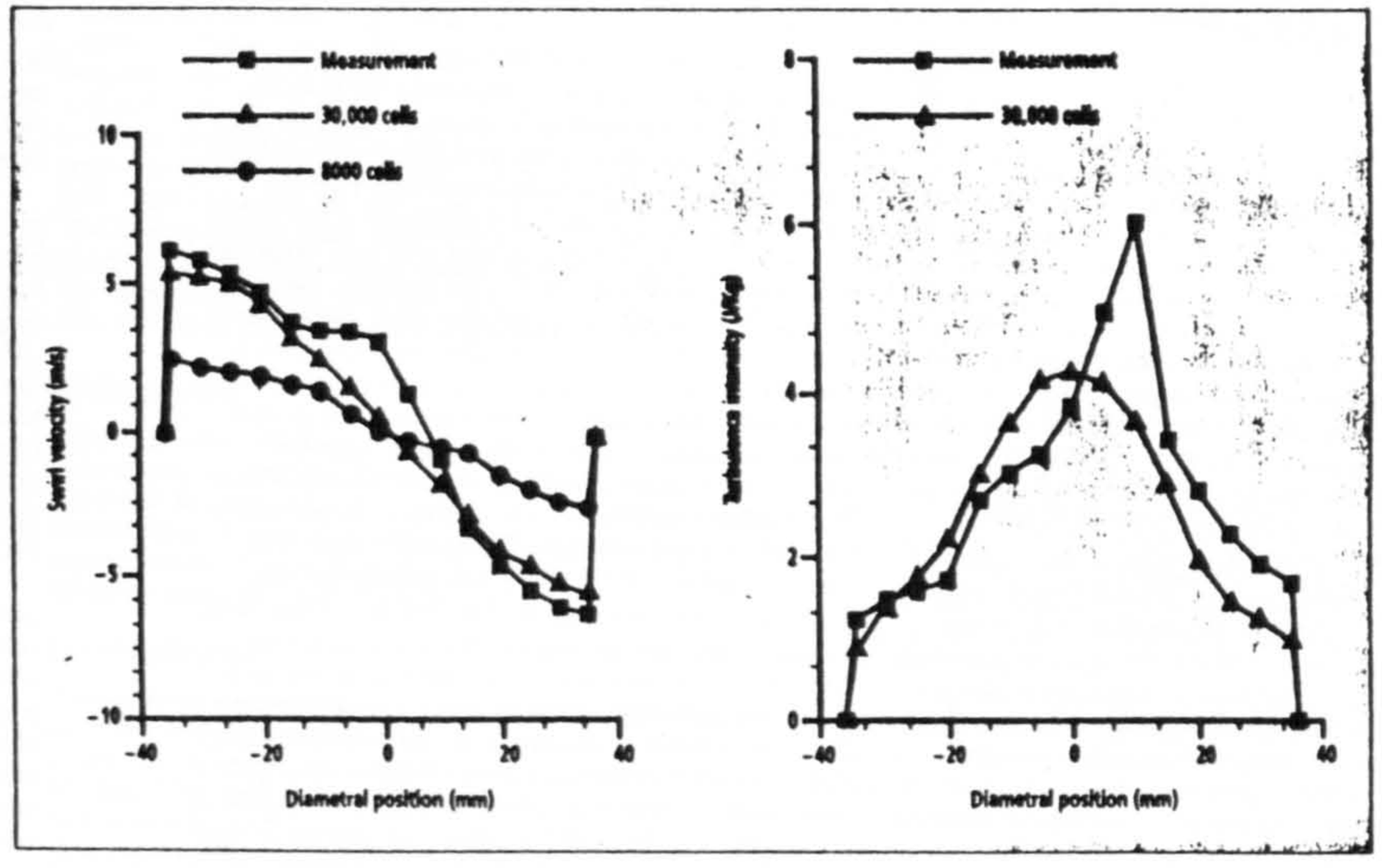
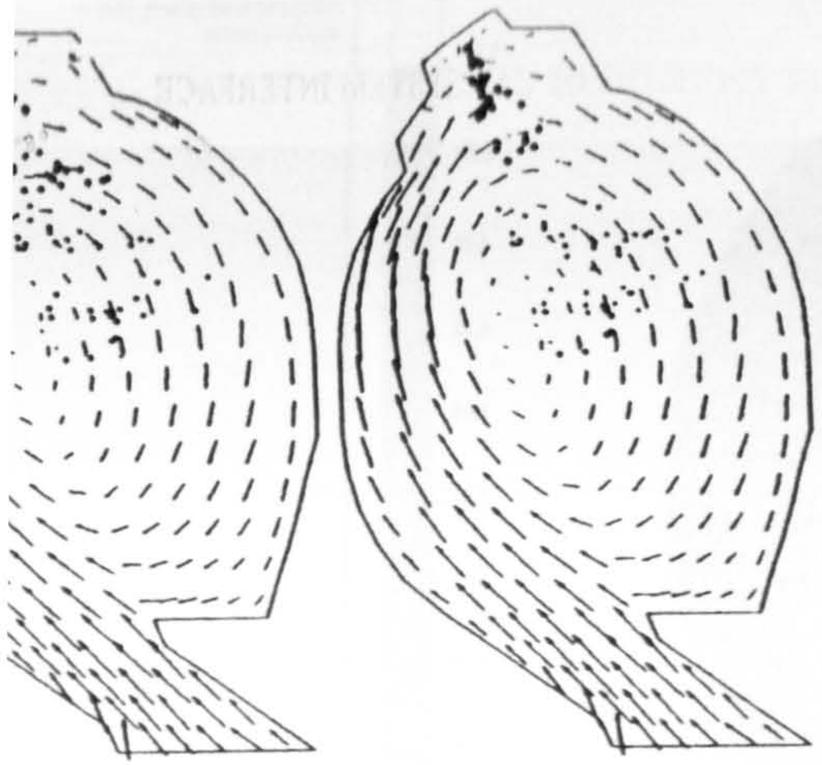
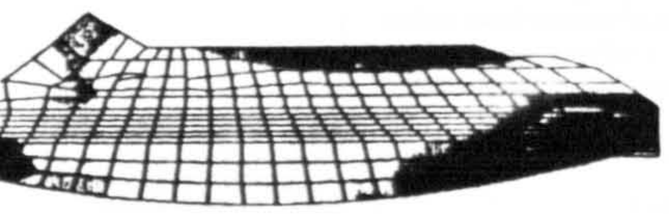
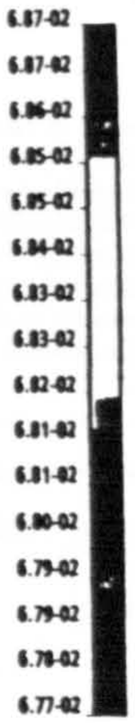


Figure 2. Comparison of measurements and computational results for top-dead-center compression in a disc chamber engine.



ection through
showing swirl
spray colored
rft) and drop





section through
park-ignition
onset of knock
uel concentration

Figure 3 illustrates the three-dimensional complexity that can be obtained for the aerodynamic flow fields. The figure shows one axial and two diametral sections through the bottom-dead-center velocity field in the bowl-in-piston geometry. These sections through the flow were selected to make comparisons with measurements for this engine[3,6] Evidently these sections do not exhaust the full complexity of the flow field, nor do they bring out changes over time. They thus point to the need for a more effective way to visualize the results.

Visualization of results

The plots of Figure 3 provide the basic information needed to evaluate the flow evolution. To understand how the flow fields evolved to this point, however, it is necessary to examine a temporal sequence of plots, using as many sections as required to map the features of the flow. Because such an evaluation can lead to the user drowning in a complexity of data, techniques are needed to package and summarize this information more directly, the most obvious one being animation. Once the problems of data storage (usually on the order of 10^5 points for each of 100-300 time steps) and automatic extraction of relevant data subsets are overcome, viewing the evolution of the flow dynamically is the most direct way to assimilate the code output.

Animated films have been made of the flow in the prechamber geometry, the evolution of a fuel spray injected into this flow, and the evolution of the flame front in a spark-ignition engine. The films were made from a sequence of frames taken directly from the calculations. An alternative method is to process the data to provide defined objects, which then are animated. This reduces storage by ensuring that only the objects of interest are stored and allows the animation package to improve image smoothness by appropriate time interpolation using the object definition.

A comparison of these films shows that the animated data are easy to comprehend for Eulerian scalar fields, such as fuel concentration, and for properly Lagrangian fields, such as the distribution of fuel spray droplets. However, a problem arises with Eulerian velocity fields, in that the fixed positions of the arrows do not represent the fluid motion very well. Although there is no simple solution to this problem, many alternative visualization techniques exist. For example, an Eulerian solution involves filling each cell with a striped pattern, such that the stripes are perpendicular to the direction of motion, and have a speed proportional to the flow speed. In this case it is unclear whether a minimum mesh resolution exists at which this display would be effective, and it is in any case a two-dimensional method only. An alternative is to adopt a Lagrangian approach and display particle trajectories. An advantage to this approach is that, with a suitably low density of particles and appropriate perspective, the time evolution could be shown in a static picture for the full three-dimensional domain. The disadvantage is that the particles misrepresent an Eulerian field by providing a nonuniform coverage, with particles accumulating in some places and depleting in others. A technique for providing generation and extinction of tracker particles also is required.

The fuel-spray calculations shown in Figure 3 are for a prechamber without a spark plug. The representation of the droplet flow, color coded for temperature in one view and for size in the other, reveals the correlation between the two parameters. The spray is imposed on a monochrome representation of the flow field.

Figure 4 shows the results of calculations for spark-ignited combustion in a Renault F1800A engine. Color fringe plots are used to visualize an Eulerian scalar field, which in this case is the fuel concentration in the unburnt gas. This field is important in the prediction of engine knock, whose onset is indicated by the depletion zones shown. Measurements suggest that these calculations agree with observations on the actual engine and are improvements over results obtained using only the ensemble-average fuel concentration, where knock is not predicted.[3,4]

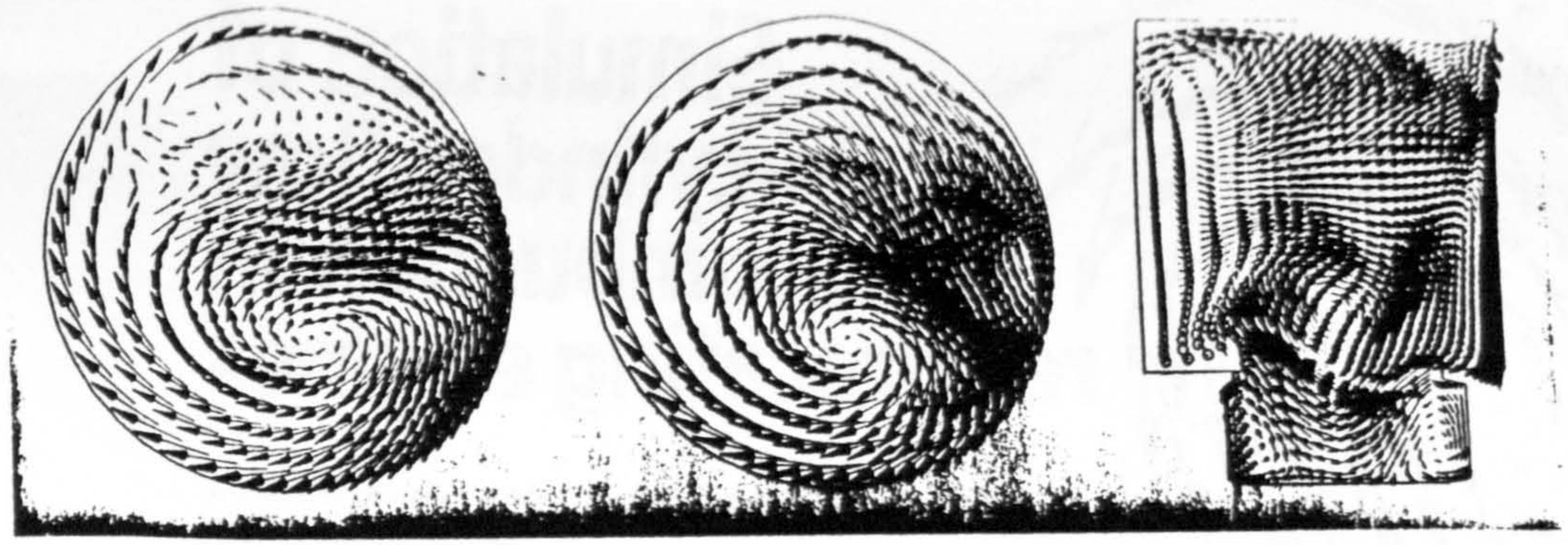


Figure 5. Sections through the flowfield at bottom-dead-center for the bowl-in-piston geometry. Shown are diametral planes 15 mm (left) and 35 mm (center) below the cylinder head and a vertical section through the symmetry plane of the engine (right).

Future developments

As the number and detail of interacting models proliferate, major emphasis will be placed on improvement of the solution algorithm for the interaction and on higher order discretization schemes to make the code more efficient. However, improvement of the output data evaluation process will remain a major area of development. The present state of development of the SPEED code relies heavily on existing standards in visualization, and it will have to be moved outside the existing capabilities of the PATRAN CAD system and of such commercially available systems generally. Although commercial CAD systems historically have been based in steady-state, often structural-mechanics based, applications, the industry increasingly is providing tools for time-dependent fluids applications. Nevertheless, the alternative of a tool-kit environment, which provides a standard graphics base on which the user can build the required applications, seems to be a more appropriately flexible way to explore visualization options, because the subjective worth of the end product is a major consideration and cannot always be evaluated prior to its realization.

Similar comments apply to mesh generation. Although we would prefer to use the advancing front option, in view of its flexibility in the body-fitting of meshes, commercial packages with the full range of visualization facilities including this option are not yet available. In addition, the problems of closure of advancing fronts are more severe for hexahedral meshes than for tetrahedral meshes. However, this option provides the most straightforward means of mesh reconstruction for dynamic local refinement, and, when more widely available, is likely to become the standard.

All of the visualization possibilities mentioned so far presume that a two-dimensional section of a three-dimensional field has to be observed as a whole. Alternative strategies based on viewing streamline trajectories as two-dimensional (space and time) objects also could provide a way of visualizing the flow. Here research is needed not only into the graphical representation options, but also into the way that the mind understands such data fields. We hope that the existing successful application of the SPEED code environment to engine problems will provide further impetus for the development of such methods, which will be applicable to a wide range of problems.

About the authors

The authors are faculty members in the fluids section of the Department of Mechanical Engineering at the Imperial College of Science, Technology, and Medicine in London.

Acknowledgments

The authors thank C. Vafidis for valuable discussions, C. Kralj for computing assistance, and the Direction des Etudes and the Direction de la Recherche of Renault for permission to use data from Reference 2, including Figure 4. The methodology embodied in the SPEED code was developed with the financial assistance of the U.K. Science and Engineering Research Council, the Commission of the European Community, and the Joint Research Committee of Austin-Rover, Fiat, PSA, Renault, Volkswagen, and Volvo.

References

1. Gosman, A. D. and C. J. Marooney, "Development and Validation of Computer Models of In-cylinder Flow and Combustion in Diesel and Spark-ignition Engines," C399/169, *IMechE*, Autotech '89, Birmingham, England, 1989.
2. Issa, R. I., "Solution of the Implicitly Discretized Fluid-flow Equations by Operator Splitting," *Journal of Computational Physics*, Vol. 62, pp 40-65, 1986.
3. Zellat, M., Régie Nationale des Usines Renault, private communication.
4. Gosman, A. D., C. J. Marooney, and H. G. Weller, "Prediction of Unburnt Gas Temperature in Multidimensional Engine Combustion Simulation," International Symposium on Diagnostics and Modelling of Combustion in Internal Combustion Engines (Comodia), Kyoto, Japan, 1990.
5. Adamson, B., A. D. Gosman, C. J. Marooney, B. Nasseri, and T. Theodoropoulos, "A New Unstructured-mesh Method for Flow Prediction in Internal Combustion Engines," International Symposium on Diagnostics and Modelling of Combustion in Internal Combustion Engines (Comodia), Kyoto, Japan, 1990.
6. Vafidis, C., G. Vorropoulos, J. Whitelaw, and E. Nino, "In-cylinder Flowfield in a Motored Engine Equipped with a Helical Port and with Re-entrant and Square Piston-bowl Configurations," Imperial College, Department of Mechanical Engineering, Fluids Section Report F5/87/41, 1987.

Interactive computational fluid dynamics tools for engine design

A D GOSMAN, PhD, DSc, C HILL, BSc, MDES, DIC, C J MAROONEY, BA, PhD
and M SARANTINOS, BSc
Department of Mechanical Engineering, Imperial College of Science, Technology and Medicine, London

SYNOPSIS Computational fluid dynamics (CFD) codes need an easy-to-use and flexible user interface, to compete with laboratory techniques for engine evaluation and design. Computer-aided design (CAD) systems provide the basic functions of mesh generation and field visualisation, but their present design precludes development of a comprehensive graphical user interface in which the CFD developers can apply their own algorithms where desired. We compare this with the emerging toolkit environment paradigm which promises much improved interactive techniques for engine design through its open modular design.

1. INTRODUCTION

The work of designing internal combustion engines has been enhanced in recent years by the emergence of realistic numerical simulation techniques for the evaluation and design of reacting and non-reacting flow processes. The engineer can now draw upon not only laboratory-based experimental facilities, but also CFD systems with capability to model increasingly realistic engine geometries. CFD offers greater predictability and reduced lead times in engine design through reduced reliance on expensive prototypes. It also promises greater insight in the design problem. As CFD systems become increasingly reliable and thus relied on in engine design, it becomes more important to provide easy-to-use CFD systems tailored to the particular needs of the design engineer. Indeed, the ease with which an engineer can use CFD will be a determining factor in its usefulness and its cost-effectiveness compared with conventional laboratory techniques.

However, whereas much effort has been concentrated on the development and implementation of CFD methodologies, less attention has so far been paid to the overall design of the engineer's computational tools. In this paper we are concerned with ways in which CFD 'software products', as distinct from raw methodology, may be designed for ease and flexibility of use. We are interested in the design of software to support interactions between the engineering designer and a CFD code: in short, the design and technology of the CFD user interface. This includes not only facilities which have been expressly designed, but all aspects of a system with which users interact.

The requirements for such software are exemplified by an engine design code called 'SPEED', which is being developed at Imperial College. This paper is an introduction to our research into suitable interactive support for SPEED.

In the present academic and industrial implementations of SPEED, a 'monolithic' CAD system is used, in conjunction with specially written software, to define simulation problems and to view results. Although this software provides a proven, portable, facility for engine design, the CAD systems used are restrictive in their capabilities and difficult to extend. They inhibit the development of facilities for data visualisation, database management and the creation of tailored CFD applications. In the following section SPEED is described, and in section 3 the present interactive software is described and discussed.

Further research is underway into more comprehensive interactive facilities for codes like SPEED. Software is being developed to provide a flexible application framework for engine design, based on the emerging paradigm of the toolkit environment. The work anticipates distributed computation using networks of computers of varying architectures including parallel computers and affordable high-speed graphics systems. The research is described, and the design contrasted with that of monolithic CAD systems. The paper concludes with some remarks concerning the development of both styles of software to serve engine designers more fully.

2. ENGINE DESIGN WITH SPEED

The SPEED code simulates three-dimensional transient gas flows and combustion in reciprocating engines. Many of the modelling and computational aspects of the code are already well documented [1, 2] and in this section we confine ourselves to a description in terms of the software required to support the engine design process.

The flow domain is represented by an unstructured body-fitted computational mesh of hexahedral (distorted cuboid) cells (figure 1). The mesh adapts to the motion of boundaries and may in the future allow adaptation to flow conditions, by being automatically refined in areas of high gradient. SPEED models a broad range of phenomena including the basic aerodynamic properties of the flow, chemical concentrations and fuel droplet distributions. These are represented mathematically as scalar and vector fields, except for fuel droplets which are presently represented as a particle population ('Monte Carlo field').

Future developments under consideration include changing SPEED's mesh to accommodate tetrahedral cells, possibly mixed with triangular prisms for reasons pertaining to mesh generation and parallel processing. Advances in the simulation of flames may require spectral fields to be incorporated and in the case of shock wave and flame simulations, discontinuous fields may occur in future modelling. Anticipating these changes demands a generalised approach to the design of the interactive software used with SPEED.

The design engineer undertakes a series of tasks, perhaps iteratively, to evaluate an engine using SPEED [3]. Here we describe these tasks in order to identify the established techniques available to accomplish them.



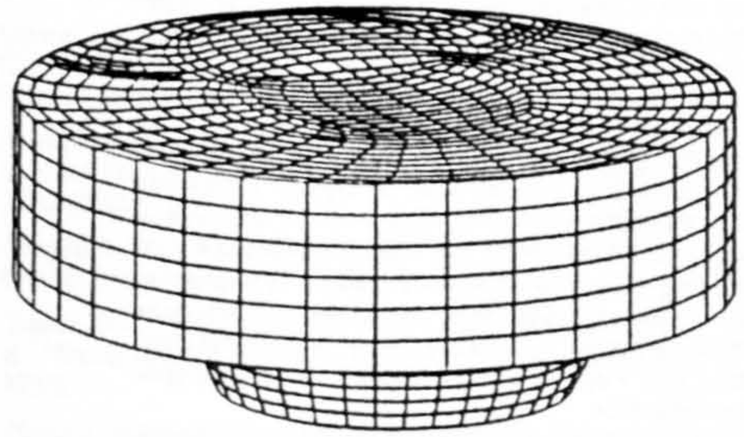


Figure 1 – A solid model of a combustion chamber and the resulting mesh.

2.1 Set-up

The engineer makes a computational mesh of the combustion chamber, labelling its physical and flow boundaries. Until recently meshes were generated using algebraic techniques, in a numerical code written by the design engineer. The need to model real engine configurations with unstructured meshes demands a different approach. One method is to generate a mesh based on an explicit model of the internal surface of the combustion chamber. The surface of the combustion chamber can be defined using solid modelling techniques [4]. Two appropriate techniques are 'analytical solid modelling' (ASM) [5] and 'trimmed surface patches' (TSP) [6]. The mesh may be generated by one of two methods: the 'composite grid' method [7], or the 'advancing front' technique as quoted in [8]. Once the mesh has been generated, initial and (possibly) time-dependent boundary conditions are defined, such as initial in-cylinder swirl, wall temperatures, and upstream manifold conditions. These may come from experimental or computational sources. More general control parameters, such as combustion model coefficients, fuel type, and valve lift profiles will also need to be inserted into the input data.

2.2 Simulation

After suitable checking, the mesh and other data are transferred to SPEED. Given the lengthy computation involved in simulations of real engine configurations, SPEED usually runs in a background process, often on a supercomputer remote from the user's workplace. There is minimal interaction with the analysis itself.

2.3 Visualisation

The result of a simulation is a set of physical fields defined over the computational domain for each of a number of timesteps, together with the meshes on which they are defined. The final task for the engineer is to explore and interpret this data using computer graphics techniques, viewing time-dependent information by animating views of 3D fields. Global parameters such as cylinder averages are also assessed, typically plotted against time. The computational results may be compared with laboratory-based experimental data. This process is 'visualisation' [9]. Many different techniques must be used to explore and visualise fields: see for example [10].

3. DEVELOPMENT OF CAD SYSTEM INTERFACE

The interactive software currently in use with SPEED was developed to serve the immediate needs of several industrial users with widely differing computing resources and applications, exploiting CAD systems already in use in the users' companies. This software has now been maintained through many releases of SPEED.

3.1 Design

The use of CAD systems was restricted to those tasks which would be most expensive to develop in-house: surface geometry definition, mesh generation, and results visualisation. This enabled custom software development efforts to address those aspects of the user interface which are special to the SPEED code, and ought therefore to be portable between different CAD systems. Hence all other numerical tasks and data format translation are performed by specially written programs which support the CAD system file formats (figure 2).

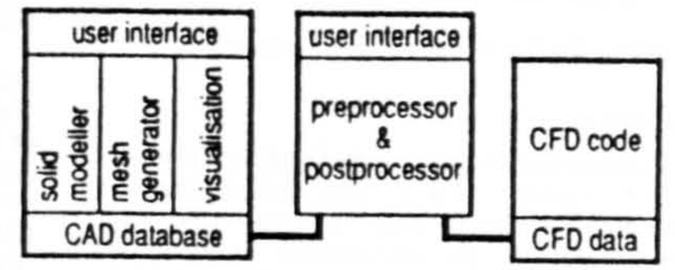


Figure 2 – The CAD system interface to SPEED.

The surface geometry is generated in the CAD system, using ASM or TSP techniques. Meshes are usually generated using the composite grid method. The mesh must have a layered structure in the stroked region to facilitate mesh motion. The distribution of cells can be controlled by local mesh refinement and through the chosen composite grid scheme, and the capabilities of the CAD system are used to label distinct aspects of the mesh such as physical and reentrant boundaries. The mesh is written to a file.

Other data for the simulation are defined in a 'data file', using a text editor. Boundary fields, when not constant in space, are defined in terms of local coordinate systems; simple representations of initial conditions such as solid body rotations around user-defined axes are also possible. Graphs are defined in terms of data tables.

Both files are read by a 'preprocessor' code which generates additional mesh addressing data needed for efficient running. This code also interrogates the quality of the mesh in terms of cell aspect ratio, orthogonality and warpage, since this may affect the time taken for the simulation to converge. Initial and boundary conditions are inserted in the data file. The main boundary conditions represented are time- and space-dependent fields on physical and reentrant boundaries, such as velocity profiles at the valve inlet, upstream manifold conditions, and wall temperature. These fields will be interpolated to mesh resolution in SPEED. The data are checked for correctness and output to a single file in SPEED input format.

After the simulation, a 'postprocessor' code correspondingly translates the results from the CFD format to that of the CAD system. Scalar fields are typically represented as distributions of colour within the mesh; vector fields as distributions of arrows (figure 3). These representations may be animated by real-time 'flip-book' display of multiple time-steps.

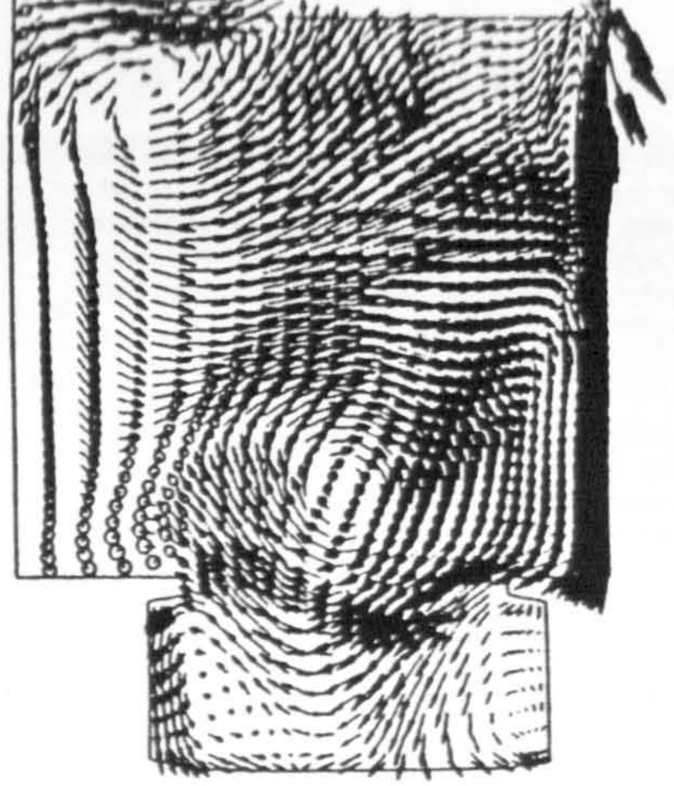


Figure 3 – Velocity field in a 'lipped bowl' engine.

3.2 Implementation

The present implementation uses the PATRAN and IDEAS-SUPERTAB CAD systems, which have predominantly command-line user interfaces. With PATRAN, command macros can be generated by the postprocessor to customise the command line for visualisation work. The pre- and postprocessors, like SPEED, are written in FORTRAN77 and share SPEED's global variable space (FORTRAN 'common blocks'). The significance of this will be discussed in section 4.4. Both programs have simple command-line user interfaces which emulate PATRAN's commands. The pre-processor's data file is based on a simple keyword language.

3.3 Evaluation

The SPEED system provides a proven, portable facility for engine evaluation and design. Where users already have an integrated CAE process, the existing CAD system may be readily exploited for CFD work. The CAD systems supported offer comprehensive product support from their suppliers, and a reusable geometry database for transfer to the structural analysis and other arenas. A practice has evolved of generating meshes by writing CAD system command language files, which can be debugged and run to create the mesh. This enables meshes to be defined parametrically, so that a fine mesh can easily be derived from the command file of a coarse one. It also enables very compact storage of mesh descriptions.

The CAD systems are however expensive in terms of licensing, maintenance and training, yet of limited suitability to the SPEED system in both functional and operational terms. Mesh generation by an explicit decomposition of the flow domain becomes very difficult with complex surface geometries, where the advancing front option might be more suitable. Visualisation facilities are quite basic, and difficult for the CFD developers to extend. There is a growing need to visualise specialised phenomena such as the evolving flame surface during combustion, and to experiment with new visualisation techniques. It would also be desirable to enter or at least visualise boundary conditions in the same interactive graphical representation as that used for mesh generation. The visualisation process may also obscure extrema and gradients in the simulation code's output data. SPEED's parameter fields are evaluated as cell averages, whereas many visualisation techniques require fields evaluated as point intensities (for example so the data is smoothed prior to visualisation. A symbolic display of the original extrema correctly located in the viewing space would be a useful adjunct to such visualisations. Also, temporal and spatial interpolation algorithms will vary from one CFD code to another. Hence CFD developers may wish to impose their preferred algorithms and techniques for some aspects of visualisation.

SPEED is being continually upgraded to incorporate improved algorithms, as these are developed or become computationally feasible. The types and representations of the phenomena modelled thus not only vary between codes but also evolve with a single code. This requires the interactive software to be adaptable to a changing code system including changes in the types and formats of data files, and changes in the parameters set by the user, without imposing excessive maintenance work on the CFD developers. The CAD developers have however been slow to adopt advances in database techniques, and the need to support a two-way *ad hoc* data translation interface has incurred the cost of developing and maintaining multiple versions of data formats, leading to unreliable software. Since CFD data has a hierarchical structure, data selection in the user interface might best be supported by an underlying hierarchical database system. This would also decouple the simulation code from its user interface.

The CAD systems are also difficult to use, requiring expensive operator training, and being time-consuming in use. To overcome these problems we perceive a need to extend to the CFD sphere the principles of the event driven, direct manipulation [11], graphical user interface (GUI) [12]. This work was pioneered by XEROX in the 1970's [13] and is perhaps best known through the APPLE MACINTOSH and its 'desktop' metaphor [14]. GUIs represent components of a software system as graphical elements ('objects') within a display. 'Directly manipulating' the graphical objects by means of a pointing device such as a mouse causes the software to operate. GUIs make a computer system easier to learn and use. Computer networks can now be fully exploited in a GUI, via the X WINDOW SYSTEM [15]. This enables both access from the interactive software to remote computers (for instance a supercomputer), and remote access to the interactive software from other computers (e.g. low-cost graphics terminals and workstations).

The SPEED developers need to provide applications tailored to different types of simulation: cold-flow with or without fuel sprays, and homogeneous, inhomogeneous and diesel combustion. The present CAD systems do not allow such custom packaging of software. It would help developers and users to be able to assemble applications from ready-made software components without recourse to a programming language.

Finally, it is becoming essential to exploit parallel processing computers to achieve accurate simulations of increasingly realistic engine configurations. This applies equally to interactive software involving heavy computational load, such as 3D geometry analysis and high-quality image rendering. The existing system does not support parallel processing.

4. DEVELOPMENT OF AN INTERACTIVE TOOLKIT FOR ENGINE DESIGN

Alternative approaches to the design of interactive software for CFD are possible in which monolithic CAD systems play a smaller role. In this section we describe research aimed at the development of some new software for the SPEED code, with a view to demonstrating better computational methods for engine design. Our initial work concentrates on the generic user interface and support for visualisation, but the long-term intention is to accommodate the entire CFD process.

The ease with which the developers can implement interactive facilities directly affects the quality and

CFD developers may rapidly implement interactive software appropriate to their end-users' needs.

4.1 Applying GUI Technology to CFD

A GUI for SPEED must of course include facilities for creating, viewing and manipulating 3D geometric entities. Such a GUI is a complex system, tightly coupled through graphics and operating system libraries to the machines on which it operates. It is unreasonable to expect CFD developers to undertake low-level coding. The software developer needs ready-made user interface components such as windows, menus, buttons, dials, and file browsers, as well as geometric primitives, from which to construct a consistent GUI.

4.2 The Toolkit Environment

The experiences with CAD systems emphasise a need to frame the diverse technologies of the user interface in some unifying paradigm. At the heart of the problem is the need to make distinct parts of the software independent of each other, so that an incremental approach can be taken to their development and maintenance. We invoke the principle of 'modularity' [16]. A 'module' is a program component written in such a way that its output parameters are defined solely as a function of its input parameters (there are no hidden 'side-effects' to a module's execution). Its virtue lies in being self-contained, defined independently of other modules. It is thus easier to debug, and new versions of a module can be created independently of others. This aids software reuse, increases reliability, and reduces maintenance work. Further, modules can be used as the building blocks of a higher-level program. This can be achieved by calling modules as subroutines, or by assigning the output data of some modules to be the inputs of others, with each executed as a separate process. Several such interconnected modules make a higher-level modular unit, as it too will be defined solely in terms of its inputs and outputs.

A modular approach leads away from the design principles of monolithic CAD systems, towards the more recent notion of 'toolkit environments'. An 'environment' manages the assembly of modular 'tools' into applications, and the execution of such applications (figure 4). A tool is defined here as a module which has 'ports' defined for its input and output data, allowing the data produced by one module to be sent directly to the input ports of others (figure 5). Each port is of a defined data type, such that only ports of the same type may be connected. Graphical user interface components representing module parameters (numbers as dials, choices as menus, filenames as file browsers, etc.) are obtained through port connections to the environment. A set of interconnected modules forms an application.

4.3 Design

The CAD system and pre- and postprocessing codes as they now exist would be replaced in the proposed scheme by an environment and a kit of tools, whose primary role would be to provide CFD codes (such as SPEED) with a direct manipulation GUI with 3D visualisation facilities. Such a design is intended to encourage a process of 'cut-and-try' visualisation, with the notion of providing a 'computational laboratory' in which to perform engine evaluations.

Appreciation of the need for a toolkit environment, which originally arose from the experiences with SPEED, has been enhanced by the emergence of a ready-made environment called the APPLICATION VISUALISATION SYSTEM (AVS) [17, 18, 19, 20]. Although designed primarily for visualisation, AVS offers many of the generic features prescribed in our design study. AVS provides an excellent visual programming methodology for assembling modular tools, by manipulating a dataflow diagram; and good supporting facilities for a GUI and 3D geometry viewing and

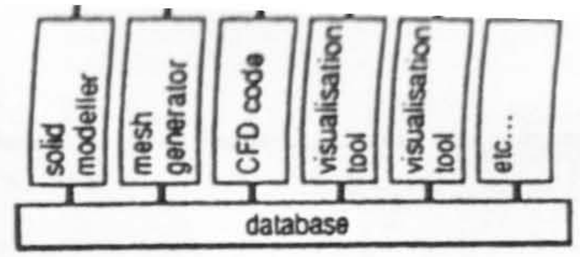


Figure 4 – A toolkit environment for CFD.

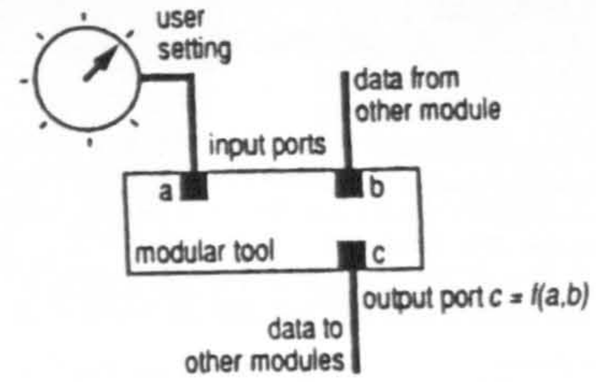


Figure 5 – A tool with two input ports and one output port.

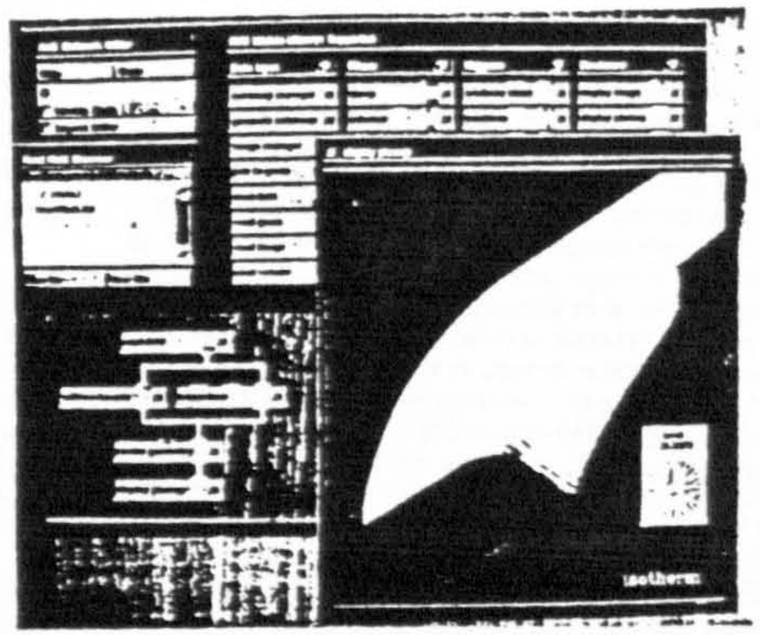


Figure 6 – A screen from AVS.

manipulation (figure 6).

This has made possible a concentration of effort on the other requirements for a complete interactive toolkit for engine design. Firstly, a hierarchical database provides access to distinct parts of CFD data (e.g. the list of points in a mesh), and manages their storage in terms of filing system location, and file formats and versions. The database is supported by a 'dictionary' which enables users to select data by comprehensible names, and also enables the CFD programmer to select sets of data for animation. Secondly, a library has been developed whose programming interface gives the CFD developer access to techniques for processing mesh and field data. These include operations involving a knowledge of the CFD simulation algorithms such as interpolation and field algebra, and operations involving graphical interpretation of the data, such as analytical geometry for visualisation programming. Programs written using this library can easily be installed as tools in AVS.

Parallel processing for both shared and distributed memory architectures is anticipated in the library's design (although not yet implemented) in a fine-grain, multiple-instruction multiple-data scheme as already described for SPEED itself [22]. The cells of a mesh are distributed by a master processor to several parallel slave processors, on the basis of minimum communication between processors. An

asynchronously. A complete program invocation consists of a sequence of master processes.

4.4 Implementation

A prototype system is being developed based on AVS, in combination with UNIX, the X WINDOW SYSTEM, and POSTSCRIPT. The database and field manipulation software are being implemented as a class library written in C++ [21]. The use of FORTRAN in the present pre- and processors (as mentioned in section 3.2) has made it difficult to maintain code quality, and has restricted programming style. Use of C++ enables a more rigorous methodology to be applied, while simplifying the library programming interface.

Because of the very large size of CFD data, the software must automatically manage in-core memory usage. Memory management is being implemented in the container classes of the library (such as lists), and can be automatic or application-controlled.

4.5 Evaluation

Apart from commercial considerations like current availability, the design of AVS is generally validated by the present research, and initial impressions are that the functional and operational problems of the CAD system interface are overcome. One issue is raised: if the toolkit approach proves to be useful, we foresee a need for standardisation in the interface between tools and environments. In principle a set of tools might be ported to several environments. A toolkit environment developed by a third party software developer would make broad porting of a single environment more likely.

The main omission of our research is mesh generation in a toolkit environment. Given a surface geometry describing a flow domain, mesh generation could be undertaken using an implicit solution method such as the advancing front technique, which is desirable anyway for complex engine configurations. A mesh generation tool could then be installed in the environment.

This leaves the problem of geometry definition. Presently available solid modellers tend towards two extremes: integrated CAE systems, and imprecise modellers for applications like video animation. For our purposes a simple but precise modeller is needed, limited to the specific task of engineering geometry definition. Such a modeller might be installed in the environment as another tool, its user interface and data types integrated with the toolkit environment. A complete CFD system would thus be created with a single user interface and data environment, but separated from other CAE activities.

Alternatively, in a hybrid scheme, translators between the environment and a CAD system could be installed in the environment as tools, enabling surface and mesh descriptions as well as simulation results to pass each way. The translators would be useful to a larger number of users than ones specific to SPEED, and thus more viable to develop. There is still no fully satisfactory standard for data exchange. Possibly STEP and its associated 'information modelling language' EXPRESS [23], a developing ISO standard (and replacement for IGES), will provide a suitable base technology in this area. In the extreme, the role of the CAE system could be reduced to data transfer between different application environments.

Meanwhile, conventional CAD vendors are now developing better programming features (based on a dedicated high-level language), and specific support for CFD. It may be possible with such features to retain the conventional arrangement of cooperating CAD and CFD systems and improve the interactive facilities. Three issues persist. Any code written in the dedicated CAD system language will lock the CFD developer into the CAD product; a large CAD system interface will still be necessary; and CFD developers will probably remain restricted in their

visualisation within the CAD system. CAD systems may of course move towards the open modular design of toolkit environments. The great strength of CAD systems is their ability to integrate multi-disciplinary CAE, and they have the advantage of a large number of existing users. However, the current complexity and expense of such systems appears to limit their present suitability for CFD.

5. CONCLUDING REMARKS

We have explored two paradigms for the design of interactive software for CFD: a two-way interface between CAD and CFD systems; and an alternative based on a toolkit environment with an open modular design. The toolkit approach has been shown to be feasible, promising to unify and greatly improve the user interface of the CFD code, and thereby create a more explorative and flexible environment for CFD. By contrast, CAD systems may evolve to offer appropriate interactive facilities for CFD, and therefore fully incorporate CFD within the established CAE arena. In our view they will only do this if they offer more comprehensive facilities for application, database, and user interface management. Their designers must recognise the precept that users need to implement their own algorithms in a programming language of their choice for many aspects of the CFD user interface. The toolkit approach, which we have tested through use of the AVS visualisation system, is clearly desirable from a user's perspective. Its potential may best be realised if geometry modelling tools are included, either within the environment, or through a well engineered interface to a conventional CAD system.

Acknowledgements

The contributions of B.N. Adamson and H.G. Weller are gratefully acknowledged. We would also like to acknowledge funding by the Joint Research Committee (a consortium of Rover, Peugeot, Fiat, Renault, Volkswagen and Volvo), the European Commission and Hewlett Packard, and the loan of computing equipment by the Rutherford Appleton Laboratory.

References

(1) Gosman, A. D. and Marooney, C. J., "Development and Validation of Computer Models of In-cylinder Flow and Combustion in Diesel and Spark Ignition Engines". *Autotech 89*, Birmingham, England, 1989 (IMechE C399/169)

(2) Adamson, B., Gosman, A. D., Marooney, C. J., Nasseri, B., Theodoropoulos, T., "A New Method for Flow Prediction in Internal Combustion Engines". *Comodia 90*, Kyoto, Japan, 1990

(3) Adamson, B., Gosman, A. D., Hill, C., Marooney, C. J., Nasseri, B., Sarantinos, M., Theodoropoulos, T., and Weller, H. "Simulation of in-cylinder flow and combustion in reciprocating engines", *Science and Engineering on Supercomputers*, October 1990, Ed. Pitcher, E. J. (Computational Mechanics Publications & Springer Verlag)

(4) Riquicha, A. and Voelcker, H., "Solid Modeling: A Historical Summary and Contemporary Assessment". *Computer Graphics and Applications*, 1982, 2, 2, 9-24 (IEEE)

(5) Casale, M.S., Stanton, E.L., "An overview of Analytic Solid Modeling". *Computer Graphics and Applications*, Feb. 1985 (IEEE)

(6) Casale, M.S., "Free-Form Solid Modeling with Trimmed Surface Patches". *Computer Graphics and Applications*, Jan 1987 (IEEE)

(7) Thompson J.F., Warsi Z.U.A, Mastin C.W., "Numerical Grid Generation, Foundations and Applications". 1985 (North-Holland)

(8) Lohner, R., "Fast Graphics Algorithms Using Unstructured Grids", SIGGRAPH 90, course notes 27, Chapter 7, 1990 (ACM SIGGRAPH)

(9) McCormick, B., DeFanti, T., Brown, M., "Visualisation in Scientific Computing". *Computer Graphics special issue*, 1987, 21, 6 (ACM SIGGRAPH)

(10) Upson, C., "The Visualisation of Volumetric Data". Computer Graphics 89, London, 29-38, 1989 (Blenheim Online Publications, Middx. UK)

(11) Schneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages". *Computer*, August 1983 (IEEE)

(12) Verplank, W., "Designing Graphical User Interfaces". *CHI 88 tutorial notes*, 1988 (ACM SIGCHI)

(13) Smith, D. C., Harslem E. F., Irby, C. H., Kimball, R. B., "Designing the Star User Interface". *Byte*, April 1982.

(14) *Human Interface Guidelines: The Apple Desktop Interface*, 1987 (Addison-Wesley)

(15) O'Reilly, T., Quercia, V., Lamb, L., *X Window System User's Guide for version 11*, 1988, (O'Reilly and Associates, Inc.)

(16) Parnas, D., "On the criteria to be used in decomposing systems into modules". *Commun. ACM*, 1972, 15, 12, 1055-1058 (ACM)

(17) Upson, C., Faulhaber, T. Jnr., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., van Dam, Andries, "The Application Visualization System: A Computational Environment for Scientific Visualisation". *Computer Graphics and Applications*, July 1989, 30-42 (IEEE)

(18) Van de Wettering, M., "The Application Visualization System – AVS 2.0", *Pixel* July/August 1990 (Pixel Communications, Inc., Watsonville CA.)

(19) Gelberg, L., Kamins, D., Parket, D., Sacks, J., "Visualization Techniques for Structured and Unstructured Scientific Data". *SIGGRAPH 90, course notes 27*, 1990 (ACM SIGGRAPH)

(20) *AVS Tutorial Guide (version 3.0 Beta Test)*, November 1990 (Stardent Computer)

(21) Stroupstrup, B. *The C++ Programming Language*, 1987 (Addison-Wesley)

(22) Weller, H., and Marooney, C. J., "Parallelism and CFD", *EASE 90 Symposium*, April 1990, UMIST, Manchester, England.

(23) International Standards Organisation *Information Modelling Language Express, Language Reference Manual*, May 1989 (ISO TC184/SC4/WG1 - N362)

BIBL LONDON UNIV

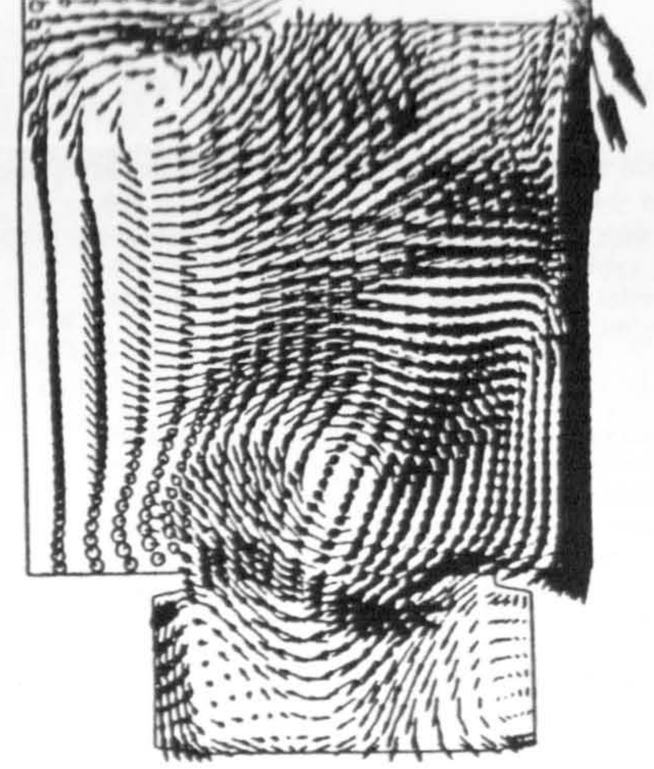


Figure 3 – Velocity field in a 'lipped bowl' engine.

3.2 Implementation

The present implementation uses the PATRAN and IDEAS-SUPERTAB CAD systems, which have predominantly command-line user interfaces. With PATRAN, command macros can be generated by the postprocessor to customise the command line for visualisation work. The pre- and postprocessors, like SPEED, are written in FORTRAN77 and share SPEED's global variable space (FORTRAN 'common blocks'). The significance of this will be discussed in section 4.4. Both programs have simple command-line user interfaces which emulate PATRAN's commands. The pre-processor's data file is based on a simple keyword language.

3.3 Evaluation

The SPEED system provides a proven, portable facility for engine evaluation and design. Where users already have an integrated CAE process, the existing CAD system may be readily exploited for CFD work. The CAD systems supported offer comprehensive product support from their suppliers, and a reusable geometry database for transfer to the structural analysis and other arenas. A practice has evolved of generating meshes by writing CAD system command language files, which can be debugged and run to create the mesh. This enables meshes to be defined parametrically, so that a fine mesh can easily be derived from the command file of a coarse one. It also enables very compact storage of mesh descriptions.

The CAD systems are however expensive in terms of licensing, maintenance and training, yet of limited suitability to the SPEED system in both functional and operational terms. Mesh generation by an explicit decomposition of the flow domain becomes very difficult with complex surface geometries, where the advancing front option might be more suitable. Visualisation facilities are quite basic, and difficult for the CFD developers to extend. There is a growing need to visualise specialised phenomena such as the evolving flame surface during combustion, and to experiment with new visualisation techniques. It would also be desirable to enter or at least visualise boundary conditions in the same interactive graphical representation as that used for mesh generation. The visualisation process may also obscure extrema and gradients in the simulation code's output data. SPEED's parameter fields are evaluated as cell averages, whereas many visualisation techniques require fields evaluated as point intensities (for example so the data is smoothed prior to visualisation. A symbolic display of the original extrema correctly located in the viewing space would be a useful adjunct to such visualisations. Also, temporal and spatial interpolation algorithms will vary from one CFD code to another. Hence CFD developers may wish to impose their preferred algorithms and techniques for some aspects of visualisation.

SPEED is being continually upgraded to incorporate improved algorithms, as these are developed or become computationally feasible. The types and representations of the phenomena modelled thus not only vary between codes but also evolve with a single code. This requires the interactive software to be adaptable to a changing code system including changes in the types and formats of data files, and changes in the parameters set by the user, without imposing excessive maintenance work on the CFD developers. The CAD developers have however been slow to adopt advances in database techniques, and the need to support a two-way *ad hoc* data translation interface has incurred the cost of developing and maintaining multiple versions of data formats, leading to unreliable software. Since CFD data has a hierarchical structure, data selection in the user interface might best be supported by an underlying hierarchical database system. This would also decouple the simulation code from its user interface.

The CAD systems are also difficult to use, requiring expensive operator training, and being time-consuming in use. To overcome these problems we perceive a need to extend to the CFD sphere the principles of the event driven, direct manipulation [11], graphical user interface (GUI) [12]. This work was pioneered by XEROX in the 1970's [13] and is perhaps best known through the APPLE MACINTOSH and its 'desktop' metaphor [14]. GUIs represent components of a software system as graphical elements ('objects') within a display. 'Directly manipulating' the graphical objects by means of a pointing device such as a mouse causes the software to operate. GUIs make a computer system easier to learn and use. Computer networks can now be fully exploited in a GUI, via the X WINDOW SYSTEM [15]. This enables both access from the interactive software to remote computers (for instance a supercomputer), and remote access to the interactive software from other computers (e.g. low-cost graphics terminals and workstations).

The SPEED developers need to provide applications tailored to different types of simulation: cold-flow with or without fuel sprays, and homogeneous, inhomogeneous and diesel combustion. The present CAD systems do not allow such custom packaging of software. It would help developers and users to be able to assemble applications from ready-made software components without recourse to a programming language.

Finally, it is becoming essential to exploit parallel processing computers to achieve accurate simulations of increasingly realistic engine configurations. This applies equally to interactive software involving heavy computational load, such as 3D geometry analysis and high-quality image rendering. The existing system does not support parallel processing.

4. DEVELOPMENT OF AN INTERACTIVE TOOLKIT FOR ENGINE DESIGN

Alternative approaches to the design of interactive software for CFD are possible in which monolithic CAD systems play a smaller role. In this section we describe research aimed at the development of some new software for the SPEED code, with a view to demonstrating better computational methods for engine design. Our initial work concentrates on the generic user interface and support for visualisation, but the long-term intention is to accommodate the entire CFD process.

The ease with which the developers can implement interactive facilities directly affects the quality and the present research is thus to develop methods by which CFD developers may rapidly implement interactive software appropriate to their end-users' needs.

4.1 Applying GUI Technology to CFD

A GUI for SPEED must of course include facilities for creating, viewing and manipulating 3D geometric entities. Such a GUI is a complex system, tightly coupled through graphics and operating system libraries to the machines on which it operates. It is unreasonable to expect CFD developers to undertake low-level coding. The software developer needs ready-made user interface components such as windows, menus, buttons, dials, and file browsers, as well as geometric primitives, from which to construct a consistent GUI.

4.2 The Toolkit Environment

The experiences with CAD systems emphasise a need to frame the diverse technologies of the user interface in some unifying paradigm. At the heart of the problem is the need to make distinct parts of the software independent of each other, so that an incremental approach can be taken to their development and maintenance. We invoke the principle of 'modularity' [16]. A 'module' is a program component written in such a way that its output parameters are defined solely as a function of its input parameters (there are no hidden 'side-effects' to a module's execution). Its virtue lies in being self-contained, defined independently of other modules. It is thus easier to debug, and new versions of a module can be created independently of others. This aids software reuse, increases reliability, and reduces maintenance work. Further, modules can be used as the building blocks of a higher-level program. This can be achieved by calling modules as subroutines, or by assigning the output data of some modules to be the inputs of others, with each executed as a separate process. Several such interconnected modules make a higher-level modular unit, as it too will be defined solely in terms of its inputs and outputs.

A modular approach leads away from the design principles of monolithic CAD systems, towards the more recent notion of 'toolkit environments'. An 'environment' manages the assembly of modular 'tools' into applications, and the execution of such applications (figure 4). A tool is defined here as a module which has 'ports' defined for its input and output data, allowing the data produced by one module to be sent directly to the input ports of others (figure 5). Each port is of a defined data type, such that only ports of the same type may be connected. Graphical user interface components representing module parameters (numbers as dials, choices as menus, filenames as file browsers, etc.) are obtained through port connections to the environment. A set of interconnected modules forms an application.

4.3 Design

The CAD system and pre- and postprocessing codes as they now exist would be replaced in the proposed scheme by an environment and a kit of tools, whose primary role would be to provide CFD codes (such as SPEED) with a direct manipulation GUI with 3D visualisation facilities. Such a design is intended to encourage a process of 'cut-and-try' visualisation, with the notion of providing a 'computational laboratory' in which to perform engine evaluations.

Appreciation of the need for a toolkit environment, which originally arose from the experiences with SPEED, has been enhanced by the emergence of a ready-made environment called the APPLICATION VISUALISATION SYSTEM (AVS) [17, 18, 19, 20]. Although designed primarily for visualisation, AVS offers many of the generic features prescribed in our design study. AVS provides an excellent visual programming methodology for assembling modular tools, by manipulating a dataflow diagram; and good supporting facilities for a GUI and 3D geometry viewing and

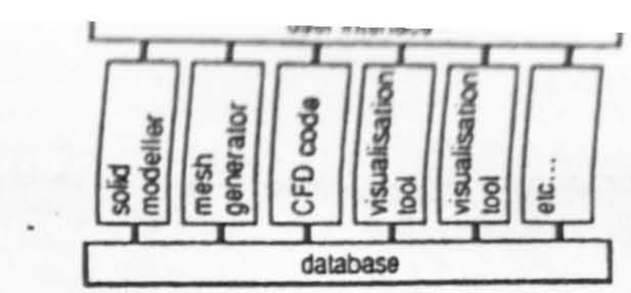


Figure 4 – A toolkit environment for CFD.

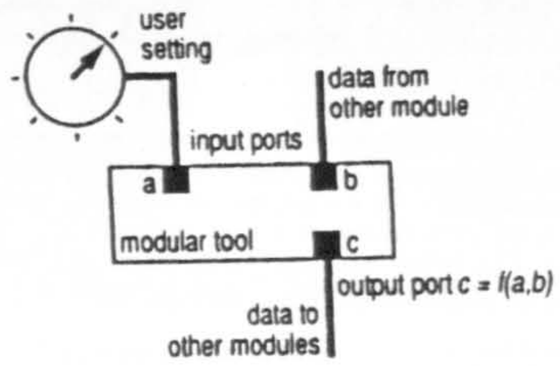


Figure 5 – A tool with two input ports and one output port.

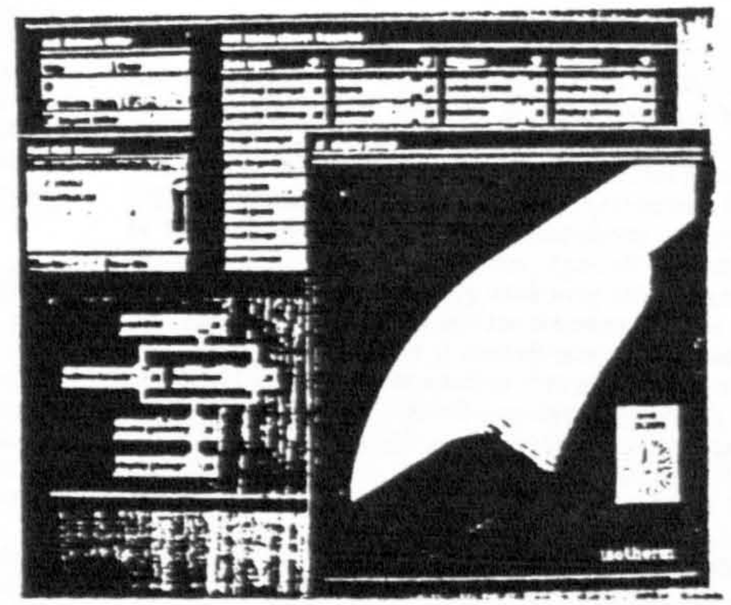


Figure 6 – A screen from AVS.

manipulation (figure 6).

This has made possible a concentration of effort on the other requirements for a complete interactive toolkit for engine design. Firstly, a hierarchical database provides access to distinct parts of CFD data (e.g. the list of points in a mesh), and manages their storage in terms of filing system location, and file formats and versions. The database is supported by a 'dictionary' which enables users to select data by comprehensible names, and also enables the CFD programmer to select sets of data for animation. Secondly, a library has been developed whose programming interface gives the CFD developer access to techniques for processing mesh and field data. These include operations involving a knowledge of the CFD simulation algorithms such as interpolation and field algebra, and operations involving graphical interpretation of the data, such as analytical geometry for visualisation programming. Programs written using this library can easily be installed as tools in AVS.

Parallel processing for both shared and distributed memory architectures is anticipated in the library's design (although not yet implemented) in a fine-grain, multiple-instruction multiple-data scheme as already described for SPEED itself [22]. The cells of a mesh are distributed by a master processor to several parallel slave processors, on the basis of minimum communication between processors. An

operation on a field is transformed by the master process into operations on several subfields communicating asynchronously. A complete program invocation consists of a sequence of master processes.

4.4 Implementation

A prototype system is being developed based on AVS, in combination with UNIX, the X WINDOW SYSTEM, and POSTSCRIPT. The database and field manipulation software are being implemented as a class library written in C++ [21]. The use of FORTRAN in the present pre- and processors (as mentioned in section 3.2) has made it difficult to maintain code quality, and has restricted programming style. Use of C++ enables a more rigorous methodology to be applied, while simplifying the library programming interface.

Because of the very large size of CFD data, the software must automatically manage in-core memory usage. Memory management is being implemented in the container classes of the library (such as lists), and can be automatic or application-controlled.

4.5 Evaluation

Apart from commercial considerations like current availability, the design of AVS is generally validated by the present research, and initial impressions are that the functional and operational problems of the CAD system interface are overcome. One issue is raised: if the toolkit approach proves to be useful, we foresee a need for standardisation in the interface between tools and environments. In principle a set of tools might be ported to several environments. A toolkit environment developed by a third party software developer would make broad porting of a single environment more likely.

The main omission of our research is mesh generation in a toolkit environment. Given a surface geometry describing a flow domain, mesh generation could be undertaken using an implicit solution method such as the advancing front technique, which is desirable anyway for complex engine configurations. A mesh generation tool could then be installed in the environment.

This leaves the problem of geometry definition. Presently available solid modellers tend towards two extremes: integrated CAE systems, and imprecise modellers for applications like video animation. For our purposes a simple but precise modeller is needed, limited to the specific task of engineering geometry definition. Such a modeller might be installed in the environment as another tool, its user interface and data types integrated with the toolkit environment. A complete CFD system would thus be created with a single user interface and data environment, but separated from other CAE activities.

Alternatively, in a hybrid scheme, translators between the environment and a CAD system could be installed in the environment as tools, enabling surface and mesh descriptions as well as simulation results to pass each way. The translators would be useful to a larger number of users than ones specific to SPEED, and thus more viable to develop. There is still no fully satisfactory standard for data exchange. Possibly STEP and its associated 'information modelling language' EXPRESS [23], a developing ISO standard (and replacement for IGES), will provide a suitable base technology in this area. In the extreme, the role of the CAE system could be reduced to data transfer between different application environments.

Meanwhile, conventional CAD vendors are now developing better programming features (based on a dedicated high-level language), and specific support for CFD. It may be possible with such features to retain the conventional arrangement of cooperating CAD and CFD systems and improve the interactive facilities. Three issues persist. Any code written in the dedicated CAD system language will lock the CFD developer into the CAD product; a large CAD system interface will still be necessary; and CFD developers will probably remain restricted in their ability to use their own algorithms for mesh generation and visualisation within the CAD system. CAD systems may of course move towards the open modular design of toolkit environments. The great strength of CAD systems is their ability to integrate multi-disciplinary CAE, and they have the advantage of a large number of existing users. However, the current complexity and expense of such systems appears to limit their present suitability for CFD.

5. CONCLUDING REMARKS

We have explored two paradigms for the design of interactive software for CFD: a two-way interface between CAD and CFD systems; and an alternative based on a toolkit environment with an open modular design. The toolkit approach has been shown to be feasible, promising to unify and greatly improve the user interface of the CFD code, and thereby create a more explorative and flexible environment for CFD. By contrast, CAD systems may evolve to offer appropriate interactive facilities for CFD, and therefore fully incorporate CFD within the established CAE arena. In our view they will only do this if they offer more comprehensive facilities for application, database, and user interface management. Their designers must recognise the precept that users need to implement their own algorithms in a programming language of their choice for many aspects of the CFD user interface. The toolkit approach, which we have tested through use of the AVS visualisation system, is clearly desirable from a user's perspective. Its potential may best be realised if geometry modelling tools are included, either within the environment, or through a well engineered interface to a conventional CAD system.

Acknowledgements

The contributions of B.N. Adamson and H.G. Weller are gratefully acknowledged. We would also like to acknowledge funding by the Joint Research Committee (a consortium of Rover, Peugeot, Fiat, Renault, Volkswagen and Volvo), the European Commission and Hewlett Packard, and the loan of computing equipment by the Rutherford Appleton Laboratory.

References

(1) Gosman, A. D. and Marooney, C. J., "Development and Validation of Computer Models of In-cylinder Flow and Combustion in Diesel and Spark Ignition Engines". *Autotech 89*, Birmingham, England, 1989 (IMechE C399/169)

(2) Adamson, B., Gosman, A. D., Marooney, C. J., Nasseri, B., Theodoropoulos, T., "A New Method for Flow Prediction in Internal Combustion Engines". *Comodia 90*, Kyoto, Japan, 1990

(3) Adamson, B., Gosman, A. D., Hill, C., Marooney, C. J., Nasseri, B., Sarantinos, M., Theodoropoulos, T., and Weller, H. "Simulation of in-cylinder flow and combustion in reciprocating engines", *Science and Engineering on Supercomputers*, October 1990, Ed. Pitcher, E. J. (Computational Mechanics Publications & Springer Verlag)

(4) Riquicha, A. and Voelcker, H., "Solid Modeling: A Historical Summary and Contemporary Assessment". *Computer Graphics and Applications*, 1982, 2, 2, 9-24 (IEEE)

(5) Casale, M.S., Stanton, E.L., "An overview of Analytic Solid Modeling". *Computer Graphics and Applications*, Feb. 1985 (IEEE)

(6) Casale, M.S., "Free-Form Solid Modeling with Trimmed Surface Patches". *Computer Graphics and Applications*, Jan 1987 (IEEE)

(7) Thompson J.F., Warsi Z.U.A, Mastin C.W., "Numerical Grid Generation, Foundations and Applications". 1985 (North-Holland)

(8) Lohner, R., "Fast Graphics Algorithms Using Unstructured Grids", *SIGGRAPH 90, course notes* 27, Chapter 7, 1990 (ACM SIGGRAPH)

(9) McCormick, B., DeFanti, T., Brown, M., "Visualisation in Scientific Computing". *Computer Graphics special issue*, 1987, 21, 6 (ACM SIGGRAPH)

(10) Upson, C., "The Visualisation of Volumetric Data". Computer Graphics 89, London, 29-38, 1989 (Blenheim Online Publications, Middx. UK)

(11) Schneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages". *Computer*, August 1983 (IEEE)

(12) Verplank, W., "Designing Graphical User Interfaces". *CHI 88 tutorial notes*, 1988 (ACM SIGCHI)

(13) Smith, D. C., Harslem E. F., Irby, C. H., Kimball, R. B., "Designing the Star User Interface". *Byte*, April 1982.

(14) *Human Interface Guidelines: The Apple Desktop Interface*, 1987 (Addison-Wesley)

(15) O'Reilly, T., Quercia, V., Lamb, L., *X Window System User's Guide for version 11*, 1988, (O'Reilly and Associates, Inc.)

(16) Parnas, D., "On the criteria to be used in decomposing systems into modules". *Commun. ACM*, 1972, 15, 12, 1055-1058 (ACM)

(17) Upson, C., Faulhaber, T. Jnr., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., van Dam, Andries, "The Application Visualization System: A Computational Environment for Scientific Visualisation". *Computer Graphics and Applications*, July 1989, 30-42 (IEEE)

(18) Van de Wettering, M., "The Application Visualization System - AVS 2.0", *Pixel* July/August 1990 (Pixel Communications, Inc., Watsonville CA.)

(19) Gelberg, L., Kamins, D., Parket, D., Sacks, J., "Visualization Techniques for Structured and Unstructured Scientific Data". *SIGGRAPH 90, course notes* 27, 1990 (ACM SIGGRAPH)

(20) *AVS Tutorial Guide (version 3.0 Beta Test)*, November 1990 (Stardent Computer)

(21) Stroupstrup, B. *The C++ Programming Language*, 1987 (Addison-Wesley)

(22) Weller, H., and Marooney, C. J., "Parallelism and CFD", *EASE 90 Symposium*, April 1990, UMIST, Manchester, England.

(23) International Standards Organisation *Information Modelling Language Express, Language Reference Manual*, May 1989 (ISO TC184/SC4/WG1 - N362)

UNIV LONDON BIBL